Taxonomy of DEVS Subclasses for Standardization

Moon Ho Hwang
Arizona Center for Integrative Modeling and Simulation
Electrical and Computer Engineering
University of Arizona, Tucson, AZ, USA
Email: mhhwang@ece.arizona.edu

Abstract—This paper clarifies the class hierarchy among DEVS subclasses in terms of their expressiveness. We define the expressiveness of a given formalism as the scope of accepting event segments by the formalism. In this paper, we interpret that DEVS formalism can be nondeterministic as well as deterministic. Based on this interpretation, inclusion relationship between several different formalisms including Timed Automaton are shown. As a consequence, this paper contributes to DEVS standardization at a level of formalism. The classes clarify the power of DEVS and enable the development of powerful sub-languages that make it easier to work in particular domains - and to teach the basics of DEVS - as we do with a subclass, called FDDEVS.

I. INTRODUCTION

Since 1976 when Zeigler introduced Discrete Event System Specification (DEVS) to the public[18], a number of variants [15][13] [5] [10] [14] [8] have come from the formalism. We can understand some of these variants as extensions of, the equivalent classes of, or subclasses of DEVS depending on interpretation of DEVS. The comparison requests against different formalisms are raised not just from the DEVS variations but from other formalisms including Timed Automata that has been paid tremendous attentions in two decades even from the DEVS community [3][7][4].

The contribution of the paper is to provide taxonomy of DEVS subclasses, especially focusing on those having finite events and states. Based on the expressiveness clarification of DEVS classes, we are able to define their syntax standardization using any programming or markup languages, as we did for a subclass, called FDDEVS [11][16][2].

The paper is organized as follows. Section II defines a general class of timed event systems and its behavior based on the set of admissible event segments. Based on the general class of timed event systems, Section III gives a review of DEVS classes, we are able to define their syntax standardization using any programming or markup languages, as we did for a subclass, called FDDEVS.

Section II defines a general class of timed event systems, Section III gives a review of DEVS classes, we are able to define their syntax standardization using any programming or markup languages, as we did for a subclass, called FDDEVS.

II. TIMED EVENT SYSTEMS

To support the mathematical common foundation across different formalisms, we define the prerequisite framework along with trajectories and a general timed system class in this section.

A. Trajectories and Event Segments

First, we would like to introduce two symbols of time:

\[ T = [0, \infty) \] and \[ T^{\infty} = [0, \infty) \cup \{\infty\} \]

where \( T \) is the set of non-negative real numbers, while \( T^{\infty} \) is \( T \) with infinity \( \infty \) such that \( \infty + t = \infty \) and \( \infty - t = \infty \) for any \( t \in T \).

1) Trajectories: Given an arbitrary set \( Z \) and a time base \( T \), a segment is a sequence of pairs of \( (z, t) \) where \( z \in Z \) and \( t \in T \), which are treated as one group within a certain time period \( [t_i, t_n] \subseteq T \). A segment is called linear if one pair \( (z, t) \) defines all pairs \( (z + c \times dt, t + dt) \) over time \( [t, t + dt] \), \( dt \in T \), where \( c \) is its slope (see Figure 1(b)). A segment is called constant if the slope is zero (see Figure 1(c)). Since our interesting formalism, DEVS, assumes the slope of a given piecewise linear segment is either one of zero or one [17], from now on, a linear segment means a linear segment with \( c=1 \), differentiating the constant segment.  \(^1\)

We introduce an operator \( \oplus \) to symbolize the time passage \( dt \in T \) over arbitrary \( Z \) within a segment. For \( z \in Z \), \( z \oplus dt = z \) if \( Z \) is piecewise constant, \( z \oplus dt = z + dt \) otherwise. Let \( Z = \times Z_i \) be a compound set of arbitrary \( Z_i \). Then, for an element \( z = (\ldots, z_i, \ldots) \in Z \) at time \( t \in T \), its time passage by \( dt \in T \) within a segment is \( z \oplus dt = (\ldots, z_i', \ldots) \) where

\[ z_i = \begin{cases} z_i = z_i & \text{if } Z_i \text{ is piecewise constant} \\ z_i = z_i + dt & \text{otherwise}. \end{cases} \]

\(^1\)Two variant classes of DEVS can handle the general continuous trajectory: GDEVS [14] uses the \( n \)-order polynomials; DEV/DESS [15] utilizes differential equations. However, GDEVS and DEV/DESS are out of this paper’s scope.
A trajectory is a sequence of segments. A trajectory is called piecewise constant (respectively, piecewise linear) if it consists of constant segments (respectively, linear segments). There is a special class of constant trajectories over an event set, called the event trajectory as addressed in the following section.

2) Event Trajectories:
   a) Events and Time: An event is a label that abstracts a change of systems under consideration. A timed event is a pair \((z, t)\) of an event \(z\) in \(Z\) and a time \(t\) in \(T\) when the event occurs. The symbol \(e \notin Z\) is called the null event for indicating “nothing changes” so \((e, t)\) indicates nothing changes at time \(t\) in \(T\). \(e_{[t_1,t_u]}\) is called a null event segment over a time interval \([t_1, t_u]\) \(\subseteq T\), which denotes “nothing but time passage over \([t_1, t_u] \)”. An unit event segment is either a timed event or a null event segment. Therefore, an unit event segment informs us that “something happens or time is passing in a time interval”.

   b) Concatenation of Event Segments: Given an event set \(Z\), concatenation of two unit event segments \(\omega\) over \([t_1, t_2]\) and \(\omega'\) over \([t_3, t_4]\) is denoted by \(\omega \omega'\), and defined if \(t_2 = t_3\). A multi-event segment \((z_1, t_1)(z_2, t_2) \ldots (z_n, t_n)\) over \(Z\) and \([t_1, t_u] \subseteq T\) is concatenations of unit event segments \(e_{[t_1,t_1]}, (z_1, t_1)(z_2, t_2) \ldots (z_n, t_n)\) and \(e_{[t_n,t_u]}\) where \(t_1 \leq t_2 \leq \ldots \leq t_{n-1} \leq t_n \leq t_u\). Similarly, concatenation of two multi-event segments: \(\omega\) over \([t_1, t_2]\) and \(\omega'\) over \([t_3, t_3]\) is defined as \(\omega \omega'\) if \(t_2 = t_3\).

   For example, let \(Z=\{\text{push}, \text{pop}\}\) be an event set for the toaster shown in Figure 1(a). If we observe an event segment \(\omega_1 = (\text{push},25)(\text{pop},50)\) during \([0,60]\), and another \(\omega_2 = (\text{push},80)(\text{pop},105)\) during \([60, 120]\), then we say that we observe \(\omega = \omega_1 \omega_2 = (\text{push},25)(\text{pop},50)(\text{push},80)(\text{pop},105)\) over \([0,120]\) (see Figure 1(d)).

3) Universal Timed Languages: The universal timed language over an event set \(Z\) and a time interval \([t_1, t_u] \subseteq T\) is denoted by \(\Omega_{Z,[t_1,t_u]}\), and is defined as the set of all possible event segments. Formally,

\[
\Omega_{Z,[t_1,t_u]} = \{ (z, t)^* : z \in Z \cup \{ \epsilon \}, t \in [t_1, t_u] \}
\]

where \((z, t)^*\) denotes a none or multiple concatenations of null or timed events. A timed language over an event set \(Z\) and a timed interval \([t_1, t_u] \subseteq T\) is a set of event segments over \(Z\) and \([t_1, t_u]\). If \(L\) is a language over \(Z\) and \([t_1, t_u]\), then \(L \subseteq \Omega_{Z,[t_1,t_u]}\).

B. Timed Event Systems

A timed event system (TES) is a system that changes its state along with associated event segments. We would take a look at it from the standpoints of structure and behavior.

1) The structure of TESs:

**Definition 1 (Timed Event Systems)** A timed event system (TES) is a structure

\[
G = (Z, Q, q_0, Q_A, \Delta)
\]

2Note that a timed event \((z, t)\) of an unit event segment can be written \((z, [t, t])\) by using the singleton interval \([t, t]\) which is equivalent to \(t\).

where

- \(Z\) is the set of events;
- \(Q\) is the set of states; \(q_0 \in Q\) is the initial state variable;
- \(Q_A \subseteq Q\) is the set of accept states;
- \(\Delta : Q \times \Omega_{Z,T} \rightarrow Q\) is the state trajectory function that defines how a state \(q\) changes to another \(q'\) along with an event segment \(\omega \in \Omega_{Z,T}\).

If \(\omega\) is concatenation of two event segments, i.e. \(\omega = \omega_1 \omega_2\), then \(\Delta(q, \omega) = \Delta(\Delta(q, \omega_1), \omega_2)\). In general if \(\omega\) is concatenation of \(n\)-event segments, i.e. \(\omega = \omega_1 \omega_2 \ldots \omega_n\), where \(n > 1\) then

\[
\Delta(q, \omega) = \Delta(\ldots \Delta(\Delta(q, \omega_1), \omega_2) \ldots), \omega_n)
\]

To define determinism and nondeterminism of TES, we define those of a function as the following definition.

**Definition 2 (Nondeterminism and Variables)** Given two arbitrary sets (including the empty set \(\emptyset\)) \(A\) and \(B\), a function \(f : A \rightarrow B\) is called deterministic if for an element \(a \in A\), \(f(a)\) is identical any time. Otherwise, \(f\) is called non-deterministic. The function \(f\) is called a variable if the domain of \(f\) is the empty set \(\emptyset\) which means a variable \(f\) is a domain independent function. In addition, we call the function \(f\) a constant variable if \(f\) is a variable and deterministic.

For example, assume that \(A\) and \(B\) are real numbers, then \(f(a) = a + 5\) is deterministic. Given two sets \(A = \{\text{coin}, \text{dice}\}\) and \(B = \{\text{head}, \text{tail}, 1,2,3,4,5,6\}\), if the function \(f\) indicates outcomes of tossing a coin or a dice, \(f\) is non-deterministic.

**Definition 3 (Deterministic and Non-Deterministic TESs)** A TES \(G = (Z, Q, q_0, Q_A, \Delta)\) is deterministic if (1) \(q_0\) is a constant variable, and (2) \(\Delta\) is deterministic. Otherwise, \(G\) is non-deterministic.

2) Behaviors of TESs: Given a TES \(G = (Z, Q, q_0, Q_A, \Delta)\), an event segment \(\omega \in \Omega_{Z,T}\) is called a behavior if there is such a case that \(\Delta(q_0, \omega)\) reaches an accept state \(q \in Q_A\). The set of behaviors of \(G\) is called its language. The language of a given TES \(G\) can be depend on the observation time of behaviors.

Let \(t\) be the observation time length. If \(0 \leq t < \infty\), \(t\)-length observation language of \(G\) is denoted by \(L(G, t)\), and defined

\[
L(G, t) = \{ \omega \in \Omega_{Z,[0,t]} : \exists \text{ the case : } \Delta(q_0, \omega) \in Q_A \}.\]

Notice that the reason why we need “there exists the case” is that we allow \(G\) to be non-deterministic as well. \(^3\)

Based on Equation (3), the infinite time length is defined by sending \(t\) to infinity. Giving an infinite-observation event segment \(\omega \in \lim_{t \rightarrow \infty} \Omega_{Z,[0,t]}\) and a TES \(G\), let \(inf(\Delta(q_0, \omega))\) denote the set of \(G\’s\) states which \(\omega\) visits infinitely many

\(^3\) A TES \(G\) is called legitimate or a non-Zeno system if the number of events in any \(\omega \in L(G, t)\) is finite.
times or stays infinitely long. Then the infinite length observation language of \( G \) is denoted by \( L(G, \infty) \), and defined

\[
L(G, \infty) = \{ \omega \in \lim_{t \to \infty} \Omega_{Z,[0,t)} : \exists \text{ the case s.t. } in\phi(\Delta(q_0, \omega)) \subseteq Q_A \}. 
\]

Let us use the notation \( L(G) \) denoting the set of behaviors of \( G \) regardless of the observation time \( t \) in case \( t \) is not significant. The language that is all accepting behaviors of an instance or a class of TESs provides their expressiveness.

**Definition 4 (Expressiveness Inclusion)** Suppose that \( A \) and \( B \) are two TES classes. Then we say that the expressiveness of \( A \) is less than or equal to that of \( B \), denoted by \( E(A) \subseteq E(B) \), if any given instance \( A \) of \( B \) exists a homomorphism \( \omega \) such that \( L(a) = L(b) \). We say that the expressiveness of \( A \) is less than that of \( B \), denoted by \( E(A) \subset E(B) \), if \( E(A) \subseteq E(B) \) but there is no instance \( A \) of \( B \) such that \( L(a) = L(b) \). We say that the expressiveness of two classes \( A \) and \( B \) are the same, denoted by \( E(A) = E(B) \), if \( E(A) \subseteq E(B) \) and \( E(B) \subseteq E(A) \). \( \square \)

**Definition 5 (Subclass, Equivalent class, and Superclass)** Suppose that \( A \) and \( B \) are two TES classes. Then \( A \) is called a subclass of \( B \) and \( B \) is called a superclass of \( A \) if \( E(A) \subset E(B) \). \( A \) is called a subclass or equivalent class of \( B \) and \( B \) is called a superclass or equivalent class of \( A \) if \( E(A) \subseteq E(B) \). \( A \) and \( B \) are called the equivalent classes if \( E(A) = E(B) \). \( \square \)

**Definition 6 (Homomorphic Systems)** Given two TESs \( G = (Z,Q,q_0,Q_A,\Delta) \) and \( H = (Z,Q',Q'_A,q'_0,\Delta') \), the homomorphism is a mapping \( f : Q \rightarrow Q' \) such that \( \{f(q) : q \in Q_A\} \subseteq Q'_A \) and \( f(\Delta(q_0,\omega)) = \Delta'(q'_0,\omega) \) for all \( \omega \in \Omega_{Z,T} \). \( H \) is called a homomorphic system of \( G \) if there exists a homomorphism \( f \).

\( \square \)

**Proposition 1** \( L(G) \subseteq L(H) \) if \( H \) is a homomorphic system of \( G \).

**Proof:** Let \( f \) be the existing homomorphism \( f(\Delta(q_0,\omega)) = \Delta'(q'_0,\omega) \) for all \( \omega \in \Omega_{Z,T} \) and \( \{f(q) : q \in Q_A\} \subseteq Q'_A \). For \( \{q_0,0\} \in \Omega_{Z,T} \),

\[
 f(\Delta(q_0,e_{[0,0]})) = f(q_0) = q'_0 = \Delta'(q'_0,0) \text{ } \Delta(q_0,\omega) \in Q_A \text{ implies } f(\Delta(q_0,\omega)) = \Delta'(q'_0,\omega) \in Q'_A. 
\]

In other words, if \( \omega \in L(G) \) then \( \omega \in L(H) \) which means \( L(G) \subseteq L(H) \). \( \square \)

### III. Discrete Event System Specification (DEVS)

Because of page limitation we just give a review of the DEVS formalism in terms of its atomic structure in this paper. For coupled DEVS models, the reader can refer to [17]. DEVS is a TES class, which defines system dynamics associating I/O events and time.

**Definition 7 (DEVS)** A DEVS model is given by the 8-tuple

\[
M = (X,Y,S,s_0,\tau_a,\tau_{ext},\tau_{int},\lambda)
\]

where

- \( X \) and \( Y \) are the set of input events and the set of output events, respectively;
- \( S \) is the set of states;
- \( s_0 \in S \) is the initial state variable;
- \( \tau_a : S \rightarrow T^\infty \) is the time advance function which is used to determine the lifespan of a state;
- \( \tau_{ext} : Q \times X \rightarrow S \) is the external transition function which defines how an input event changes a state of the system, where \( Q = \{ (s,e) \in Q, e \in (T \cap [0,\tau_a(s)]) \} \) is the set of total states, and \( e \) is the piecewise linear elapsed time since last event;
- \( \tau_{int} : S \rightarrow S \) is the internal transition function which defines how a state of the system changes internally (when the elapsed time reaches to the lifetime of the state);
- \( \lambda : S \rightarrow Y^\phi \) is the output function where \( Y^\phi = Y \cup \{ \phi \} \) and \( \phi \notin Y \) is a silent event or an unobservable event. \( \square \)

**Definition 8 (Deterministic and Non-deterministic DEVSs)** A DEVS model \( M \) is called deterministic if (1) \( s_0 \) is a constant variable, and (2) characteristic functions \( \tau_a, \tau_{ext}, \tau_{int} \) and \( \lambda \) are deterministic. Otherwise, \( M \) is called nondeterministic. \( \square \)

**Definition 9 (Behavior of DEVS)** Let \( M = (X,Y,S,s_0,\tau_a,\tau_{ext},\tau_{int},\lambda) \) be a DEVS model. Then the behavior of \( M \) is explained by a TES \( G(M) = (Z,Q,q_0,Q_A,\Delta) \) where the event set \( Z = X \cup Y^\phi \);

The state set \( Q = Q_A \cup Q_{\bar{A}} \) where \( Q_{\bar{A}} = \{ s \in S \} \) called the non-accept state in which \( s \) is piecewise constant.

The initial state variable \( q_0 = (s_0,0) \in Q_A \).

The state trajectory function \( \Delta : Q \times \Omega_{Z,T} \rightarrow Q \) is defined for a total state \( q = (s,e) \in Q \) at time \( t \in T \) and an event segment \( e \in \Omega_{Z,[t,t+dt]} \).

For a null event segment, i.e. \( \omega = e_{[t,t+dt]} \),

\[
\Delta(q,\omega) = q \oplus dt = \begin{cases} (s \oplus dt, e + dt) & \text{if } q \in Q_A \\ (x) & \text{otherwise} \end{cases}
\]

which is a timed passage where \( \oplus \) is defined in Equation (1).

For a timed input event, i.e. \( \omega = (x,t) \) where \( x \in X \)

\[
\Delta(q,\omega) = \begin{cases} (\tau_{ext}(s,e,x),0) & \text{if } q \in Q_A, \\ (x) & \text{otherwise} \end{cases}
\]

For a timed output or silent event, i.e. \( \omega = (y,t) \) where \( y \in Y^\phi \)

\[
\Delta(q,\omega) = \begin{cases} (\tau_{int}(s),0) & \text{if } q \in Q_A, e = \tau_a(s), y = \lambda(s) \\ (x) & \text{otherwise} \end{cases}
\]

If \( \omega \) is a multi-event segment, we can apply Equation (2) using above three primitive cases described in Equations (5), (6), and (7).

Observe the condition when \( G(M) \) moves from an accept state \( q \in Q_A \) into the non-accept state \( s \in Q_{\bar{A}} \) is the negation of
the first statement’s condition of Equation (7); either one of \( e \neq ta(s) \) nor \( y \neq \lambda(s) \). Once \( G(M) \) gets into \( \pi \), the last two statements of Equations (6) and (7) as well as the time passage of Equation (5) make \( G(M) \) stay on the non-accept state \( s \) forever.

For the sake of simplicity, we will use \( L(M) \) for the behavior of the DEVS model \( M \) instead of using \( L(G(M)) \) where \( G(M) \) denotes the TES corresponding to \( M \) defined in Definition 9.

IV. CLOCK-BASED DEVS (CDEVS)

As we mentioned earlier, we assume that there exist two different state trajectories in our DEVS framework: one is piecewise constant, the other is piecewise linear whose slope \( c \) is one. A variable having piecewise linear together with \( c = 1 \) can be understood an elapsed time of a clock since its last resetting. From the viewpoint of the internal transition of DEVS, the elapsed time can reach up to a time out or a schedule value of a clock. Thus we would pay a special attention to clocks.

Based on the observation of clocks, we partition the state of DEVS into two groups: one is non-clock related, and the other is clock-related. For each clock, there are two variables: the schedule and the elapsed time as the following formal definition.

**Definition 10 (CDEVS)** A clock-based DEVS (CDEVS) is a tuple

\[
M_C = (X,Y,S,s_0,\delta_x,\delta_y)
\]

where

- \( X \) and \( Y \) are the input and output events sets, respectively.
- \( S = (S_d, \times \ (T^\infty, T))_c \) is the set of states that consists of two disjoint sets
  - \( S_d \) is the set of piecewise constant states which is called the set of discrete states.
  - \( C \) is the set of clock names. Each clock \( c \in C \) has two clock variables
    * \( \sigma_c \in T^\infty \): the schedule of clock \( c \in C \), which is piecewise constant.
    * \( e_c \in T \cap [0,\sigma_c] \): the elapsed time of clock \( c \in C \), which is piecewise linear.
- Thus \( s = (s_d,\ldots,\sigma_c,e_c,\ldots) \) denotes at phase \( s_d \in S_d \), each clock \( c \)’s schedule \( \sigma_c \) and the elapsed time \( e_c \).
- \( s_0 = (s_{d_0},\ldots,\sigma_{c_0},0,\ldots) \in S \) is the initial state variable where \( s_{d_0} \in S_d \) is the initial discrete state variable and \( \sigma_{c_0} \) is the initial schedule of clock \( c \in C \) where the elapsed time \( e_c = 0 \) for all \( c \in C \).
- \( \delta_x : S \times X \rightarrow S \) is the external transition function. For a given \( s \in S \) and \( x \in X \), \( \delta_x(s,x) = s' \) defines how an input \( x \) changes the state \( s \) to \( s' \).
- \( \delta_y : S \rightarrow Y^\phi \times S \) is the output and internal transition function; For a given \( s \in S \), \( \delta_y(s) = (y,s') \) defines how this system model generates an output event \( y \), at the same time, internally changes the state from \( s \) to \( s' \).

**Definition 11 (Behavior of CDEVS)** Given a CDEVS \( M_C = (X,Y,S,s_0,\delta_x,\delta_y) \), let the remaining time function \( tr : S \rightarrow T^\infty \) be

\[
tr(s_d,\ldots,\sigma_c,e_c,\ldots) = \min_{\nu \in C} \{ \tau_c - \epsilon_c \}
\]

where \( (s_d,\ldots,\sigma_c,e_c,\ldots) \in S \).

Then the TES \( G(M_C) = (Z,Q,A_\tau,\tau_0,\Delta) \) defines the behavior of \( M_C \) as follows. The set of events is \( Z = X \cup Y \).

The set of states is \( Q = Q_A \cup Q_\tau \) where \( Q_A = \{ (s,t_s,t_e) : s \in S, t_s \in T^\infty, t_e \in T \cap [0,t_s] \} \) and \( Q_\tau = \{ s \notin S \} \) in which \( \sigma \) and \( s \in S \) is piecewise constant, and \( e \) is piecewise linear. The initial state variable is given

\[
g_0 = (s_0,t_{s_0},t_{e_0}) = ((s_{d_0},\ldots,\sigma_{c_0},0,\ldots),tr(s_0),0).
\]

The state trajectory function \( \Delta : Q \times \Omega_{\tau,T} \rightarrow Q \) is defined for \( q \in Q \) and an unit segment \( w \) as below.

For a null segment \( w = \epsilon_{[t,t+d]} \) and \( t,d \in T \),

\[
\Delta(q,w) = \begin{cases} (s_d,\ldots,\sigma_c,e_c + dt,\ldots),t_s,t_e + dt) & \text{if } q \in Q_A \\ (\tilde{s}) & \text{otherwise} \end{cases}
\]

For a timed input event \( \omega = (x,t) \), \( x \in X \), and \( t \in T \),

\[
\Delta(q,\omega) = \begin{cases} (\delta_x(s,x),tr(\delta_x(s,x)),0) & \text{if } q \in Q_A \\ (\tilde{s}) & \text{otherwise} \end{cases}
\]

For a timed output event \( \omega = (y,t) \), \( y \in Y^\phi \), and \( t \in T \),

\[
\Delta(q,\omega) = \begin{cases} (s',tr(s'),0) & \text{if } t_x = t_s,\delta_y(s) = (y,s') \\ (\tilde{s}) & \text{otherwise} \end{cases}
\]

Equation (11) says that the enabling condition of the output and internal transition \( \delta_y \) is \( t_x = t_s \), i.e., \( 3c^* \in C : \sigma_{c^*} = e_{c^*} \).

In other words, executing the timed output event transition when \( t_x \neq t_s \), the system gets into the non-accept state \( \tilde{s} \).

If \( \omega \) is a multi-event segment, we can apply Equation (2) using above three primitive cases described in Equations (9), (10), and (11).

As we defined deterministic and nondeterminism for DEVS class, a CDEVS model is called deterministic if (1) \( s_0 \) is a constant variable, (2) characteristic functions \( \delta_x \) and \( \delta_y \) are deterministic. Otherwise, it is called non-deterministic.

**Example 1 (CDEVS Toaster Model \( M_1 \))** Figure 2(a) illustrates a toaster which is a nondeterministic CDEVS \( M_1 = (X,Y,S,s_0,\delta_x,\delta_y) \) where \( X = \{\text{?push,?repair}\} \); \( Y = \{\text{?pop}\} \); \( S = \{ (s_d,\sigma_c,e_c) : s_d \in S_d,\sigma_c \in T^\infty, e_c \in T \cap [0,\sigma_c] \} \) where \( S_d = \{ \text{idle, busy, down} \} \) and \( C = \{ c \} \), i.e, there is only one clock name that is ‘c’. The initial state \( s_0 = (\text{idle}\infty,0) \). The external transition function is defined \( \delta_y(\text{idle}\infty,0,c) \) either one of \( \{ \text{down}\infty,0,c \} \) or \( \{ \text{busy},t,0 \} \) where \( t \in (20,30) \).

The \text{push} input event at \( s_d \in \{ \text{down}, \text{busy} \} \) is ignored such that \( \delta_x(s_d,\sigma_c,e_c) = (s_d,\sigma_c,e_c) \)
when \( s_d \in \{ \text{down}, \text{busy} \} \). The ?repair input event makes the toaster return to work such that \( \delta_e(s_d, \infty, e_c, ?\text{repair}) = (\text{idle}, \infty, e_c) \), however ?repair is ignored when \( s_d \in \{ \text{idle}, \text{busy} \} \) like \( \delta_s(s_d, \infty, e_c, ?\text{repair}) = (s_d, \infty, e_c) \).

The active discrete state triggering output and internal transition is only the state \( \text{busy} \) having \( \delta_y(\text{busy}) = (\text{pop}, \text{idle}, \infty, 0) \). In other words, there is no output or internal transition for the discrete state \( s_d \in \{ \text{down}, \text{idle} \} \) such that \( \delta_y(s_d, \infty, e_c) = (\varnothing, \infty, e_c) \).

To draw CDEVS model \( M_1 \), we use the conventions for in Figure 2(a): (1) Each node represents a discrete state \( s_d \in S_d \). (2) Each arc is either one of \( \delta_e \) or \( \delta_y \) together with [pre-conditions]/[post-conditions]. To make the figure simpler, we omitted the obvious conditions of clocks in the DEVS context: (1) \( e_c \in \mathbb{T} \land [0, \sigma_c] \) of the precondition of external transitions, (2) \( e_c = \sigma_c \) for the precondition of internal transitions, (3) \( e_c = 0 \) for the postcondition of internal transitions.

In \( M_1, \omega = (?\text{push}, 70)(?\text{push}, 80)(?\text{push}, 90)(?\text{pop}, 100) \in L(M_1), \omega = (?\text{push}, 10)(?\text{repair}, 55)(?\text{push}, 60)(?\text{pop}, 85) \in L(M_1) \) but \( \omega = (?\text{push}, 70)(?\text{pop}, 200) \notin L(M_1) \).

At this moment, we may have a question about the expressiveness of CDEVS. Is it greater than, or lesser than, equal to that of DEVS? Following lemmas and a theorem brings the answer.

**Lemma 1** \( E(\text{CDEVS}) \subseteq E(\text{DEVS}) \).

**Proof:** Let \( M_{\text{CDEVS}} = (X,Y,S,s_0,\delta_e,\delta_y) \) be a CDEVS model. Then we need to show there exists a DEVS \( M = (X,Y,S_{\text{DEVS}},s_0,\delta_e,\delta_y,\lambda) \) that behaves identically to the given \( M_{\text{CDEVS}} \), which means that \( L(M_{\text{CDEVS}}) \subseteq L(M) \), i.e., \( E(\text{CDEVS}) \subseteq E(\text{DEVS}) \). To show \( E(\text{CDEVS}) \subseteq E(\text{DEVS}) \), we will prove \( M \) is a homomorphic system of \( M_{\text{CDEVS}} \) based on Proposition 1.

Let \( M \)'s the set of states \( S_{\text{DEVS}} = \{(s,\sigma) : s \in S, \sigma \in \mathbb{T}^\infty \} \) where \( \sigma \) is piecewise constant. The idea introducing \( \sigma \) is to trace the remaining time of \( s \) given by \( t_r(s) \).

Therefore, the initial state variable

\[
s_{0\text{G}} = (s_0, \sigma_0) = (s_0, t_r(s_0)). \tag{12}
\]

The time advance function \( t_a : S_G \to \mathbb{T}^\infty \) returns just the \( \sigma \) value such that given a state \( s_G = (s, \sigma) \in S_G \),

\[
ta(s_G) = ta(s, \sigma) = \sigma. \tag{13}
\]

The external transition function \( \delta_e : \mathcal{Q} \times X \to S_{\text{DEVS}} \) is given for \( q = (s, \sigma, e) \in \mathcal{Q}, s \in X \),

\[
\delta_e(s, \sigma, e, x) = (\delta_e(s, x), tr(\delta_e(s, x))). \tag{14}
\]

The internal transition function \( \delta_i : S_G \to S_{\text{DEVS}} \) is given for \( (s, \sigma) \in S_G \) and \( \delta_i(s) = (y, s') \)

\[
\delta_i(s, \sigma) = (s', tr(\delta_i(s, \sigma))). \tag{15}
\]

The output function \( \lambda : S_{\text{DEVS}} \to \mathbb{Y}^o \) is given for \( (s, \sigma) \in S_G \) and \( \delta_y(s) = (y, s') \),

\[
\lambda(s, \sigma) = y. \tag{16}
\]

Let \( G(M_C) = (Z,Q_0,A,T,\Delta) \) and \( G(M) = (Z', Q', A_0, T, \Delta') \) be the TES corresponding CDEVS \( M_C \) and DEVS \( M \) introduced in Definitions 11 and Definition 9, respectively.

Let’s define a function \( f : Q \to Q' \) such that for a given \( q = (s,t_x,t_e) \in Q \),

\[
f(q) = \begin{cases} 
(s,\sigma,e) = (s,t_x,t_e) & \text{if } q \in Q_A \\
\sigma' & \text{otherwise}
\end{cases}
\]

where \( s = (s_d,\ldots,\sigma_c,e_c,\ldots) \). We are going to show \( f \) is a homomorphism from \( M_C \) to \( M \).

Based on Definitions 9 and 11, since both models of \( M \) and \( M_C \) get trapped once they get into the non accept states \( \bar{s} \) and \( \bar{s}' \) for respectively \( M \) and \( M_C \), \( \Delta(\bar{s},\omega) = \bar{s} \) and \( f(\bar{s}) = \bar{s}' = \Delta'(\bar{s}',\omega) \). Let’s investigate the cases of the accept states \( q \in Q_A \) and \( q' \in Q_A' \).

It’s fact that \( f(q) : q = (s,t_x,t_e) \in Q_A \} = \{(q,t_x,t_e) : s \in Q_A \} = Q_A' = M.Q \} = \{q,\sigma,e) : q \in Q, \sigma \in \mathbb{T}, e \in \mathbb{T} \} \because \) because (1) when \( t_x \) is updated by \( tr(s) \), so is \( \sigma \); (2) when \( t_e \) is reset; (3) \( t_x \) and \( \sigma \) are piecewise constant, \( t_e \) and \( e \) are piecewise linear.

To show \( f(\Delta(q_0,\omega)) = \Delta'(q_0',\omega) \), we would like to use induction.

1. Checking the initial condition: For the initial state \( q_0 = (s_0, tr(s_0),0) \in Q_A, q'_0 = (s_0, tr(s_0),0) = f(q_0) \) by Equation (12).

2. Checking the general case: Suppose that for \( \omega_1 \in \Omega_{Z,T}\), \( \Delta(q_0,\omega_1) = q = (s,t_x,t_e) \in Q_A \) and \( \Delta'(q_0',\omega_1) = q' = (s,t_x,t_e) = f(q) \) by Equation (12).

2.1. Time passage Case: If \( \omega_2 = \omega_1 \epsilon_{t+dt} \) where \( t, dt \in \mathbb{T} \), \( \Delta(q_0,\omega_2) = \Delta(\Delta(q_0,\omega_1), \epsilon_{t+dt}) = \Delta(q, \epsilon_{t+dt}) = q + dt = (s + dt, t, t + dt) \) by Equation (9). In the meantime, by Equation (5), \( \Delta'(q_0',\omega_2) = \Delta'(\Delta'(q_0',\omega_1), \epsilon_{t+dt}) = \Delta'(q', \epsilon_{t+dt}) = q' + dt = (s' + dt, t_e + dt) = (s + dt, t_e + dt) \).

2.2. Timed Input Event: If \( \omega_2 = \omega_1 (x,t) \) where \( x \in X, t \in \mathbb{T} \), \( \Delta'(q_0,\omega_2) = \Delta(\Delta(q_0,\omega_1), (x,t)) = \Delta(q, (x,t)) = (\delta_x(s, x), tr(\delta_x(s, x))) \).
2.3. Timed Output Event: If \( \omega_2 = \omega_1(y, t) \) where \( y \in Y^e, t \in T \), \( \delta_y(s) = (y', s') : \Delta(q_0, \omega_2) = \Delta(\Delta(q_0, \omega_1), (y, t)) = \Delta(q_0, (y, t)) = \delta_{int}(s) = (s', \text{tr}(s), 0) \) by Equation (11). In the meantime, by Equations (7) and (15), \( \Delta'(q_0, \omega_2) = \Delta'(\Delta'(q_0, \omega_1), \omega_2) = \Delta'(q', (y, t)) = \delta_{int}(s, \sigma) = (s', \text{tr}(s), 0) = f(\Delta(q_0, \omega_2)). \)

By induction \( f(\Delta(q_0, \omega) = \Delta'(q_0', \omega) \) for all \( \omega \in \Omega_{ZT} \), so we can say \( f \) is a homomorphism and \( G(M) \) is a homomorphic system of \( G(M_C) \), in other word, \( L(G(M_C)) \subseteq L(G(M)) \). Since \( M_C \) and \( M \) are the representatives of the classes CDEVS and DEV, respectively, we can say \( E(CDEVS) \subseteq E(DVS) \).

Lemma 2 \( E(DEV) \subseteq E(CDEVS) \).

Proof: As we proved in Lemma 1, we use again the homomorphism, however, the direction is the opposite way.

Let \( M = (X, Y, S, s_0, \text{ta}, \text{dt}, \text{int}, \phi) \) be a DEV model. Then we need to show there exists a corresponding DEV \( M_C = (X, Y, S, s_0, \text{ta}, \delta_x, \text{dt}, \text{int}, \phi_C) \) where the set of states \( S_C = S \times \{C \in \{c'\} \} \), i.e., there is only one clock name is \( c' \).

We are going to synchronize \( \sigma_c = \text{ta}(s) \) and \( e_c = e \) where \( e \) is the elapsed time of a total state in \( M \).

First, the initial state \( s_0G = (s_0, \text{ta}(s_0), 0) \).

The external transition function \( \delta_x : S_C \times X \rightarrow S_C \) is defined for \( (s, \sigma_c, e_c) \in S_C \)

\[ \delta_x(s, \sigma_c, e_c, x) = (\delta_{ext}(s, \sigma_c, x), \text{ta}(\delta_{ext}(s, \sigma_c, x)), 0). \]

The output internal transition function \( \delta_y : S_C \rightarrow Y^e \times S_C \) is defined for \( (s, \sigma_c, e_c) \in S_C \), then

\[ \delta_y(s, \sigma_c, e_c) = (\lambda(s), \delta_{int}(s), \text{ta}(\delta_{int}(s)), 0). \]

Let \( G(M) = (Z, Q, q_0, Q_A, \Delta) \) and \( G(M_C) = (Z, Q', q_0', \Delta') \) be the TES corresponding DEV \( M \) and CDEVS \( M_C \) introduced in Definitions 9 and Definition 11, respectively.

Let’s define a function \( f : Q \rightarrow Q' \) such that for a given \( q \in Q \),

\[ f(q) = (s, \sigma_c, e_c, x, t_e) = (s, \text{ta}(s), \sigma_c, e_c, x, t_e) \]

if \( q = (s, \sigma_c, e_c, t_e, t_e) \). Otherwise \( f(q) = s \).

Then \( \{ f(q) \} = \{ (s, \sigma_c, e_c, x, t_e) : q \in Q, (s, \sigma_c, e_c, x, t_e) \} \subseteq Q'. \)

To show \( f(\Delta(q_0, \omega) = \Delta'(q_0', \omega) \), we would like to use induction.

1. Checking the initial condition : For the initial state \( q_0 = (s_0, 0) \in Q_A, q_0' = (s_0, \text{ta}(s_0), 0, \text{ta}(s_0), 0) = f(q_0) \) by Equation (12).

2. Checking the general case : Suppose that for \( \omega_1 \in \Omega_{ZT} \), \( \Delta(q_0, \omega) = q = (s, e) \in Q_A \) and \( \Delta'(q_0, \omega) = q' = (s, \text{ta}(s), e, \text{ta}(s), e) = f(q) \).

2.1. Time passage case: If \( \omega_2 = \omega_1(\delta_{int}(s), t, dt) \) where \( t, dt \in T \):

\[ \Delta(q_0, \omega_2) = \Delta(\Delta(q_0, \omega_1), (\delta_{int}(s), t, dt)) = \Delta(q_0, (\delta_{int}(s), t, dt)) = (s, \text{ta}(s), e, \text{ta}(s), e) = q = \delta_{int}(s) = (s', \text{tr}(s), 0) = f(\Delta(q_0, \omega_2)). \]

2.2. Timed Input Event: If \( \omega_2 = \omega_1(x, t) \) where \( x, t \in T \):

\[ \Delta(q_0, \omega_2) = \Delta(\Delta(q_0, \omega_1), (x, t)) = \Delta(q_0, (x, t)) = (s, \text{ta}(s), e, \text{ta}(s), e) = q = \delta_{int}(s) = (s', \text{tr}(s), 0) = f(\Delta(q_0, \omega_2)). \]

2.3. Timed Output Event: If \( \omega_2 = \omega_1(y, t) \) where \( y \in Y^e, t \in T \):

\[ \Delta(q_0, \omega_2) = \Delta(\Delta(q_0, \omega_1), (y, t)) = \Delta(q_0, (y, t)) = \delta_{int}(s) = (s', \text{tr}(s), 0) = f(\Delta(q_0, \omega_2)). \]

By induction \( f(\Delta(q_0, \omega) = \Delta'(q_0', \omega) \) for all \( \omega \in \Omega_{ZT} \), so we can say \( f \) is a homomorphism and \( G(M_C) \) is a homomorphic system of \( G(M) \), in other word, \( E(M) \subseteq E(M_C) \). Since \( M \) and \( M_C \) are the representatives of the classes DEV and CDEVS, respectively, we can say \( E(CDEVS) \subseteq E(DEV) \).

Theorem 1 CDEVS is an equivalent class of DEV, i.e., \( E(CDEVS) = E(DEV) \).

Proof: By Lemmas 1 and 2.

Remark 1 CDEVS is nothing but DEV in terms of expressiveness. Thus, all DEVs researches that have been made, such as the abstract simulation algorithm, the property of closure under coupling operation [17], the hybrid system simulation methodology [12] are still valid to CDEVS.

Remark 2 The reason why we are going to use CDEVS instead of DEV is that CDEVS has the explicit expressiveness over the clocks. This characteristic brings us easier comparability against other formalisms especially if they are also based on explicit clocks, for example, Timed Automata.

V. Subclasses of DEV

This section clarifies the expressiveness of subclasses of DEV through CDEVS’s equivalent class, CDEVS. Moreover, we would focus on CDEVS having finite components.

Definition 12 (FCDEVS) A Finite CDEVS (FCDEVS) is a subclass of CDEVS \( M_{FC} = (X, Y, S, s_0, \delta_x, \delta_y) \) such that the sets of \( X, Y, S_d \) and \( C \) are finite.
A. Timed Automata (TA)

The timed automata [1] is another formalism in which a state transition along with an event can be enabled by a set of clock constraints written as intervals.

Let $t_i, t_u > 0$ be a rational-bounded interval then $t_i, t_u$ are non-negative rational numbers with infinity and $t_i \leq t_u$, the boundary conditions are \(\leq\) and \(\geq\). Let \(\mathbb{I}_Q\) be the all of possible rational-bounded intervals, and \(C\) be a set of clocks. Then a pair of \((c, \text{inv})\) where \(c \in C\) and \(\text{inv} \in \mathbb{I}_Q\) defines a clock \(c\)'s constraint with the interval \(\text{inv}\).

**Definition 13 (Timed Automata (TA))** A timed automaton is a tuple

\[
TA = (Z, C, P, p_0, I, T)
\]

where
- \(Z\) is the finite sets of events.
- \(C\) is the finite set of clocks.
- \(P\) is the finite set of phases which are piecewise constant.
- \(p_0 \in P\) is the initial phase variable.
- \(I : P \to \Phi(C)\) is the phase clock-constraint function where \(\Phi(C) = \{C \to \mathbb{I}_Q\}\) is the set of partial clock constraints. For example, \(C = \{c_1, c_2\}, \varphi = \{(c_2, [2, 5])\} \in \Phi(C)\). In this case of \(\varphi\), we say \(\varphi(c_2) = [2, 5]\), but \(\varphi(c_1)\) is undefined.
- \(T \subseteq P \times Z^\phi \times \Phi(C) \times \mathcal{P}(C) \times P\) is a set of transitions. A transition \((p, z, \varphi, C_R, p') \in T\) can be also inter-changeably represented by the notation \(\varphi^{-1}(c, \text{inv}(c)); C_R \times p'\), requires the enabling condition of \(I(p)\) and \(\varphi\) as a precondition, and the resetting clocks in \(C_R\) as a postcondition.

To define the behavior of a TA model, we need a set of mathematical building blocks. The notation \(\text{inv} = \triangleq t_i, t_u \triangleq \emptyset\) denotes the interval \(\text{inv}\) is empty. Notice that the interval class is closed under intersection.

Let \(\text{inv}(c)\) be an interval of a clock \(c\). Then a time zone of a finite set of clocks \(C\), denoted by \(\text{Zone}(C)\), is defined by \(\text{Zone}(C) = \times \text{inv}(c)\). \(\text{Zone}(C)\) is called empty (denoted by \(\text{Zone}(C) = \emptyset\)) if \(\forall c \in C : \text{inv}(c) = \emptyset\). Intersection of two clock zones \(\text{Zone}_1(C) \cap \text{Zone}_2(C) = \times \text{Zone}_2(C)|_c \cap \text{Zone}_1(C)|_c\), where \(\text{Zone}_1(C)|_c\) denotes \(\text{inv}(c)\) in \(\text{Zone}_1(C)\).

The zone making function \(M : \Phi(C) \to \mathcal{T}_{C,|c|}\) makes \(\text{Zone}(C)\) from a clock constraint function \(\varphi \in \Phi(C)\)

\[
M(\varphi) = \begin{cases} 
\varphi(c) & \text{if } \varphi(c) \text{ is defined} \\
[0, \infty) & \text{otherwise}
\end{cases}
\]

**Definition 14 (Behavior of TA)** Given a TA \(A = (Z, C, P, p_0, I, T)\), there exists a corresponding FCDEVS \(B = (X, Y, S, s_0, \delta)\) that defines the behavior of \(A\). We consider all events in \(Z\) of \(A\) as output events of \(B\) so \(X = \emptyset\) and \(Y = Z\). The state set \(S = (P, \times)\) \((T^\infty, T^\infty)\).

The initial state variable \(s_0 = (p_0, \ldots, \text{su}(p_0, c), 0, \ldots)\) where \(\text{su} : P \times C \times T \to T^\infty\) is called the clock-schedule update function that is given for a phase \(p \in P\) and a clock \(c \in C\)

\[
\text{su}(p, c) = \min\{t_S((M(I(p)) \cap M(\varphi)), c, [e_c, \infty)) : (p, z, \varphi, C_R, p') \in T\}
\]

where \(M\) is defined in Equation (20) and \(t_S : \mathcal{P}(T^\infty) \to T^\infty\) is the sampling function that is given for a set of time values \(t \subseteq T^\infty\) which can be an time interval,

\[
t_s(t) = \begin{cases} 
\infty & \text{if } t = \emptyset \\
t & \text{otherwise } t \in t.
\end{cases}
\]

The schedule update function \(su\) for TA computes the minimum remaining time considering (1) phase \(p\)'s clock constraints, (2) enabling clock interval for the given clock \(c\) for outgoing all transitions, and (3) the elapsed time \(e_c\) of the clock \(c\).

The output and internal transition function \(\delta_y : S \to Y^\phi \times S\) is given for \(s = (p, \ldots, \sigma_c, e_c, \ldots)\), \(y \in Y^\phi\): If \(\exists (p, y, \varphi, C_R, p') \in T\), then

\[
\delta_y(s) = (p', \ldots, \sigma'_c, e'_c, \ldots)
\]

where \(e'_c = t_R(c, C_R)\) where \(t_R : C \times \mathcal{P}(C) \to T\) is called the rest function that is defined for \(c \in C\) and \(C_R \subseteq C\),

\[
\begin{align*}
t_R(c, C_R) &= \begin{cases} 0 & \text{if } c \in C_R \\
e_c & \text{otherwise}
\end{cases} \\
&= (p, \ldots, \sigma_c, e_c, \ldots)
\]

**Example 2 (TA Toaster Model \(M_2\))** Figure 2(b) shows a toaster model which is a timed automaton \(A = (Z, C, P, p_0, I, T)\) where \(Z = \{\text{push}, \text{repair}, \text{pop}\}; C = \{c\}; P = \{\text{idle, busy, down}\}; I(\text{down}) = I(\text{idle}) = I(\text{busy}) = \emptyset\) which is illustrated \(\{\}\) inside of nodes of Figure 2(b). It means that, by \(M\) of Equation (20), \(0 \leq e_c < \infty\) is generated for the clock constraints for staying at these phases.

The set of transitions \(T = \{\text{idle} \downarrow \text{push, } \varphi, \varphi, \text{idle}, \text{busy} \downarrow \text{pop, } \varphi, \varphi, \text{idle}, \text{busy} \downarrow \text{repair, } \varphi, \varphi, \text{idle}\}\) where \((c, (20, 30))\) implies \(\varphi(c) = (20, 30)\).

We can find that the enable condition of the transition \(\text{busy} \downarrow \text{pop, } \varphi, \varphi, \text{idle}\) is \(e_c \in (20, 30)\) where \((20, 30) = (I(\text{busy}) = [0, \infty)) \cap (20, 30)\), and as the post condition, \(e_c\) is reset zero.

**Theorem 2** TA is a subclass of FCDEVS, i.e., \(E(TA) \subseteq E(FCDEVS)\).

**Proof:** According to Definition 14, TA is a subclass or equivalent class of FCDEVS. We need to check if TA is equivalent to FCDEVS.
Let’s assume that an FCDEVS $M_C = (X,Y,S,s_0,\delta_x,\delta_y)$ which has only one cycle $C = \{c\}$ and only one discrete state $S_d = \{s_{d_0}\}$, so it is the initial state $s_0 = (s_{d_0},\sigma_{o_0},0) \in S$ where $\sigma_{o_0} = \sqrt{T}$, $s_{d_0} = t_S([\sqrt{T},10])$, or $\sigma_{o_0} = ts\{1,2,3\}$ setting $\sqrt{T}$, sampling from an interval $[\sqrt{T},10]$ or sampling from 1,2, and 2, respectively. Since $T$ needs to describe clock constraints using rational-bounded intervals, no $T$ can define such constraints of $\sigma_{o_0}$. As a result, $E(TA) \subset E(FCDEVS)$

\section{Finite Graph-based DEVS (FGDEVS)}

In this section, we consider an FCDEVS class in which its state transitions are described as a set of relations. Each transition relation can be seen as an edge of a graph, so it is called a Finite Graph-based DEVS, abbreviated by FGDEVS, in which the numbers of vertices and edges are finite.

Before defining the FGDEVS class, let $\Psi(C) = \{C \rightarrow \mathcal{P}(T_\infty)\}$ be the set of partial clock schedule constraints $T_\infty$. For example, for a clock set $C = \{c_1,c_2\}$, if $\psi = \{(c_2,10,20)\} \in \Psi(C)$, then we say $\psi(c_2) = 10$ or $20$, $\psi(c_1)$ is undefined.

\begin{definition}[FGDEVS]
An FGDEVS model is a tuple
\[ M_{FG} = (X,Y,S,s_0,\delta) \]
where
\begin{itemize}
  \item $X,Y,S$ and $s_0$ are the same as those of FCDEVS of Definition 12.
  \item $\delta \subseteq S_d \times Z_+ \times \Psi(C) \times \mathcal{P}(C) \times S_d$ is the finite set of transition relations where $Z = X \cup Y$. A transition $\sigma_{c} = \psi(c)\{s_{d_0},\sigma_{c_0},0,\ldots\}$ or its graphical notation $s_{d_0},\sigma_{c_0},0,\ldots\rightarrow s'_{d_0},\sigma'_{c_0},0,\ldots\rightarrow$ denotes that the discrete state changes $s_{d_0}$ to $s'_{d_0}$ associated with an event $z$, together with two post-conditions: updating the schedule $\sigma_{c} = \psi(c)$ of the clock $C$, and resetting the elapsed time $e_{c}$ of each clock $c \in C_R$.
\end{itemize}
\end{definition}

\begin{definition}[Behavior of FGDEVS]
The behaviors of an FGDEVS $M_{FG} = (X,Y,S,s_0,\delta)$ model are given through an FCDEVS $M_{FC} = (X,Y,S,s_0,\delta_x,\delta_y)$ as follows.

The initial state is variable $s_0 = (s_{d_0},\ldots,\sigma_{o_0},0,\ldots)$.

The external transition function $\delta_x : S \times X \times S$ is given for $s = (s_{d_0},\ldots,\sigma_{c},e_{c},\ldots) \in S$ and $x \in X$, if $\exists s_{d_0},\sigma_{c},e_{c},\ldots \in S$, then
\[ \delta_x(s,x) = (s'_{d_0},\ldots,\sigma'_{c},e'_{c},\ldots) \]
where
\[ \sigma'_{c} = \begin{cases} t_S(\psi(c)) & \text{if } \psi(c) \text{ is defined} \\ \sigma_{c} & \text{otherwise} \end{cases} \]

where $t_S$ is the sampling function defined in Equation (22), and $e'_{c} = t_R(c,C_R)$ where $t_R(c,C_R)$ is the resetting function defined Equation (23). If $\exists s_{d_0},\sigma_{c},e_{c},\ldots \in S$, then
\[ \delta_x(s,x) = (s_{d_0},\ldots,\sigma_{c},e_{c},\ldots) \]

The output and internal transition function $\delta_y : S \rightarrow Y \times S$ is given for $s = (s_{d_0},\ldots,\sigma_{c},e_{c},\ldots) \in S$ and $y \in Y$, if $\exists s_{d_0},\sigma_{c},e_{c},\ldots \in S$, then
\[ \delta_y(s) = (y,s'_{d_0},\ldots,\sigma'_{c},e'_{c},\ldots) \]
where $\sigma'_{c} = \psi(c)$ if $\psi(c)$ is defined, otherwise $\sigma'_{c} = \sigma_{c}$ and $e'_{c} = t_R(c,C_R)$. If $\exists s_{d_0},\sigma_{c},e_{c},\ldots \in S$, then
\[ \delta_y(s) = (\delta, (s_{d_0},\ldots,\sigma_{c},e_{c},\ldots)) \]
\end{definition}

\begin{example}[FGDEVS Toaster Model $M_3$]
Figure 2(c) illustrates an FGDEVS model $M_{FG} = (X,Y,S,s_0,\delta)$ of a toaster where $X = \{\text{?push},\text{?repair}\}; Y = \{\text{?pop}\}; S_d = \{\text{down, idle, busy}\}; C = \{c\}; s_0 = (idle,\infty,0)$ because $\delta = \{(idle,\text{?push},\sigma_{c},idle),(idle,\text{?push},(\{0\}),\sigma_{c},idle),(idles,\text{?pop},(\{\infty\}),\sigma_{c},idle)\}$ where $\{(c,20,30)\}$ and $\{(\infty,\infty)\}$ means $\psi(c) = (20,30)$ that is an opened interval 20 to 30, and $\psi(c) = \infty$.
\end{example}

\section{Finite Deterministic DEVS (FDDEVS)}

A subclass of DEVS, called Finite Deterministic DEVS was introduced originally in [9], and applied to generate a reachability graph [8] for verification[6], and provided a DEVS XML schema[16]. This section investigates the expressiveness of FDDEVS comparing other formalisms.

\begin{definition}[FDDEVS]
An FDDEVS model is a tuple
\[ M_{FD} = (X,Y,S,s_0,\tau,\delta_x,\delta_y) \]
where
\begin{itemize}
  \item $X,Y$ are the same as those of FCDEVS.
  \item $S$ is the finite discrete states which are piecewise constant.
  \item $s_0 \in S$ is the constant initial state.
  \item $\tau : S \rightarrow Q_{[0,\infty)}$ is the time schedule function where $Q_{[0,\infty)}$ is the none negative rational numbers plus infinity.
  \item $\delta_x : S \times X \rightarrow S \times \{0,1\}$ is the external transition function.
  \item $\delta_y : S \rightarrow Y \times S$ is the output and internal transition function.
\end{itemize}
As explained the name, $\tau,\delta_x$ and $\delta_y$ of FDDEVS are deterministic.
\end{definition}

To clarify the expressiveness inclusion relation, we would define the behavior of FDDEVS models using FGDEVS class.

\begin{definition}[Behavior of FDDEVS]
Given an FDDEVS model $M_{FD} = (X,Y,S,s_0,\tau,\delta_x,\delta_y)$, there is a corresponding FGDEVS $M_{FG} = (X,Y,S,G,s_0,G,\delta)$ can describe the behavior of the original model $M_{FD}$ as follows. The events sets of $M_{FG}$ are the same those of $M_{FD}$. The state set of $S_G = \{(s,\sigma_{c},e_{c}) : s \in S, c \in C\}$ where $C_{c} = \{c'\}$. The initial state $s_0G = (s_0,\tau(s_0),0)$. The state transition relation $\delta$ of $M_{FG}$ is defined corresponding to each state transition.
\[ \delta_x(s,x) = (s',0) \implies \mathcal{P}_{x,G,s'} \in \delta, \]

\end{definition}
Theorem 3 $E(\text{FDDEVS}) \subseteq E(\text{FGDEVS})$. 

Proof: $E(\text{FDDEVS}) \subseteq E(\text{FGDEVS})$ because the behavior of an FDDEVS model is given by an FGDEVS model in Definition 18. The FGDEVS class allows nondeterministic characteristics, but FDDEVS doesn’t. Therefore $E(\text{FDDEVS}) \subseteq E(\text{FGDEVS})$.

VI. CONCLUSIONS

a) Summary: Starting from trajectories and the event segments, we defined the language of a class of timed event systems, and relations of subclass, superclass and equivalent classes in terms of their expressiveness based on the language. A DEVS equivalent class called Clock-based DEVS has been investigated to allow us to have an explicit representation of DEVS dynamics based on a set of clocks. Using such a DEVS class, we clarified expressiveness of other formalisms such as FCDEVS, TA, FGDEVS and FDDEVS.

b) Contribution: We conclude expressiveness of DEVS classes as the following theorem.

Theorem 4 $E(\text{DEVS}) = E(\text{CDEVS}) \supset E(\text{FCDEVS}) \supset E(\text{TA}). E(\text{FCDEVS}) \supset E(\text{FGDEVS}) \supset E(\text{FDDEVS})$.

Proof: Provided in Section V.

The expressiveness inclusion relationship shown in Theorem 4 combining $E(\text{TA}) \supset E(\text{FDDEVS})[7]$ and $E(\text{FDDEVS}) \supset E(\text{SPDEVS})[6][8]$ is illustrated in Figure 3.

This clarifying expressiveness inclusions among DEVS subclasses contributes to DEVS standardization at the formalism level. As we did along with FDDEVS class [2][16], based on the semantic consequences of this paper, we are able to standardize syntax of FCDEVS and FGDEVS using some markup languages such as XML and general-purpose programming languages like C/C++, Java, Ruby, Python, and so on.

c) Future Research: From the theoretical point of views, comparison of expressiveness between TA and FGDEVS needs to be investigated more thoughtfully, as well as between FCDEVS and FGDEVS for checking if $E(\text{FCDEVS}) = E(\text{FGDEVS})$.

Although it is not exactly a taxonomy research topic, identifying equivalence and/or distance (or metric) between two DEVS instances is expected to get started from the formal approach similar to that we took here.

REFERENCES