The DEVS framework for Discrete Event Systems Control *

Hae Sang Song and Tag Gon Kim
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
Taejon 305-701, Korea
tkim@ee.kaist.ac.kr

Abstract

This paper proposes a new methodology for analysis of discrete event systems and design of discrete event systems controllers. The methodology is based on the sound semantics for specification of discrete event systems called the DEVS formalism. It introduces concepts of inverse DEVS and defines controllability of discrete event systems expressed in the DEVS formalism. These two concepts, inverse DEVS and controllability of discrete event systems, play important roles in designing a discrete event controller. An example for appreciating the concepts is presented.

1 Introduction

Discrete event systems (DES) have taken a more important part in managing the contemporary world, most of which are man-made systems such as multicomputer systems, communication networks, traffic systems and manufacturing systems. In such a DES, computation is done by interactions between components to achieve a given goal. Such a goal can be an operation range of system behavior or optimization of system performance. An operation range of a DES can be specified by a state trajectory which is piecewise constant in time function. In the control system's viewpoint, a DES can be divided into two subsystems, plant and controller. Thus, the DES control problem is as follows. Given a DES plant, design a DES controller to meet specified objectives.

*ISBN 0-8186-6440-1.

Compared to numerous efforts to develop formalisms to specify the discrete event systems, there are a few researches on design of a discrete event controller (DEC). [1] is the first work in the field, where a DES plant is specified based on automata theory and a discrete event controller is designed based on language theory. The essential feature is that a controller has the desired behavior of a plant with respect to objectives. The controller supervises output events occurring in the plant and produce required control outputs to the plant.

This paper proposes a framework for specification and design of discrete event systems control based on the DEVS (discrete event systems specification) formalism [2]. Unlike the methodology based on automata theory [1], we base DES controller design on the system-theoretic DEVS formalism. DEVS, in nature, has more information than that of automata in specification, especially timing information. It also distinguishes state transition into two different ones, i.e., internal and external transitions and discriminates input events from output events. Our approach inherits diverse advantages of the DEVS formalism.

2 Supervisory control of discrete event systems

Supervisory control [3] is a kind of feedback control that supervises the behavior of a discrete event plant to control the plant in desired objectives. It is assumed that the behavior of the plant is known. Thus the controller can deduce the current behavior of the plant from observed output events. Then it compares the behavior with desired one and generates appropriate next control outputs. A design problem of a discrete event controller is as follows: given a discrete event system with known dynamics and control objectives, design a discrete event controller that satisfies the control objectives.

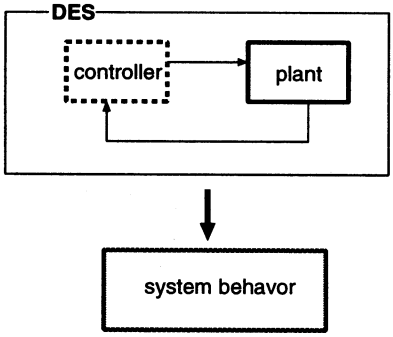

Figure 1: Supervisory control of a discrete event system

Fig. 1 shows concepts of supervisory control. To design a discrete event controller, we have to specify 1) dynamics of a plant 2) controlled behavior of the plant that satisfies the control objectives. In this paper, we specify plant behavior based on the DEVS formalism. We call it a plant DEVS. From informal control objectives and the plant DEVS, we obtain a desired state trajectory that satisfies the control objectives. A discrete event controller would be obtained by transformation of the desired state trajectory into a DEVS model. Connecting the plant with the designed controller through proper interfaces, called ports, results in an overall control system satisfying the desired system behavior. We will describe concepts and design steps in more detail in the following sections.

3 The DEVS formalism

This section briefly reviews the DEVS formalism and introduces a graphical notations representing the DEVS formalism.

3.1 Review of the DEVS formalism

The DEVS formalism specifies discrete event models in a hierarchical, modular form. Formally, an atomic model M is specified by a 7-tuple :

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

where

X : input events set;
S : states set
Y : output events set;
$\delta_{ext} : Q \times X \rightarrow S$: external transition function;
$\delta_{int} : S \rightarrow S$: internal transition function;
$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$: total states;
$\lambda : S \rightarrow Y$: output function;
$ta : S \rightarrow Real$: time advance function.

The four elements in the 7-tuple, namely, δ_{int}, δ_{ext}, λ, ta are called characteristic functions, and S is set of state variables, $X(Y)$ is set of input(output) events. An atomic model represents a corresponding discrete event process and connections between processes is represented by a coupled model DN

$$DN =< X, Y, M, EIC, EOC, IC, SELECT >$$

where

X : input events set;
Y : output events set;
M : DEVS components set;
$EIC \subseteq DN.IN \times M.IN$: external input coupling relation;
$EOC \subseteq M.OUT \times DN.OUT$: external output coupling relation;
$IC \subseteq M.OUT \times M.IN$: internal coupling relation;
$SELECT : 2^M - \emptyset \rightarrow M$: tie-breaking selector.

The three elements, EIC, EOC, IC means the connections between set of models M and input, output ports X, Y. $SELECT$ function acts as a tie-breaking selector.

3.2 Dynamics of a DEVS model

The dynamics of a discrete event systems specified in the DEVS formalism can be interpreted by the abstract simulator in [2]. We mention it briefly here and define some terminologies to be used in the following sections. The dynamics of an atomic DEVS is determined by a sequence of internal or external transitions. An internal transition is spontaneous after fulfilling its current activity and an external transition is caused by an external event.

An atomic DEVS M_i spontaneously changes its current activity(or state) s_i to $\delta_{int}(s_i)$ after completing the activity at the predefined sojourn time $ta(s_i)$. Just before the transition, M_i produces an output event $\lambda(s_i)$. The stimulus that makes this transition to occur is generated at which an activity has just completed. We call the stimulus a * event. It's not visible to us and other DEVS models, so we call it a *hidden event.* This transition, $* \rightarrow \lambda(s_i) \rightarrow \delta_{int}(s_i)$, is called an *internal transition.* From the above statement, we can define *internal transition relation* for a DEVS M_r as follows: $T_{r,int} = \{(s_i, p!m, s_j) | (s_i, s_j) \in \delta_{r,int},\ p!m = \lambda_r(s_i) \in Y_r, s_i \in S_r\}$.

An output event $p_i!m_i$ resulting from an internal transition of M_i is converted into an input event $p?m$ of another DEVS M_j connected with M_i through the port p. If M_j is ready to receive the input event $p?m$ at the current state s_j, it changes its state into $\delta_{ext}(s_j, p?m)$. We call the transition an *external transition*. We define *external transition relation* of a DEVS M_r as follows: $T_{r,ext} = \{(s_i, p?m, s_j) | (s_i, p?m, s_j) \in \delta_{r,ext}, s_i \in S_r\}$.

We can see that an external transition of a DEVS is caused by an output event resulting from an internal transition of another DEVS stimulated by the * event. That is, with the * event, two DEVSs are changing their states at the same time. This phenomenon is *concurrency*. We deal with concurrency later in more detail. A pair of input and output events, $p_i!m_i$, $p_j?m_j$, that make concurrency satisfies the following properties: $dom(p) = dom(p_i) = dom(p_j)$ and $m = m_i = m_j$, where $dom(p)$ is the domain of port p that messages can reside. We call the pair a *dual event*.

3.3 Graphical notations

An atomic model representing a process is enclosed by a box with input and output ports in the wall. Such Ports are entrances or exits for messages on its own to represent X and Y in DEVS. Combinations of ports and messages by ?(input) and !(output) represent input events on X and output events on Y, respectively. Each state variable in the state set S of DEVS is represented by a small box in the atomic model and has its name in it. The behavioral description of an atomic model is represented by an *activity transition diagram* or an state transition diagram, which consists of nodes and two-colored edges. Each node represents an activity or a state, dotted arc denotes an internal transition and solid arc an external transition. In Fig. 2, (a)

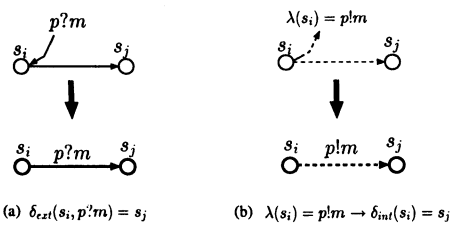

Figure 2: graphical notation: (a) external transition (b) internal transition

represents an external transition and (b) denotes an internal transition consisting of $\lambda(s) \rightarrow \delta_{int}$. An output event is specified on a dotted line by an output port followed by a message name with output operator '!'. For example, an output event $out!m$ means that a message m is output at the port *out*. Similarly an input event is specified on a solid line by an input port followed by a message name with input operator '?'. An input event $in?m$ means that a message m is input at the input port *in*. The change of values of state variables are enclosed by '[' and ']'. If it is on the dotted (solid) line, it represents a state transition specified by the internal (external) transition function δ_{int} (δ_{ext}) of DEVS, respectively. Optionally, transition condition g can be specified after input or output event with a seperator notation @. It is natural that a time advance in a state be attached to the state node because it represents a sojourn time to fulfill its activity. An empty output event is denoted by $\emptyset$.

4 Concurrency

Consider two atomic models M_i and M_j, $M_i = < X_i, S_i, Y_i, \delta_{i,int}, \delta_{i,ext}, \lambda_i, ta_i >$, $M_j =< X_j, S_j, Y_j, \delta_{j,int}, \delta_{j,ext}, \lambda_j, ta_j >$. Assume that $M_i||M_j$ be a composition of M_i and M_j. Let state transitions of M_i, M_j be $(q_i, a, r_i) \in (T_i - T_j) = T_{i,int} \cup T_{i,ext}$, $(q_j, b, r_j) \in (T_j - T_i) = T_{j,int} \cup T_{j,ext}$. Then state transition relation T of the composition $M_i||M_j$ has the following transition rules:

(1) if $a = p!m$ and $b \neq p?m$ then
$\rightarrow ((q_i, q_j), p!m, (r_i, q_j)) \in T$, for all $q_j \in S_j$

(2) if $b = p!m$ and $a \neq p?m$ then
$\rightarrow ((q_i, q_j), p!m, (q_i, r_j)) \in T$, for all $q_i \in S_i$

(3) if ($a = p!m$ and $b = p?m$) or ($b = p!m$ and $a = p?m$) then
$\rightarrow ((q_i, q_j), p\#m, (q_i, r_j)) \in T$

(4) No other transition in T

(3) means if a, b is dual event and connected, then both are transiting concurrently. That is, if an output event resulting from an internal transition of one model becomes an input event of the other waiting for it, then the two models changes their states concurrently. This is called concurrency and the event communicating the message is called *concurrent event*, denoted by $p\#m$. (1) and (2) represent the cases even if an output event of one model is produced but the other one is not waiting for the event in its current state. In this case, the output event takes no effect on

the other model and disappeared. We call the disappeared event a *dangling event.*

The concurrency rules described above will be used to analyze the dynamics of a system consisting of several subsystems.

5 Controllability and inverse DEVS

Controllability of a discrete event system specified in DEVS is exploited to check if a discrete event controller satisfying a desired state trajectory exists or not. If controllable, concepts of inverse DEVS would transform the desired state trajectory into the behavior of a discrete event controller. In this section, we define these two concepts.

5.1 Controllable DEVS

Let T be the global state trajectory of a discrete event plant $M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$. Let $ST \subseteq T$ be a desired state trajectory.

Definition 1 (strong controllability)
A desired state trajectory ST is said to be strongly controllable *if the following condition is hold. For all s_i on ST, if $\exists\ s_j = \delta_{int}(s_i)$, then s_j is on ST. A discrete event plant is said to be strongly controllable if $\forall\ ST \subseteq T$, ST is strongly controllable.*

Definition 2 (weak controllability)
A desired state trajectory ST is said to be weakly controllable *if the following condition is hold. For all s_i on ST, if $\exists\ s_j = \delta_{int}(s_i)$ not on ST, then $\exists$ at least one $s_k = \delta_{ext}(s_i, e, p?m)$ such that s_k is on ST and $e < ta(s_i)$. A discrete event plant is said to be weakly controllable if $\forall\ ST \subseteq T$, ST is weakly controllable.*

Note that controllability in Def. 1 does not depend on time, but Def. 2 is time-dependent.

5.2 Inverse DEVS

Consider two DEVS M_i, M_j operating concurrently. If $M_i || M_j$ is running with transitions caused only by concurrent events (see Section 4), then we call $M_i(M_j)$ an inverse DEVS of $M_j(M_i)$, respectively.

Definition 3 (inverse DEVS) *$M_i(M_j)$ is said to be an* inverse DEVS *of $M_j(M_i)$ iff the following properties are hold:*

(i) State set morphism

$$S_i \rightarrow S_j$$

(ii) Dual I/O events set

$$X_i = \{p?m | p!m \in Y_j\}$$
$$Y_i = \{p!m | p?m \in X_j\}$$

(iii) Dual transition relation

$$T_{i,int} = \{(q, p!m, r) | (q, p?m, r) \in T_{j,ext}\}$$
$$T_{i,ext} = \{(q, p?m, r) | (q, p!m, r) \in T_{j,int}\}$$

(i) states that there is one-to-one correspondence between states of two DEVSs. (ii) indicates that input events of one would be converted into output events of the other, and vice versa. (iii) denotes that an internal transition of one DEVS would be changed to an external transition of the other.

A transformation from an atomic DEVS M_i to M_j satisfying the above properties is called an *inverse DEVS transformation.* The concepts of inverse DEVS is used to obtain a discrete event controller from a desired controlled state trajectory.

6 Design methodology for DEC

A methodology for design of a DES controller bases on the DEVS formalism and the concepts of inverse DEVS. Fig. 3 shows the steps for design of a discrete event controller. First, a plant is specified in the DEVS formalism, which consists of one or more atomic models that are connected each other. We can obtain a global state space at this step, from which we shall extract a controlled state trajectory satisfying given objectives. We call it a desired state trajectory of the plant. The next step is to check the controllability of the desired state trajectory by the controllability definition defined in the paper. And the supervisor DEVS transformation described above is used to directly transform the desired state trajectory to a DES controller. After the transformation we relabel states of the controller after suitable state reduction. Finally we can obtain a controller. Note that if a desired state trajectory of a plant is given, then we can directly obtain a corresponding DES controller from it using the inverse DEVS transformation.

7 A simple example: control of water supply system

We will clarify the approach described previous sections by a simple example of a water supply system.

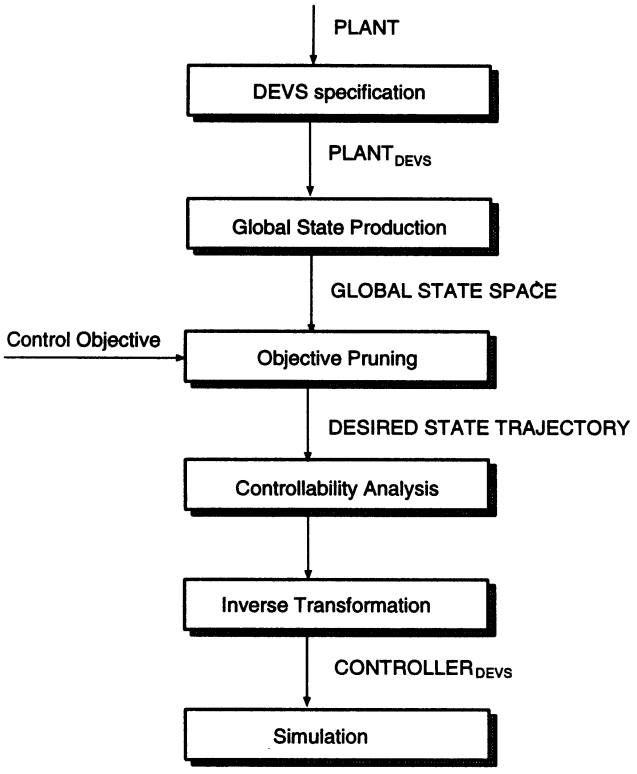

Figure 3: Design steps for a discrete event controller

Following subsections show how each step in Fig. 3 performs through the example. We use the graphical notations rather than set-theoretical notations, for graphical notations show more clear model descriptions.

7.1 Problem description

An informal problem description is a starting point for the design of a controller for the water supply system. In Fig. 4, a water supply plant consists of two subsystems, a water pump and a water tank. In the water tank, there are two water level sensors (low, high) and a water pipe through which water flows in. The water pump takes electric energy from outside world, which can be controlled by a ON/OFF switch. It fills the water tank through the water pipe.

The purpose of the system is to keep the water level between the low and high sensors.

7.2 DEVS model of the plant

Specification of a plant in the DEVS formalism is the first step to design a discrete event controller. Input and output events, state variables, output function, internal and external transition functions and time advance function should be identified.

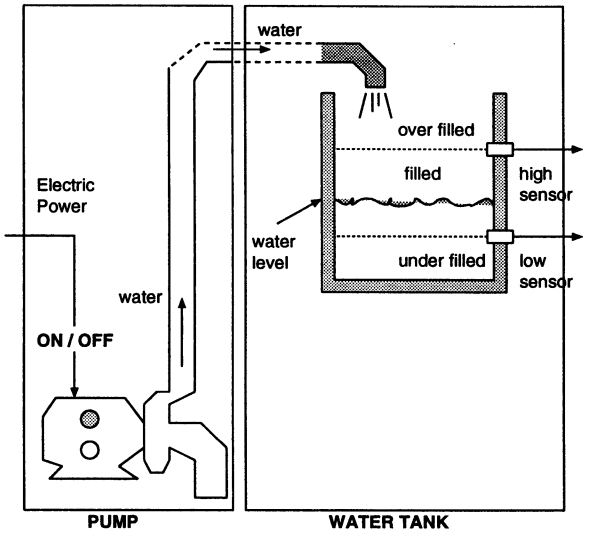

Figure 4: Water pump plant

To specify a plant in the view of a discrete event system, we have to identify events occuring in the plant. The events set of the water pump consists of ON/OFF switching events of the motor, and events that water starts and stops to flow out through the water pipe. The events set of the water tank consists of the low sensor ON/OFF, the high sensor ON/OFF, and events that water starts and stops flowing in through the water pipe.

Fig. 5 shows a DEVS model of the plant. If the water pump is turned on ("pw"?ON), it starts pumping water through the water pipe ("w"!WSTRT). If it is turned down ("pw"?OFF), it stops pumping and the flow of water will be stopped ("w"!STOP). Initially, the water tank changes INIT state into BS, FS, VS according to the initial water level producing appropriate sensor signals to outside world. The letter B means under-filled, F filled, V over-filled and the second letter S means that the in-flow of water is currently stopped. While the water level lowers, corresponding level sensor signals would be generated. With water flowing in, the state is changed into BF, FF, VF generating proper sensor signals. If the in-flow is continued even though the water level reaches far over the high sensor, the water tank eventually overflows (TOP). In the opposite case, the water level would reach to the bottom (BOT).

The two subsystems, the water pump and the water tank, are connected through the pipe ("w"). The dynamics of the composite subsystem can be obtained

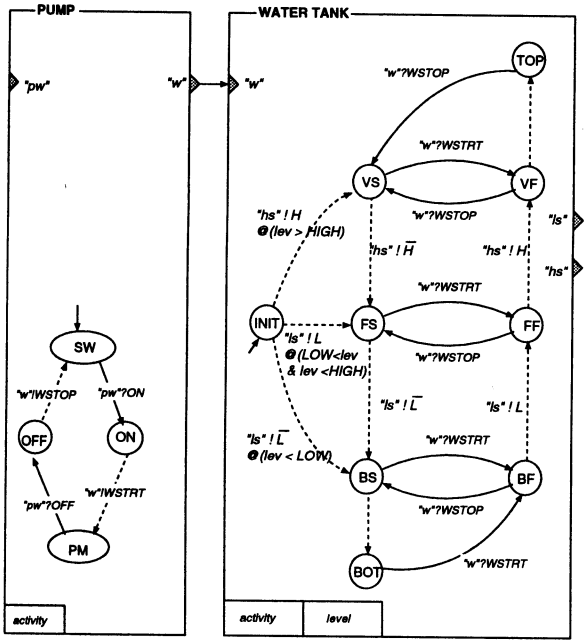

Figure 5: Specification of the water pump plant

by concurrency analysis.

7.3 Controlled state trajectory

The next step is to get a global state transition diagram (GSTD) from the plant specification. It may be obtained by a Cartesian product of state diagrams of each subsystem. Alternatively, if we use the concurrency analysis, we can obtain far less number of states. In the case of no connection, there would be $4 \times 9 = 36$ states in the example plant. However, there is one connection("w") and we obtained a GSTD of the plant with only 17 states by the concurrency analysis.

Meanwhile, the informal description of control objectives is to be re-described based on the DEVS model of the plant. Let a controlled state trajectory be K, and s on GSTD be $(P.s, W.s)$, where $P.s$ denotes a state of the water pump and $W.s$ a state of the water tank. Then K should satisfy the following conditions:

1. If $\exists$ `(`x`, BS)`, then eventually `(SW, VS)`, where x means *don't care*.

2. $\exists$ no s such that `(`x`,BOT) or (`x`,TOP)`.

3. hysteresis: $\exists$ no transition between `(`x`,FF)` and `(`x`,FS)`

Now we can extract a controlled state trajectory from GSTD and the control objectives. Fig. 6 is the

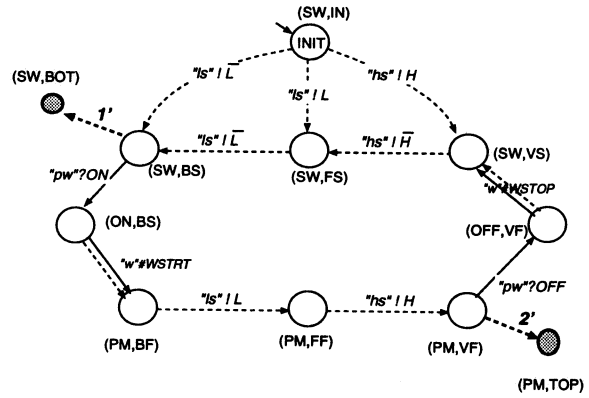

Figure 6: desired state trajectory of the plant

desired state trajectory (In the figure, we exclude transition 1' and 2' from the desired state trajectory, which were added for convenience to explain controllability.). Initially, the state is changed into (SW,BS) after a few internal transitions. Then the motor should be ON, which is followed by a concurrent event "w"#WSTRT. If the water level reaches to the over-filled region, the motor should be OFF. Then, water stops flowing in. By internal transitions, the water level is to be lowered again, and then the motor should be ON, and so on. This state trajectory satisfies the three objectives.

The next step is to check if the desired state trajectory is controllable (strongly controllable). Unfortunately, there are two internal transitions into states not on the desired state trajectory (1', 2'). We can not control these transitions to occur because they are spontaneous. Thus it is impossible to design a discrete event controller satisfying the desired state trajectory.

However, if water can be supplied or stopped before those states are reached, then the two internal transitions, 1' and 2' can be controlled not to occur. It means that the controls depend on time. By Def. 2, we know the desired state trajectory is weakly controllable. Usually, controllability means strongly controllable unless explicitly specified.

In the following section, we assume that the time interval necessary to stop the water flow is sufficiently small, compared to the time interval that the water tank overflows from the position of the high sensor. Of cause, we assume that the opposite case is also true. Then, we know that the internal transitions 1' and 2' could not occur physically and be eliminated from the plant DEVS. Thus, the desired state trajectory is controllable with the assumption.

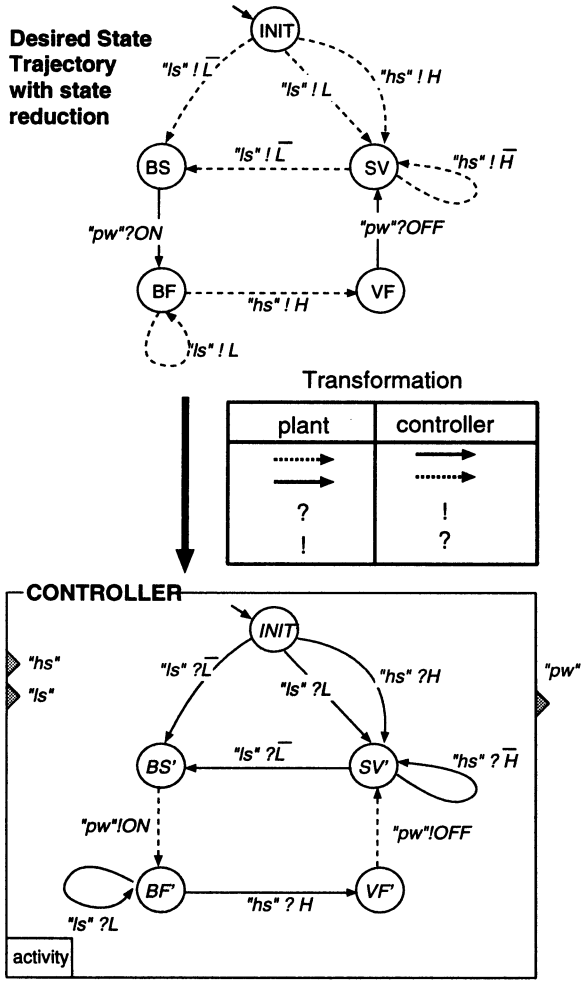

Figure 7: Design of a DES controller

7.4 Getting a discrete event controller

Until now, we specified a discrete event plant in DEVS, obtained GSTD, re-described the control objectives based on the plant DEVS, and got a desired state trajectory satisfying the control objectives. Now, we can get a discrete event controller from the trajectory using inverse DEVS transformation.

Before the transformation, we can reduce the states of the desired state trajectory. The upper part of Fig. 7 is obtained by suitable state reduction from the trajectory in Fig. 6.

A discrete event controller is obtained by inverse DEVS transformation to the desired state trajectory with reduced states shown in the upper part of the Fig. 7. The lower part of Fig. 7 shows the discrete event controller after the transformation. Note that the transformation is so straightforward and intuitive.

The three subsystems, the controller, the water pump, and the water tank are to be connected through ports that have the same name, which results in an overall system. If we analyze the dynamics of the overall system, we know that it follows the desired state trajectory only with concurrent events.

8 Conclusion

The paper proposes a new approach for analysis of discrete event systems and design of discrete event system controllers based on the DEVS formalism. Controllability and concept of inverse DEVS are defined and introduced. These two concepts play key roles in designing a discrete event controller.

A DES controller designed under the framework will be simulated in a straightforward way using DEVSim++[4] to analyze the performance of the system. But it was omitted here.

More formal description of the methodology and extension to timed systems are remained as further research.

References

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes", *SIAM J. Control and Optimization*, vol. 25, no. 1, pp. 206–230, Jan. 1987.

[2] B. P. Zeigler, *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, Orlando, FL, 1984.

[3] Cristos G. Cassandras, *Discrete Event Systems*, IRWIN, 1993.

[4] Tag G. Kim and Sung B. Park, "The DEVS formalism: Hierarchical modular systems specification in C++", *Proc. of the 1992 European Simulation Multiconference*, pp. 152–156, 1992.

[5] Bernard P. Zeigler, *Theory of Modelling and Simulation*, John Wiley & Sons, Inc., 1985.