DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems

Bernard P. Zeigler
Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721, U.S.A.

Hae Sang Song and Tag Gon Kim
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
Taejon 305-701, Korea

Herbert Praehofer
Systems Theory and Information Engineering
Johannes Kepler University of Linz
A-4040 Linz, Austria

Abstract

We make the case that Discrete Event System Specification (DEVS) is a universal formalism for discrete event dynamical systems (DEDS). DEVS offers an expressive framework for modelling, design, analysis and simulation of autonomous and hybrid systems. We review some known features of DEVS and its extensions. We then focus on the use of DEVS to formulate and synthesize supervisory level controllers.

1 Introduction

Formal treatment of discrete event dynamical systems is receiving ever more attention[5]. Work on a mathematical foundation of discrete event dynamic modeling and simulation began in the 70s [13, 14, 16] when DEVS (discrete event system specification) was introduced as an abstract formalism for discrete event modeling. Because of its system theoretic basis, DEVS is a universal formalism for discrete event dynamical systems (DEDS). Indeed, DEVS is properly viewed a short-hand to specify systems whose input, state and output trajectories are piecewise constant[17]. The step-like transitions in the trajectories are identified as discrete events.

Recently, interest in hybrid systems (mixed discrete-continuous) systems has been growing. Such systems are prominent in such areas as intelligent control [1, 7] and reactive system design [3]. This article proposes that DEVS-based systems theory provides a sound, general framework within which to address modelling, simulation, design, and analysis issues for hybrid systems.

Fig. 1 shows the basic approach to hybrid systems research proposed here. We briefly address the ideas raised in this figure.

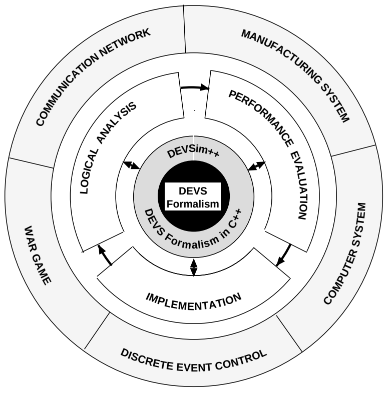


Figure 1: DEVS-Centered Hybrid Systems Research

1.1 Modelling

Discrete event modelling has its origin in job shop scheduling simulations in operations research. More recently control theorists developed their own discrete event formalisms[5]. The universality claims of the DEVS just cited are addressed by characterizing the class of dynamical systems which can be represented by DEVS models. Prahofer and Zeigler (in press) showed that any causal dynamical system which has piecewise constant input and output segments can be represented by DEVS. We call this class of systems *DEVS-Representable*[17]. In particular, Differential Equation Specified Systems (DESS) are usually used to represent the plant (system under control) in hybrid systems and are controlled by high-level, symbolic, event-driven control schemes. Sensing and actuation in such systems are event-like and they have piecewise constant input and output trajectories when viewed within the frame of their sensing/actuator interface. Consequently, within such an interface, the plant is representable by a DEVS. Likewise, the controller has natural DEVS representation.

Closure under coupling[14, 9] is a desirable property for subclasses of dynamical systems since it guarantees that coupling of class instances results in a system in the same class. The class of DEVS-representable dynamical systems is closed under coupling. This justifies hierachical, modular construction of both DEVS models and the (continuous or discrete) counterpart systems they represent.

1.2 Simulation

DEV&DESS is an extension of the DEVS formalism for combined discrete/continuous modeling and simulation[9, 4]. DEV&DESS includes both the DEVS and DESS classes of systems. In addition it provides a means of specifying systems with interacting continuous and discrete trajectories. The formalism is fully expressive of hybrid systems in that it is shown to be closed under coupling. In particular, the resultants of coupled models that contain components of either class are expressibel as basic models in the formalism. Since it is also implemented in software, DEV&DESS offers a powerful modeling formalism for simulating hybrid systems.

1.3 Design and Analysis

Ramage and Wonham[11] were the first control theorists to develop an approach to designing discrete event controllers. They modelled the plant as an automaton and used language theory to design a controller that forces the plant to exhibit behavior consistent with given objectives. The goal of this paper is to demonstrate the applicability of DEVS as unifying framework for hybrid control. We do so by reformulating the Ramage-Wonham approach to controller design within the DEVS formalism.The resulting methodology is both more general and more intuitively appealing than the original. Since it is universal for general piecewise constant systems, DEVS, can represent timing information about the plant that eludes an automaton formulation. Such information can be employed to improve controller designs. Moreover, DEVS distinguishes between internal and external state transitions, a fact that enables us to derive a controller DEVS from a plant DEVS model in a rather direct manner.

Since the appeal of the proposed methodology rests upon the universality of DEVS representation, we first briefly review systems formalisms and their mapping into DEVS. Then we discuss the DEVS-based methodology for discrete event control of hybrid plants. We conclude with open questions for future research.

2 Review of General Dynamical Systems and DEVS

2.1 General Dynamical Systems

Based on[8, 13, 12] we define a general dynamical system as follows:

$$DS = (T, X, Y, \Omega, Q, \Delta, \Lambda)$$

with

- T is the time base,
- X is the set of input-values,
- Y is the set of output-values,
- Ω is the set of admissible input segments $\omega :< t_1, t_2 > \rightarrow X$ over T and X and Ω is closed under concatenation as well as under left segmentation [13],
- Q is the set of states;
- $\Delta : Q \times \Omega \rightarrow Q$ is the global state transition function,
- $\Lambda : Q \times X \rightarrow Y$ is the output function.

The global state transition function of the general dynamical system DS has to fulfill the following properties [8, 13]:

Consistency: $\Delta(q, \omega_{(t,t>}) = q,$ (1)

Semigroup property: $\forall \omega :< t_1, t_2 > \rightarrow X \in \Omega, t \in < t_1, t_2 >: \Delta(q, \omega_{<t_1,t_2>}) = \Delta(\Delta(q, \omega_{<t_1,t>}), \omega_{<t,t_2>}),$(2)

Causality: $\forall \omega, \bar{\omega} \in \Omega$ if $\forall t \in < t_1, t_2 >: \omega(t) = \bar{\omega}(t)$ then $\Delta(q, \omega_{<t_1,t_2>}) = \Delta(q, \bar{\omega}_{<t_1,t_2>}).$ (3)

Causality, the semigroup property and closure of admissible segments under left segmentation justifies defining the state trajectory resulting from every initial state $q \in Q$ and input segment $\omega :< t_1, t_2 > \rightarrow X \in \Omega$ in the following way:

$$STRAJ_{q,\omega} :< t_1, t_2 > \rightarrow Q \quad \text{with} \quad \forall t \in < t_1, t_2 > STRAJ_{q,\omega}(t) = \Delta(q, \omega_{<t_1,t>}). \quad (4)$$

Similarly we define the output trajectory $OTRAJ_{q,\omega} :< t_1, t_2 > \rightarrow Y$ by: for every initial state $q \in Q$, input segment $\omega :< t_1, t_2 > \rightarrow X \in \Omega$ and $t \in < t_1, t_2 >$

$$OTRAJ_{q,\omega}(t) = \Lambda(STRAJ_{q,\omega}(t), \omega(t)). \quad (5)$$

Now the input/output behavior R_{DS} of the dynamical system is given by

$$R_{DS} = \{(\omega, OTRAJ_{q,\omega}) : \omega \in \Omega, q \in Q\}. \quad (6)$$

2.2 The Discrete Event System Specification (DEVS) Formalism

An atomic DEVS is a structure [13, 14, 16]

$$DEVS = (X_M, Y_M, S, \delta_{ext}, \delta_{int}, \lambda, ta)$$

where

- X_M is the set of inputs,
- Y_M is the set of outputs,
- S is the set of sequential states,
- $\delta_{ext} : Q \times X \rightarrow S$ is the external state transition function,
- $\delta_{int} : S \rightarrow S$ is the internal state transition function,
- $\lambda : S \rightarrow Y$ is the output function,
- $ta : S \rightarrow \mathcal{R}_0^+ \cup \{\infty\}$ is the time advance function.

A DEVS specifies a dynamical system DS in the following way:

- the time base T is the real numbers $\mathcal{R}$;
- $X = X_M \cup \{\Phi\}$, i.e., the input set of the dynamical system is the input set of the DEVS together with a symbol $\Phi \notin X_M$ specifying the non-event,
- $Y = Y_M \cup \{\Phi\}$, i.e., the output set of the dynamical system is the output set of the DEVS together with Φ.
- $Q = \{(s, e) : s \in S, 0 \leq e \leq ta(s)\}$ the set of states of the dynamical system consists of the sequential states of the DEVS paired with a real number e giving the elapsed time since the last event,
- the admissible input segments is the set of all DEVS-segments over X and T (DEVS-segments [13] are characterized by the fact that for any $\omega :< t_1, t_2 > \rightarrow X \in \Omega$, there is only a finite number of event times $\{\tau_1, \tau_2, \ldots, \tau_n\}, \tau_i \in < t_1, t_2 >$ with $\omega(\tau_i) \neq \Phi$. As a special case, a so-called Φ segment has no events.
- for any DEVS input segment $\omega :< t_1, t_2 > \rightarrow X \in \Omega$ and state $q = (s, e)$ at time t_1 the global state transition function Δ is defined as follows:
  $\Delta(q, \omega_{<t_1,t_2>}) =$

$$(s, e + t_2 - t_1) \tag{7}$$

$$\text{if } e + t_2 - t_1 < ta(s) \wedge \omega \text{ is the } \Phi \text{ segment,}$$

$$\Delta((\delta_{int}(s), 0), \omega_{<t_1+ta(s)-e,t_2>}) \tag{8}$$

$$\text{if } e + t_2 - t_1 \geq ta(s) \wedge \neg\exists t \in < t_1, t_1 + ta(s) - e >: \omega(t) \neq \Phi,$$

$$\Delta((\delta_{ext}(s, e + t - t_1, \omega(t)), 0), \omega_{(t,t_2>}) \tag{9}$$

$$\text{if } \omega(t) \neq \Phi \wedge t \leq t_1 + ta(s) - e \wedge \omega \text{ restricted to } < t_1, t > \text{ is a } \Phi \text{ segment.}$$

- the output function Λ of the dynamical system is given by

$$\Lambda((s,e),x) = \lambda(s). \tag{10}$$

For a given input segment, ω three different cases can be identified:

- First, if ω is a non-event segment and an internal event does not occur in interval $< t_1, t_2 >$, then the sequential state is left unchanged but the elapsed time component e of the total state has to be updated (7).
- Second, if a time event scheduled by the time advance function does occur in the time interval $< t_1, t_2 >$ and there is no input event prior to the internal event time, then the internal transition function defines a new sequential state, the elapsed time component is set to zero, and the rest of the input segment is applied to this new state (8).
- Third, if the input segment ω defines an input event prior to the occurrence of an internal event, then the external state transition function is applied with the extenal input value and elapsed time and the elapsed time is set to zero. The rest of the input segment is applied to this new state (9).

2.3 DEVS-Representation of Constant Input/Output Dynamical Systems

We define a *piecewise constant trajectory* $\omega :< t_1, t_2 > \rightarrow X$ in the following way: there is a finite (possibly empty) subset $\{\tau_1, \tau_2, \ldots, \tau_n\}, \tau_i \in < t_1, t_2 >$ for which $\omega(\tau_i) \neq \omega(\tau_i - \epsilon)$ and $\epsilon \in \mathcal{R}^+$ is arbitrarily small. Piecewise constant trajectories are equivalent to DEVS-segments in the sense that they can be transformed into each other. The transformations are as follows:

- *Piecewise constant trajectory ω to DEVS-segment $\bar{\omega}$*: For every event time $\tau_i \in \{\tau_1, \tau_2, \ldots, \tau_n\}$ of ω there is an value $\bar{\omega}(\tau_i) = \omega(\tau_i)$ and for all $\tau \neq \tau_i \Rightarrow \bar{\omega}(\tau) = \Phi$.
- *DEVS-segment $\bar{\omega}$ to piecewise constant trajectory ω*: Let $\{\tau_1, \tau_2, \ldots, \tau_n\}, \tau_i \in < t_1, t_2 >$ be the event times of the DEVS-segment $\bar{\omega}$ with $\bar{\omega}(\tau_i) \neq \Phi$. Then for every $\tau \in < t_1, t_2 >$ we define the respective value of the piecewise constant trajectory $\omega(\tau)$ by the value $\bar{\omega}(\tau_x)$ with τ_x being the largest time in $\{\tau_1, \tau_2, \ldots, \tau_n\}$ with $\tau_x \leq \tau$. If such a number τ_x does not exist then $\omega(\tau) = \Phi$.

For $x \in X$ let $x_{<t_1,t_2>}$ denote the segment $\omega :< t_1, t_2 > \rightarrow X$ with constant value x, i.e., $\forall t \in < t_1, t_2 >$ $\omega(t) = x$. If it is clear from the context, we write x for $x_{<t_1,t_2>}$.

We now define a *constant input/output dynamical system* as a dynamical system whose input trajectories $\omega \in \Omega$ and associated output trajectories $OTRAJ_{q,\omega}$ are piecewise constant only. Our main interest is to show how such a system can be represented in the DEVS formalism.

Given a constant input/output dynamical system $DS = (T, X, Y, \Omega, Q, \Delta, \Lambda)$, we define a $DEVS_{DS} = (X_M, Y_M, S, \delta_{ext}, \delta_{int}, \lambda, ta)$

in the following way:

- $X_M = X$,
- $Y_M = Y$,
- $S = Q \times X$, and

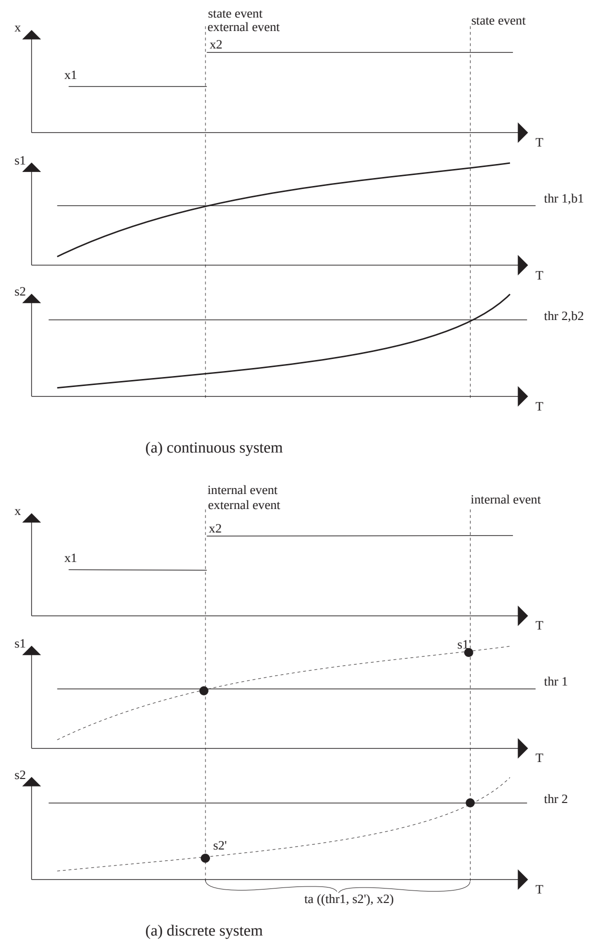


Figure 2: Illustrating Discrete Event Trajectories

- for every total state $((q,x),0)$ and input segment $\omega :< t_1, t_2 >\rightarrow X$ we define

$$ta((q,x)) = min\{e|OTRAJ_{q,x}(t_1) \neq OTRAJ_{q,x}(t_1+e), \tag{11}$$

$$\delta_{int}((q,x)) = (STRAJ_{q,x}(t_1+ta(s)), x), \tag{12}$$

$$\delta_{ext}((q,x),e,x') = (STRAJ_{q,x}(t_1+e), x'), \tag{13}$$

$$\lambda((q,x)) = \Lambda(q,x). \tag{14}$$

Theorem 1: The $DEVS_{DS}$ as constructed above and the original DS are behaviorally equivalent, i.e., they have the same input/output behavior when started in the same state.

Proof: The main work is to show that for all input trajectories, ω and initial states, q the state in the constructed discrete event system $DEVS_{DS}$ is equal to the state of the original dynamical system DS at both the event times $\tau_e \in \{\tau_{e1}, \tau_{e2}, \ldots, \tau_{en}\}$ of ω and the event times $\tau_i \in \{\tau_{i1}, \tau_{i2}, \ldots, \tau_{in}\}$ of the associated output trajectory $OTRAJ_{q,\omega}$. This is done by examining the state and output trajectories of the dynamical system $\bar{DS} = (\bar{T}, \bar{X}, \bar{Y}, \bar{\Omega}, \bar{Q}, \bar{\Delta}, \bar{\Lambda})$ specified by $DEVS_{DS}$. From the definition of the output in (14), and from the fact that outputs do not change between event times it follows that the output behavior of the discrete event system $DEVS_{DS}$ is equal to that of the original dynamical system DS. The formal proof is presented in Prahofer and Zeigler (in press).

2.4 Coupled Models

Models constructed from components are formalized in the DEVS formalism as coupled models, 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;
$SELECT : 2^M - \emptyset \to M$: tie-breaking selector.

Here DN.IN, DN.OUT, M.IN, and M.OUT refer to the input and output ports of the newly constructed coupled model and component models, respectively. The three elements, EIC, EOC, IC specify the connections between the set of models M and input, output ports X, Y. $SELECT$ function acts as a tie-breaking selector.

Closure under coupling of DEVS-representable systems is stated as:

Theorem 2: A modular coupled system whose components are DEVS- representable and which does not contain algebraic loops is a DEVS-representable dynamical system.

The definitions required and formal proof are presented in Praehofer and Zeigler (in press).

2.5 DEVS-based Combined Discrete/Continuous Modeling of Hybrid Systems

In this section, we demonstrate how the DEVS representation applies to a special subclass of dynamical systems – those whose state dynamic is defined by a set of first order differential equations and whose input and output trajectories are piecewise constant. We introduce the *output-partitioned DEV&DESS* formalism for hybrid system modeling and show how a DEVS abstraction is constructed.

Let us consider a continuous system with an n-dimensional continuous state space $S_c = S_{c1} \times S_{c2} \times ... \times S_{cn} = R^n$ whose continuous behavior can be modeled by a set of first order differential equations $\phi = \{\phi_1, \phi_2, ..., \phi_n\}$. However, we assume that the system receives input stimulations which are piecewise constant time functions. Moreover, the system is outfitted with a set of threshold like sensors. Such a threshold sensor only changes its state at particular fixpoints f_{i,b_i} - *threshold values* - of continuous state variables s_i. For simplicity lets assume that only boolean valued sensors $thr_{f_{i,b_i}}$ are used which define a boolean value $x = thr_{f_{i,b}}$ for all $s_i < f_{i,b_i}$ and the opposite value $\neg x$ for all values $s_i > f_{i,b_i}$.

Then we define the output partitioned DEV&DESS in the following way:

Definition: An output-partitioned DEV&DESS is a structure:

$$op - DEV\&DESS = (X, Y, Q, \phi, \lambda, F) \tag{15}$$

where

- X is the set of input values which is an arbitrary set,
- $S = S_1 \times S_2 \times \ldots \times S_n = R^n$, i.e., is the continuous state space which is an n-dimensional vectorspace (it has n state variables),
- $\phi = \{\phi_1, \phi_2, ..., \phi_n\}$ is a set of rate of change function with $\phi_i : S \times X \rightarrow S_{ci}$,
- $F = \bigcup_{i \in \{1,...,n\}} F_i$ with $F_i = \{f_{i,1}, ..., f_{i,m_i}\}$ is the set of threshold values for all dimensions i,
- $Y = \mathcal{B}^F$ is the output value set which is the crossproduct of all boolean value sets of the threshold sensors $thr_{f_{i,b_i}}, f_{i,b_i} \in F$,
- $\lambda : S \rightarrow Y_M$ is the output function which is defined by

  $$\lambda(s) = (thr_{f1,1}(s_1), thr_{f1,2}(s_2), ..., thr_{fi,b_i}(s_i), ..., thr_{fn,b_n}(s_n)) \tag{16}$$

  i.e., the output is defined by the values of the threshold sensors.

Such a specification defines a dynamical system DS in the following way:

- the time base T is the real numbers $\mathcal{R}$;
- input set X, output set Y, and state set $Q = S$ of the dynamical system are identical to those of the DEV&DESS,
- the admissible input segments Ω is the set of all piecewise constant segments over X and T,
- for any piecewise constant input segment $\omega :< t_1, t_2 > \rightarrow X \in \Omega$ and state s at time t_1 the global state transition function Δ is defined by the set of first order differential equations as follows:

  $$\Delta(s, \omega_{<t_1,t_2>}) = s + \int_{t_1}^{t_2} f(STRAJ_{s,\omega}(\tau), \omega(\tau))d\tau, \tag{17}$$

- the output function Λ of the dynamical system is given by

  $$\Lambda(s, x) = \lambda(s). \tag{18}$$

By the threshold values $f_{ib_i} \in F$ each dimension i of the continuous state space is partitioned into a set of *output blocks* [1]

$$OB_i = \{(f_{i,0}, f_{i,1} >, < f_{i,1}, f_{i,2} >, ..., < f_{i,b_i-1}, f_{i,b_i} >, ..., < f_{i,m_i}, f_{i,m_i+1})\} \tag{19}$$

with $f_{i,0} = -\infty$ and $f_{i,m_i+1} = +\infty$. The cross product $OB = \times_{i \in \{1,...,n\}} OB_i$ of the output form the so-called *output partition* of the state space. An output block $(< f_{1,b_1-1}, f_{1,b_1} >, ..., < f_{i,b_i-1}, f_{i,b_i} >, ..., < f_{n,b_n-1}, f_{n,b_n} >)$ of the output partition is characterized by the fact that all states in the same output block have the same threshold values, i.e.,

$$\lambda_d(s) = \lambda_d(\bar{s}) \Leftrightarrow \forall i \exists f_{i,b_i} : s_{ci} \in < f_{i,b_i-1}, f_{i,b_i} > \wedge \bar{s}_{ci} \in < f_{i,b_i-1}, f_{i,b_i} > . \tag{20}$$

The output values do not change as long as the continuous states stay in one output block and, therefore, the output trajectory is a piecewise constant time function $OTRAJ_{s,\omega} :< t_1, t_2 > \rightarrow Y$ with $OTRAJ_{s,\omega}(t) = \lambda(STRAJ_{s,\omega}(t))$.

[1] For simplicity reasons, we do not distinguish between open or closed intervals in this work.

2.6 DEVS Abstraction of Output-Partitioned DEV&DESS Models

Since an output partitioned DEV&DESS is a piecewise constant input/output dynamical system if has a DEVS representation. The transition functions of a DEVS are derived from an DEV&DESS in the following way: for every total state $((s,x),0)$ and input segment $\omega :< t_1, t_2 > \rightarrow X$ we define:

- the time advance is the time interval to the next output change, i.e., threshold crossing, provided a constant input x,

$$ta((s,x)) = min\{e|\lambda(s) \neq \lambda(s + \int_{t_1}^{t_1+e} f(STRAJ_{s,x}(\tau), x)d\tau\}, \tag{21}$$

- the internal transition function computes the state at the next threshold crossing occuring at time $t_1 + ta(s, x$

$$\delta_{int}((s,x)) = (s + \int_{t_1}^{t_1+ta(s,x)} f(STRAJ_{s,x}(\tau), x)d\tau, x), \tag{22}$$

- the external transition function updates the state at the input event

$$\delta_{ext}((s,x), e, x') = (s + \int_{t_1}^{t_1+e} f(STRAJ_{s,x}(\tau), x)d\tau, x'), \tag{23}$$

- the output function is inherited from the DEV&DESS

$$\lambda((s,x)) = \Lambda(s,x). \tag{24}$$

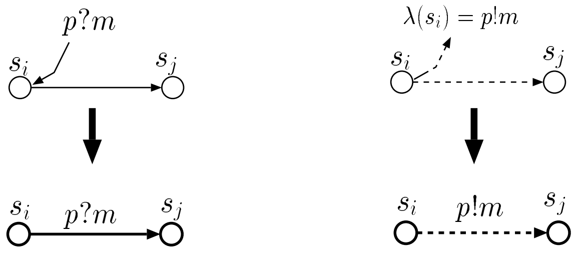
3 Supervisory control of discrete event systems

Supervisory control [2] is a kind of feedback control that supervises the behavior of a plant to achieve desired objectives. A design problem for a discrete event controller can be formulated as follows: given a plant with known dynamics and given control objectives, design a discrete event controller that satisfies the control objectives. To design a discrete event controller, we have to specify both the dynamics of the plant as well as the behavior of the plant that satisfies the control objectives. Given its universality, we model the plant behavior using the DEVS formalism. From the informally stated control objectives and the plant DEVS model, we specify a desired state trajectory that satisfies the control objectives. A discrete event controller is then derived by transforming the desired state trajectory into a DEVS model. The plant and the designed controller are coupled through proper interfaces resulting in a controlled system that satisfies the required behavior. We describe the basic concepts and design steps in more detail in the following sections.

3.1 Behavior of Atomic and Coupled Models

This section makes the connection between DEVS atomic and coupled models on the one hand and concurrent system concepts on the other. We present a graphical notation that exposes the duality between external and internal transition that will play an important role in the design methodology to be discussed later.

Recall that the dynamic behavior of a DEVS is determined by a sequence of internal or external transitions. An internal transition happens autonomously after a state determined sojourn time has expired and an external transition is caused by an external event.



(a) $\delta_{ext}(s_i, p?m) = s_j$ (b) $\lambda(s_i) = p!m \rightarrow \delta_{int}(s_i) = s_j$

Figure 3: graphical notation: (a) external transition (b) internal transition

In Fig. 3, (a) represents an external transition. An input event is specified using '?'. For example, an input event $in?m$ means that a message m is input at the input port in. Fig. 3(b) denotes an internal transition. An output event is specified using '!'. For example, an output event $out!m$ means that a message m is output at the port out. A dotted line represents an internal state transition specified by the internal transition function, δ_{int}. A sold line represents a state transition specified by the external transition function, δ_{ext}. This notation derives from CSP (Communicating Sequential Processes)[6] where a receiving process waits at a '?' step in its program until the sending process sends a matching '!' message. Although DEVS allows arbitrary communication between sender and receiver, we will adopt the CSP synchronous communication protocol restrictions for the following presentation. Thus, to communicate, we require that a component M_i sends an output event, $p_i!m_i$ and a receiving component M_j be waiting for it as an input event $p_j?m_j$, where the ports and messages must match (i.e., $p_i = p_j$ and $m_i = m_j$). In the case of a successful communication such as this, M_i undergoes an internal transition denoted as in Fig. 3(b) while concurrently, M_j undergoes an external transition denoted as in Fig. 3(a).

Since only the control portion of the state is typically shown graphically, we indicate transition conditions that depend on other state variables with a separator @. A time advance is attached to a state node to represent its sojourn time. An empty output event is denoted by $\emptyset$.

The concept of it controllability will be formulated to ascertain whether or not a discrete event controller exists that can enforce a desired state trajectory. If a plant is controllable, the concept of an it inverse DEVS can be used to transform the desired state trajectory into the specification of a discrete event controller. In this section, we define these two concepts.

3.2 Controllable DEVS

Given a discrete event plant $M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$, a *path* is a sequence of elements of S.

Definition 1 (weak controllability) *A path ST is weakly controllable if a series of internal and external transitions can take each state on the path to the next state on the path. More formally, a path ST is weakly controllable if:*
for all s_i, s_{i+1} such that s_{i+1} follows s_i in ST, either:
a) $\delta_{int}(s_i)= s_{i+1}$, or
b) there is a pair (e, x), where $0 < e < ta(s_i)$. and $x = p?m$, such that $\delta_{ext}(s_i, e, x) = s_{i+1}$, or
c) the path $\delta_{int}(s_i)$, s_{i+1}, $path(i + 2)$ is weakly controllable, or

d) there is a pair (e,x), where $0 < e < ta(s_i)$. and $x = p?m$, such that the path $\delta_{ext}(s_i,e,x)$, s_{i+1}, $path(i+2)$ is weakly controllable,
where $path(i)$ is the subsequence of the path following s_i.

Definition 2 (strong controllability) *A path is strongly controllable if either an internal or external transition can take each state on the path to the next state on the path. More formally, a path ST is strongly controllable if:*
for all s_i, s_{i+1} such that s_{i+1} follows s_i in ST, either:
a) $\delta_{int}(s_i) = s_{i+1}$, or
b) there is a pair (e,x), where $0 < e < ta(s_i)$. and $x = p?m$,
such that $\delta_{ext}(s_i,e,x) = s_{i+1}$.

The states on a strongly controllable path never veer away from those specified in the path. We now strengthen the requirements further by removing the ability of the controller to find an input after some elapsed time to keep the plant on the desired path:

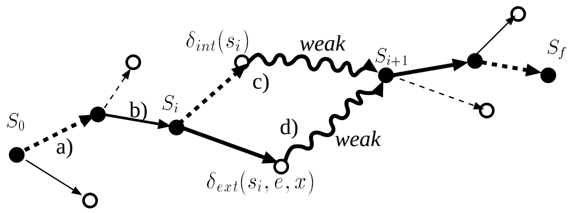
Definition 3 (very strong controllability) *A strongly controllable path ST is very strongly controllable if:*
for all s_i, s_{i+1} such that s_{i+1} follows s_i in ST, either:
a) $\delta_{int}(s_i) = s_{i+1}$, or
b) there is an input $x = p?m$ such that $\delta_{ext}(s_i,e,x) = s_{i+1}$ for all $0 < e < ta(s_i)$.

Figure4 shows cases for the three definitions of controllability. Filled circles represent states on the desired path, $ST = (s_0, s_1, \cdots, s_i, s_{i+1}, \cdots, s_f)$. Thick lines denote internal or external transitions that eventually lead one state on ST to the next. The annotations a), b), c) and d) on Figure (a) correspond to four cases of the definition of weak controllability. The curly thick line on the path marked c) and d) illustrate weak controllability since there is at least one sequence of internal and external transitions that drive every state on the path to the next. In Figure (b), illustrating strong controllability, there are direct internal or external transitions from every state on the desired path to the next. Note that although there are internal transitions from states on the path to states not on the path (for instance, from s_i to s_k) , a controller can schedule a command within the plant's holding time, $ta(s_i)$ to force it onto the desired path, (i.e., to s_{i+1}). Figure (c) illustrates very strong controllability. Here, every state on the desired path with path-straying internal transitions is controllable by a command that does not depend on the time the plant has been in the state. As far as the controller is concerned, this is equivalent to such states being passive (i.e., $ta(s_i)$ is infinity). After entering a passive state, the plant merely waits for the proper command to drive it to the next desired state. In other words for very strong controllability, the plant behavior must be such that either it moves through the desired state path autonomously or if it needs an input to do so it passively waits for the command to arrive.

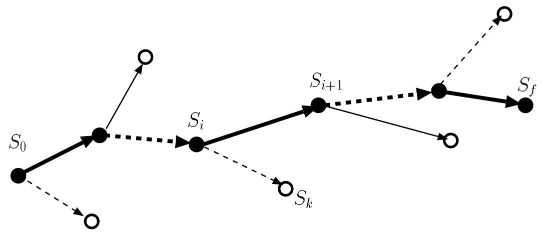
In the following, we focus on designing controllers for very strongly controllable plants. We remark that a weakly controllable plant may have to be enhanced with additional sensors and actuators to make it very strongly controllable. Thus there is a tradeoff between complicating the controller and complicating the plant.

3.3 Inverse DEVS

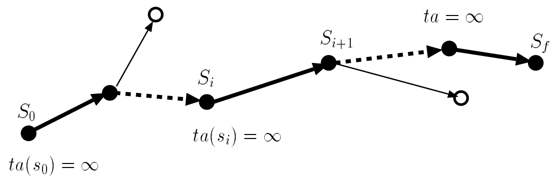
Definition 4 (inverse DEVS) *DEVS $M_i(M_j)$ is said to be an* inverse DEVS *of $M_j(M_i)$ iff the following properties hold:*
(a) State set bijection



(a) weakly controllable



(b) strongly controllable



(c) very strongly controllable

Figure 4: Controllability Definition: (a) weakly controllable (b) strongly controllable (c) very strongly controllable path. Annotations a),b),c) and d) on Figure (a) shows four cases of the definition of weak controllability.

$S_i \to S_j$
(b) Dual I/O events set
$X_i = \{p?m | p!m \in Y_j\}$
$Y_i = \{p!m | p?m \in X_j\}$
(c) Dual transition relations
$T_{i,int} = \{(q, p!m, r) | (q, p?m, r) \in T_{j,ext}\}$
$T_{i,ext} = \{(q, p?m, r) | (q, p!m, r) \in T_{j,int}\}$

(a) states that there is one-to-one correspondence between states of two DEVSs. (b) indicates that input events of one correspond to the output events of the other, and vice versa. (c) denotes that internal transitions match external transitions and conversely.

More specifically, the transition relations for DEVS $M_k, k = i, j$ above are defined by:
$T_{k,int} = \{(s_i, p!m, s_j) \mid s_j = \delta_{k,int}(s_i),\ p!m = \lambda_k(s_i) \in Y_k,\ \ s_i, s_j \in S_k\}$,
$T_{k,ext} = \{(s_i, p?m, s_j) \mid s_j = \delta_{k,ext}(s_i, e, p?m),\ 0 < e < ta(s_i),\ p?m \in X_k,\ \ s_i, s_j \in S_k\}$.

When inversely related DEVS are started in corresponding states, they undergo corresponding transitions which leave them in corresponding states.

An operation on a DEVS M_i creating an inverse M_j is called an *inverse DEVS transformation*. The inverse transformation is used to obtain a discrete event controller from a desired plant strongly controllable state trajectory.

4 Design methodology for DEC

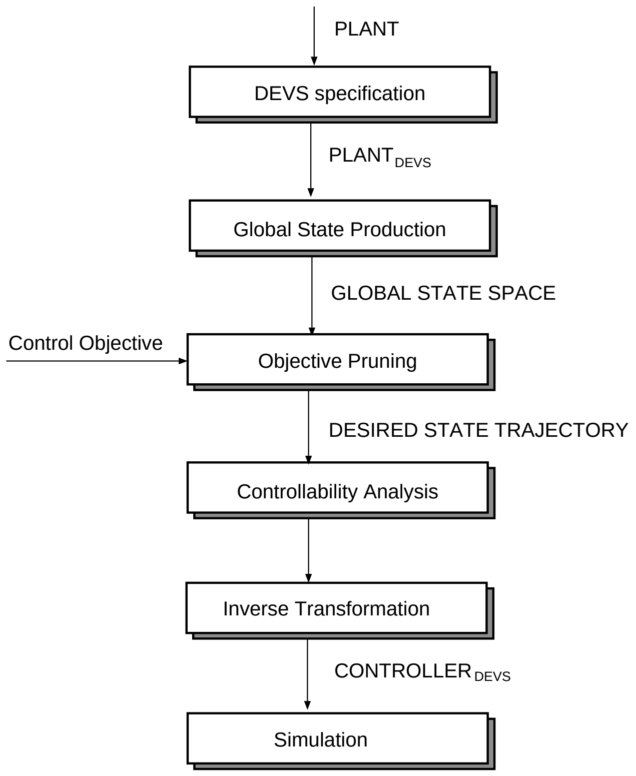


Figure 5: Design steps for a discrete event controller

A methodology for design of a discrete event controller based on the DEVS formalism and the concepts of inverse DEVS is outlined in Fig. 5. First, a plant is represented in the DEVS formalism based on the theory discussed earlier. Typically such a representation is a coupled model consisting of several atomic models. We then obtain a global state space which is a crossproduct of the component state sets, from which we extract a desired state path satisfying given objectives. The next step is to check the controllability of the desired state path. If the plant is very strongly controllable, we map the desired state path to a DES controller using the inverse DEVS transformation. State reduction is then applied to produce a minimal controller. If the plant is not very strongly controllable, we consider augmenting it with further sensors and actuators. The performance of the controller may be checked by simulation with a realistic model of the plant.

5 A simple example: control of water supply system

We will clarify the methodology with a simple example of a water supply system.

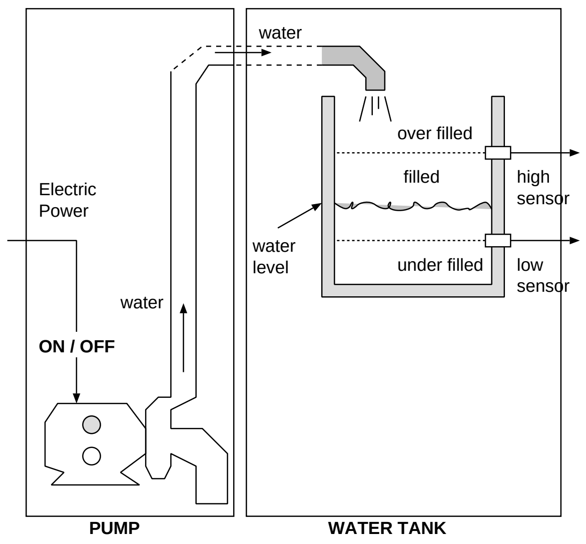


Figure 6: The plant of a water supply system

5.1 Problem description

An informal problem description is a starting point for the design of a controller for the water supply system. In Fig. 6, 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 pipe. The water pump is controlled by a ON/OFF switch. It fills the water tank through the water pipe.

5.2 DEVS model of the plant

Representation of the plant in the DEVS formalism is the first step in the design a discrete event controller whose objective is to keep the water level between the low and high sensors.

To specify a plant in the view of a discrete event system, we have to identify events occuring in the plant. These are:
1) the ON/OFF position changes of the motor switch.
2) the starting and stopping of water flow through the pipe.
3) low sensor switching between ON/OFF, the high sensor between ON/OFF

Fig. 7 shows a DEVS model of the plant. The initial state of the water tank is designated INIT. The level is represented by:

Low = under-filled,
Filled = filled,
High = over-filled

The input flow is represented by :

Out = stopped
In = flowing

With water flowing in, the state transitions from Low-In to Filled-In to High-In, generating proper sensor signals. If the in-flow is continued even though the water level exceeds the level of the high sensor, the water tank eventually overflows (Overflow). With the water pump stopped, the water level recedes as customers draw upon it. The state transitions from High-Out to Filled-Out to Low-Out. If the pump is not turned back on the water level reaches the bottom (Exhausted). As the water recedes, corresponding level sensor signals are generated.

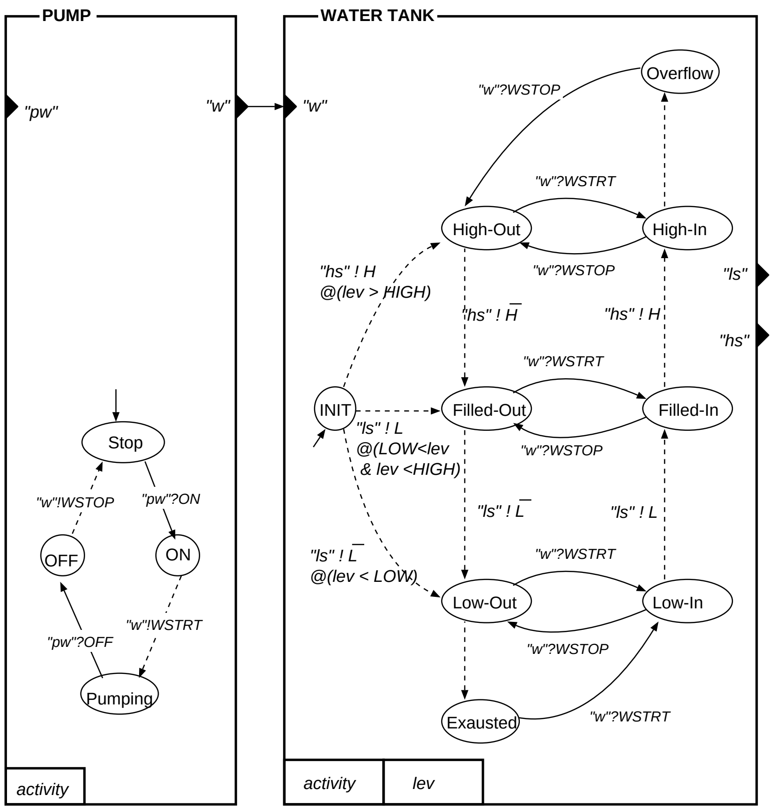


Figure 7: DEVS Representation of the water supply plant

The input port of the PUMP is "pw" and the output port is "w". If the water pump is turned on ("pw"?ON), it starts pumping water through the water pipe ("w"!WSTRT). If it is turned off ("pw"?OFF), it stops pumping and the flow of water will be stopped ("w"!STOP). The PUMP has four states

Stop = passive
ON = transient, issue a start output
Pumping = active,
OFF = transient, issue a stop output.

The two components, pump and tank, are coupled through port "w" (the the pipe). The coupled model, which is the plant representation, has one input port "pw" (power ON/OFF) and two output ports for the high and low sensors.

5.3 Controlled state trajectory

The next step is to get a global state transition diagram (GSTD) from the plant representation. This is obtained by a Cartesian product of state diagrams of each component. (Alternatively, we can use an approach called concurrency analysis that we have developed which significantly reduces the states that need to be examined.)

Let a state s in the 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. We formulate the desired state path, denoted K, by the constraints:

1. If $\exists$ `(`x`, Low-Stop)`, then eventually `(Stop, High-Out)`, where x means *don't care.*

2. $\exists$ no s such that (x,`Exhausted`) or (x,`Overflow`).

3. hysteresis: $\exists$ no transition between (x,`Filled-In`) and (x,`Filled-Out`)

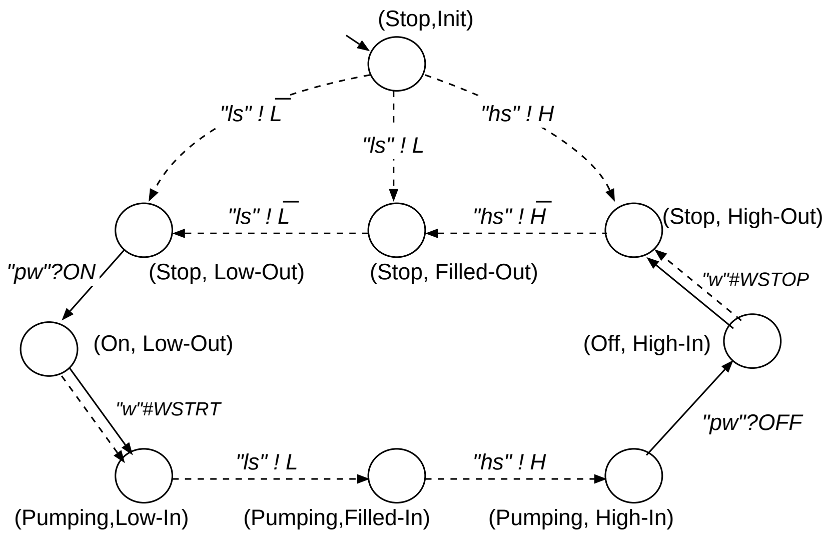


Figure 8: desired state trajectory of the plant

Now we can extract a controlled state trajectory from GSTD and the control objectives. Fig. 8 is the desired state trajectory. Initially, the state is changed into (Stop,Low-Out) after a few internal transitions. Then the motor should be ON, which is followed by a concurrent event "w"#WSTRT ("w"!WSTRT and "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 recedes, and then the motor should be turned back ON, and so on. This cyclic state path satisfies the three objectives.

The next step is to check if the controllability of the desired state path. Since every successive pair of states is connected by an internal or external transition with unconditional input, the plant is very strongly controllable.

However, consider Figure 9 in which the transition marked 1' occurs from (Stop, High-Out) to (Stop, Overflow). This is an internal transition which results from the remaining water in the pipe continuing to flow into the tank after the pump is turned off. This transition will occur if the high sensor is too close to the top of the tank. However, this internal transition is not on the desired path which requires (Stop,High-Stop) to be followed by (Stop, Filled-Out), i.e., for the water to start receding after the influx ceases **without** first overflowing. Moreover, there is **no** external transition that can rectify the situation since the pump has already been turned off. A similar internal transition, marked 2', occurs when the water reaches bottom despite the pump being turned back on due to the transport delay in the pipe.

We see that in this case, the plant is not weakly controllable. However, it can be made very strongly controllable by placing the high and low sensors sufficiently far away from the top and bottom of the tank, respectively.

5.4 Deriving a Discrete Event Controller

At this point we have represented a plant as a coupled DEVS, obtained the state transition behavior of its resultant, and interpreted the informally stated control objectives as a desired path in the state

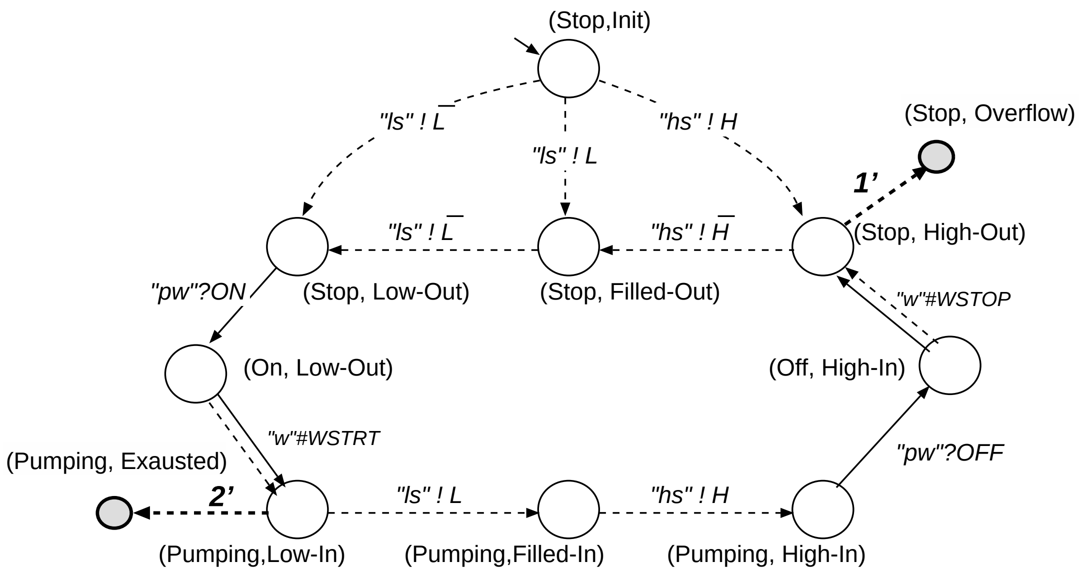


Figure 9: Possible state trajectory with high sensor failure

space of the resultant. This path is strongly controllable (assuming proper placement of the high/low sensors) so an associated state trajectory exists. Now, we derive a discrete event controller from the state trajectory using the inverse DEVS transformation.

Before the transformation, we can reduce the states of the desired state trajectory. The upper part of Fig. 10 is obtained by suitable state reduction from the trajectory in Fig. 8.

A discrete event controller is obtained by inverse DEVS transformation of the desired reduced state trajectory. The lower part of Fig. 10 shows the DEC after the transformation. Note that the transformation is straightforward and intuitive.

The two components, the controller, and the plant are to coupled through ports that have the same name to form a hierarchical coupled DEVS, shown in Figure 11.

The dynamics of the resultant follow the desired state trajectory since the controller and plant, once started in corresponding states, maintain this correspondence forever.

6 Discussion

More realistic DEVS models include states that generate null outputs. This can represent internal transitions in the plant which are not disclosed to the outside world via sensors. The inverse transformation may fail to produce a proper controller since the plant cannot generate an informative input to it.

An example of the above may occur when a sensor fails. To add more autonomy to the controller, it may be augmented with event-based control[15, 16, 10]. The controller then has a time-window in which it expects a sensor response. If this response does not occur within the expected window, the controller knows a failure has occurred and can initiate a diagnosis while taking corrective action. This enables the DEVS event-based controller to be used for monitoring the plant for occurrence of malfunctions. The controller employs a DEVS model which is an abstraction of the plant and which takes into account normal variations in its response time. The error messages issued by the DEVS-based controller contain important information about possible causes for the systems' malfunctions and can

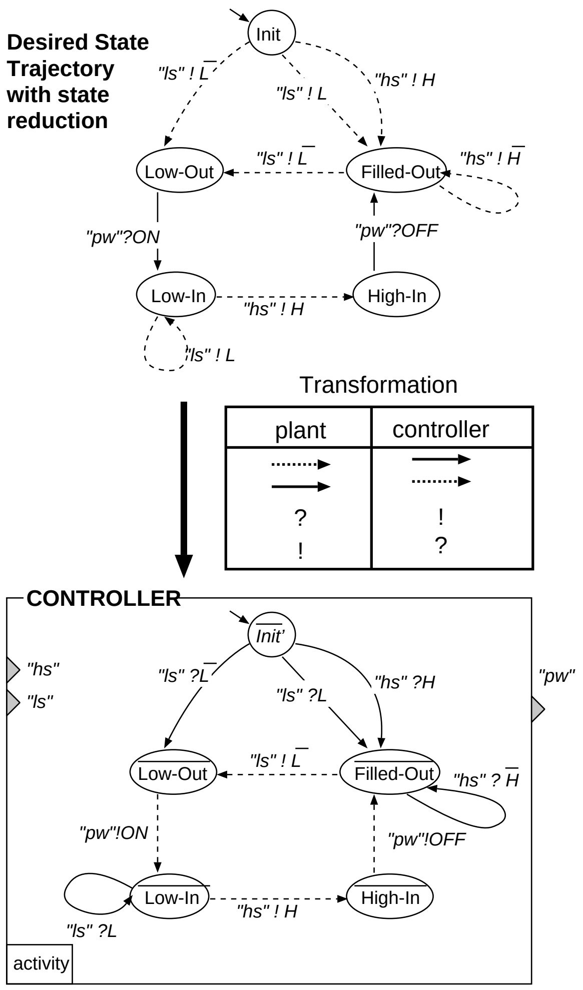


Figure 10: Design of a DES controller

be used by a diagnosing unit.

References

[1] P.J. Anstaklis, M.D. Passino, and S.J. Wang. Towards intelligent autonomous control systems. *Journal of Intelligent and Robotic Systems*, 1(4):315–342, 1989.

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

[3] D. Harel et al. Statemate: A working environment for the specification of compex reactive system. *IEEE Trans. on SE*, 16(4):403–414, 1990.

[4] Fr.Pichler and H. Schwaertzel, editors. *CAST Methods in Modelling*, chapter 3, pages 123–241. Springer-Verlag, 1992.

[5] Y. C. Ho. Special issue on discret event dynamic systems. *Proceedings of the IEEE*, 77(1), 1989.

[6] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

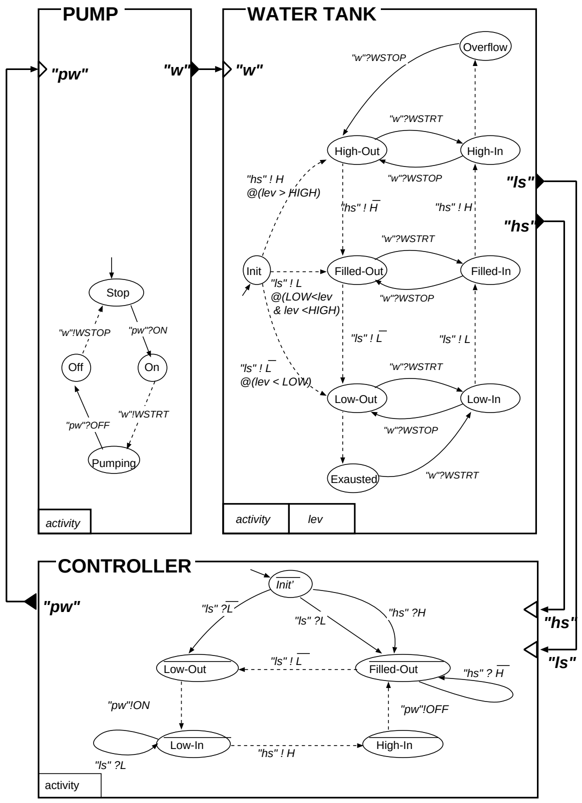


Figure 11: Whole water supply system : coupling the plant and the controller designed

[7] A. Meystel. Intelligent control: A sketch of the theory. *Journal of Intelligent and Robotic Systems*, 2(2):97–107, 1989.

[8] L. Padula and M. A. Arbib. *Systems Theory.* Saunders, Philadelphia, 1974.

[9] H. Praehofer. *System Theoretic Foundations for Combined Discrete-Continuous System Simulation.* PhD thesis, Johannes Kepler University of Linz, Linz, Austria, 1991.

[10] H. Praehofer and B.P. Zeigler. Automatic abstraction of event-based control models from continuous base models. *submitted to IEEE Trans. Systems, Man and Cybernetics*, 1995.

[11] P.J.G. Ramadge and W.M. Wohnham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[12] G. Wunsch, editor. *Handbuch der Systemtheorie.* Akademie-Verlag, Berlin, 1986.

[13] B. P. Zeigler. *Theory of Modelling and Simulation.* John Wiley, New York, 1976.

[14] B. P. Zeigler. *Multifacetted Modelling and Discrete Event Simulation.* Academic Press, London, 1984.

[15] B. P. Zeigler. DEVS representation of dynamical systems: Event-based intelligent control. *Proceedings of the IEEE*, 77(1):72–80, 1989.

[16] B. P. Zeigler. *Object-Oriented Simulation with Hierarchical, Modular Models.* Academic Press, London, 1990.

[17] B. P. Zeigler and W. H. Sanders. Preface to special issue on environments for discrete event dynamic systems. *Discrete Event Dynamic Systems: Theory and Application*, 3(2):110–119, 1993.