Modeling Combined Time- and Event-Driven Dynamic Systems

Mieczyslaw M. Kokar and Kenneth Baclawski
Northeastern University
Boston, MA 02115

September 17, 2000

Abstract

Real-time distributed systems such as logistical systems, distributed sensor systems and intelligent highway vehicle systems, are complex dynamic systems that not only evolve in real-time but also must respond to unanticipated events. Furthermore, these systems require modeling real-world phenomena accurately as well as solving resource allocation problems for which it is infeasible to find the optimum solution. Modeling such systems requires combining classical methods for analyzing continuous dynamic systems with methods from computer science for analyzing discrete systems. Furthermore, new techniques must be introduced that reconcile these two very different kinds of system, and a new methodology is needed to prove that systems meet requirements other than the classical optimum or reachability properties, such as achieving a solution that is "good enough" as well as "soon enough." To contribute to this goal, we present an ontology for classifying general systems. This ontology provides a common terminology for understanding different kinds of dynamic system. In addition, we propose an outline of how systems such as real-time distributed resource management systems can be modeled.

1 Introduction

Modeling and simulation of dynamic systems has always been an important issue in both science and engineering. General methodologies of modeling and simulation were studied extensively in systems science and general systems theory (cf. [12, 10]), control (cf. [14, 6]), artificial intelligence (cf. [17]). The systems science approach is based on the mathematical formulation of a general dynamic system. Most of the scientific effort in systems science was focused upon quantitative models and automatic quantitative simulation. The main thrust of the systems science research has been to provide models that would give accurate representations of the behavior of a real physical system. In this approach, future behaviors are generated through quantitative simulation which "executes" a simulation model, typically at fixed time steps, to obtain quantitative values of state and/or output variables.

The purpose of this paper is to introduce the notion of a general dynamic systems and to assess the adequacy of the current modeling techniques for complex dynamic systems such as real-time distributed systems. We begin by introducing the terminology of general systems theory in Section 2. The terminology is then used as a means of classifying general systems in Section 3. Many examples of general systems are described and classified according to the terminology and classification scheme we introduce. The main paradigms for modeling such systems are discussed in Section 4, and in the concluding section we assess the adequacy of these modeling techniques for systems such as real-time distributed systems.

2 General Systems

A *system* is a combination of components that act together to perform a function not possible with any of the individual parts. A system can be either a mathematical system or a physical system. *Modeling* or *abstraction* (cf. [7]) is a relationship between two systems such that one system abstracts some features of another system, while preserving certain desirable properties. Abstraction is the "process of suppressing irrelevant detail to establish a simplified model" [9]. Abstraction levels and viewpoints, together with their appropriate structuring, lead to clarity and understandability. This is most often used for the case of a physical system whose behavior is described by a mathematical system. The first system is called the *ground* system and the second system is called the *abstract* system. The abstract system is said to *model* the ground system. When the ground system is a physical system, then it is common to refer to the abstract system as being the *mathematical model*. However, both the ground system and the abstract system can be mathematical systems, and the purpose of modeling is to replace a more complicated system by a system that is simpler and easier to handle.

Because of the huge variety of systems, it is helpful to classify systems by using properties that help to distinguish different kinds of system from each other. The rest of this section introduces some of the most commonly used properties for classifying systems. The properties introduced below refer only to mathematical systems. However, it is not uncommon to speak of them as being properties of the system being modeled by a mathematical system.

2.1 Static Systems

A *static* system has the property that the current output is entirely determined by the current input. Mathematically, one can express this as follows.

Definition 1 *A* static system $\mathcal{S}$ *is a triple* $\mathcal{S} = (I, O, F)$ *for which*

1. I *and* O *describe the* input space *and the* output space*, respectively.*
2. F *is a* global function $F\colon I \to O$.

There are several definitions of the notion of a static system in the systems science literature, but they are generally variations on the Definition 1. For example, [12] allows the global function of a static system to depend on an "initial state." In effect, each initial state determines a static system in the sense defined above. In other words, it is a parametrized collection of static systems.

It is common for the input and output spaces to have the form of a cartesian product (or a subset of a cartesian product). For example, the input space might have the form $I = I_1 \times I_2 \times \ldots \times I_n$, where each component I_j is more elementary (e.g., the set of real numbers or the integers). Each component corresponds to an *input variable* of the system. Similar notation is used for the output space.

Compared to dynamic systems (defined below), static systems can be characterized as systems for which the output depends only on the current input. It is, of course, unrealistic for a physical system to be static in any of the senses considered above. A physical system inevitably changes eventually. However, one can abstract away enough of the features of a physical system for it to be static (or nearly static).

2.2 Dynamic Systems

While a static system is a useful notion, it is not very effective for modeling most physical systems because they tend change over time. A system is said to be *dynamic* if it evolves with respect to some notion of time. Incorporating time into the notion of a system complicates it a great deal, but the modeling power is substantially greater as a result.

There are several definitions of the notion of a dynamic system in the systems science literature [12, 10, 14]. The differences between various definitions are primarily stylistic rather than fundamental. The definition presented below is closest to the one presented in [16].

Definition 2 *A* general dynamic system, $\mathcal{S}$, *is an 8-tuple*

$$\mathcal{S} = (T, I, O, Q, P, F, g, \leq),$$

where

1. *T is a totally ordered set with order relation $\leq \subset T \times T$. The ordered set T is called the* time set.

2. *I and O describe the* input space *and the* output space*, respectively. The input space is also called the* event space*, and in this case, elements of I are called* events.

3. *Q describes the* space of inner states *(or more simply, the* state space*) of the dynamic system.*

4. *P is a subset of $\{p : T \to I\}$ such that P is closed under* splicing. *The splicing operation is a binary operation on functions $p : T \to I$ that produces a new function by using the first function before a specified time, and a second function after that time. In other words, the splicing of any two functions p_1 and p_2 at time t is*

$$p = p_1 \perp_t p_2 = \begin{cases} p_1(\tau), \tau < t \\ p_2(\tau), \tau \geq t. \end{cases}$$

The set P is called either the set of input processes *or the* set of event streams*, depending on whether elements of I are referred to as inputs or events, respectively.*

5. *A function $F\colon T\times T\times Q\times P \to Q$, called the* global state transition function*, which satisfies these three conditions:*

   (a) *(Consistency) For every $t \in T, q \in Q, p \in P, F(t,t,q,p) = q$;*

   (b) *(Semigroup) For every $t_0 \in T, t_1 \in T, t_2 \in T, q \in Q, p \in P$, if $t_0 < t_1 \leq t_2$, then $F(t_2,t_0,q,p) = F(t_2,t_1,F(t_1,t_0,q,p),p)$;*

   (c) *(Causality) For every $t_0 \in T, t_1 \in T, q \in Q, p_1 \in P, p_2 \in P$, if $p_1(\tau) = p_2(\tau)$, for every $\tau \in (t_0,t_1]$, then $F(t_1,t_0,q,p_1) = F(t_1,t_0,q,p_2)$,*

6. *g is a function $g : T\times Q \to O$, called the* output function*.*

The most significant difference between a static system and a dynamic system is that a dynamic system has an internal state that evolves over time and that determines the output. The structure of the internal state space as well as its evolution function can be difficult to construct. While the input and output variables of a dynamic system are "visible" outside the system, the state is, by its nature, hidden within the system. Sometimes it can be helpful to think of each element $q \in Q$ of the state space as being that part of the history of the system that is sufficient for computing the current output of the system.

As in the case of a static system, the internal state space Q can be a cartesian product (or subset of a cartesian product) of simpler components. The individual components are sometimes called the *state variables*. The input, output and state variables are collectively called the *system variables*. Note that the four conditions that define the notion of dynamic system constrain the choice of state variables.

The global state transition function $F(t_1,t_0,q,p)$ specifies the state of the system at time t_1 if the system is in state q at time t_0 and the system receives the input process p. Note that the input process p specifies *all* inputs over the entire time set. However, the causality condition specifies that only the inputs between t_0 and t_1 can influence the state transition from time t_0 to time t_1. Furthermore, the consistency condition specifies that state transitions cannot be instantaneous, which is a natural condition for any realistic dynamic system.

The splicing property of the input processes gives the flexibility of choosing input values during any small period of time without any undue restriction on the choices of input values at earlier or later times. This is particularly important for control models in which the input values are influenced by the past behavior of the system (i.e., by "feedback").

The semigroup condition ensures that one can compute state transitions on any time interval $(t_0,t_1]$ either in a single computation $F(t_1,t_0,q,p)$, or by any sequence of steps obtained by partitioning the time interval $(t_0,t_1]$ into subintervals.

2.3 Time-Varying and Time-Invariant Dynamic Systems

A dynamic system is *time-invariant* if its *behavior* is independent of time. Like a static system, the behavior of a time-invariant system does not explicitly depend on any input except the current input. However, unlike a static system, a time-invariant system may have an internal state which can affect the output. A *time-varying* system is a system that is not time-invariant. The mathematical model of general dynamic system given by Definition 2 is for a time-varying system. A time-invariant system can be defined using a a simplification of that model. Mathematically, a time-invariant system is a dynamic system for which the following hold:

1. The time set T is a totally ordered Abelian group. In particular, this means that one can compute the difference $t_1 - t_0$ when $t_0, t_1 \in T$.

2. The global transition function F satisfies the condition: for every $t_i \in T$, if $t_1 - t_0 = t_3 - t_2$, then $F(t_1, t_0, q, p) = F(t_3, t_2, q, p)$. In other words, $F(t_1, t_0, q, p)$ depends only on the difference $t_1 - t_0$. As a result, F is uniquely determined by the function $F_0 \colon T \times Q \times P \to Q$ defined by $F_0(t, q, p) = F(t, 0, q, p)$.

3. The output function g satisfies the condition: for every $t_0, t_1 \in T$, $g(t_0, q) = g(t_1, q)$. It follows that g is uniquely determined by the function $g_0 \colon Q \to O$ defined by $g_0(q) = g(0, q)$.

For every time-varying system there is an equivalent time-invariant system obtained by replacing the state space Q of the time-varying system by $T \times Q$. In other words, the variation on time can be transformed into a variation of state by simply incorporating the time set as an additional state variable. Incorporating the time set as a state variable does simplify the model, but at the cost of reducing the modeling power. Whatever is known about the behavior of such a system for time t is unique to this time instant and cannot be used to predict the behavior of the system at any other time instant. Nevertheless, it is common to assume that a dynamic system is time-invariant, and we do so for the rest of this paper.

2.4 Time-Driven and Event-Driven Systems

Time-driven systems change state in response to a uniformly progressing physical time. The physical time is a global variable that has exactly the same value at any level of the system. *Event-driven* systems (cf. [6]) change state in response to the occurrence of asynchronous discrete events that result in instantaneous state transitions. The state of an event-driven system is unchanged between event occurrences.

The distinction between these two notions is the result of two fundamentally different concepts of clocks and time as follows:

1. A clock ticks at a uniform rate (which could be continuous as in physical time, or discrete as if the clock were a metronome). At every clock tick, an input (also called an event) e is selected from the input set I. If events occur for only some clock ticks and not others, then simply introduce a "null event" to represent the

case of a clock tick having no corresponding event. In this case, state transitions are *synchronized with* or *driven by* the clock ticks. The clock alone is responsible for (drives) state transitions. For this reason, such a system is said to be *time-driven*.

2. A sequence of events is presented to the system. These events are not necessarily known in advance and need not occur at a clock tick. Indeed, there need not be a clock at all in the usual sense. In such a system, events occur *asynchronously*. The notion of time, if it is explicitly defined at all, is simply an index that determines the position of an event occurrence within the sequence of event occurrences. In other words, the event stream drives the clock rather than the other way around. Such a system is said to be *event-driven*.

The notions of time- and event-driven system are not complementary. A system can be both time-driven and event-driven. We introduce the term *fully-driven* for a dynamic system that is both time-driven and event-driven.

It is helpful to consider some examples to illustrate the distinctions among the three kinds of system introduced above.

- There are various ways one can model a computer processor. One example is the RISC (reduced instruction set computer) architecture. In this architecture, the instructions are simple enough that at every clock tick just one instruction is executed by the processor. If interrupts are disabled, then such a processor can be modeled as being time-driven. In this case, the clock is a physical device in the processor that synchronizes the instruction executions. If some instructions require more than one clock tick, then one can introduce null events called "wait states" that allow this system to continue be modeled as a time-driven system. If interrupts are enabled, then the processor is a fully-driven system since there is now an asynchronous event stream, in addition to the processor clock, that is driving the system. If one focuses on just the interrupt stream as being the driver of the system, then the system can be modeled as an event-driven system.

- Consider next a software system called a compiler. The purpose of a compiler is to convert an input text file into a program that can be executed by the computer. To a compiler, the events are syntactic structures occurring in the input text file. An event occurrence is a particular occurrence of a syntactic structure within the input text, and the notion of time in this case is simply the position of the syntactic structure within the input text file. A compiler is just one example of a large class of system in computer science that takes a "language view" toward dynamic behavior. Such a system is modeled without there being any direct connection between physical time and the event occurrences. Introducing physical time into such a system requires reconciling the fact that there are two very different notions of time: the abstract time used by the computer program and the "real time" of the physical world in which computation is taking place.

The impact of the distinction between time-driven and event-driven systems is especially important when one is composing systems from smaller systems. In time-driven

systems, the time is a global variable that is the same for every component of a system. In event-driven systems, on the other hand, every component has a different notion of time, since the time for a component is determined by the event stream that is presented to it. So not only is the notion of time different on different levels of a dynamic system, but the notion of time is different for every component on the same level as well.

2.5 Deterministic and Stochastic Systems

A system is *stochastic* if at least one of its system variables is a random variable. A probabilistic framework is required to model the behavior of a stochastic system. Mathematically, a stochastic system is a system in which some of the system variables are random variables.

A system is *deterministic* if its output variables are all completely determined by the input and system state. Note that this includes the somewhat inappropriately named "nondeterministic" systems used in finite state machines (automata) in Computer Science. We discuss this class of dynamic system in the next section below.

For every stochastic dynamic system, one can define a deterministic dynamic system that is equivalent to it. This is done by replacing the state space Q with the set of probability measures on Q. The notion of a state is then a probability measure, and state transitions take a probability measure on the original state space Q to another probability measure on Q. This mechanism is very useful both theoretically and in practice. However, it has the disadvantage that it replaces a finite-dimensional space by an infinite-dimensional one.

While a system can have both deterministic and stochastic output variables, if any of the output variables are stochastic, then the entire system is regarded as being stochastic. Indeed, since every stochastic system can be converted to a deterministic one (as discussed above), it may be more relevant to classify systems by whether their system spaces are finite-dimensional rather than whether they are deterministic.

All physical systems will necessarily have some degree of randomness, so they are all stochastic systems. However, the stochastic variability might not be significant or might not be relevant. In this case, it is reasonable to refer to the system as being deterministic, even though it is only approximately so.

2.6 Continuous and Discrete Systems

If the state variables of a system are real or complex, then the system is said to be a *continuous-state* system. If every state variable of a system takes values in a discrete (possibly infinite) set, then the system is said to be a *discrete-state* system. A *finite-state* system is a special case of a discrete-state system for which the state space is finite. Note that the word "continuous" here refers only to the type of a state variable and does not imply that any functions are continuous in the mathematical sense. The term "continuous" is short for the mathematical term "continuous manifold," also called a "real manifold."

Similar distinctions can be made for the input and output variable of a system. If all of the system variables are continuous, then the system is said to be a *continuous*

system. Similarly, if all of the system variables are discrete, then the system is called a *discrete* system. If some of the system variables are continuous and others are discrete, then the system is called a *hybrid* system. We consider hybrid systems in more detail in Section 4.

Note that the time set is not included in the requirement above. In other words, a continuous system can have a time set that is discrete, and a discrete system can have a time set that is continuous.

2.7 Continuous-Time and Discrete-Time

The behavior of a dynamic system is described by a totally ordered set of inputs, outputs and states. By analogy with physical time, the totally ordered set is called the time set of the system. However, it is not necessary for the system time to have any relationship to physical time. If the time set T is the real numbers or an interval of real numbers, then the system is said to be a *continuous-time* system. If the time set is discrete (i.e., the set of integers or an interval of integers), then the system is said to be a *discrete-time* system.

The semigroup property of the global transition function F implies that F is determined by transitions on arbitrarily small intervals. In the case of discrete-time systems, this means that F is uniquely determined by transitions from the *current time* to the *next time*. Furthermore, the causality condition implies that F only depends on the current input. As a result, the function F is uniquely determined by a *local transition function* $f: Q \times I \to Q$.

The analysis above holds whether the dynamic system is continuous or discrete. In the continuous case, the local transition function is often stated in terms of the difference between the current state and the new state, and the evolution of the system is given in terms of a "difference equation." This is closely analogous to the differential equation technique for expressing the evolution of a continuous-time system to be introduced below.

For continuous-time systems, this same analysis can be performed provided that the global transition function is sufficiently smooth. In particular, this requires that the dynamic system must be continuous (i.e., the input, output and state spaces must be continuous). When this is the case, the semigroup condition allows the decomposition to subintervals of time to be carried out down to infinitesimal intervals. The result is the "differential" form of the global transition function. If the input process is also assumed to be smooth, then the global transition function is uniquely determined by a local transition function $f: Q \times I \to Q$ just as in the discrete case above. The global transition function is then determined by the following differential equation:

$$Q'(t) = f(Q(t), p(t)).$$

Many authors define continuous dynamic systems using the differential form of the mathematical model, even though this does not define the most general continuous dynamic system. For a continuous dynamic system, if there is a local transition function f and if it is linear, then the dynamic system is said to be a *linear* dynamic system. Continuous systems that are not linear are called *nonlinear* dynamic systems.

3 Classification of General Systems

The properties of general systems discussed above, subdivide all general systems along the following classification dimensions:

1. Dynamics. Two cases: static and dynamic.
2. Driving Type. Three cases: time-driven, event-driven, fully-driven.
3. Randomness. Two cases: deterministic and stochastic.
4. Type of Variables and Linearity. Five cases: finite, (infinite) discrete, continuous linear, continuous nonlinear, hybrid.
5. Time. Two cases: continuous-time and discrete-time.

In theory, there are 120 cases defined by the classification dimensions above. However, not all the cases are meaningful or distinguishable from one another.

3.1 Classification of Static Systems

In theory there are 60 kinds of static system determined by the classification above. However, there are actually only 10 distinguishable possibilities because there is no dependence on time. In particular, there is no significant distinction between continuous-time and discrete-time or between time-driven and event-driven for static systems. The following table shows typical functions that are used in static system models. The slots have been filled in with examples of the kind of function in each case. The examples are not meant to be exhaustive.

	Deterministic	Stochastic
Finite	Finite Function	Finite Random Variable
Discrete	Discrete Function	Discrete Random Variable
Linear	Linear Transformation	Stochastic Linear Function
Nonlinear	Nonlinear Vector Function	Real-Valued Stochastic Function
Hybrid	General Function	General Random Variable

3.2 Classification of Dynamic Systems

Many combinations of the properties in Section 2, and listed at the beginning of this section, have their own names, and substantial infrastructures exists for these cases.

- Classical Physics. These systems are characterized by being dynamic, time-driven, continuous and continuous-time systems. Within this class of system, linear systems have been studied much longer and are easier to deal with than nonlinear systems. One common technique for approximating classical physical models is to replace differential equations with difference equations. In other words, continuous-time is approximated using discrete-time.

- Discrete Event Systems (DES). These systems are dynamic, event-driven, discrete systems [6, page 35].

- Markov Chains. Such a system is a dynamic, event-driven, stochastic, discrete system. In other words, a Markov chain is a stochastic DES. Queuing theory uses continuous-time Markov chains, while discrete-time Markov chains have been used successfully in many diverse areas such as speech recognition and natural language processing, In the latter application domain, the time set of the Markov chain has no relationship to physical time.

- Automata. In this case, the system is a dynamic, event-driven, deterministic, discrete, finite-state, discrete-time system. In other words, a deterministic, finite-state, discrete-time DES. Automata are often used in Computer Science to model programs. For this reason, automata are also called *finite state machines*. One variation on this class of dynamic system is the "nondeterministic" automaton. In such an automaton, the local state transition function allows a transition from one state to an arbitrarily chosen state taken from a set of allowable states. This is very different from a stochastic dynamic system in which the state transition function chooses a state according to a probability distribution. In automata theory, nondeterministic automata are simply a convenient device for constructing deterministic automata. One can transform a nondeterministic automaton into a deterministic one by simply replacing the state space by the power set (i.e., the set of all subsets of the state space). The resulting automaton is considerably more complex since the state space has exponentially more states than the original. Note that this mechanism is very similar to the one discussed in Section 2.5 for converting a stochastic system into a deterministic one.

- Computer. A computer that does not allow interrupts can be modeled as a time-driven, deterministic, finite-state, discrete-time system. Note that this is not a special case of automata or even of DES, which are not time-driven. When asynchronous interrupts are allowed, then the model for a computer is fully-driven.

- Real-Time Resource Managers. Systems that manage resources in real-time are fully-driven, hybrid, continuous-time systems. They may be either deterministic, as in the case of logistical systems, or they may be stochastic, as in the case of sensors. These systems are often highly distributed, but that issue is not considered here.

- Computer Device Managers. The hardware that manages computer devices are usually deterministic, fully-driven, hybrid, discrete-time systems. These managers are often called *controllers* although their role is closer to management than to control as defined by the control theory literature. They typically have their own clock hardware, although they can also depend on the processor clock for their discrete time set. Much of their functionality is determined by the clock, so they are time-driven. However, they must also respond to asynchronous commands and other input, so they are fully-driven. While some controllers are purely digital,

and so qualify to be considered discrete systems, most have to deal with devices that are not entirely digital, so most controllers are hybrid. Video cards and disk controllers for a PC are typical examples of such controllers.

- Hysteresis. Systems such as switches and relays have both continuous and discrete variables. The continuous variables of the system have no time-dependence and so form a static continuous system. However, a hysteresis system also has a finite state that is determined by a continuous, asynchronous event process. So such systems are deterministic, event-driven, hybrid, continuous-time systems.

- Bouncing Ball. A "bouncing" ball traveling in a room will change its velocity discontinuously when it hits a boundary. Such a system is deterministic, time-driven, hybrid and continuous-time.

The properties in Section 2 classify the dynamic systems into 60 cases shown in the following two tables, in which "discrete" is short for "infinite discrete."

Deterministic			
	Event-Driven	Time-Driven	Fully-Driven
Finite, Discrete-Time Finite, Continuous-Time	Automata DES	Computers	Computers
Discrete, Discrete-Time Discrete, Continuous-Time	DES DES		Device Managers Device Managers
Linear, Discrete-Time Linear, Continuous-Time		Physics Physics	
Nonlinear, Discrete-Time Nonlinear, Continuous-Time		Physics Physics	
Hybrid, Discrete-Time Hybrid, Continuous-Time	Hysteresis	Bouncing Ball	Device Managers RTDRM (Logistics)

Stochastic			
	Event-Driven	Time-Driven	Fully-Driven
Finite, Discrete-Time Finite, Continuous-Time	Markov Markov		
Discrete, Discrete-Time Discrete, Continuous-Time	Markov Markov		
Linear, Discrete-Time Linear, Continuous-Time		Physics Physics	
Nonlinear, Discrete-Time Nonlinear, Continuous-Time		Physics Physics	
Hybrid, Discrete-Time Hybrid, Continuous-Time			RTDRM (Sensors)

4 Paradigms for Modeling Fully-Driven and Hybrid Dynamic Systems

The problem of modeling fully-driven dynamic systems requires that one reconcile the two different notions of time that are driving the system. This problem has been addressed by hybrid systems research where the issue was apparently first observed and techniques were developed for dealing with them. In this section, we discuss the approaches that have been developed for hybrid system modeling, with emphasis on those modeling techniques that are most useful for modeling fully-driven dynamic systems. Most of the treatment in this section was derived from [2].

There are five basic approaches (or *paradigms*) for studying hybrid systems. One can either suppress the hybrid nature of the system and convert it into a purely discrete or purely continuous system, or one can combine the power of the two kinds of system by treating the system as a continuously interacting set of automata or a discretely interacting set of continuous systems, or a combination of these two. More specifically, the approaches are as follows:

1. *Aggregation.* Suppress the continuous dynamics. This is the most common approach in the literature (e.g., [5]). There are many drawbacks to this approach:

   - *Nondeterminism.* The automaton one obtains is usually nondeterministic (see [1]). While nondeterministic automata can be converted to deterministic automata (as in Section 3.2), the resulting automaton is more complex, and the analysis of such an automaton produces much weaker consequences.
   - *Nonexistence.* There may not be any finite automaton that adequately models the behavior of the ground system.
   - *Partition Problem.* Even when a finite automaton exists, it can be a very deep and difficult problem to find one such that it adequately models the ground system.

   Generally speaking, aggregation can be fully carried out only under strong assumptions on the hybrid system.

2. *Continuation.* This is the opposite of aggregation (cf. [4]). The discrete variables are converted to continuous ones so that the system becomes a purely continuous one. The fact that there are simple continuations of finite automata, pushdown automata and Turing machines makes this approach more compelling than aggregation. However, it also has serious drawbacks:

   - *Arbitrariness.* How the continuation is accomplished is largely arbitrary, and care must be taken to insure that the continuation does not introduce behavior that is an artifact of the continuation.
   - *Complexity.* Continuation does not eliminate inherent complexity. The complexity is simply shifted to another part of the system where it must eventually be dealt with.

- *Artificiality.* It can lead to an unnatural analytical loop of going from discrete to continuous and back to discrete.

3. *Automatization.* Model the system as a collection of automata interacting within the framework of a time-driven continuous dynamic system (cf. [13]). The main difficulty with this approach is the inherent incompatibility between the event-driven nature of the automata and the time-driven nature of the dynamic system that connects the automata with each other. There are two main ways to reconcile this incompatibility:

   - Force synchronization at regular clock ticks.
   - Attach elapsed times to discrete event transitions.

   This paradigm includes notions such as *timed automata* and *qualitative dynamic systems*.

4. *Systemization.* (e.g., [3]) This is the opposite of automatization. In this case, the component systems are continuous dynamic systems that interact within the framework of an automaton. As with automatization, it is still necessary to reconcile the incompatibility between the time-driven components and the event-driven framework. The main ways to achieve this reconciliation are:

   - Force a uniform timing structure via "timing maps."
   - Sequentially synchronize the continuous dynamic systems at event times when the dynamic systems enter prescribed subsets of the state space.

   In the latter technique above, the synchronization points are of two kinds:

   (a) A *jump* or *impulse* is a discontinuous change in the state as a result of entering a prescribed subset of the state space. The "bouncing ball" system is an example of a system with jump discontinuities.

   (b) A *switch* is a discrete change in the dynamic system as a result of entering a prescribed subset of the state space. This is a more dramatic discontinuity than a jump, for it causes the entire dynamic system to be replaced by another one. An example of a switch is a hysteresis model. Such a model has a discrete "memory" that determines which transition function is applied at any given point in time.

5. *Hybridization.* (cf. [13]) This is a combination of the two previous approaches. Such a model contains both automata and dynamical systems. This model is the basic model used in the Hybrid Systems community (cf. [8]). In this model, discrete systems, represented by finite automata, and continuous systems, represented by ordinary differential equations, interact through two kinds of *interface*, a digital-to-analog (DA) or an analog-to-digital (AD) interface. One of the issues in this approach is how to specify the two kinds of interface. The dynamical system's output must be unified with the finite automaton's input (and vice versa). This

unification process is not a simple matter. One way to approach this problem is to define the interfaces through *abstraction*. Abstractions of dynamical systems were investigated, for instance, in [15]. In [11] a AD interface was specified as an abstraction of a general dynamic system so that an associated automaton can be constructed that is provably consistent with the underlying GDS. The shortcoming of this approach is, as we mentioned earlier, the lack of good tools for analyzing such hybrid systems.

5 Proposal for Modeling Fully-Driven Hybrid Dynamic Systems

The paradigm that appears to be the most effective for modeling fully-driven systems is hybridization, which combines automatization and systemization. While this paradigm has been effective for modeling many hybrid systems, it does not completely address several important issues that arise in real-time systems, especially in real-time systems that involve one or more of the following:

1. Resource management and other intractable computational problems. It could be known that an optimal solution to a problem exists, and it may be easy to check that a solution is optimal once the solution has been found. However, the problem of finding the solution may be intractable, as in NP-complete problems. Many resource allocation problems are NP-complete and so are intractable in this sense. As a result, optimization is no longer feasible even when an optimum exists, and some other criterion for adequacy of a solution is necessary. The best known example of such criteria are the "good enough, soon enough" criteria, in which a suboptimal solution is used if it meets specified minimal requirements and if it can be computed before a deadline is reached.

2. Distributed systems with limited communication. Multiple autonomous systems that communicate over a network are not fully modeled by dynamic systems for which there is a single event stream, but one can obtain a reasonable model when the network is reliable and has high bandwidth. However, when the network is very widely distributed or when communication channels are unreliable and have limited bandwidth, then it is no longer adequate to model the system using a single global state and a single event stream. It is no longer possible for any one node of such a network to have global knowledge of the state of the entire system. Furthermore, communication becomes a resource that must be allocated along with other resources.

3. Self-awareness. Generally speaking, the dynamic systems literature does not consider systems that know their own structure and can restructure themselves in response to changing requirements.

We now outline a proposal for how to address the inadequacies above. Our proposal has three main components which we introduce here, and then we elaborate on each of them in the rest of this section:

1. Models of Continuous Dynamic Systems. Modeling languages such as the Unified Modeling Language (UML) do not currently include any diagrams that can express continuous-time, time-driven or stochastic systems. We propose to use extended data-flow diagrams to model these kinds of system. Data-flow is also useful for modeling distributed systems.

2. Mixed Modeling Strategy. In Section 4 a strong case was made that simply reducing to either a purely discrete or a purely continuous system is not an effective modeling technique for hybrid systems. The techniques that have proven to be effective for modeling hybrid systems mix discrete and continuous by using a continuous model of discrete components and by using a discrete model of continuous components. Accordingly we propose using a mixed modeling strategy in which the components of a discrete model (such as an automaton) may be continuous and vice versa.

3. Reflection. The modeling language must allow its own structure to be included within the model. This can be accomplished by using reflective modeling languages and tools.

5.1 Models of Continuous Dynamic Systems

The first important step in modeling fully-driven systems is to support the modeling of continuous and time-driven systems. Many modeling languages already exist for modeling such systems, some of which are even executable. However, these modeling languages are not currently integrated with UML whose behavioral diagrams are limited to automata. This is essential for supporting the mixing of both time-driven and event-driven components in the same diagram.

As a first step toward better integration, we propose adapting and extending data-flow diagrams as a means of modeling time-driven and continuous dynamic systems. While data-flow diagrams are not equivalent to dynamic systems diagrams, we are investigating the extensions that will be needed to achieve the appropriate semantics.

5.2 Mixed Modeling Strategy

Top-down design, also known as the "divide-and-conquer" design strategy is a commonly used technique for dealing with design complexity. However, in most modeling languages, the same design language is employed at every level of detail. As a result, top-down elaboration is only a means of communicating and understanding (which are certainly important) and not an element of the design language. In principle, the whole model could be expressed on a single, very complex, level.

We propose to use top-down elaboration as a means of mixing distinct modeling languages and formalisms. In such a model, the elaboration is not just a convenience, and it is not possible, even in principle, to express the model on a single level. The modeling languages we are especially interested in are the discrete modeling languages (such as statecharts and activity diagrams) and the continuous modeling languages (such as the model of continuous dynamic systems discussed in the previous subsection above).

Mixing modeling languages is not simply a matter of allowing both to be used in the same modeling tool. It is important to reconcile the differing notions of time that drive the system. This reconciliation is accomplished by specifying conditions for event transitions as follows:

- A continuous component in a discrete system is synchronized at event times within the discrete system which can be triggered in several ways:
  1. An input event of the discrete system causes a change of state. This is the normal way in which a discrete event system changes state.
  2. The current state of the continuous component enters a prescribed subset of the state space.
- A discrete component in a continuous system changes its state as a result of the following kinds of event:
  1. At regular clock ticks determined by the continuous system.
  2. After an elapsed time determined by the amount of computation performed by the discrete systems.

5.3 Reflection

One of the themes that underlies the inadequacies in the list at the beginning of this section is "computational reflection." By this we mean that the dynamic system includes its own computation within its model of itself. This is most clearly stated in the case of self-awareness, but it is fundamental in the other issues as well. By emphasizing optimal solutions and ignoring the cost of computing an optimal solution, the literature on hybrid systems fails to recognize that optimality may be undesirable. This is the result of separating the mathematical model of a system from the system being modeled. When the mathematical model is embodied in a computer program, then the program becomes part of the system being modeled, and it must be modeled (and controlled) along with the physical system.

The hybridization paradigm is a powerful modeling approach, but it is not reflective. The modeling language must allow its own structure to be included within the model. To accomplish this objective the modeling language must include:

1. Formal specifications of the system components and how they interact with one another.
2. A meta-level for component specifications that allows the system to examine and reason about the specifications at run-time.

6 Conclusion

In this paper we have introduced the notions and terminology of general systems and used this terminology as a means of classifying general systems. Effective techniques are

available for modeling and analyzing many of the classes of general system that we have described. However, real-time distributed resource management and other fully-driven hybrid dynamic systems cannot be adequately modeled using existing methodologies and tools. Yet the existing methodologies are very powerful, so it makes sense to try to build on these methodologies whenever possible. Accordingly, we have outlined a proposal for how to build on existing methodologies to model and analyze fully-driven hybrid systems.

Acknowledgments

This research was partially supported by the Defense Advanced Research Projects Agency under contract F330602-99-C-0167.

References

[1] P. J. Antsaklis, J. A. Stiver, and M. D. Lemmon. Hybrid system modeling and autonomous control systems. In L. Grossman, R., A. Nerode, P. Ravn, A., and H. Rischel, editors, *Lecture Notes in Computer Science: Hybrid Systems*, volume 736, pages 366–392. Springer Verlag, 1993.

[2] M. S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control*. PhD thesis, MIT, Cambridge, MA, 1995.

[3] M. S. Branicky, B. S. Borkar, and S. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, 1998.

[4] R. W. Brockett. Hybrid models for motion control systems. In H. L. Trentelman and J. C. Wilems, editors, *Essays in Control: Perspectives in the Theory and its Applications*. Birkhauser, Boston, 1993.

[5] R. W. Brockett. Dynamical systems and their associated automata. In U. Helmke, R. Mennicken, and J. Saurer, editors, *Systems and Networks: Mathematical Theory and Applications*. Akademie Verlag, 1994.

[6] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[7] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence Journal*, 57, 1992.

[8] D. N. Godbole, J. Lygeros, and S. Sastry. Hierarchical hybrid control: a case study. In P. A Antsaklis, Kohn W., Nerode A., and S. Sastry, editors, *Lecture Notes in Computer Science: Hybrid Systems II*, volume 999, pages 166–190. Springer Verlag, 1995.

[9] ISO/IEC JTC1/SC21. Open Distributed Processing - Reference Model: Part 2: Foundations (ITU-T Recommendation X.902 — ISO/IEC 10746-2).

[10] G. Klir. *Architecture of Systems Problem Solving*. Plenum Press, 1985.

[11] M. M. Kokar. On consistent symbolic representations of general dynamic systems. *IEEE Transactions on Systems, Man and Cybernetics*, 25, no. 8:1231–1242, 1995.

[12] M. D. Mesarovic and Y. Takahara. *Abstract Systems Theory*. Springer Verlag, 1989.

[13] A. Nerode and W. Kohn. Models for hybrid systems: Automata, topologies, stability. In L. Grossman, R., A. Nerode, P. Ravn, A., and H. Rischel, editors, *Lecture Notes in Computer Science: Hybrid Systems*, volume 736, pages 317–356. Springer Verlag, 1993.

[14] L. Padulo and M. A. Arbib. *Systems Theory: A Unified State Space Approach to Continuous and Discrete Systems*. W. B. Saunders, Philadelphia, PA, 1974.

[15] G. J. Pappas and S. Sastry. Towards continuous abstractions of dynamical and control systems. Technical Report UCB/ERL M96/53, University of California at Berkeley, 1996.

[16] F. Pichler. Dynamic systems: A survey. In Singh M., editor, *Systems and Control Encyclopedia: Theory, Technology and Applications*, pages 1282–1286. Pergammon Press, 1987.

[17] D. S. Weld and J. de Kleer. *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, CA, 1990.