Modeling of Combined Discrete/Continuous Processes

P. I. Barton and C. C. Pantelides
Centre for Process Systems Engineering, Imperial College of Science, Technology and Medicine, London SW7 2BY, England

The dynamic behavior of processing systems exhibits both continuous and significant discrete aspects. Process simulation is therefore a combined discrete/continuous simulation problem. In addition, there is a critical need for a declarative process modeling environment to encompass the entire range of processing system operation, from purely continuous to batch. These issues are addressed by this article.

A new formal mathematical description of the combined discrete/continuous simulation problem is introduced to enhance the understanding of the fundamental discrete changes required to model processing systems. The modeling task is decomposed into two distinct activities: modeling fundamental physical behavior, and modeling the external actions imposed on this physical system. Both require significant discrete components. Important contributions include a powerful representation for discontinuities in physical behavior, and the first detailed consideration of how complex sequences of control actions may be modeled in a general manner.

Introduction

The utility of general-purpose software packages that model the dynamic behavior of processing systems has now been recognized for many years. Potential or actual applications of such tools have been discussed by many authors (Perkins, 1985; Perkins and Barton, 1987; Mani et al., 1990; Naess et al., 1992). Changes in the process industries, such as more process integration and concerns over safety and the environment, have also provided increased incentives for this form of analysis. However, it is only relatively recently that advances in numerical methods and computer hardware have provided the means to perform dynamic process simulations on a realistic scale.

Despite the potential benefits of the application of dynamic simulation technology, academic authors remain dissatisfied with the extent to which this technology has been adopted by the process industries (Perkins, 1985; Marquardt, 1991). The reluctance to adopt this technology can in part be attributed to the fact that dynamic simulation is still considered a costly activity that can only be justified in special cases. In turn, the inadequacies of currently available process simulation packages help to perpetuate this viewpoint. This article attempts to address some of these deficiencies. First, it is worthwhile to examine briefly the techniques currently employed. The reader is directed to Marquardt's (1991) review for a comprehensive list of the packages available.

Software packages specifically designed for the activity of dynamic simulation improve productivity by allowing the engineer to concentrate on the correct formulation of the process model, as opposed to the numerical algorithms and coding required to achieve a solution. Several *continuous process simulation packages* have been reported in the recent literature. They are mainly suitable for the continuous simulation of processing systems, although discrete changes to the inputs and the model are tolerated to a limited extent, and all enable a process model to be built in a block oriented manner, through the interconnection of instances of library unit operation models in a process flowsheet. Examples include GEPURS (Shinohara, 1987), DIVA (Holl et al., 1988), and DYNSIM (Gani et al., 1992). In addition to these facilities, some packages allow the user to create new unit operation models expressed in terms of high-level equation-based symbolic languages, for instance

Correspondence concerning this article should be addressed to C. C. Pantelides.
Current address of P. I. Barton: Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139.

SpeedUp (Perkins and Sargent, 1982) and DPS (Wood et al., 1984).

The ability to create new unit operation models is particularly important because the construction of a complete library of unit operation models is probably impossible due to the complexity and diversity of dynamic process models. In order to realize this level of flexibility, it is necessary to decouple completely the description of a physical system from the numerical methods employed to achieve a solution of the simulation problem. It is now well recognized that this decoupling offers the further advantage that the same model may be reused for a wide range of activities, such as steady-state simulation and design, steady-state and dynamic optimization, data reconciliation, and so on, in addition to dynamic simulation. This leads to the conclusion that the notion of a dynamic simulation package should be broadened to that of a general-purpose *process modeling environment* that will encompass a broad range of activities employing a common process model.

Few processes can be considered to operate in an entirely continuous manner. Even the majority of "continuous" processes experience significant discrete changes superimposed on their predominantly continuous behavior. Such changes typically arise from the application of digital regulatory control, plant equipment failure, or as a consequence of planned operational changes, such as startup and shutdown, feedstock and/or product campaign changes, process maintenance, and so on. Moreover, the situations in which these discrete components significantly affect the overall process behavior are often precisely those that could benefit most from detailed dynamic simulation studies (for example, in order to verify safe operation or assess environmental impact).

Any process that is operated in an essentially dynamic manner, such as a batch process or a periodic separation process, will always experience frequent control actions of a discrete nature in order to maintain operation in a time dependent, often cyclic, mode. The fact that the behavior of batch and semicontinuous processes has both significant discrete and continuous components has been recognized for many years (Fruit et al., 1974), and this has been reflected in the design of special purpose dynamic simulation packages for this class of processing system, such as BOSS/BATCHES (Joglekar and Reklaitis, 1984) and UNIBATCH (Czulek, 1988). In addition, Sørensen et al. (1991) have recently extended DYNSIM to encompass the dynamic simulation of batch processes.

From the above discussion, we argue that, in most situations, an engineer involved in the analysis of the dynamic behavior of a processing system wishes to pose a *combined discrete/continuous* simulation problem as opposed to a purely continuous simulation problem, regardless of the nominal mode of operation of the process under investigation. It is therefore necessary to consider the development of a general-purpose combined discrete/continuous process modeling environment that will also encompass the declarative modeling capabilities of existing continuous process simulation packages. Such an environment will support the study of arbitrarily operated processing systems within a unified framework.

A number of combined discrete/continuous simulation languages have been described in the system simulation literature. Fahrland (1970) was the first author to advocate in detail the development of "combined discrete event and continuous" simulation languages. The majority of subsequent developments have concentrated on augmenting existing discrete event simulation languages to include rather primitive continuous simulation capabilities (Cellier, 1979b). Examples include GASP IV (Pritsker and Hurst, 1973), DISCO (Helsgaun, 1980), and SLAM II (Pritsker, 1986). In more recent years, a small number of truly combined simulation languages, such as SYSMOD (Smart and Baker, 1984) and COSMOS (Kettenis, 1992), have emerged that offer continuous simulation capabilities comparable to those of the languages originating from the CSSL (continuous system simulation language) standard (Strauss, 1967). However, the effective use of even these latter languages for the detailed simulation of complex *processing* systems is problematic, often for the same reasons that have precluded the widespread application of CSSL-type languages in the process industries (Perkins, 1986). Moreover, as will be demonstrated later in this article, the "world view" adopted by all these languages is not well suited to the demands of process simulation.

The objective of this article is to present a sound formal basis for the description of combined discrete/continuous process simulation problems, and to articulate these notions in the form of a modeling language. We begin with a formal mathematical description of the combined simulation problem. We then proceed to consider the modeling of this class of systems, and demonstrate how this can be decomposed into two distinct activities: modeling the fundamental physical behavior of a processing system, and modeling the external actions imposed on this physical system. These discussions draw on the mathematical formalism, but also consider the important practical issues of model complexity and reusability. Barton and Pantelides (1994) consider how models of physical behavior and external actions may be brought together to form the description of complete dynamic simulation experiments, and then proceed to illustrate the usefulness and necessity of combined discrete/continuous process simulation with a detailed example developed with a prototype process modeling environment that implements the above notions.

Mathematical Formulation of the Combined Process Simulation Problem

The physicochemical mechanisms that govern the time dependent behavior of processing systems are predominantly continuous. Modeling of these mechanisms from first principles typically yields large, sparse sets of nonlinear equations representing conservation laws, physical constraints, equilibrium, and thermodynamic relationships, and so on. More specifically, a process that can be described entirely in terms of lumped parameters will give rise to a model composed of a mixed set of ordinary differential and algebraic equations (Pantelides et al., 1988). On the other hand, a process that also contains unit operations with variables distributed in one or more spatial dimensions will typically yield a mixed set of partial differential, ordinary differential, and algebraic equations (Heydweiller et al., 1977). Terms that must be integrated over one or more spatial dimensions, or particulate system modeling by means of population balances, may also add integrals to the above set of equations (Marquardt, 1991). This latter class of problems is considered to be beyond the scope of this article, primarily because general-purpose methods for the direct solution of equations with one or more spatial independent variables in addition to time are still in their infancy (see, for example, Pipilis (1990)).

A natural formalism for this continuous behavior is expressed mathematically by a set of nonlinear differential-algebraic equations (DAEs) of the form:

$$\begin{aligned} f(x, \dot{x}, y, u, t) &= 0 \\ u &= u(t) \end{aligned} \tag{1}$$

where: $x \in X \subseteq \mathbb{R}^n$, $y \in Y \subseteq \mathbb{R}^m$, $u \in U \subseteq \mathbb{R}^l$, $t \in T = [t^{(0)}, t^{(f)}]$ and f: $X \times \mathbb{R}^n \times Y \times U \times T \rightarrow \mathbb{R}^{n+m}$. The unknowns x and y are usually referred to as the differential variables and algebraic variables respectively, u are the known system inputs, and t is the independent variable time. It should be noted that well-posed purely differential ($m=0$) and purely algebraic ($n=0$) equation sets are encompassed by this formalism. Equations of the above form can be classified according to their *index*, which may be defined as the smallest non-negative integer I such that Eq. 1 and its first I derivatives with respect to time define $\dot{x}$ and $\dot{y}$ as locally unique functions of x, y, and u (and its time derivatives), and t (Brenan et al., 1989).

Combined discrete/continuous process simulation requires the solution of a sequence of initial value problems, described by equations of the above form, interspersed by instantaneous *events* that may cause some form of discrete change to the initial value problem currently being solved. Most importantly, these changes may involve manipulation of the functional form of the continuous mathematical model. As a consequence, the time domain of interest $[t^{(0)}, t^{(f)}]$ is partitioned into NCD continuous subdomains $[t^{(k-1)}, t^{(k)}]$ $\forall k = 1 \ldots NCD$. This is illustrated by Figure 1. The initial time $t^{(0)}$ is assumed given, whereas the final time $t^{(f)} \equiv t^{(NCD)}$ may be either given or specified implicitly through one or more termination conditions. The subdomain boundaries $t^{(k)}$, $k = 1 \ldots NCD-1$, may also be specified explicitly or determined implicitly during the course of a simulation.

The combined simulation problem can therefore be defined as the solution of the following sequence of DAE sets:

$$\left.\begin{aligned} f^{(k)}(x^{(k)}, \dot{x}^{(k)}, y^{(k)}, u^{(k)}, t) &= 0 \\ u^{(k)} &= u^{(k)}(t) \end{aligned}\right\} \quad t \in [t^{(k-1)}, t^{(k)}) \qquad \forall k = 1 \ldots NCD \tag{2}$$

With the above formulation, it is possible for both the set of variables and the set of equations that relate them to vary from subdomain to subdomain in a completely general manner. The describing equations and initial condition of the first subdomain are determined by an individual simulation description. The describing equations and initial condition of the succeeding subdomains will be determined from a combination of the final state of the system in the preceding subdomain and the consequences of the corresponding event(s). It is also important to note that the index of the describing equations can vary from one subdomain to the next.

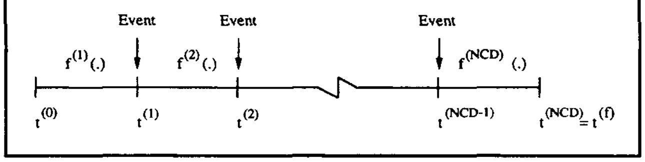

Figure 1. Time domain partitioning in combined discrete/continuous simulation.

There are three issues associated with the solution of the above mathematical problem. First, the condition of the system at the beginning of each subdomain must be determined, secondly the system behavior over the subdomain must be calculated, and finally the precise end point of the subdomain must be located. These issues are considered in more detail in the following subsections. In addition to the solution of the simulation problem, it is important for a combined discrete/continuous process modeling environment to detect and diagnose automatically those problems that are either inherently badly posed, or cannot readily be solved using currently available techniques. Pantelides and Barton (1993) review some of the criteria and algorithms that may be employed for this purpose.

Consistent initialization

Before simulation can commence, a set of consistent initial values for the describing variables at the beginning of the first subdomain must be determined. A necessary condition for a set of initial values to be consistent is that they satisfy Eq. 2 at $t^{(0)}$. This condition represents a set of $n+m$ equations in the set of $2n+m$ unknowns $\{x^{(1)}(t^{(0)}), \dot{x}^{(1)}(t^{(0)}), y^{(1)}(t^{(0)})\}$. A number of additional relationships, known as the initial condition specification, are therefore usually necessary.

For a set of explicit ordinary differential equations (ODEs) (of the form $\dot{x} = f(x, t)$) the initial condition specification normally involves a set of values assigned to the variables x at $t^{(0)}$. For a system of DAEs, a more general approach is both possible and desirable. After all, steady-state (for example, $\dot{x}(t^{(0)}) = 0$) is probably the most frequently encountered initial condition specification. The continuous process simulation package SpeedUp (Pantelides, 1988b) provides a flexible facility for the specification of the initial condition: the requisite number of initial *values* may be assigned to any subset of variables in the set $\{x^{(1)}(t^{(0)}), \dot{x}^{(1)}(t^{(0)}), y^{(1)}(t^{(0)})\}$, subject to the nonsingularity of the sufficient consistency conditions in the remaining variables.

In more general terms, the initial condition specification can be expressed as a set of nonlinear equations of the form:

$$C^{(1)}(x^{(1)}(t^{(0)}), \dot{x}^{(1)}(t^{(0)}), y^{(1)}(t^{(0)}), u^{(1)}(t^{(0)}), t^{(0)}) = 0 \tag{3}$$

The definition of a sufficient condition for the consistency and completeness of a set of initial values is particularly complex, and is closely related to the categorization of DAEs according to their index. For all DAE sets of index greater than one, and even for some index one sets, consistency demands that the initial conditions not only satisfy the original Eqs. 2, but also the first, and possibly higher-order, time differentials of some of these equations (Pantelides, 1988a). The existence of these "hidden" constraints implies that, unlike ODEs and most index one DAEs, the number of initial condition specifications 3 must be smaller than the number of differential variables appearing in the DAE set.

The problem of determining consistent initial values for the describing variables at the beginning of subdomains $k=2...NCD$ (*reinitialization*) is slightly different to that of the first subdomain. If the composition of the set of differential variables appearing in the describing equations remains unchanged across a subdomain boundary, it is normal practice to assume that the values of these variables are continuous across the boundary:

$$x^{(k)}(t^{(k-1)}) = x^{(k-1)}(t^{(k-1)}) \tag{4}$$

the physical reasoning being that the differential variables represent conserved quantities (such as mass, energy, or momentum) or are directly related to them. For most cases, this condition is sufficient to define uniquely the state of the system at the beginning of the next subdomain.

The assumption of continuity is, however, problematic in at least two situations. If Eq. 2 implies "hidden" consistency relations of the type mentioned above, some of the continuity relations 4 may be redundant or inconsistent. A simple illustration is provided by a vessel which is partially full of an incompressible liquid mixture in the $(k-1)$th subdomain that becomes full at the boundary with the kth domain. At this point, the index of the DAE system increases from one to two, as the previously independent molar component holdups (differential variables) become constrained by the fixed volume of the vessel and the incompressibility of the liquid. In this case, one of the component holdup continuity relations is redundant.

In addition, the continuity assumption precludes situations in which a discrete change to the values of one or more differential variables at $t^{(k-1)}$ occurs as a consequence of an impulsive input to the system corresponding to a Dirac Delta function. This may be used to model a phenomenon that takes place on a much smaller time scale than that of primary interest (Mattsson, 1989). For example, if a quantity of a reactant is added instantaneously to a reactor vessel, then neither the holdup of the reactant, nor the internal energy of the reactor contents will be continuous across the subdomain boundary. The condition of the reactor at the beginning of the new subdomain is determined by an instantaneous material balance on the reactant, an instantaneous energy balance, and the continuity of the holdups of all the other components. Another example is provided by an analog control loop with integral action. When the loop is switched to automatic control, the integral of the controller error is normally initialized to zero in order to eliminate any reset windup that may have occurred while the loop was under manual control. This situation is most easily modeled by replacing the continuity assumption on the variable representing this integral (a differential variable) with an equation constraining its value to zero at the beginning of the next subdomain.

Another problem is encountered when the composition of the set of differential variables changes at a subdomain boundary. Variables that are dropped from this set do not cause problems, because the number of relationships required to determine a set of consistent initial values at the beginning of the next subdomain will also decrease by an equal number. However, if variables are added to this set, additional relationships *may* be required to define uniquely the state of the system at the beginning of the next subdomain. In the most general terms, these additional relationships will take the form shown in Eq. 5 below. If the new differential variables belonged to the set of algebraic variables in the preceding subdomain, continuity could be the default assumption.

In summary, a set of consistent initial values for the describing variables at time $t^{(k-1)}$ can in principle be determined from the simultaneous solution of:

(a) The describing Eqs. 2.

(b) The hidden consistency relations (if any) implied by these equations.

(c) The requisite number of additional equations of the general form:

$$C^{(k)}(x^{(k-1)}(t^{(k-1)}), \dot{x}^{(k-1)}(t^{(k-1)}), y^{(k-1)}(t^{(k-1)}), u^{(k-1)}(t^{(k-1)}), x^{(k)}(t^{(k-1)}), \dot{x}^{(k)}(t^{(k-1)}), y^{(k)}(t^{(k-1)}), u^{(k)}(t^{(k-1)}), t^{(k-1)}) = 0 \tag{5}$$

In many cases Eq. 5 takes the simpler form shown in Eq. 4, while at the beginning of the first subdomain ($k=1$) it is replaced by Eq. 3. However, in general Eq. 5 may involve the known values of the describing variables immediately before the subdomain boundary.

Numerical solution of the DAEs

A combined discrete/continuous simulation is advanced between subdomain boundaries by the numerical solution of the describing equations that characterize the particular subdomain. Strategies for this task fall broadly into two categories (Marquardt, 1991): direct integration, whereby the equations are solved simultaneously by the same algorithm, or modular integration, whereby the equations are suitably partitioned and solved by different algorithms applied to each subsystem. Most dynamic simulation packages for continuous processes employ the former strategy (for example, SpeedUp, DIVA), although recent progress on the use of modular integration strategies has been reported by Laganier et al. (1993).

Pantelides and Barton (1993) review several techniques currently employed for the direct integration of DAEs, including a discussion of the difficulties associated with the numerical solution of DAEs of index exceeding unity.

Event location

The end of each subdomain in the sequence shown in Eq. 2 is marked by the occurrence of an event. Each event takes place instantaneously, and will cause some form of discrete change to the functional form of the mathematical model, and/or the scheduling of new event(s) to occur at some future time. Events may be distinguished by the manner in which the time of occurrence is determined:

- *Time Events*—the exact time of occurrence of time events is known in advance, so solution of the subdomains can proceed to these events in time order. They can be either *exogenous*, if the time of occurrence is known a priori, or *endogenous*, if the time of occurrence is computed as a consequence of some earlier event during the simulation (Cellier, 1979a).
- *State Events*—the time of occurrence of state events is never known in advance because it is dependent on the system fulfilling certain conditions (known as *state conditions*). The numerical solution of the equations in a subdomain must in-

stead be advanced speculatively until the state condition becomes satisfied.

A modeling environment for combined discrete/continuous processing systems should provide facilities for state conditions to be expressed in the most general terms possible. A very broad range of these conditions can be expressed in terms of classical propositional logic. The boundary between the kth and the $(k+1)$th subdomain $(k<NCD)$, and also the end of the last subdomain, can therefore be determined by a scalar logical expression of the general form:

$$l^{(k)}(x^{(k)}, \dot{x}^{(k)}, y^{(k)}, u^{(k)}, t) \tag{6}$$

becoming true.

The nature of state events dictates that all pending state conditions should ideally be monitored continuously throughout the solution of the equations in a subdomain. However, numerical integration algorithms advance time in a stepwise fashion, so state events can only practically be detected at the earliest discrete point in time that the system status indicates a state condition has become satisfied. A significant body of literature exists on the problem of detecting and then locating state events, and this is reviewed briefly by Pantelides and Barton (1993).

Model Decomposition

The mathematical formulation of the combined simulation problem significantly enhances our understanding of the fundamental discrete changes required to model processing systems and the techniques necessary to achieve a numerical solution. We now consider the task of developing a practical engineering tool—a high-level language that can provide extensive support for the modeling activity, while still encompassing the expressive power of the above formulation.

In his original article, Fahrland (1970) identifies the fundamental characteristics of combined discrete/continuous systems, and proposes a modeling methodology based on decomposition into a series of continuous subsystems and discrete subsystems which are then allowed to interact as equals during the course of a simulation. This original decomposition has been reflected in the design of all subsequent combined simulation languages, and is particularly suitable for those systems in which multiple entities with identical continuous behavior periodically enter and leave the overall system.

We argue, however, that processing systems are more naturally viewed as a single physical subsystem (the plant equipment) on which external actions are imposed in order to achieve certain objectives. Examples of such actions include disturbances (for example, equipment failure), or control actions (for example, the action of a digital controller at the end of each sampling interval).

A common feature of all processing systems is the occurrence of discontinuities in the fundamental physical behavior. These *physicochemical discontinuities* typically arise from thermodynamic (for example, phase) and fluid mechanical (for example, from laminar to turbulent regime) transitions, or from irregularities in the geometry of process vessels (for example, nonuniform cross-sections), and form an integral part of any declaration of physical behavior. It is also important to recognize that the external actions experienced by a processing system are again combined discrete/continuous in nature. Some, such as the opening and closing of manual valves, can be viewed as purely discrete, whereas others, such as an input that is ramped between two steady values, have both discrete (initiation and termination) and continuous (ramping) characteristics.

Decomposition of a process model into a continuous subsystem that partially represents the physical behavior, and a discrete subsystem representing the external actions and physicochemical discontinuities therefore seems rather arbitrary. Instead, we propose that the description of a process model should be decomposed into the underlying combined discrete/continuous physical behavior of the plant, and the external actions imposed on it by its environment. In this manner, all the knowledge concerning the physical behavior of a system can be encapsulated by a single entity. The decomposition also significantly increases the scope for the reuse of this information within a process modeling environment: it becomes completely decoupled from the external actions that are applied during a particular simulation experiment.

In reflection of reality, physical behavior should be considered subordinate to and manipulated by external actions. The external actions can then drive the plant through a simulation experiment by direct manipulation of its characteristics. These conclusions lead to the fundamental structuring concepts of the simulation language:

- *Model entities* encapsulate a description of the physicochemical mechanisms governing the behavior of unit operations, including any discontinuities in these mechanisms.
- *Task entities* encapsulate a description of the control actions or disturbances imposed on this physical system by its environment.

Figure 2 contrasts the above decomposition with that proposed by Fahrland (1970). The remainder of this article will consider the modeling of these two aspects of process behavior in more detail.

Modeling Physical Behavior

A model entity encapsulates a reusable declaration of the physical behavior of a processing system. This declaration includes the complete set of variables required to *describe* the time dependent behavior of the system, and a set of DAEs that *determine* this behavior. The latter set is typically underdetermined with respect to the variables, additional equations and/or specifications being required to form a well-posed simulation problem.

A high-level equation-based symbolic language, similar to that employed by SpeedUp (Perkins and Sargent, 1982) and ASCEND (Piela et al., 1991), has been designed for the dec-

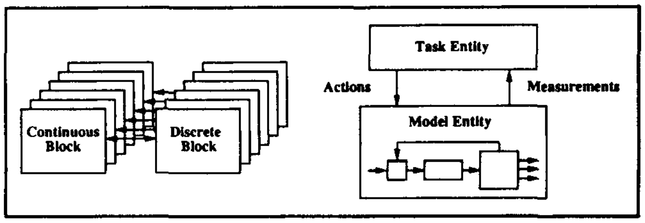

Figure 2. Alternative model decompositions.

```
MODEL Vessel_With_Safety_Valve

PARAMETER
  Vessel_Volume, R              AS REAL
  Set_Press, Reseat_Press       AS REAL
  Valve_Const                   AS REAL

VARIABLE
  Flow_In, Flow_Out,
  Relief_Flow                   AS Molar_Flowrate
  Holdup                        AS Moles
  Temp                          AS Temperature
  Press, Press_In               AS Pressure

STREAM
  Inlet  : Flow_In, Press_In    AS MainStream
  Outlet : Flow_Out, Press      AS MainStream
  Relief : Relief_Flow, Press   AS MainStream

SELECTOR
  Valve_Flag                    AS (Closed,Open)

EQUATION

  # Mass balance
  $Holdup = Flow_In - Flow_Out - Relief_Flow ;

  # Equation of state (perfect gas)
  Press*Vessel_Volume = R*Holdup*Temp ;

  # Safety relief valve with hysteresis
  CASE Valve_Flag OF
    WHEN Closed : Relief_Flow = 0 ;
                  SWITCH TO Open IF Press >= Set_Press ;
    WHEN Open   : Relief_Flow = Valve_Const*Press/Temp^0.5 ;
                  SWITCH TO Closed IF Press <= Reseat_Press ;
  END # case

END # Vessel_With_Safety_Valve
```

Figure 3. Model of pressure vessel fitted with safety relief valve.

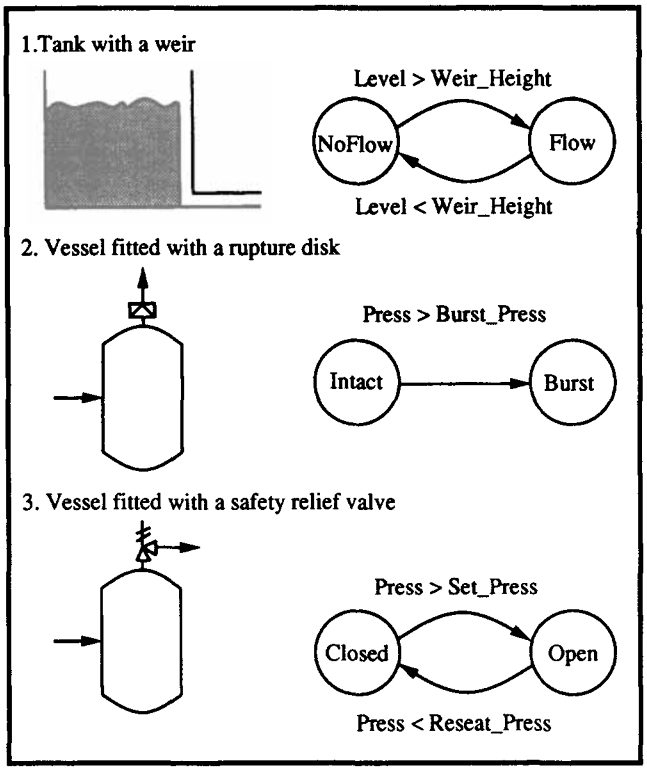

Figure 4. Examples of models containing physicochemical discontinuities.

laration of this information, although a number of significant enhancements have been made, which include extensive support for the description of general physicochemical discontinuities, and complex regular structures in the form of multidimensional arrays. The reader is directed to Barton (1992) for a complete description. Figure 3 illustrates the declaration of a model for a pressure vessel fitted with a safety relief valve.

Physicochemical discontinuities

An examination of the nature of physicochemical discontinuities leads to the conclusion that they can be modeled by dynamic manipulations of the functional form of Eq. 1. For example, the transition from laminar to turbulent flow in a pipe involves a discrete change in the relationship between the friction factor and the Reynolds number. At a phase change, not only does the functional form of the equations change, but it is also possible that the number of variables, and hence equations, required to describe the system changes.

At any given point in time, a set of variables and a set of equations that relate these variables will characterize the current *state* of a model. Many models have a unique state: the composition of the sets of variables and equations remains unchanged (although, of course, the *values* of the variables will change with time). On the other hand, models that involve physicochemical discontinuities must be declared in terms of several states, each characterized by a different set, and possibly number, of describing equations. For example, the model of a flash drum will be declared in terms of at least three states corresponding to the presence of both vapor and liquid phases, subcooled liquid only, and superheated vapor only. During a simulation, the active state of a model will determine the equations that describe the system at that point in time. Events may result in changes to the active state of a model, and the attendant changes to the describing equations.

In addition to providing facilities for the declaration of the equations that characterize each model state, it is necessary to provide a sufficiently general representation of the mechanisms that result in transitions between model states. These mechanisms are best illustrated by the three simple examples shown in Figure 4, where models containing physicochemical discontinuities are represented by digraphs, with nodes denoting model states and arcs signifying instantaneous transitions between these states.

In the first example, a vessel containing an overflow weir is declared in terms of two states, corresponding to whether or not liquid flows over the weir. This is termed a *reversible discontinuity* because the condition for one state transition is the negation of the condition for the other. Hence, the two transitions can be characterized by a single logical expression—a fact that is reflected by language structures of both SpeedUp and ASCEND. However, this is also the limitation of these language structures: they are only suitable for the declaration of reversible discontinuities.

The second example is that of a vessel fitted with a bursting (or rupture) disc. The disc can either be intact or burst. However, in this case there is only one possible state transition. When the pressure in the vessel rises above the set pressure, the disc shatters. This is termed an *irreversible discontinuity* because the model can never return to the intact state once the disc has shattered: the plant must be shut down and the disc replaced.

Finally, a vessel fitted with a safety relief valve has states corresponding to whether or not the relief valve is open. A transition from the closed to the open state occurs when the pressure in the vessel rises above the set pressure, and a transition from the open to the closed state occurs when the pressure falls below a (lower) reseat pressure. This is termed an *asymmetric and reversible discontinuity* because, although there are possible transitions in both directions, the conditions that must be satisfied are not directly related.

A formalism that does support this wide range of mechanisms was proposed by Pearce (1978) and is equivalent to a deterministic finite automaton (Hopcroft and Ullman, 1979), albeit one whose states are immensely complex according to the nature of the describing equations. Generalizing this for a set of DAEs, the model of a system (or subsystem) is declared in terms of a finite number of distinct states, each of which is characterized by:

(a) A set of variables x, $\dot{x}$, y, and u.

(b) A set of equations $f(x, \dot{x}, y, u, t) = 0$.

(c) A (possibly empty) set of *transitions* to other states.

A transition is characterized by:

(a) An initial state S^I.

(b) A terminal state S^T.

(c) A scalar logical expression $l(x^I, \dot{x}^I, y^I, u^I, t)$.

(d) A set of relationships allowing the determination of consistent initial values for the variables in S^T from the final values of the variables in S^I.

If we assume that the set of variables is the same for all states, and that the differential variables are continuous across all transitions, consistent initial values for the terminal state S^T can be determined from the mapping:

$$x^T(t^*) = x^I(t^*) \qquad (7)$$

where t^* is the time at which the transition occurs.

From a practical point of view, many of the describing equations of a system will remain unchanged regardless of the state the system is in. A model can therefore be considered to consist of *variant* and *invariant* sets of equations. The variant set of equations are described in terms of one or more *finite automata*, each of which is characterized by one or more *states*. At any point in time, the describing equations of a model are determined from the union of the invariant equations and the equations that characterize the active states of the finite automata. This definition is also recursive—each state of a finite automaton can be declared with variant and invariant parts.

The above formalism can be implemented by a CASE language structure, similar to that first used in the combined simulation language COSY (Cellier and Bongulielmi, 1980). Figure 3 illustrates how this language structure can model an asymmetric and reversible discontinuity describing the hysteresis in a safety relief valve.

Complexity and reusability—model hierarchies

The declarative language described so far provides an adequate tool for the modeling of primitive unit operations. Models of more complex unit operations or complete flowsheets are also characterized by sets of variables and equations, and could, in principle, be described in a similar manner. However, the efficient and error-free construction of such a description is an overwhelming task in most cases.

In recent years, a new generation of languages suitable for modeling continuous process systems, such as OMOLA (Nilsson, 1989), MODEL.LA (Stephanopoulos et al., 1990), and ASCEND (Piela et al., 1991), has emerged. A key contribution of these languages is the use of *hierarchical submodel decomposition* and *inheritance* in order to manage the complexity of the model building activity and to facilitate the reuse of existing models. These notions are strongly supported in the simulation language described here, both in the development of models for physical behavior and models for external actions. The former will only be described to the extent necessary to support the discussion concerning complexity and reusability of models for external actions in the next section.

In order to manage complexity, an engineer will analyze the structure of a system and divide it into a set of connected components. The simpler task of modeling each component may then be considered in isolation. Hierarchical submodel decomposition is a formalization of this activity. It enables a complex model to be constructed from the interconnection of a set of submodels with continuous flows, or *streams*. These connections can be viewed as additional algebraic equations in the system model, so any operation that can be applied to an equation can also be applied to a stream connection. Any model may include the declaration of component models, or even the declaration of component model arrays. In turn, these submodels may also be declared in terms of a set of component models, so a hierarchy of arbitrary depth may evolve. Figure 5 demonstrates how this concept is employed for the declaration of a distillation column model.

Inheritance, a concept first popularized by the object-oriented programming languages, has been discussed frequently in the recent literature. Through inheritance, a new type may be declared as an extension or restriction of one or more previously declared types. An inheritance hierarchy evolves as new types are declared that inherit the characteristics of existing types.

```
MODEL Distillation_Column

PARAMETER
  No_Trays, Feed_Position   AS INTEGER

UNIT
  Condenser                 AS Total_Condenser
  Top, Bottom               AS Linked_Trays
  Feed                      AS Feed_Tray
  Reboiler                  AS Partial_Reboiler

VARIABLE
  Net_Energy_Requirement    AS Energy_Flow

STREAM
  Feed_Stream               IS Feed.Feed_Stream
  Top_Product               IS Condenser.Liquid_Product
  Bottom_Product            IS Reboiler.Liquid_Product

SET
  Top.No_Trays    := NoTrays - Feed_Position ;
  Bottom.No_Trays := Feed_Position - 1       ;

EQUATION
  # Define the net energy requirement for the column
  Reboiler.Heat_Load + Condenser.Heat_Load = Net_Energy_Requirement ;

  # Stream connections
  Condenser.Reflux      IS Top.Liquid_In     ;
  Condenser.Vapour_Feed IS Top.Vapour_Out    ;
  Top.Vapour_In         IS Feed.Vapour_Out   ;
  Top.Liquid_Out        IS Feed.Liquid_In    ;
  Feed.Vapour_In        IS Bottom.Vapour_Out ;
  Feed.Liquid_Out       IS Bottom.Liquid_In  ;
  Reboiler.Vapour_Out   IS Bottom.Vapour_In  ;
  Reboiler.Liquid_Feed  IS Bottom.Liquid_Out ;

END # Distillation_Column
```

Figure 5. Model of a distillation column.

In the present context, the facility to construct a more detailed model from a simpler one by "inheriting" and "refining" information is of primary relevance. The construction of model hierarchies in this manner has at least three advantages:

(a) It enables the same information to be reused in a number of similar models, thereby reducing both the possibility of error and the effort required to develop new models.

(b) It ensures that changes to a simple model at a later date can automatically propagate to all its descendants.

(c) It serves as a mechanism for grouping similar models into sets, each member of which is guaranteed to have a minimum set of attributes (for example, variables, equations), namely those of any common ancestor.

This third feature enables us to establish the notion of a model *type*, one application of which will be demonstrated in the following section.

Modeling External Actions

In comparison to the attention devoted to modeling the physical behavior of processing systems by workers interested in the realization of continuous process simulation packages, the modeling of the external actions experienced by such systems has been largely neglected to date. The rather modest capabilities of these packages only extend to the discrete manipulation of the functions of time assigned to system inputs as a consequence of exogenous time events alone. On the other hand, workers interested in developing simulation packages for batch processes have always had to consider complex sequences of control actions. Below we consider briefly one of these packages, BATCHES (Clark and Kuriyan, 1989).

Most representations of batch processing systems are centered on tracking batches of material as they move through a plant: a material oriented approach as opposed to the equipment (or unit operation) oriented approach adopted by continuous process simulation packages. A BATCHES simulation description is therefore based on the set of products produced by a facility. Each product has a network of *tasks* associated with it, and each task describes an operation carried out in its entirety in a single item of equipment. Tasks are further decomposed into the sequence of elementary processing steps (or *subtasks*) required to complete the operation. A subtask is characterized by the set of DAEs that determine the continuous time dependent behavior of an operation during the step in question. A library of subtasks that model elementary processing steps frequently encountered in batch processing systems is made available to the user, from which the more complex structures can be constructed.

However, a methodology based on subtasks, each characterized by a unique set of describing equations, has considerable drawbacks, especially for a modeling environment targeted at the entire range of process operations. The key disadvantage lies in the fact that a model of physical behavior must be posed separately for each elementary step involved in an operation. This can be tedious even for small models in which most of the describing equations are common to all steps, and becomes impracticable as the number of equations increases, in terms of both the sheer programming effort required, and in ensuring correctness of the models. For example, it would be both highly undesirable and unnecessary to restate the model of a continuous process each time a control action is conducted during a complex startup schedule, especially as each control action would usually affect only a small part of the overall system.

A subtask describes a period of continuous simulation terminated by some form of discrete change to the model. Usually these discrete changes will only affect a subset of the describing equations, and in many cases all that will distinguish one elementary processing step from another is the set of forcing functions. As has already been discussed earlier, it seems more natural to pose one model of physical behavior for an entire facility, and then to provide a means for the incremental manipulation of this model or its forcing functions at the end of each elementary processing step.

This conclusion is fundamental to the design of the simulation language described in this article. Accordingly, the item of process equipment under investigation (which could be a single batch unit or an entire continuous plant) is considered to be described by a single combined discrete/continuous model entity over the entire time horizon or interest. The simulation description is then completed by a *schedule of task entities* representing the external actions applied to the system during this period. Execution of the schedule will drive the predominantly continuous model entity through the desired operational changes. This approach is also introduced in an attempt to emulate the manner in which chemical processes are operated in reality: a schedule of tasks mirrors the action of an operator or control system on a process. To accommodate the diverse needs of many applications, it has global access to any feature of the underlying model entity and modifies it according to the desired objectives. For example, during the simulation of a startup procedure the schedule of tasks will manipulate the model entity just as operators manipulate the plant itself by opening valves or placing control loops under automatic control.

The rest of this section considers the practical task of developing language structures for the description of external actions according to this philosophy.

Elementary tasks

Before discussing these language structures, it is helpful to use two examples to illustrate how the external actions imposed on a processing system might be represented at a fundamental level. First, consider how the opening of a manual valve by an operator might be modeled. Physically, this control action moves the valve stem from the fully closed to the fully open position. A model of the physical behavior of the valve would include an equation that, according to the value of the variable representing the stem position and the inherent valve characteristic, relates the flow rate through the valve to the pressure drop across the valve. The action of the operator could therefore be modeled by a *persistent* change to the value of the stem position variable. If the value were determined by another equation, the latter would replace the original equation in the model.

Next, consider how the instantaneous addition of a small quantity of catalyst slurry to a batch reactor might be modeled. If we compare the physical state of the reactor immediately after the action with that immediately before, we observe that the component holdups of the catalyst and its solvent, and the total internal energy holdup, will all have increased slightly.

This phenomenon could be modeled as an *impulsive* increase in the values of the variables representing these quantities, provided it takes place on a small time scale in comparison to that of primary interest. However, the model equations remain unchanged.

Therefore, at the most fundamental level, external actions can be modeled in terms of discrete manipulations of some aspect of the underlying mathematical model of the physical system (see Eq. 1). *Elementary tasks* represent a formalism of these basic mathematical manipulations, and are the primitive language structures from which all tasks are ultimately constructed.

It has already been observed that mathematically there are only two fundamental aspects of the model: the set of variables, and the set of equations. For convenience, each equation in the model of the physical system is considered to belong to one of three categories:

- The model or *general equations* expressing general relationships between variables (cf. the perfect gas equation of state in Figure 3).
- The *connecting equations* established by the existence of continuous connections (for example, streams) between submodels (cf. the stream connections in Figure 5).
- The *input equations* (or forcing functions) defined by the assignment of an explicit function of time to a single system variable ($u=u(t)$), thus rendering the latter an *input variable* for the subsequent period of simulation.

A discrete change to the set of equations takes the form of the instantaneous disposal of a subset of these equations and its replacement with an equal number of new equations, similar to the effect of a physicochemical discontinuity. Changes to the set of input equations could equally be regarded as discrete changes to the values (or functional time variation) of input variables, but their effect is more similar to changes to the set of equations. Moreover, considering the input equations as interchangeable with other equations enables the number or identity of input variables to vary during the course of a simulation.

The above discussion has highlighted the fact that primitive actions can be distinguished as either *persistent* or *impulsive*. A persistent action is defined as a modification to the composition or status of certain elements of the variable or equation sets that has a finite *nonzero* duration. On the other hand, an impulsive action leaves the variable and equation sets unaltered, but changes the time trajectory being followed by the system.

Persistent primitive actions

The language definition currently includes two structures that enable the declaration of persistent changes to the composition of the set of equations. The REPLACE task defines the replacement of one or more equations with an equal number of new equations. The category to which these equations belong is irrelevant to this operation. It is therefore possible to replace an input equation involving one variable with a new input equation defining a specification on another variable (thus changing the status of both variables), or to replace a general equation with an input equation, or vice versa. Of course, in practical terms it must be possible to identify an equation uniquely in order to replace it. An input equation can be

```
# Close the control loop
REPLACE
  Control_Valve.Position
WITH
  AutomaticControl AS Control_Valve.Position = Control_Valve.Signal ;
END

# Set the controller bias
REPLACE
  Controller.Error
WITH
  Controller.Bias := OLD(Controller.Bias) ;
END
```

Figure 6. Applications of the REPLACE task.

identified by the associated input variable, whereas general and connectivity equations must be associated with an identifier when declared. Figure 6 applies a REPLACE task to model the switching of a control loop from manual to automatic analog control. Variables are identified using a pathname mechanism, for example, Control_Valve.Position denotes the variable representing the stem position in the physical model of a control valve. Note that the task discards a specification on the stem position of the control valve, and inserts an equality constraint (called AutomaticControl) into the model. This latter relationship will then hold until either it is also discarded, or the simulation terminates.

The second example in Figure 6 illustrates how the steady-state bias of an analog controller might be calculated automatically. In order to determine this bias, a steady-state initialization calculation is performed in which the controller error is constrained to zero by an input equation, and the bias is considered to be a calculated (algebraic) variable. The value for the bias determined by this calculation corresponds to the correct steady-state bias for the controller. Before solution of the first subdomain commences, execution of the REPLACE task can automatically replace the input equation involving the controller error with a new input equation constraining the bias to its current value. The controller error, now an algebraic variable, is then free to fluctuate as disturbances are introduced and the controller attempts corrective action. These two examples also demonstrate how changes to the set of equations can have the side effect of persistent changes to the status of members of the variable set (for example, from algebraic to input or vice versa).

Probably the simplest and most common manipulation a model may experience is the replacement of one or more input equations with an equal number of new input equations involving the *same* set of variables. In fact, this is the only form of manipulation tolerated by continuous process simulation packages such as SpeedUp. The ubiquity of this type of manipulation, and the considerably simpler syntactic form required to express it, has motivated the introduction of the RESET task as an abbreviated form of a REPLACE task which defines discrete changes to the forcing functions assigned to input variables.

Figure 7 demonstrates two applications of the RESET task. In the first example, the opening of a manual valve by an operator is modeled by manipulating the value of the variable representing the position of the valve stem. The second example illustrates how the action of a digital controller at the end of its sampling interval might be modeled. The value of the signal to the control valve is updated discretely according to the value of the expression on the righthand side at the time of execution of the task.

```
# Open a manual valve instantaneously
RESET
  Valve.Position := 1.0 ;
END

# Action of a digital controller at the end of its sampling interval
RESET
  Control_Valve.Signal := Bias + Gain*(Error + Integral_Error/Reset_Time) ;
END
```

Figure 7. Applications of the RESET task.

Of course, there are also certain phenomena that are best modeled as persistent changes to the composition of the set of variables. Examples include batch processes in which units are frequently brought on- and off-line, so only a subset of the total equipment inventory is active at any point in time, or systems in which significant changes occur to the number and/or nature of the thermodynamic phases present. At present, only limited facilities are provided for the declaration of these changes. The user is required to model a system in terms of a maximal set of variables, and then it is possible to "deactivate" or "activate" subsets of these variables by the insertion or deletion of special UNDEFINED equations that specify a subset of variables as being inactive (Barton, 1992).

Impulsive primitive actions

As has already been mentioned, impulsive actions cause instantaneous changes to the time trajectory of the system, but leave the composition and status of the variable and equation sets unchanged. The starting point of the new trajectory is defined by a set of *instantaneous* relationships between the describing variables. These relationships may involve the known *values* of the variables immediately before the impulsive action. For example, when the catalyst slurry is added to the batch reactor, the new value of the solvent holdup is equal to that just before the action, plus the amount added.

The REINITIAL task defines a subset of the differential variables that are to be considered discontinuous at the event that triggered its execution. The declaration also includes an equal number of nonlinear equations that will replace the continuity assumptions in determining consistent initial values for the system variables at the discontinuity (cf. Eq. 5). Note that these equations are only employed for the reinitialization calculation, and are not used for determining the dynamic behavior thereafter. Two examples of its application are shown in Figure 8.

The first example models the elimination of reset windup when an analog control loop with integral action is switched from manual to automatic control. In the second example, a small quantity (10 mol) of catalyst (component No. 2) is added instantaneously to a batch reactor, under the assumption that the temperature of the contents remains constant. It is important to recognize that this assumption leads to a discontinuity in the internal energy holdup. The special function OLD is introduced to include the known values of the describing variables immediately before the event in the relationships that replace the continuity assumptions.

```
# Initialise the integral error of an analogue controller
REINITIAL
  PI_Controller.Integral_Error
WITH
  PI_Controller.Integral_Error = 0 ;
END

# Instantaneous isothermal addition of catalyst
REINITIAL
  BatchReactor.Holdup(2), Batch_Reactor.U_Holdup
WITH
  BatchReactor.Holdup(2) - OLD(BatchReactor.Holdup(2)) = 10 ;
  Batch_Reactor.Temp = OLD(Batch_Reactor.Temp) ;
END
```

Figure 8. Applications of the REINITIAL task.

Task schedules

A *schedule* provides the means by which the ordering and execution of elementary tasks in the time domain may be declared. The control structures employed are almost identical to those used by modern general-purpose programming languages, and control languages for sequential process operations (Rosenof and Ghosh, 1987), providing facilities for sequential, iterated, and conditional execution of tasks. The concurrent nature of the real world is reflected by the provision of a control structure for the explicit declaration of tasks to be executed in parallel, similar to those provided by some of the parallel programming languages, such as OCCAM (Inmos, 1984). Intuitively this explicit declaration of concurrency is more helpful for the description of operating procedures, although scope also exists for the addition of language structures that mimic the model of concurrency supported by the process interaction world view of discrete event simulation languages (Kreutzer, 1986).

The execution of all elementary tasks takes place instantaneously with respect to the simulation clock. The final vital component is therefore a mechanism by which the duration of periods of continuous change between subdomain boundaries can be specified. This is provided by the CONTINUE task, which suspends execution of the control structure in which it appears, while scheduling the event that will result in the resumption of execution. In order for execution to resume, the simulation clock must be advanced to this event by numerical integration of the underlying mathematical model. This task comes in two forms, distinguished by the nature of the event scheduled. The first form:

CONTINUE FOR ⟨real expression⟩

schedules a time event, whereas the second form:

CONTINUE UNTIL ⟨logical expression⟩

schedules a state event. These two forms may also be combined in a single task through the use of AND and OR operators. For example, the CONTINUE FOR ⟨n⟩ OR UNTIL ⟨x⟩ form is particularly useful in speculative simulations in which the user is uncertain whether a certain state event will ever occur, and therefore wishes to specify a certain maximum time beyond which the simulation should not proceed.

We are now in a position to define a complete task entity. Figure 9 illustrates the declaration of a task entity that models the application of a simple startup procedure to a flowsheet composed of a storage vessel with feed and product pumps. The schedule that it encapsulates describes the evolution of this procedure.

```
TASK Simple_Start_Up

SCHEDULE
  SEQUENCE
    # Start feed pump to tank - outlet valve closed
    RESET
      Feed_Pump.Status := Feed_Pump.On ;
    END

    # Wait until the tank reaches its level set point
    CONTINUE UNTIL Tank.Liquid_Level > Level_Control.Set_Point

    # Concurrently...
    PARALLEL
      # Start product pump
      RESET
        Product_Pump.Status := Product_Pump.On ;
      END

      # Close the level control loop
      REPLACE
        Level_Control.Control_Action
      WITH
        Level_Control.Bias := 12.7 ;
      END # replace

      # Eliminate any reset windup accumulated while control loop was open
      REINITIAL
        Level_Control.Integral_Error
      WITH
        Level_Control.Integral_Error = 0 ;
      END # reinitial
    END # parallel

    # Continue simulation for another 100 time units
    CONTINUE FOR 100
  END # sequence
END # Simple_Start_Up
```

Figure 9. Simple startup schedule.

Initially, both pumps are idle and the level control loop is under manual control (the outlet valve is closed). The feed pump is started, and we wait until the level in the storage tank reaches its setpoint (a state event). Then, simultaneously, the product pump is started and the level control loop is switched from manual to automatic control. In order to model these actions, three elementary tasks are executed concurrently: a RESET task changes the outlet pump's status to operating, a REPLACE task replaces a specification on the controller's signal with a specification on the value of the controller's bias, and finally a REINITIAL task eliminates any reset windup that may have occurred while the control loop was under manual control. Finally, we wait another 100 time units until (presumably) the system approaches steady state.

Complexity and reusability—task hierarchies

We have already highlighted the importance of addressing complexity in models of the physical behavior of processing systems. Combined discrete/continuous process simulation faces the additional problem of dealing with the potential complexity of the external actions imposed on the system. For all but the most trivial simulation experiments, the complexity of these external actions may grow just as rapidly as that of the model of the physical behavior. The management of complexity in the physical model of a system provides useful insights regarding possible mechanisms for the management of task complexity—a complex operation on an item of process equipment can usually be decomposed in terms of lower level operations applied to the structural components of this equipment. Here, however, the decomposition is procedural as opposed to structural: the role of the more complex operation is to define the ordering in the time domain of the lower level operations. Each of the lower level operations may in turn be decomposed in terms of other operations, the decomposition continuing until all operations can be described in terms of discrete manipulations of the physical model that are implementable in terms of the elementary tasks introduced earlier.

The construction of these *task hierarchies* is facilitated by the introduction of the notion of a parameterized *task entity*. All task entities are user-defined, and encapsulate a declaration of a complete schedule and a set of parameters that permit its use in a number of different applications. In addition to the basic parameter types common to many programming and modeling languages, we introduce parameters which define the model entity (or entities) manipulated by the task. This important feature enables the declaration of task entities that may operate on any instance of a particular model entity, and therefore facilitates the reuse of descriptions of frequently occurring operations. The hierarchical decomposition can extend to an arbitrary number of intermediate levels since any schedule may be defined in terms of the execution of instances of other task entities.

Figure 10 illustrates the declaration of parameterized task entities that model the switching of an analog control loop from manual to automatic control, and the startup of a pump. Note that the actual model instances on which these tasks operate are passed as parameters, so that the schedule in Figure 9 could be rewritten as shown in Figure 11. In particular, notice that the task entity Start_Pump has been reused and applied to two different pumps. The extensive use of this facility is demonstrated by the detailed example presented in the companion article (Barton and Pantelides, 1994).

Earlier, we discussed some of the advantages of declaring models of physical behavior in inheritance hierarchies. For example, imagine that a collection of models for pumps that occur frequently in processing systems have been developed in an inheritance hierarchy in order to make the most efficient use of shared information. If we now want to model the startup and switching off of all these different pumps, it would be highly undesirable if separate task entities had to be developed for each member of the hierarchy.

Accordingly, the simulation language provides a powerful facility that obviates the need to declare all these separate task

```
# Task to close an analogue control loop
TASK Close_Loop

PARAMETER
  Controller      AS MODEL PI_Controller
  Steady_Bias     AS REAL

SCHEDULE
  PARALLEL
    # Close the control loop
    REPLACE
      Controller.Control_Action
    WITH
      Controller.Bias := Steady_Bias ;
    END # replace

    # Eliminate any reset windup accumulated while control loop was open
    REINITIAL
      Controller.Integral_Error
    WITH
      Controller.Integral_Error = 0 ;
    END # reinitial
  END # parallel
END # Close_Loop

# Task to start a pump
TASK Start_Pump

PARAMETER
  Pump         AS MODEL Generic_Pump

SCHEDULE
  RESET
    Pump.Status := Pump.On ;
  END
END # Start_Pump
```

Figure 10. Parameterized task entities.

```
TASK Simple_Start_Up

PARAMETER
  Plant AS Tank_Flowsheet

SCHEDULE
  SEQUENCE
    # Start feed pump to tank - outlet valve closed
    Start_Pump(Pump IS Plant.Feed_Pump) ;

    # Wait until the tank reaches its level set point
    CONTINUE UNTIL Tank.Liquid_Level > Level_Control.Set_Point

    # Concurrently...
    PARALLEL
      # Start product pump
      Start_Pump(Pump IS Plant.Product_Pump) ;

      # Close the outlet level control loop
      CloseLoop(Controller  IS Plant.Level_Control,
                Steady_Bias IS 12.7          ) ;
    END # parallel

    # Continue simulation for another 100 time units
    CONTINUE FOR 100
  END # sequence
END # Simple_Start_Up
```

Figure 11. Simple startup schedule that employs parameterized tasks.

entities. In particular, the concept of a model type, also introduced in the previous section, allows the actual value assigned to any model parameter to be an instance of any of the model entities descended by inheritance from the original parameter type. This facility enables a task entity to describe the same operation applied to a broad range of similar submodels, provided that this operation can be described in terms of the basic attributes of the original model entity. For example, notice that the task Start_Pump in Figure 10 is declared in terms of a parameter of type Generic_Pump. If this is the root of the above inheritance hierarchy of pump models, the task entity may be applied to any member of the hierarchy. This feature is closely related to the notion of a polymorphic entity in object-oriented programming languages (Meyer, 1988).

Beyond Process Simulation

The examples presented so far have all been built around process engineering applications, which provided the initial motivation for this research. However, we believe that the modeling methodology introduced in this article is suitable for application in a much wider range of engineering disciplines. This belief is based on the two key features of the methodology:

(1) The very general nature of the mathematical formulation, and the discrete changes that are tolerated, encompasses the features of many physical systems.

(2) The partitioning of the system under investigation into a model of a physical behavior, and a model of the external actions applied to this system is much more suitable for modeling many engineering systems than current approaches.

Typically, the methodology is most useful where a complex series of external actions or stimuli are applied to a single physical system. Some interesting application areas include mechanical systems, electrical circuits, environmental modeling, and biological systems (for example, multiple stimuli applied to the human body).

To illustrate this, we will use a simple example of a mechanical system drawn from the literature (Mattsson, 1989). The example was originally introduced to illustrate the deficiencies of current simulation languages, and involves two rotating masses that may be switched between a rigid coupling and a slip coupling. When the masses are connected by a slip coupling, the physical system is described by the following equations:

$$J_1\dot{\omega}_1 = Q_{l1} + Q_{r1} \tag{8}$$

$$J_2\dot{\omega}_2 = Q_{l2} + Q_{r2} \tag{9}$$

$$Q_{r1} = -Q_{l2} \tag{10}$$

$$Q_{r1} = d(\omega_2 - \omega_1) \tag{11}$$

where ω_1 and ω_2 represent the angular velocities of the two bodies, J_1, J_2, and d are parameters, and the torques Q_{l1} and Q_{r2} are known functions of time. On the other hand, when the masses are rigidly connected, Eq. 11 is replaced by the following equation:

$$\omega_2 = \omega_1 \tag{12}$$

Note that the index of this latter set of equations is 2, so the index of the overall system will fluctuate as the type of coupling is switched dynamically.

Consider the schedule shown in Figure 12, where the masses are initially coupled rigidly. After a short period, the masses are switched to a slip coupling, and this is a modeled by the first REPLACE task. In the second subdomain, the two angular velocities are initially equal, but may subsequently vary independently. At some later point in time, the masses are again switched back to a rigid coupling. The modeling of this control action is somewhat more complicated because the two masses may now be rotating at different angular velocities. Not only must one of the describing equations be replaced, it is also necessary to use a REINITIAL task to force conservation of angular momentum at the discontinuity.

```
SCHEDULE
  SEQUENCE
    # Masses initially rigidly coupled
    CONTINUE FOR 10

    # Switch to slip coupling
    REPLACE
      Rigid_Coupling
    WITH
      Slip_Coupling   AS  Mass1.Right_Torque =
                                  Damping*(Mass2.Ang_Velocity - Mass1.Ang_Velocity) ;
    END # replace

    CONTINUE FOR 10

    # Switch to rigid coupling
    PARALLEL
      # Switch the equation
      REPLACE
        Slip_Coupling
      WITH
        Rigid_Coupling AS  Mass2.Ang_Velocity = Mass1.Ang_Velocity  ;
      END # replace

      # Conservation of angular momentum
      REINITIAL
        Mass1.Ang_Velocity
      WITH
        OLD(Mass1.Inertia*Mass1.Ang_Velocity + Mass2.Inertia*Mass2.Ang_Velocity) =
                                  (Mass1.Inertia + Mass2.Inertia)*Mass1.Ang_Velocity ;
      END # reinitial
    END # parallel

    CONTINUE FOR 10
  END # sequence
```

Figure 12. Model of control actions applied to two rotating bodies.

Conclusions

To a greater or lesser extent, all processing systems experience significant discrete changes superimposed on their predominantly continuous dynamic behavior. Currently available general-purpose dynamic simulation packages are either suitable for modeling small perturbations around the nominal steady-state of a continuous process, or tailored to the particular demands of batch process simulation. However, as argued by Marquardt (1991): "Future simulation packages must support the analysis of arbitrarily operated processes within a unified framework." In addition, the substantial investment associated with the development of a process model is motivating the evolution of process modeling environments in which the same model may be reused for a broad range of activities. The design and implementation of the first general-purpose combined discrete/continuous process modeling environment was therefore undertaken, with particular emphasis on the representational methodologies required to facilitate this technology.

The new mathematical formulation of the process simulation problem as a multistage DAE system identifies the fundamental discrete changes required to model process behavior, and introduces the notion that the initial condition of a set of DAEs can be expressed in the most general terms possible by the requisite number of nonlinear relations between the initial values of the variables and their time derivatives. Furthermore, it provides some clarification of the various numerical techniques required to achieve a solution.

Most processing systems are conveniently decomposed into a single physical subsystem, and the external actions imposed on this physical subsystem. However, both have significant discrete components. The paper considers in detail the development of language structures for modeling these two aspects of process behavior.

The above decomposition can be contrasted to both that adopted by general-purpose combined simulation languages (Fahrland, 1970), and that suggested by object-oriented principles, which state that data (or declarative knowledge) should be specified with the routines (or procedural knowledge) that operate on that data, in a single entry known as a class. Applying these notions to the language structures described in this article would suggest that the "data" concerning physical behavior described by a model entity should be encapsulated in a single structure with the task entities that operate on it. For example, the physical model of a valve could be encapsulated with tasks that model the valve being opened and closed. However, one of the fundamental issues addressed by this work is the evolution of a process modeling environment that may reuse the information concerning the physical behavior of a system in a wide range of activities. If this is to be achieved, the information encapsulated by a model entity must be decoupled from information specific to certain activities (such as the discrete manipulations imposed during a dynamic simulation). Hence, the decoupling of model and task entities in the manner described above.

In considering the modeling of physical behavior, a formalism is introduced that provides a much more general representation of discontinuities in physical behavior than previously reported. The recognition that external actions can be modeled in terms of discrete manipulations of the underlying mathematical model has led to the development of a set of general-purpose primitives that provide considerable flexibility in the scope of these manipulations, albeit to a mathematical model of fixed dimensionality. Facilities to express complex sequences of control actions do not exist in any other process simulation package currently reported in the literature. Particular attention is also paid to the important practical issues of complexity and reusability in both aspects of the process model.

An interesting question concerns the completeness of the modeling structures defined in the language presented in this article. We note that attention here was focused entirely on systems whose physical behavior is described in terms of differential-algebraic equations. Clearly, more general descriptions in terms of sets of delay differential-algebraic equations or partial differential-algebraic equations may be necessary for some systems (for example, those including distributed variables). On the other hand, the language *is* complete with respect to the mathematical problem defined by Eqs. 2, 3, 5, and 6, in the sense that any such mathematical problem can, in principle, be expressed in this language. Conversely, any problem expressed in the language gives rise to a mathematical problem of the above form. Of course, such completeness, albeit highly desirable, is no guarantee for the language providing a natural or convenient manner for the definition of realistic engineering problems. This can only be established by practical experience.

The companion article (Barton and Pantelides, 1994) considers how models of physical behavior and models of external actions can be brought together to form the description of a complete dynamic simulation experiment. In addition to these models, such an experiment will require a specification of the initial state of the system under investigation. This will be composed of a specification of the initial condition, and the initial set and time variation of the inputs (or degrees of freedom). The development and capabilities of a software package (gPROMS) based on these notions will be presented.

Acknowledgments

This work was supported by the Science and Engineering Research Council (UK). P. I. Barton is grateful for the partial financial support provided jointly by Air Products PLC and Air Products and Chemicals, Inc.

Literature Cited

Barton, P. I., and C. C. Pantelides, "The Simulation of Combined Discrete/Continuous Processes," in preparation (1994).

Barton, P. I., "The Modelling and Simulation of Combined Discrete/Continuous Processes," PhD Thesis, Univ. of London (1992).

Brenan, K. E., S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, North-Holland (1989).

Cellier, F. E., and A. P. Bongulielmi, "The COSY Simulation Language," *Simulation of Systems '79*, L. Dekker, G. Savastano, and G. C. Vansteenkiste, eds., North-Holland, p. 271 (1980).

Cellier, F. E., "Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools," PhD Thesis, Swiss Fed., Inst. of Technol., Zurich (1979a).

Cellier, F. E., "Combined Discrete/Continuous System Simulation Languages—Usefulness, Experiences and Future Development," *Methodology in Systems Modelling and Simulation*, B. P. Ziegler, M. S. Elzas, G. J. Klir, and T. I. Oren, eds., North-Holland, p. 201 (1979b).

Clark, S. M., and K. Kuriyan, "BATCHES—Simulation Software for Managing Semicontinuous and Batch Processes," *Proc. AIChE Meeting* (Apr., 1989).

Czulek, A. J., "An Experimental Simulator for Batch Chemical Processes," *Comp. & Chem. Eng.*, **12**, 253 (1988).

Fahrland, D. A., "Combined Discrete Event Continuous Systems Simulation," *Simulation*, **14**, 61 (1970).

Fruit, W. M., G. V. Reklaitis, and J. M. Woods, "Simulation of Multiproduct Batch Chemical Processes," *The Chem. Eng. J.*, **8**, 199 (1974).

Gani, R., E. L. Sørensen, and J. Perregaard, "Design and Analysis of Chemical Processes through DYNSIM," *Ind. Eng. Chem. Res.*, **31**, 244 (1992).

Helsgaun, K., "DISCO—a SIMULA Based Language for Continuous Combined and Discrete Simulation," *Simulation*, **35**, 1 (July, 1980).

Heydweiller, J. C., R. F. Sincovec, and L. T. Fan, "Dynamic Simulation of Chemical Processes Described by Distributed and Lumped Parameter Models," *Computers Chem. Eng.*, **1**, 125 (1977).

Holl, P., W. Marquardt, and E. D. Gilles, "Diva—a Powerful Tool for Dynamic Process Simulation," *Computers Chem. Eng.*, **12**, 421 (1988).

Hopcroft, J. E., and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley (1979).

Inmos, *OCCAM Programming Manual*, Prentice-Hall, Englewood Cliffs, NJ (1984).

Joglekar, G. S., and G. V. Reklaitis, "A Simulator for Batch and Semi-Continuous Processes," *Computers Chem. Eng.*, **8**, 315 (1984).

Kettenis, D. L., "COSMOS: A Simulation Language for Continuous, Discrete and Combined Models," *Simulation*, **58**, 32 (1992).

Kreutzer, W., *System Simulation Programming Styles and Languages*, Addison-Wesley (1986).

Laganier, F. S., J. M. Le Lann, X. Joulia, B. Koehret, and M. Morari, "Simultaneous Modular Dynamic Simulation: Application to Interconnected Distillation Columns," *Computers Chem. Eng.*, **17**, S287 (1993).

Mani, S., S. K. Shoor, and H. S. Pedersen, "Experience with a Simulator for Training Ammonia Plant Operators," *Plant/ Operations Progress*, **10**, 6 (1990).

Marquardt, W., "Dynamic Process Simulation—Recent Progress and Future Challenges," in *Proceedings CPC IV*, TX (1991).

Mattsson, S. E., "On Modelling and Differential/Algebraic Systems," *Simulation*, 24 (Jan., 1989).

Meyer, B., *Object-Oriented Software Construction*, Prentice Hall, Englewood Cliffs, NJ (1988).

Naess, L., A. Mjaavatten, and J. Li, "Using Dynamic Process Simulation from Conception to Normal Operation of Process Plants," *Computers Chem. Eng.*, **16**, S119 (1992).

Nilsson, B., "Structured Modelling of Chemical Processes—An Object-Oriented Approach," PhD thesis, Lund Institute of Technology, Sweden (1989).

Pantelides, C. C., and P. I. Barton, "Equation-Oriented Dynamic Simulation Current Status and Future Perspectives," *Computers Chem. Eng.*, **17**, S263 (1993).

Pantelides, C. C., D. Gritsis, K. R. Morison, and R. W. H. Sargent, "The Mathematical Modelling of Transient Systems Using Differential-Algebraic Equations," *Computers Chem. Eng.*, **12**, 449 (1988).

Pantelides, C. C., "The Consistent Initialization of Differential/Algebraic Systems," *SIAM J. Sci. Stat. Comput.*, **9**, 213 (1988).

Pantelides, C. C., "SpeedUp—Recent Advances in Process Simulation," *Computers Chem. Eng.*, **12**, 745 (1988).

Pearce, J. G., "Computer Simulation of Multi-State Systems," in *Proceedings UK Simulation Council Conference on Computer Simulation*, IPC Science and Tech Press, pp. 484-493 (1978).

Perkins, J. D., and G. W. Barton, "Modelling and Simulation in Process Operation," *Foundations of Computer Aided Process Operations*, G. V. Reklaitis and H. D. Spriggs, eds., Cache-Elsevier, pp. 287-316 (1987).

Perkins, J. D., and R. W. H. Sargent, "SPEEDUP: A Computer Program for Steady-State and Dynamic Simulation and Design of Chemical Processes," in *AIChE Symposium Series*, No. 78 (1982).

Perkins, J. D., "Dynamic Simulation of Chemical Plants: Why and How?" in *Proceedings Multi-Stream '85*, London (1985).

Perkins, J. D., "Survey of Existing Systems for the Dynamic Simulation of Industrial Processes," *Modeling, Identification and Control*, **7**, 71 (1986).

Piela, P. C., T. G. Epperly, K. M. Westerberg, and A. W. Westerberg, "ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis: The Modeling Language," *Computers Chem. Eng.*, **15**, 53 (1991).

Pipilis, K. G., "Higher Order Moving Finite Element Methods for Systems Described by Partial Differential-Algebraic Equations," PhD thesis, University of London (1990).

Pritsker, A. A. B., and R. E. Hurst, "GASP IV: A Combined Continuous-Discrete FORTRAN-Based Simulation Language," *Simulation*, **21**, 65 (1973).

Pritsker, A. A. B., *Introduction to Simulation and SLAM II*, John Wiley (1986).

Rosenof, H. P., and A. Ghosh, *Batch Process Automation Theory and Practice*, Van Nostrand Reinhold, New York (1987).

Shinohara, T., "A Block Structured Approach to Dynamic Process Simulation," in *Foundations of Computer Aided Process Operations*, G. V. Reklaitis and H. D. Spriggs, eds., Cache-Elsevier, p. 317 (1987).

Smart, P. J., and N. J. C. Baker, "SYSMOD—An Environment for Modular Simulation," in *Proceedings Summer Computer Simulation Conference*, North-Holland, p. 77 (1984).

Sørensen, E. L., S. W. Sørensen, and R. Gani, "Simulation and Analysis of Batch Chemical Processes," in *Computer-Oriented Process Engineering*, L. Puigjaner and A. Espuña, eds., Elsevier (1991).

Stephanopoulos, G., G. Henning, and H. Leone, "MODEL.LA. A Modeling Language for Process Engineering—I. The Formal Framework," *Computers Chem. Eng.*, **14**, 813 (1990).

Strauss, J. C., "The SCi Continuous System Simulation Language (CSSL)," *Simulation*, **9**, 281 (1967).

Wood, R. K., R. K. M. Thambynayagam, R. G. Noble, and D. J. Sebastian, "DPS—A Digital Simulation Language for the Process Industries," *Simulation*, 221 (May 1984).

Manuscript received Dec. 29, 1992, and revision received Aug. 16, 1993.