Conditional Modeling, Part I: Requirements for an Equation-Based Environment

V. Rico-Ramírez, B. A. Allan and A. W. Westerberg[1]

Department of Chemical Engineering and
the Institute for Complex Engineered Systems
Carnegie Mellon University, Pittsburgh, PA 15213

Abstract: In a series of two papers, we investigate the setting up and solving of conditional models within an equation-based modeling environment. Conditional models arise in chemical engineering when modeling systems involve physicochemical discontinuities, such as phase transitions. Grossmann and Turkay (1996) show how one can represent conditional models as algebraic systems of disjunctive equations. In this paper we identify the modeling capabilities needed for the efficient representation and solution of conditional models. We then describe modeling tools for the performance of each of the identified tasks: conditional configuration of a model structure, conditional compilation and conditional execution of procedural statements. We next describe the details of the computer implementation of these tools and show how the expressiveness of an equation-based modeling language increases with their incorporation. Finally we present several chemical engineering examples to demonstrate the scope of application of the proposed extensions.

In Part II we investigate how to solve these models using a new complementarity formulation and the boundary crossing algorithm of Zaher (1995).

Keywords: Modeling, Conditional models, Equation-based environment.

1. Author to whom all correspondence should be addressed.

1 INTRODUCTION

Modeling is the process of mapping reality into a representation that is thought to be useful for understanding that reality (Abbott, 1996). Process modeling is an important task in many process engineering activities. The state of the art as well as future trends in process modeling and simulation have been reviewed in a number of publications (Piela, 1989; Boston *et al*., 1993; Pantelides and Britt, 1994; Marquardt, 1996).

The modeling tools in current simulators may roughly be classified into two groups: modular oriented and equation-based approaches (Boston *et al.*, 1993). A detailed discussion of the modular approach versus the equation-based approach can be found in Marquardt (1996).

On the one hand, modular approaches address modeling on the flowsheet level. Every process is abstracted by a block diagram consisting of standardized blocks which model the behavior of a process unit or a part of it. All the blocks are linked by connections representing the flow of information, material and energy employing standardized interface and stream formats (Marquardt, 1996). This modular approach, though powerful and easily accessible to many engineers for the solution of standard flowsheet problems, does not adequately support the solution of more involved problems. This is due to the lack of precoded models for many unit operations of adequate level of detail (Marquardt, 1996).

On the other hand, equation-based modeling tools are motivated by the fact that, at the lowest level, process models are represented by a large set of variables and a large system of linear and nonlinear equations that relate them. Thus, equation-based approaches support the implementation of unit models by means of declarative modeling languages. In the equation-based modeling approach, the definition of the system of equations is independent of any particular application or solution algorithm that may be used for their solution. For that reason, the solution to equation-based models has been demonstrated as effective in solving simulation, optimization, parameter estimation and data reconciliation problems, all using a single set of equations (Allan, 1997). Recognition of the potential benefits of the equation-based technology has led to the development of equation-based modeling tools such as SpeedUP (Pantelides, 1988), gPROMS (Barton, 1992; Oh and Pantelides, 1994), and GAMS (Brooke *et al*., 1997). Some other researches have proposed that equation-based modeling can be further facilitated by the use of

object-oriented frameworks. Examples are OMOLA (Mattsson and Andersson, 1993) and ASCEND (Piela, 1989; Allan, 1997).

Arguably, the state of the art in process modeling software at the time of writing this work is an object-oriented, equation-based modeling environment (Abbott, 1996; Allan, 1997).

1.1 CONDITIONAL MODELS

A conditional model, as defined by Zaher (1995) and Grossmann and Turkay (1996), consists of a system of equations expressed by two sets: a globally defined invariant set of equations and a variant (or locally defined) set of conditional equations which are expressed as disjunctions. Grossmann and Turkay (1996) show that a conditional model can be represented as the system of disjunctive equations:

$$\begin{aligned} & \underline{h}(\underline{x}) = 0 \\ & \bigvee_{i \in D_k} \begin{bmatrix} r_{i_{jk}}(\underline{x}) = 0 \\ g_{i_{lk}}(\underline{x}) \le 0 \end{bmatrix} k \in K \qquad \begin{matrix} \forall j \in [1 \ldots \beta_k] \\ \forall l \in [1 \ldots \gamma_k] \end{matrix} \\ & \underline{x} \in R^n \end{aligned} \tag{1}$$

where $\underline{h}(\underline{x})$ and $\underline{r}_{i_k}(\underline{x})$ represent the vectors of the invariant and the variant sets of equations respectively, K represents the set of disjunctions and the index i is used to indicate the i-th disjunctive term in each disjunction D_k. The domain of validity of each set of variant equations is given by the inequality constraints $\underline{g}_{i_k}(\underline{x})$.

Examples of conditional models in chemical engineering are systems involving physicochemical discontinuities such as flow and phase transitions. There are also cases where conditional cost functions are used in the optimization of process flowsheets. Figure 1, taken from the work of Zaher (1995), illustrates the case of a flow transition. In that case, we must incorporate alternative equations for the transport properties and velocity bounds within the model, using the simultaneously calculated value of the flow type indicator (Reynolds or Mach number) relative to some critical value to determine which is active. A simple flash equilibrium calculation also represents a conditional model commonly encountered in chemical engineering. Figure 2

illustrates that example. Depending on the value of the temperature (relative to the dew point temperature and bubble point temperature at a given pressure), the system may be present as a subcooled liquid, as a superheated vapor, or as a liquid-vapor mixture in equilibrium. The system of equations of the model may be different (depending on the solution method employed) for each of the alternatives. The problem is that the temperature may be unknown, and, therefore, there is no way of knowing *a priori* which set of equations one needs to satisfy.

According to the above, the representation of conditional models is particularly difficult because it is necessary to declare alternative modeling equations which become active depending on the values of the modeling variables. Moreover, the solution to conditional models involves simultaneously selecting the equations to be solved and solving them.

In this work, we review the conditional modeling capabilities of the existing equation-based modeling languages. Also, we study the algorithmic and representational needs of the existing conditional modeling solution techniques. Based on those activities, we identify the requirements that an equation-based environment should meet in order to represent conditional models efficiently and in order to provide the flexibility needed by the solution techniques. Thus, we describe the incorporation of a series of tools which provides the user of an equation-based modeling environment with the ability to represent and solve complex conditional models.

For the purpose of model representation, in this work we extend the scope of the definition of a conditional model. Here, we consider as a conditional model any problem including a set of disjunctive statements as part of its formulation. In other words, the domain of validity of each particular set of alternatives equations does not have to be given by inequality constraints. Any kind of logical, integer, or binary variables can be used for that purpose instead. With that in mind, the expressiveness of our modeling tools can also be applied to problems on which the selection of alternative configurations is based on algorithmic and heuristic decisions, for example, as in the case of pure MINLP problems; as a consequence, potential applications of the new modeling capabilities vary from the simple substitution of one equation for another (as in the case of the laminar-turbulent flow transition) to the substitution of a section of a chemical plant for another (as can be required while analyzing and initializing a superstructure).

FIGURE 1 Fluid flow transition.

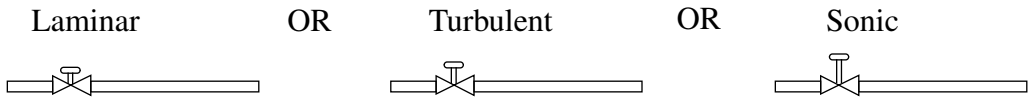
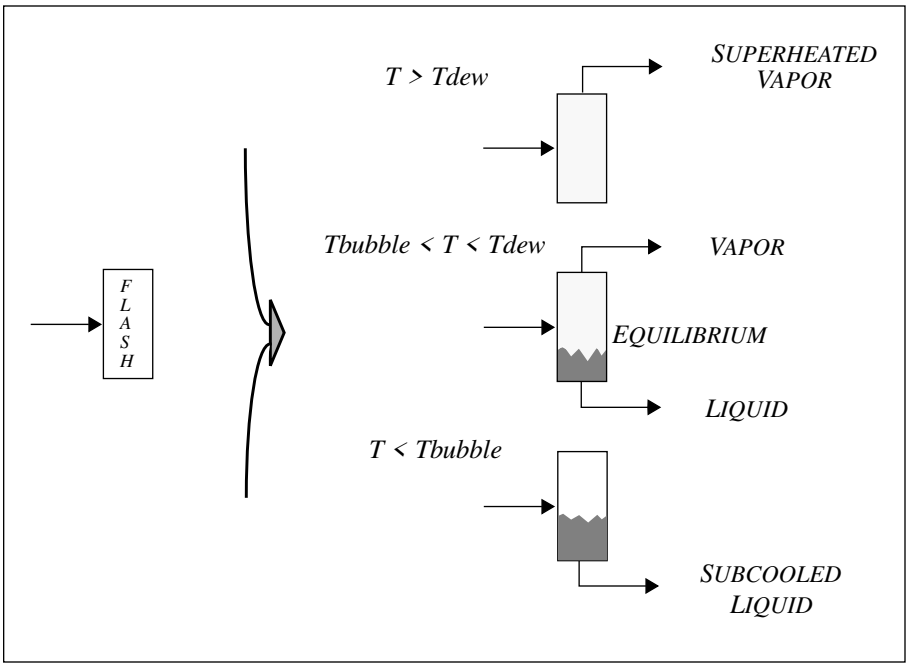


FIGURE 2 A conditional model: flash calculation.



2 BACKGROUND: CONDITIONAL MODELING IN EXISTING MODELING ENVIRONMENTS

Many currently available modeling systems have been reported in the literature. Some of the most recent approaches to computer-aided modeling tools are shown in Table 1. These modeling tools can be classified into 2 groups (Marquardt, 1996).

On the one hand, there is a group of modeling languages mainly suitable for their use in a specific application domain. In MODEL.LA and VEDA, elements tailored to chemical engineering applications are included in the language definition; In such languages, very efficient modeling constructs exist, but their main limitation is that they have been developed to match the issues of specific engineering applications. Also, it should be recognized that, in the case of VEDA,

On the other hand, there is a group of general modeling languages: ABACUSS, ASCEND, DYMOLA, GAMS, gPROMS, SpeedUp and OMOLA. In such languages, elements are not restricted to specific engineering applications.

Table 1: Recent equation-based modeling systems

System	Reference
ABACUSS	Feehery and Barton (1996)
ASCEND	Piela (1989)
DYMOLA	Elmqvist *et al*. (1993)
GAMS	Brooke *et al*. (1997)
gPROMS	Barton (1992)
MODEL.LA	Stephanopoulos *et al*. (1990)
SpeedUp	Pantelides (1988)
OMOLA	Mattson and Anderson (1993)
VEDA	Bogusch and Marquardt (1995)

All of the general equation-based modeling languages presented in Table 1 provide constructs which allow the representation of conditional dependence of some of the model equations. For instance, the IF-THEN-ELSE construct (ABACUSS, ASCEND, gPROMS, DYMOLA, SpeedUp and OMOLA), the CASE construct (gPROMS) and the WHEN construct (DYMOLA). The purpose and meaning of all these existing modeling constructs are essentially the same. For instance, the IF-THEN-ELSE construct of *SpeedUp* first described by Pantelides (1988) is:

```
IF logical_condition THEN
      equation1
ELSE
      equation2
ENDIF
```

where both the logical conditions and the equations are expressed in terms of the model variables. Such a construct defines two system states corresponding to the IF and the ELSE clauses respectively. Multiple states may be described by nesting several IF statements. As another example, Barton (1992) incorporated the CASE equation into *gPROMS*. The CASE equation is used to define both the appropriate modeling equations in each state and the logical conditions for transitions among states. It covers multiple states within only one statement and has the advantage of successfully representing irreversible discontinuities. As an aside, there is a major difficulty in both of the previous approaches as tools for conditional modeling. They only allow the substitution of one list of primitive equations (or arrays of equations) for another. In an equation-based modeling environment in which object oriented concepts like hierarchy (building complex models from small models) and inheritance are constantly in use, this approach places a significant limitation on one's modeling efficiency. For instance, it makes it very difficult to model unit replacement when searching over a superstructure.

Two different applications of the existing modeling constructs have been developed:

1. First, the conditional language constructs have been used to handle changes procedurally of the configuration of the problem because of time or state events occurring during the dynamic simulation of a system, assuming that the initial steady state of the system is known a priori (or can be estimated without the need of solving a conditional system of equations). This application constitutes the main strength of ABACUSS, gPROMS, DYMOLA, SpeedUp and

OMOLA.

2. Secondly, some of the equation-based approaches, such as gPROMS and SpeedUp, have acknowledged the difficulty of solving steady state problems where the clause of an IF-THEN-ELSE construct which is active cannot be specified explicitly, but it has to be automatically calculated (Barton, 1992). It is not clear, however, how these modeling environments handle this problem, since an explicit methodology is not provided.

Here we have to make an important distinction about these two applications of the existing conditional language constructs:

- In conditional models, as defined by equation (1), the conditional dependence of the model equations does not necessarily exist because of state or time events occurring during the dynamic simulation of a system. Hence, the problem of discrete event dynamic simulation described first is not the main focus of attention of this work.
- On the contrary, the problem described as the second application of the existing conditional language constructs fits perfectly into our definition of a conditional model given by (1).

Accordingly, we can argue that most of the existing equation-based modeling languages have not presented a formal approach to solving conditional models as formulated by equation (1). We should note though that the GAMS modeling language provides modeling and solving tools (such as the dollar condition for conditional equations and the if-else statement for flow control) powerful enough to represent and solve conditional models. The major disadvantage of the GAMS modeling language is that it does not provide an object-oriented framework to facilitate the modeling process.

This limited attention from the equation-based community to conditional models provides the main motivation for this work.

3 CONDITIONAL MODELING TOOLS

Equation oriented modeling tools support the implementation of unit models and their incorporation in a model library by means of a declarative modeling language; declarative in the sense of explicit and symbolic encapsulation of the knowledge about models (Marquardt, 1996). Moreover, methods must also be attached to a model definition for the numerical processing of the

model equations.

In this paper, we identify three modeling capabilities which support the efficient development of conditional models in both the declarative definition of equation-based models and the procedural execution of methods:

- Conditional configuration of the model structure.
- Conditional compilation.
- Conditional execution of the procedural code of methods.

In the remainder of this section, we describe the syntax and semantics of modeling tools which allow the practical implementation of those modeling capabilities.

3.1 CONDITIONAL CONFIGURATION OF A MODEL STRUCTURE

Several numerical algorithms and methodologies for the solution of conditional models have recently become available. See for example Zaher (1995), Grossmann and Turkay (1996) and Turkay and Grossmann (1996). A common characteristic of any of these methodologies is that the dynamic switching among alternative model configurations is required during the solution process. Hence, a declarative modeling language has to provide a means to represent all the alternative structural configurations of the problem as well as the conditions which trigger the switching among them. Next, we describe language constructs which fulfill this requirement:

- the WHEN statement, which provides an efficient means to declare alternative modeling configurations
- logical relations and the CONDITION statement, which can be used to represent the conditions triggering the reconfiguration of the system.

3.1.1 The WHEN Statement

Originally, the syntax for incorporating conditional dependence of some equations of a model in an equation-based environment was suggested by Piela (1989). That syntax is very similar to the CASE equation of *gPROMS* and suffers from the same limitations. Instead, in this work we represent that conditional dependence by using an object-oriented formalism. In an object-oriented approach any real or abstract entity is considered an object and any object can be referenced by a unique identifier (Marquardt, 1996). So, for instance, in an object-oriented

language representing a model superstructure, a simple equation is considered as an object and so is any submodel within the superstructure. Based on this object-oriented approach, we defined the WHEN statement in order to represent alternative configurations of a model. The syntax of such a conditional statement is:

```
definition_of_equation_1;
definition_of_model_1;
      ⋮
definition_of_model_n;

WHEN (list_of_variables)
      CASE list_of_values_1:
                        USE identifier_of_equation_1;
      CASE list_of_values_2:
                        USE identifier_of_model_1;
      ⋮
      OTHERWISE:
                        USE identifier_of_model_n;
END WHEN;
```

The following are observations about the previous definition:

1. The declaration of the objects referenced within the CASEs of the WHEN statement is independent (outside) of such a statement. As mentioned before, the solution algorithms of conditional models require the system to have available the data structures of all those objects.
2. A list of variables is used to define the domain of validity of each of the alternative configurations. The variables in that list can be of any type among boolean, integer or symbol or any combination of them. By doing that, we place problems like logic based modeling and MINLP formulations within our scope of application.
3. Practically speaking, to "USE" an object means that the variables and equations contained in that object become an active part of the system of nonlinear equations representing the current configuration of the problem.
4. Complex reconfigurations of the problem are readily represented because of the object oriented approach of the statement.

The syntax of the WHEN statement given above can represent the conditional dependence of alternative sets of equations and variables. However, it does not say anything about (and is

independent of) how to represent the conditions that trigger the dynamic switching among configurations.

3.1.2 Logical Relations

The way in which existing solution algorithms for conditional models select a particular configuration varies. MINLP algorithms employ the manipulation of binary variables. Some other approaches use logic for improving the solution of conditional models (Raman and Grossmann, 1994; Turkay and Grossmann, 1996). Also, the boundary crossing algorithm given by Zaher (1991) expresses the truth value of inequality constraints as boolean values of logical conditions.

Here we describe our approach to the incorporation of logical relations as a declarative modeling tool. This incorporation is intended to provide an equation-based environment with the ability to deal with the logic based formalisms required by most of the solution algorithms mentioned above.

The syntax for the representation of logical relations is rather simple:

```
logical term <==> logical term;
logical term  ==> logical term;
```

The symbols "<==>" and "==>" indicate that we have a logical relation between the two logical terms. In each of the two terms, logical operators among boolean variables such as AND, NOT, and OR are allowed.

The symbol "<==>" implies equality between the logical terms. Equality in a logical equation can also be interpreted as an *if and only if* implication between two logical terms expressed in clausal form. On the other hand, the symbol "==>" should be interpreted as the one sided implication *if* between the two terms of the logical relation. Also, it should be noted that we can express any logical clause using the proposed syntax by simply writing the clause in one of the terms and the constant boolean value TRUE in the other term of the logical equality.

In this implementation of logical relations, we maintain the equation-based philosophy. That is, the user states the logical relations that must be true at the solution to the problem but not how to

solve them. Each logical equation has a residual attached to it. This residual will indicate if the expression is satisfied or not. Thus, this interpretation requires that we provide a solver that knows how to deal with logical relations. Such a solver should strive for the residuals of such equations to be true as it looks for a consistent set of values of the boolean variables in the problem. The work of Raman and Grossman (1993) can be used as a starting point for such a solver implementation. They used formal procedures for performing logic inference depending on the way in which the logical relations are present (Conjunctive Normal form (CNF) or Disjunctive Normal Form (DNF)).

Hence, if a consistent set of values of the boolean variables is available after each iteration, then, by checking the value of the appropriate boolean variables, an automatic change of the structure of a conditional problem is possible in an iterative solution scheme.

3.1.3 The CONDITION Statement

We have already stated that, in some solution algorithms for conditional models, the truth value of a boolean variable may depend on constraints expressed in terms of the real variables of the model. For this reason we propose the definition of the CONDITION statement and the SATISFIED logical operator as a complement to the implementation of logical relations. The syntax of this modeling tools is as follows:

```
CONDITION
   identifier_1: real_expression;
END CONDITION;

boolean_variable <==> SATISFIED(identifier_1,tolerance);
```

A real expression is defined and labeled inside the CONDITION statement and then the logical operator SATISFIED gives the truth value of such an expression (the residual of the real expression is compared against the tolerance defined in the SATISFIED operator). The benefits of the CONDITION statement are:

1. It contributes to the separation of equations into those given in terms of real variables and those given in terms of boolean variables, making the declarative code easier to read and understand.
2. It provides a simple way of saying that the relations defined within the statement are not going

to be solved. Those relations are not a part of our nonlinear system of equations but are only used as expressions with a truth value associated to them.

3. it avoids relations containing implicit relations. Since the logical expressions are decoupled from the expressions on which they depend, it makes the life of a compiler easier.

As a summary of section 3.1, the WHEN statement provides an efficient means to declare alternative modeling configurations, while the conditions triggering the reconfiguration of the system can be represented by the use of logical relations and the CONDITION statement. When working together, these modeling capabilities meet the representation needs of all of the solution algorithms for conditional models we have found reported in the literature.

3.2 CONDITIONAL COMPILATION

Aside from the flexibility that conditional statements such as the WHEN statement give to the configuration of a model structure, another application of conditional tools is the economy of programming. An example commonly occurring in chemical engineering is the selection of the thermodynamic model to be used for equilibrium calculations. In general, it is convenient to code all of the alternative methods so that, depending on the species appearing in the equilibrium system, we can select the most appropriate method.

In these kind of problems, the decision as to which configuration should be used has to be made at the moment in which the model is created and not during the solution process. Accordingly, what we require is building only the appropriate configuration of the problem rather than having available all the possible configurations. That is, we require conditional compilation.

We want to make clear that this modeling capability is not strictly related to the conditional problem represented by equation (1), and, as we state above, it is presented here as a response to the idea of economy of programming. In section 4 we show an illustration of how this novel capability of an equation-based modeling environment may be a useful tool in the modeling process.

Hence, we propose a modeling tool to incorporate conditional compilation into an equation-based environment, the SELECT statement. While this conditional tool is flexible enough to represent all of the alternatives, its presence will indicate that only those alternatives consistent with the model

data will become available after the process of instantiation of the model. The following is the syntax proposed for the conditional compilation tool:

```
defintion_of_constants;
assignment_of_constant_values;

SELECT (list_of_constants)
      CASE list_of_values_1:
                        list1_of_declarative_statements;
      CASE list_of_values_2:
                        list2_of_declarative_statements;
      ⋮
      OTHERWISE:
                        listn_of_declarative_statements;
END SELECT;
```

Even though the syntax for the SELECT statement is similar to that described for the WHEN statement, some important differences can be identified:

- In the WHEN statement, the declaration of the object is external to the conditional statement since of all the alternatives are going to be created anyway. On the contrary, in the SELECT statement the actual declaration of an object (or any other declarative statement affecting objects) is done within each CASE of the statement, explicitly discriminating among alternative configurations.
- The selection among alternatives in the SELECT statement depends on constant booleans, integer or symbols. Since these values imply a one time structural decision, they must not be modified during the solution of the problem.

Summarizing, the SELECT statement provides the capability of conditional compilation. It allows the representation of all the structural alternatives. However, since only the desired data structure is created, it does not affect the computational requirements of the model.

3.3 CONDITIONAL EXECUTION OF PROCEDURAL CODE

Because of the use of conditional statements in the declarative description of a model, a similar feature must also exist to give the user the ability to program the conditional execution of methods. For instance, each alternative configuration of a model may require different initialization and a different selection of the independent variables for the solution process. As in

the case of conditional compilation, the conditional execution of procedural code is not necessarily applied to conditional problems as the one given by (1). Still, the conditional execution of procedural statements is without a doubt a useful tool in the modeling process. For instance, this tool can be effectively used not only for initializing conditional models (such as those defined by (1)), but also can be used in the definition of procedures employed in the solution of conventional models (solving a model several times by defining different values for some parameters each time, according to the satisfaction of logical conditions). Thus, we propose a procedural conditional statement as follows:

```
SWITCH (list_of_variables)
      CASE list_of_values_1:
                        list1_of_procedural_statements;
      CASE list_of_values_2:
                        list2_of_procedural_statements;
      ⋮
      OTHERWISE:
                        listn_of_procedural_statements;
END SWITCH;
```

Basically, this SWITCH statement has the same application as the conditional statements that already exist in procedural modeling languages such as C and FORTRAN. Procedural statements do not involve new object definitions, they are only useful for the numerical processing of objects already created.

4 DETAILS OF A COMPUTATIONAL IMPLEMENTATION

The language tools introduced in the previous section provide a general framework for the representation of conditional models. In this section, we present an overview of the details that we had to address in order to create a prototype for testing the scope and application of the modeling tools.

Some of the main issues in the computer implementation of the conditional tools are:

- The implementation of the WHEN statement must provide an efficient means to generate all of the possible alternative configurations of the problem. That is, the combinatorial nature of the

problem must be encapsulated without being memory intensive.

- The solution to a conditional model will reduce to solving a system of equations in which the variables and equations of the system may constantly change during the solution process. The issue here is how we are going to supply a conditional modeling solver with the correct set of variables and equations.
- The development of an approach to the implementation of conditional compilation in a declarative modeling language is a hard problem by itself.

4.1 IMPLEMENTATION OF THE WHEN STATEMENT: THE WHEN CLASS

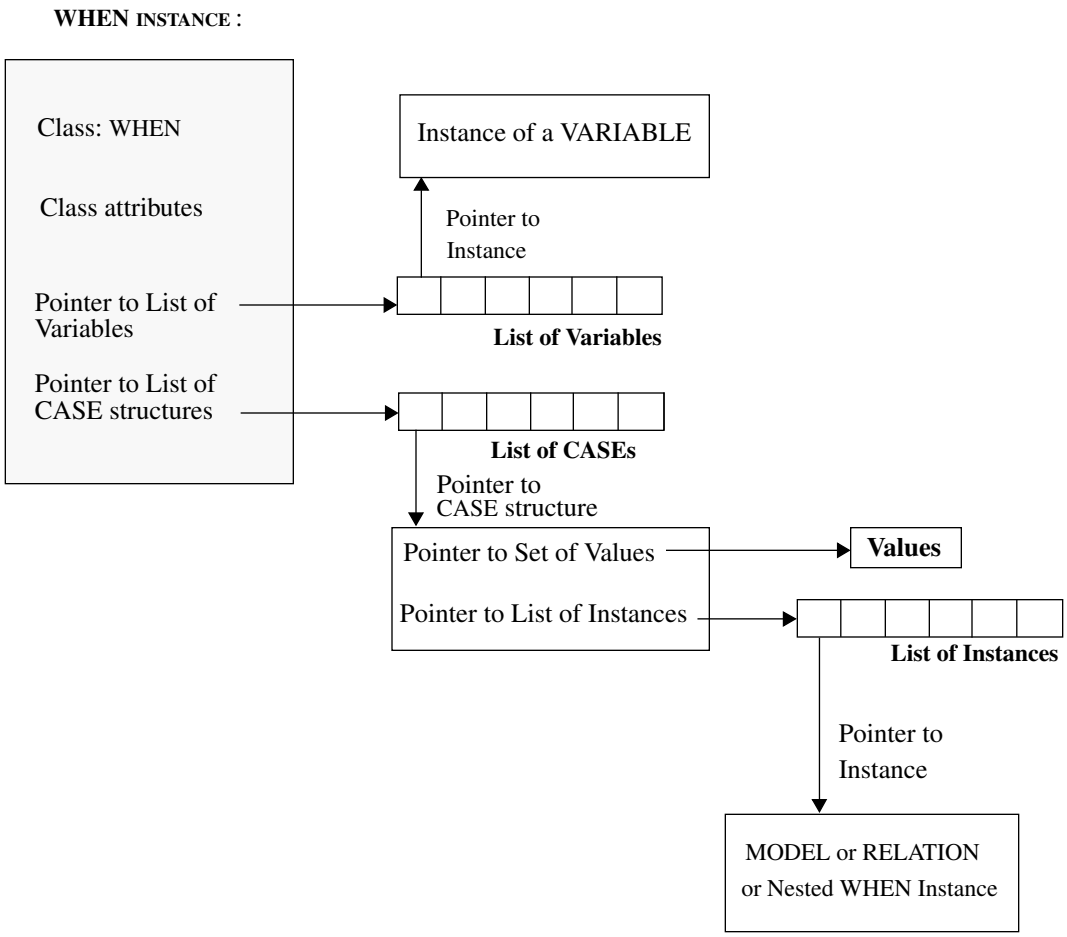
We follow the object oriented philosophy in order to implement a computer tool for the conditional configuration of a model structure. In a typical object-oriented modeling environment, all objects which share the same set of attributes can be viewed as an instance of a class (or type). Hence, each model is a structured class built hierarchically from instances of other models or elementary classes.

An early approach for the implementation of conditional modeling tools was described by Epperly (1988). He proposed to build a complete instance tree for each CASE within the conditional statement. However, he also recognized the combinatorial nature of that approach that makes it unacceptable; for example, for a type containing two conditional statements each having three CASEs, nine complete instance trees would be created.

In this work, we introduce the definition of an elementary class, the WHEN class. Instances of this class allow us to create a single instance tree in which all the structural alternatives are embedded. Figure 3 shows our approach to the implementation of a WHEN instance. Essentially, a WHEN instance is constituted by two lists of pointers: a list of pointers to instances of the conditional variables on which the WHEN statement depends and a list of pointers to CASE structures. Each CASE structure contains a list of values and another list of pointers to instances. The instances in the list of instances associated to a CASE correspond to the objects (relations, arrays of relations, models, array of models) that will be "active" when the values of the CASE list of values matches the current values of the conditional variables.

With this implementation, the data structures required for all the alternative configurations are available (*i. e.,* all the objects referenced in each CASE are compiled). By visiting the instance tree and analyzing all the WHEN statements in it, we can set as "active" only the parts of the problem corresponding to the configuration consistent with the current values of the conditional variables.

FIGURE 3 WHEN instance implementation.



4.1.1 Feeding a Conditional Model to a Nonlinear Solver

The implementation of the WHEN statement in an equation-based environment allow us to have available the data structure for all the variables and equations of the conditional model. Therefore, the first step in the implementation of a conditional solver is to decide how the solver is going to be fed with the correct structure and how to change this structure efficiently in an

iterative solution scheme. To perform this task, we will use the notion of active equation and active variable. By active we mean "it is part of the problem currently being solved." Computationally speaking, to set a relation as active or inactive implies a simple bit operation.

The following steps provide a mechanism to select the structure of the problem:

1. Initially, we consider all the equations resulting from the compilation as active.
2. Then, we set as inactive all of the equations referenced within any WHEN statement. The equations set as inactive in this step constitute the variant set of equations. All the equations which remain active constitute the invariant set of equations.
3. We analyze the WHEN statements. According to the current values of the variables on which each WHEN statement depends, we determine which of its CASEs applies. The equations stated within such a CASE are set as active.
4. All the variables incident in the active set of equations are active. The current problem to be solved consists of the active set of equations and variables.

Figure 4 shows the application of the previous steps to the example of the fluid flow transition. The mechanism outline above is independent of a particular solver or solution algorithm. However, we must emphasize that a change in the configuration of the problem during an iterative solution process may also cause a change in the partitioning of the variables and equations. Therefore, any solver using a partitioning mechanism must account for such a possibility. Zaher (1995) developed algorithms for simplifying the partitioning of variables and equations in a conditional model.

4.2 CONDITIONAL COMPILATION: THE SELECT STATEMENT

During the process of instantiation of a model, it is necessary to differentiate between those objects that are not instantiated because they are defined within nonmatching CASEs of a SELECT statement from those objects that are not instantiated because of a deficiency in their declarative definition. In order to do that, we define an elementary "dummy" class.

We find that it is necessary to build only one instance of this class, such that the dummy instance becomes a place holder for all the objects defined in all the nonmatching CASEs of the SELECT statements.

Figure 5 shows a graphic explanation of the process of instantiation in conditional compilation. Statements in the nonmatching CASEs of a SELECT statement that do not involve the creation of a new object are interpreted to determine if they are syntactically correct, but the resulting compiler actions are simply NOT executed (*i.e.* assignments, refinements, merging, etc.). On the other hand, statements in the matching CASEs lead to full compilation as if they were defined outside the SELECT statement.

FIGURE 4 Preparing a conditional model to send to a nonlinear solver.

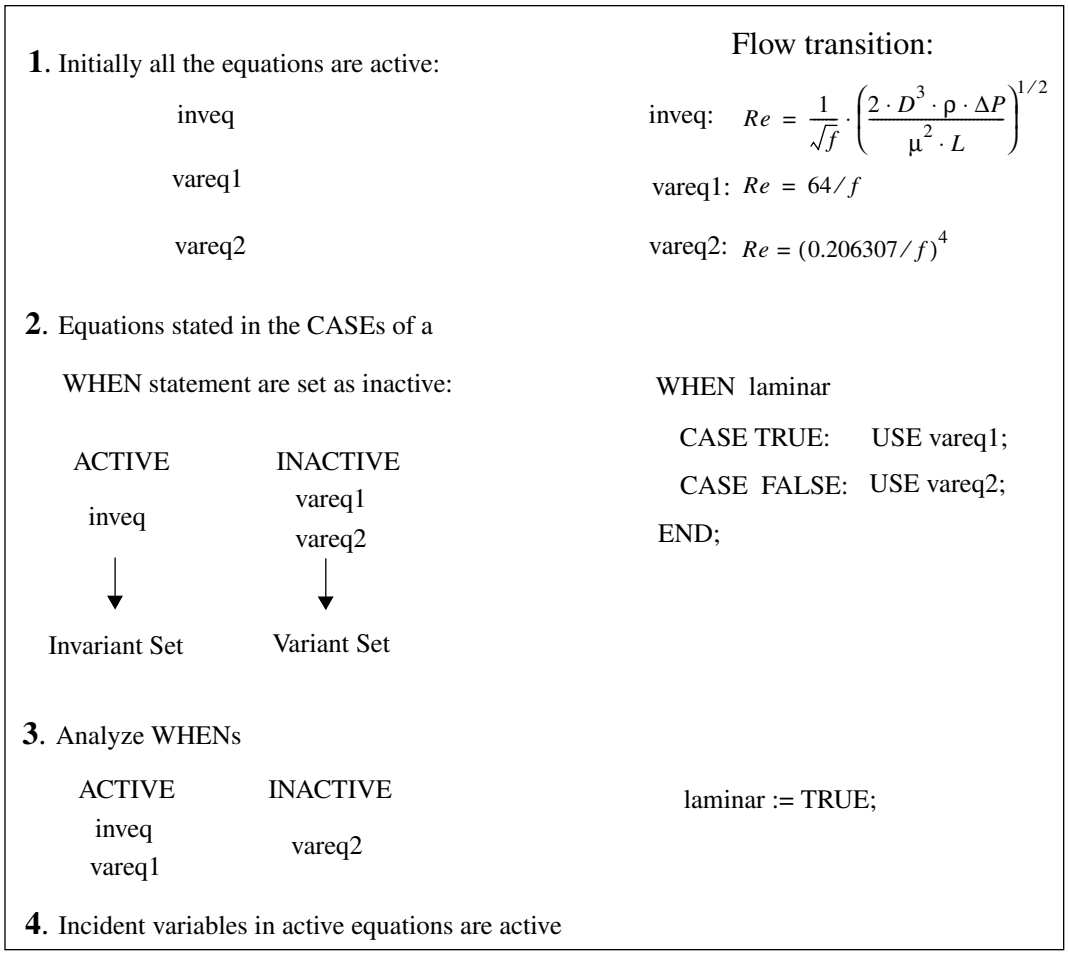
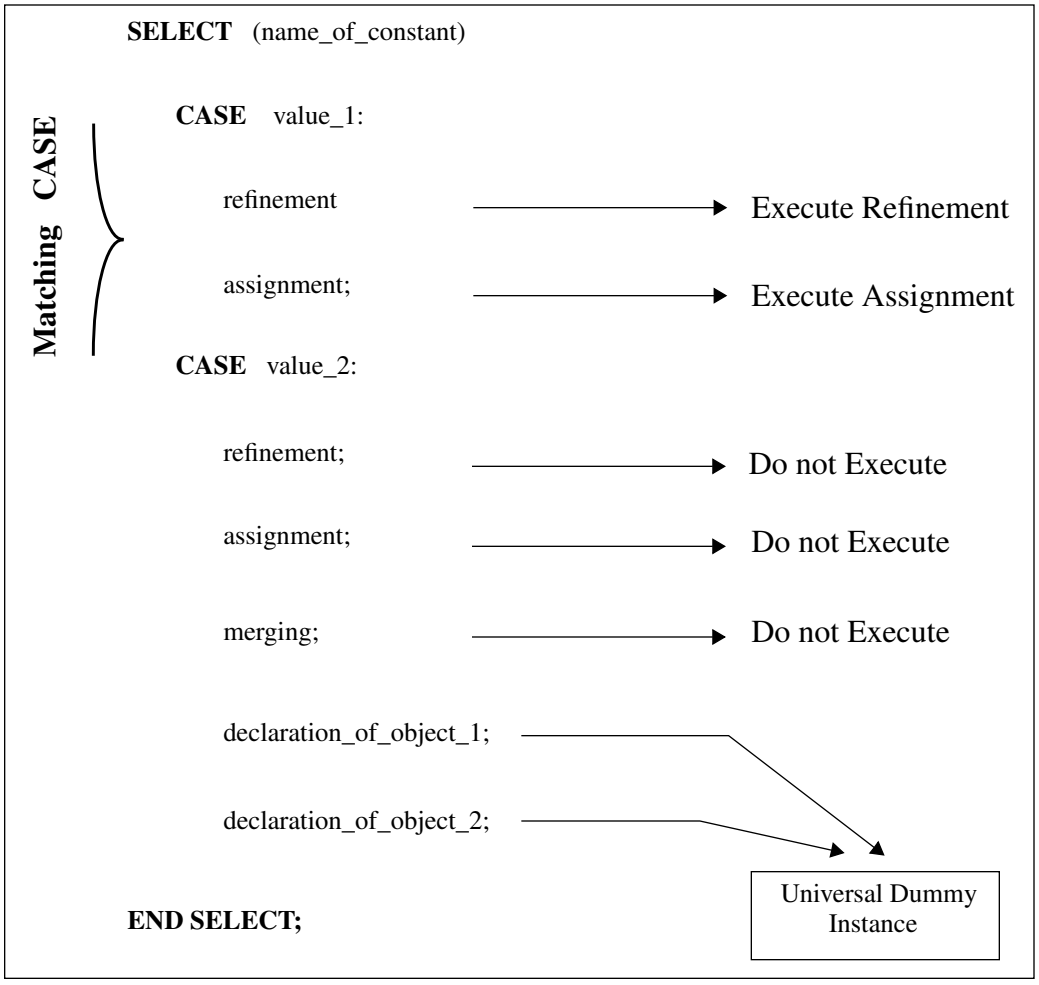


FIGURE 5 Instantiation process in conditional compilation.



5 EXAMPLES OF APPLICATION

One of the goals of our research group has been to improve one's ability to develop and solve process models. The main result of this effort has been the development of ASCEND (Piela *et al*., 1991). ASCEND is both an object-oriented mathematical modeling environment and a solving and debugging engine. Westerberg *et al.* (1994) discuss the essential features of the current ASCEND system and present several ways in which we can improve it to solve larger models and to increase its scope as a modeling environment. We implemented the conditional modeling tools explained

above into the ASCEND modeling environment. We want to emphasize, however, that the proposed modeling tools are general enough so that their practical implementation is feasible in essentially all of the existing object-oriented equation-based modeling languages that we have reviewed. In this section, we show some examples to illustrate the expressiveness of the new modeling capabilities.

5.1 THE WHEN STATEMENT

Figure 6 shows two partial ASCEND models in which the WHEN statement is used, each in a different manner.

In case (a) the value of the symbol 'method' is used to select between two alternative configurations of the problem, a flash calculation assuming constant relative volatility (VLE_flash) or a flash calculation using a more rigorous thermodynamic calculation (td_VLE_flash). Which option is going to be used and, therefore, the value of the symbol 'method', is a user decision. For example, the user may select the simplified model when looking for good initial values for the variables and then switch to the rigorous model simply by changing the value of the symbol 'method.' Once a configuration has been selected, it will be kept unless the user decides to change it. Note that the user does not have to recompile the model to switch. Since the system has compiled the data structure for both of the problems, the user can readily switch back and forth between the simplified model and the more complicated one.

FIGURE 6 Two different applications of the WHEN statement.

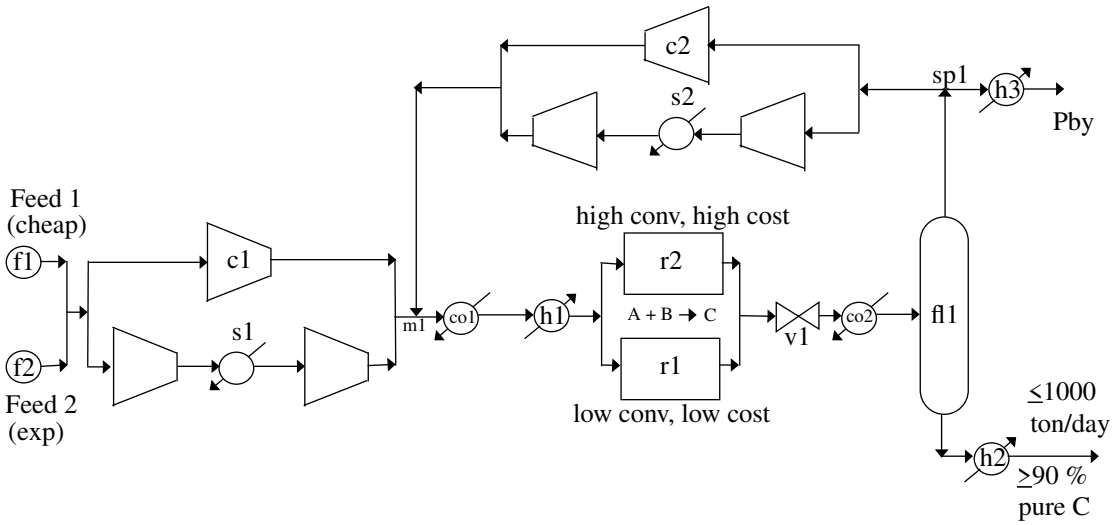
```
      (a)
method             IS_A symbol;
simplified_flash   IS_A VLE_flash;
rigorous_flash     IS_A td_VLE_flash;

WHEN (method)
      CASE 'rigorous':
            USE rigorous_flash;
      CASE 'simplified':
            USE simplified_flash;
END WHEN;
```

```
            (b)
laminar     IS_A boolean;
Re,f        IS_A factor;

eq1: Re = 64/f;
eq2: Re = (0.206307/f)^4;

WHEN (laminar)
      CASE TRUE:
            USE eq1;
      CASE FALSE:
            USE eq2;
END WHEN;
```

On the contrary, in case (b) we cannot expect the boolean value of the variable laminar to be a user decision. Its value will depend on the value of the Reynolds number which is an unknown whose value is being computed in the problem. The value of the variable laminar will be the truth value of the expression Re≤2100. In an iterative solution scheme, we expect the value of the boolean variable 'laminar' to change, and, therefore, so will the structure of the system of equations that we have to solve.

Another example of the application of the WHEN statement is the synthesis of process networks using superstructure optimization. We developed a simplified model for the superstructure given in Figure 7, taken from the work of Turkay and Grossmann (1996). In this example, there are two alternative feedstocks, two possible choices of the reactor and two choices of the compression systems. Hence, there are 4 structural decisions, and, therefore, there are 2^4 = 16 feasible configurations for the problem.

FIGURE 7 Superstructure taken from Turkay and Grossmann (1996).



All 16 configurations are encapsulated in one ASCEND model containing 4 WHEN statements which depend on the value of 4 boolean variables. Figure 8 shows this model.

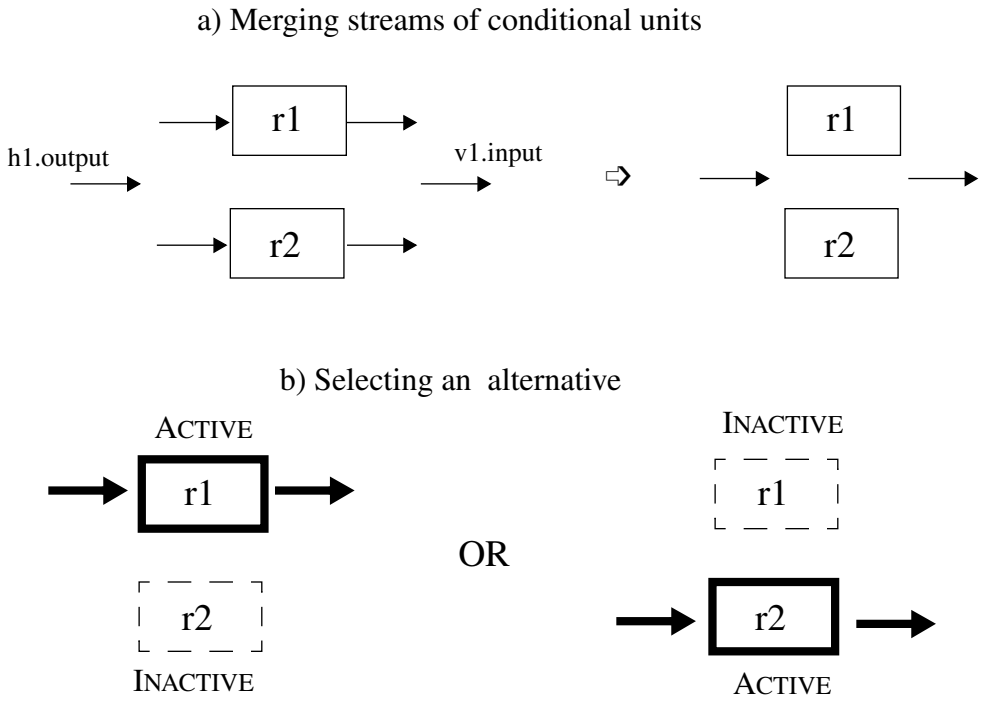
FIGURE 8 ASCEND model for the superstructure of Figure 7

```
MODEL flowsheet;
(* units *)
        f1                                  IS_A cheap_feed;
        f2                                  IS_A expensive_feed;
        c1,c2                               IS_A single_compressor;
        s1,s2                               IS_A staged_compressor;
        r1                                  IS_A cheap_reactor;
        r2                                  IS_A expensive_reactor;
        co1,co2                             IS_A cooler;
        h1,h2,h3                            IS_A heater;
        fl1                                 IS_A flash;
        sp1                                 IS_A splitter;
        m1                                  IS_A mixer;
        v1                                  IS_A expansion_valve;
(* boolean variables  *)
        select_feed1,select_single1         IS_A boolean_var;
        select_cheapr1,select_single2       IS_A boolean_var;
(* define sets  *)
        m1.n_inputs :==2;
        sp1.n_outputs :== 2;
(* wire up flowsheet *)
        f1.stream, f2.stream, c1.input, s1.input            ARE_THE_SAME;
        c1.output, s1.output, m1.feed[2]                    ARE_THE_SAME;
        m1.out,co1.input                                    ARE_THE_SAME;
        co1.output, h1.input                                ARE_THE_SAME;
        h1.output, r1.input, r2.input                       ARE_THE_SAME;
        r1.output, r2.output,v1.input                       ARE_THE_SAME;
        v1.output,co2.input                                 ARE_THE_SAME;
        co2.output, fl1.feed                                ARE_THE_SAME;
        fl1.liq, h2.input                                   ARE_THE_SAME;
        fl1.vap, sp1.feed                                   ARE_THE_SAME;
        sp1.out[1], h3.input                                ARE_THE_SAME;
        sp1.out[2],c2.input, s2.input                       ARE_THE_SAME;
        c2.output, s2.output,m1.feed[1]                     ARE_THE_SAME;
(* Conditional statements *)
        WHEN (select_feed1)
          CASE TRUE:
                USE f1;
          CASE FALSE:
                USE f2;
        END WHEN;
        WHEN (select_single1)
          CASE TRUE:
                USE c1;
          CASE FALSE:
                USE s1;
        END WHEN;
        WHEN (select_cheapr1)
          CASE TRUE:
                USE r1;
          CASE FALSE:
                USE r2;
        END WHEN;
        WHEN (select_single2)
          CASE TRUE:
                USE c2;
          CASE FALSE:
                USE s2;
        END WHEN;
END flowsheet;
```

The procedural section of the model and the model for each unit operation have been omitted for simplicity. Note that, for each structural decision, a WHEN statement allows the selection of either of two alternatives (a cheap reactor or a expensive reactor, for instance); however, the WHEN statements do not allow the selection of both alternatives simultaneously. Such a case would require the definition of splitters (for the input streams of the conditional units) and mixers (for the output streams of the conditional units) which, in the context of an equation-based approach, are generally considered as separate unit operations. Also, note the way in which the linking is done for the conditional units in the flowsheet. Consider, for instance, the case of the selection between an expensive reactor (r2) and a cheap reactor (r1) illustrated in Figure 9(a). The input streams of the two reactors are merged with the output stream of heater 1 (h1). That is, h1.output, r1.input and r2.input are merged (the ARE_THE_SAME operator in the ASCEND modeling language represents a merging operation). In practice, such a merging operation ensures that, regardless of which reactor we select (cheap or expensive), an input stream to the reactor always exists, and it is the same as the output stream from the heater 1. Similarly, a merging operation is also defined among the output streams of the reactors and the input stream to the expansion valve (v1). The combination of this merging of input and output streams with the procedure described in section 4.1.1 allows us to select a particular configuration of the flowsheet, as illustrated in Figure 9(b).

We have tested the mechanism to pass the correct submodel to a solver suggested in section 4.1.1 on several problems. As an example, we applied it to the system presented in Figure 7 and Figure 8. The value of the four boolean variables determine the structure of the problem. The values of the boolean variables can be defined interactively by the user, but they also can be defined by some logic inference algorithm which would allow the automatic change of the structure of the problem. In our model of the superstructure shown in Figure 7, the nonlinear system contains 137 invariant equations and 68 variant equations for a total of 205. The configuration defined by one of the feeds, two single-stage compressors and one of the reactors contains 169 equations, the 137 invariant and 32 of the 68 variant equations. Switching from one structure to another is done without the need to recompile and, since reconfiguring the system requires only rebuilding several list of pointers, it is computationally very efficient.

FIGURE 9 Linking of conditional units.



5.2 LOGICAL RELATIONS

Figure 10 illustrates two examples of the syntax we have used for the implementation of logical relations. Example (a) is simply to show the use of the logical operators AND, OR and NOT. Example (b) is a more complete version of the model presented in Figure 6. In this case we note that the value of the boolean variable ('laminar') depends on the truth value of a real expression (cond). In this way, since the WHEN statement depends on the boolean variable 'laminar', a change in the truth value of the real expression triggers a change in the configuration of the system.

5.3 THE SELECT STATEMENT

Figure 11 shows an ASCEND model which is similar to that shown previously in Figure 6(a). The difference is that we use the SELECT statement rather than the WHEN statement. This time, the

symbol 'method' is a constant, and, once it is defined, its value will not change. That value will always be a user decision. In the example of Figure 11, only the list of statements in the first CASE are compiled.

FIGURE 10 Examples of the syntax for the implementation of logical relations.

(a)

```
valve_open        IS_A boolean;
pump_on           IS_A boolean;
full_tank         IS_A boolean;

valve_open<==>pump_on AND
              NOT(full_tank);
```

(b)

```
laminar     IS_A boolean;
Re,f,k      IS_A factor;

CONDITION
      cond: Re <= 2100;
END;

laminar <==> SATISFIED(cond,1e-08);

eq1: Re = 64/f;
eq2: Re = (0.206307/f)^4;
WHEN (laminar)
      CASE TRUE:
            USE eq1;
      CASE FALSE:
            USE eq2;
END WHEN;
```

FIGURE 11 Application of the SELECT statement.

```
method            IS_A symbol_constant;
method :== 'rigorous';

SELECT (method)
      CASE 'rigorous':
            rigorous_flash    IS_A td_VLE_flash;
      CASE 'simplified':
            simplified_flash  IS_A VLE_flash;
END SELECT;
```

5.4 THE SWITCH STATEMENT

The use of the SWITCH statement for the conditional execution of procedural code is illustrated in Figure 12. In that example, the average value of the constant relative volatility is set to 1.5 if the simplified flash model is used. If the rigorous model is used, then a procedure defining the initial values and degrees of freedom for an adiabatic operation is executed.

FIGURE 12 Application of the SWITCH statement.

```
METHODS
      METHOD values;
            RUN reset;
            SWITCH (method)
                  CASE 'rigorous':
                        RUN adiabatic;
                  CASE 'simplified':
                        ave_alpha := 1.5;
            END SWITCH;
      END values;
```

6 SUMMARY

We developed efficient modeling tools for the representation and solving of conditional models in an equation-based environment and discussed the details of their practical implementation. The incorporation of these tools into the ASCEND environment shows how the expressiveness of an equation-based modeling language increases with this extension. Even though the new modeling capabilities are independent of any numerical technique used for the solution of a conditional model, they were developed in such a way that the needs of all of the existing solution algorithms we have found in the literature will be met. Some of the contributions of this work that we want to highlight here are:

- The object-oriented definition of the WHEN statement, which allows the conditional selection among objects; object which themselves may constitute complex models.
- The equation-based formalism that we give to our incorporation of logical relations into the

modeling language.

- The development of a tool for the incorporation of conditional compilation into an equation-based modeling language. To our knowledge, this capability is provided only by the GAMS modeling language and some specific computer applications, such as the "make" command (Stallman and McGrath, 1993).

7 ACKNOWLEDGMENTS

This work has been supported by the Engineering Design Research Center at Carnegie Mellon University (NSF Grant No. EEC-8943164), by the Computer Aided Process Design (CAPD) Consortium, and by the "Consejo Nacional de Ciencia y Tecnología (CONACYT)".

8 REFERENCES

Abbott, K. A.; Very Large Scale Modeling. Ph. D. Thesis. Department of Chemical Engineering Carnegie Mellon University, Pittsburgh, PA, 1996.

Allan, B. A.; A More Reusable Modeling System; Ph.D. thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA,1997.

Barton, P. I.; The Modeling and Simulation of Combined Discrete/Continuous Processes. Ph.D. thesis, Department of Chemical Engineering, Imperial College of Science, Technology and Medicine, 1992.

Barton, P. I. and Pantelides, C. C.; Modeling of Combined Discrete/Continuous Processes. *AIChE Journal*, 40(6):966–979, June 1994.

Bogusch, R. and Marquardt, W.; A Formal Representation of Process Model Equations. *Comput. Chem. Eng*., 19:S211-216, 1995.

Boston, J. F., Britt, H. I. and Tayyabkhan, M. T.; Software: Tackling Tougher Tasks. *Chem. Eng. Progr*. Nov., 38-49, 1996.

Brooke, A., Kendrick, D., Meeraus, A. and Raman, R.; GAMS - Language Guide. GAMS Development Corporation, 1997.

Elmqvist, H., Cellier, F.E. and Otter, M.; Object-Oriented Modeling of Hybrid Systems. ESS'93 European Simulation Symp., Delft, The Netherlands, 25-28, 1993.

Feehery, W. F. and Barton, P. I.; A Differentiation-Based Approach to Dynamic Simulation and Optimization with High Index Differential-Algebraic Equations. *Computational Differentiation*, M. Berz, C. Bischof, G.Corliss, and A. Griewank Editors, SIAM, 1996.

Epperly, T. G.; Implementation of an Ascend Interpreter. Technical Report. Engineering Design Research Center. Carnegie Mellon University. Pittsburgh, PA, 1988.

Grossmann, I. E. and Turkay, M.; Solution of Algebraic Systems of Disjunctive Equations. *Comput. Chem. Eng.*, 20:S339–44, 1996. Suppl. Part A.

Marquardt, W.; Trends in Computer-Aided Process Modeling. *Comput. Chem. Eng.*, 20(6):591-609, 1996.

Mattsson, S, E., Andersson, M.; OMOLA - An Object Oriented Modeling Language. *Recent Advances in Computer Aided Control Engineering*. Elsevier, 291-310, Amsterdam, 1993.

Pantelides, C. C.; SPEEDUP-Recent Advances in Process Simulation. *Comput. Chem. Eng.*, 12(7):745–755, 1988.

Pantelides, C. C. and Barton, P. I.; Equation-Oriented Dynamic Simulation: Current Status and Future Perspectives. *Comput. Chem. Eng.*, 17S:263–285, 1993.

Pantelides, C. C., Britt, H. I.; Multipurpose Process Modeling Environments. *FOCAPD'94*, Snowmass, CO., 1994.

Piela, P., Epperly, T., Westerberg, K., Westerberg, A. W.; An Object-Oriented Computer Environment for Modeling and Analysis: The modeling language. *Comput. Chem. Eng.*, 15(1):53–72, 1991.

Piela, P.; ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis. PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, April 1989.

Raman, R. and Grossmann, I. E.; Symbolic Integration of Logic in Mixed-Integer Linear Programming Techniques for Process Synthesis. *Comput. Chem. Eng.*, 17(9):909–927, 1993.

Raman, R. and Grossmann, I. E.; Modeling and Computational Techniques for Logic Based Integer Programming. *Comput. Chem. Eng.*, 18(7):563–578, 1994.

Stallman, R. and McGrath, R; GNU Make Manual, Version 3.69, November, 1993.

Stephanopoulos, G., Henning, H., and Leone, H.; MODEL.LA. A Language for Process Engineering, Parts I and II. *Comput. Chem. Eng.*, 14, 813-869, 1990.

Turkay, M. and Grossmann, I. E.; Logic-Based MINLP Algorithms for the Optimal Synthesis of Process Networks. Comput. Chem. Eng., 20(8):959–978, 1996.

Westerberg, A.W., Abbott, K. A., and Allan, B. A.; Plans for ASCEND IV: Our Next Generation Equational-Based Modeling Environment. Boston, Massachusetts, November 1994. AspenWorld'94.

Zaher, J. J.; Conditional Modeling. PhD thesis, Department of Chemical Engineering,Carnegie

Mellon University, Pittsburgh, Pennsylvania, May 1995.

Zaher, J. J.; Conditional Modeling in an Equation-Based Modeling Environment. The Annual AIChE National Meeting, 1991. paper 138c.

Zaher, J. J.; Conditional Programming. The Annual AIChE National Meeting, March 1993.