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.

here, but we are in the way of discovering them.

8 REFERENCES

Barton, P. I.; The Modeling and Simulation of Combined Discrete/Continuous Processes. PhD 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.

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.

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.

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.

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.

```
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 (select_single1)
          CASE TRUE:
                USE c1;
          CASE FALSE:
                USE s1;
        END;
        WHEN (select_cheapr1)
          CASE TRUE:
                USE r1;
          CASE FALSE:
                USE r2;
        END;
        WHEN (select_single2)
          CASE TRUE:
                USE c2;
          CASE FALSE:
                USE s2;
        END;
END flowsheet;
```

FIGURE 8 ASCEND model for the superstructure of Figure 6

modeling tools required to accomplish this, showing how the expressiveness of the modeling language increases with their incorporation. The computer implementation of these tools has been completed. There are surely other applications of the tools we describe that are being overlooked

All these 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. The procedural section of the model and the model for each unit operation has been omitted for simplicity.

The mechanism suggested in section 4.2 has been implemented simultaneously with the implementation of the WHEN statement and the logical relations. Its performance has been tested in several small problems. As an example, it was applied to the system presented in Figure 7 and Figure 8. The value of the four boolean variables will determine the structure of the problem to be solved. As mentioned above, the values of the boolean variables could be defined interactively by the user, but they also could be defined by some logic inference algorithm which would allow the automatic change of the structure of the problem. The NLP 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 active equations out of the 68 variant equations. Switching form one structure to another is done without the need of recompilation and, since to reconfigure the system requires only rebuilding several list of pointers, it is computationally very efficient.

6 FUTURE WORK

The WHEN statement implementation described here help us with the representation of conditional models, but, how about the solution of the same kind of problems?. Work is currently being focused in the implementation of a solver which can deal with logic inference and automatic switching among alternative configurations. Also, in the same way that we use the WHEN and the SELECT statements in the declarative section of our modeling language, a SWITCH statement will be implemented for the procedural section of the language. It will allow us the conditional execution of procedures, working as an extension to the current IF statement. Further details will appear very soon with respect to these topics.

7 CONCLUSIONS

There is a contribution to be made in the modeling world. Namely, the efficient representation of conditional models in an equation-based environment. We have described the

objects defined in all the nonmatching CASEs of the SELECT statements. The type of a dummy instance is also called a dummy type, which basically contains only an empty list of declarative statements. The list of parents of the dummy type is set to NULL, because this list would provide no useful information and would have the potential of having a very large number of elements. What we keep instead is only a counting of the number of parents, such that we can keep the sanity of the deletion process. Statements in the nonmatching CASEs of the SELECT statement that do not involve the creation of a new object, are simply NOT executed (*i.e.* assignments, refinements, merging, etc.). On the other hand, statements in the matching CASEs are executed as if they were defined outside the SELECT statement. Figure 6 shows a graphic explanation of the process of instantiation in conditional compilation.

5 AN EXAMPLE OF APPLICATION

The WHEN statement, the SELECT statement and logical relations have already been incorporated to the ASCEND system as they were outlined above. We are currently testing this initial implementation, looking for improvements that could enhance its efficiency as modeling tools. To give an example of the application of the WHEN statement, 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. Therefore, there are $2^4 = 16$ feasible configurations of the problem.

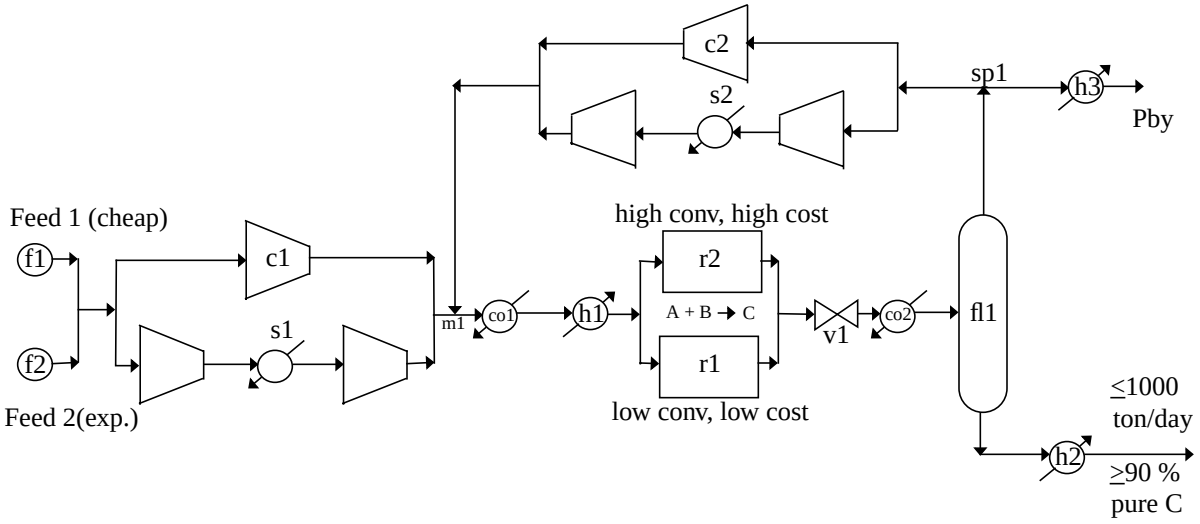

FIGURE 7 Superstructure taken from Turkay and Grossmann (1996).

The mechanism outline above is independent of a particular solver or solution algorithm. Computationally speaking, to set a relation as active or inactive implies a simple bit operation. Figure 5 shows the application of the previous steps to the example of the fluid flow transition.

4.3 CONDITIONAL COMPILATION: THE SELECT STATEMENT

During the execution of a SELECT statement, we must be able to identify if a name has not already been created because it is included in a nonmatching CASE of the SELECT statement, or because the compiler still do not execute its defining statements. In order to accomplish this, we create a UNIVERSAL dummy instance.This dummy instance is just a place holder for all the

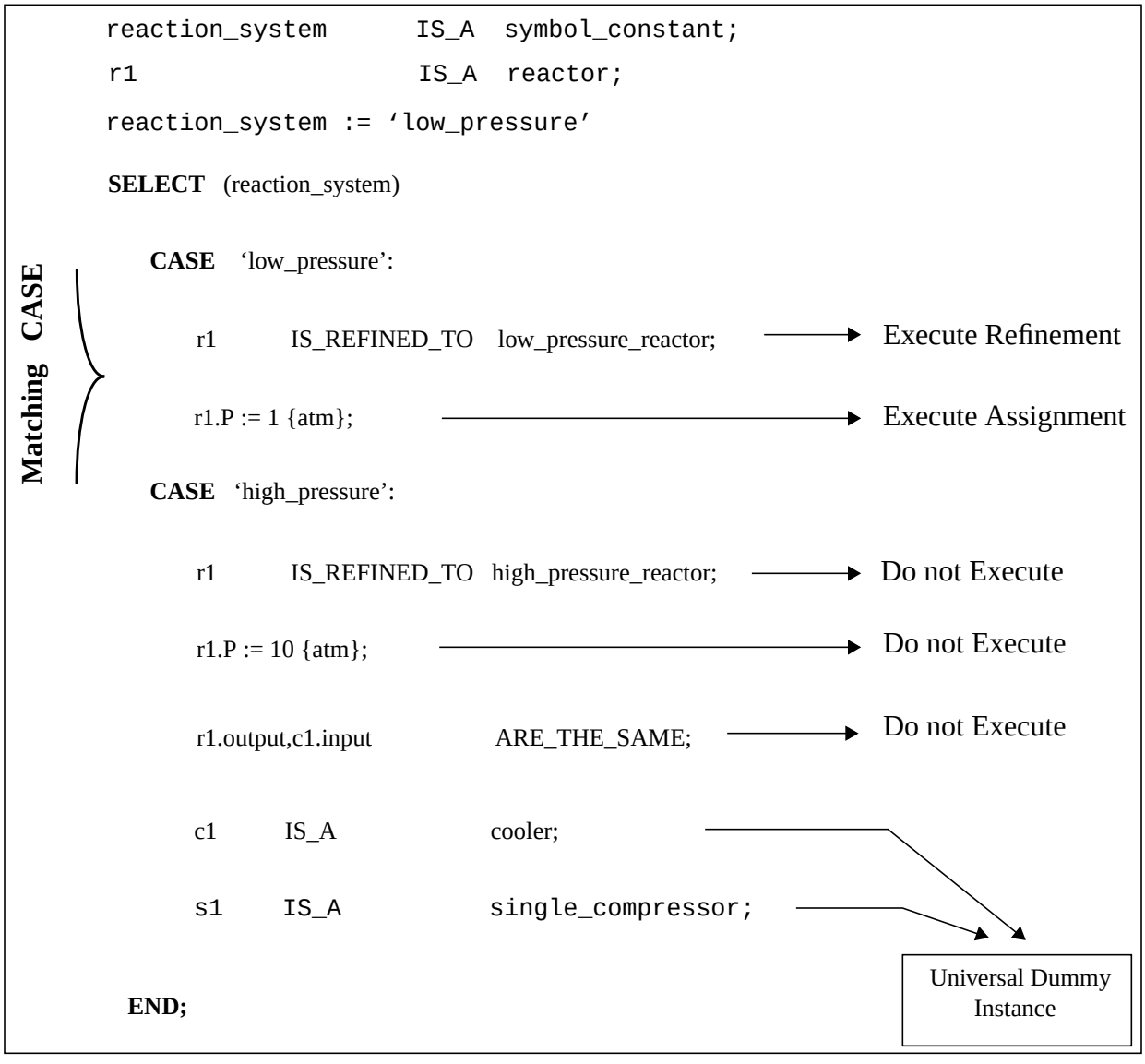

FIGURE 6 Instantiation process in conditional compilation

provide a mechanism to select the structure of the problem:

1. Initially, all the equations resulting from the compilation are considered as active.
2. Then, all the equations which are implicitly or explicitly (inside models) stated in the CASE of a WHEN statement are set as inactive. 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. Analyze the WHEN statements. According to the current values of the variables on which each WHEN statement depends, determine which of its CASEs applies. The equations stated implicitly or explicitly in 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.

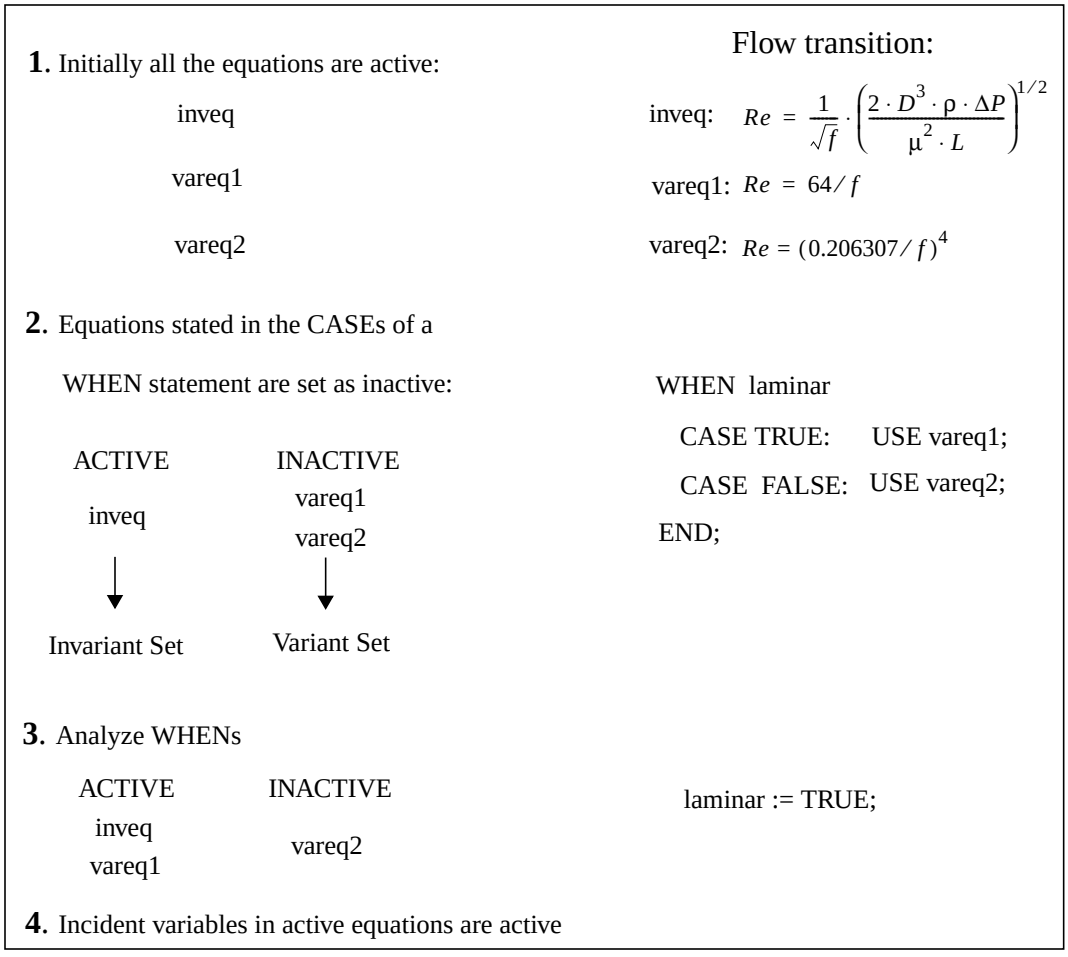

FIGURE 5 Feeding a conditional model to a NLP solver

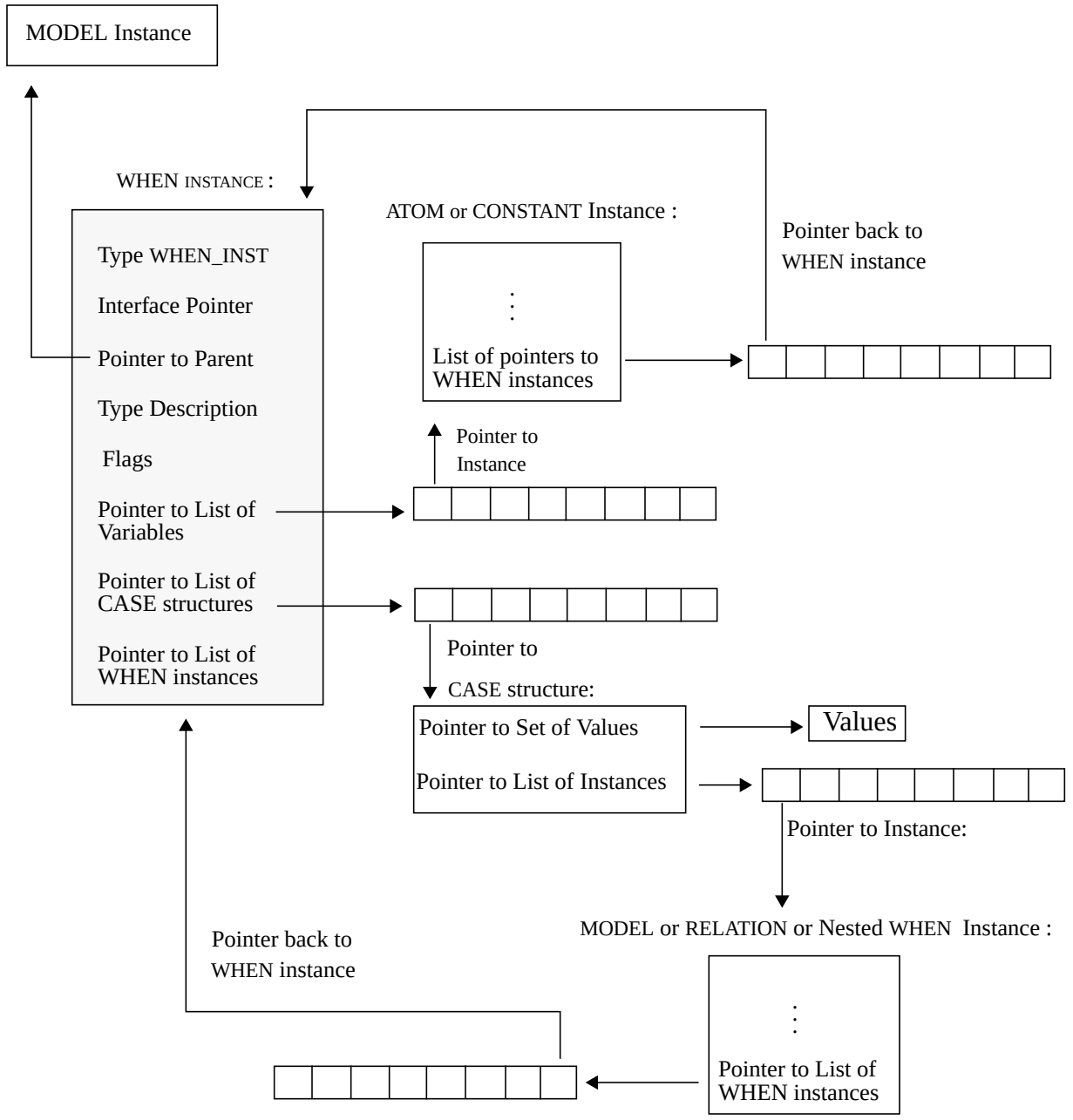

FIGURE 4 WHEN instance implementation

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". The following steps would

4 DETAILS OF THE IMPLEMENTATION

4.1 THE WHEN INSTANCE

The implementation of the ASCEND III interpreter was described by Epperly (1988). Regarding conditional modeling, he proposed to build a complete instance tree for each CASE within the WHEN statement. However, he also recognizes the combinatorial nature of that approach that makes it unacceptable: for example, for a type containing two WHEN statements each having three CASEs, nine complete instance trees would be created.

In this work, the definition of a WHEN type and a WHEN instance allows us to create a single instance tree in which all the structural alternatives are embedded. Figure 4 shows this implementation. A WHEN instance has no children and have only one parent, a MODEL instance. Basically, a WHEN instance is constituted by two list of pointers: a list of pointers to atom or constant instances on which the WHEN statement depends, and a list of pointers to CASE structures. At the same time, each CASE structure contains a list of values and another list of pointers to instances. The instances pointed for the CASE correspond to the instances (relation, arrays of relations, models, array of models) which will be “active” if the list of values of the CASE matches the current values of the conditional variables. Because of the refinement and merging operations, the instances pointed for the a WHEN instance have to be able to point back to the WHEN instance. Therefore, all these instance types (atoms, constants, models, relations) have also a list of pointers to WHEN instances.

With this implementation, the data structures required for all the alternative configurations are available (*i. e.* all the models or relation in each CASE are compiled), however, 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. In the next section, the steps used to determine the current configuration of the problem are described.

4.2 FEEDING A CONDITIONAL MODEL TO AN NLP SOLVER

The implementation of the WHEN statement in ASCEND would allow us to have available the data structure for all the variables and equations of the conditional model. Therefore, the first

However, the goal of the SELECT statement is completely different from that of the WHEN statement: the SELECT statement is used to express conditional compilation. The following are important observations:

1. The selection of the statements to be executed is computed in terms of a ordered list (note that the list is enclosed in parentheses) of constant integer, booleans or symbols.
2. Any declarative statement is allowed in the list of statements of each CASE, including WHEN and nested SELECT statements.

Also, two main limitations should be identified:

1 The same name cannot be created in more than one CASE. We suggest instead that a name should be defined outside the SELECT statement and refined differently in each CASE.

2 SELECT statements are not allowed inside a FOR loop.

Figure 3 shows a similar ASCEND model from that shown previously in Figure 1(b), the difference is that we use the SELECT statement rather than the WHEN 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;
```

FIGURE 3 Application of the SELECT statement.

This time, the symbol method is a constant, once it is defined, its value will not change. Obviously, that value will always be a user decision. In the example of Figure 3, only the list of statements in the first CASE will be executed, *i.e.*, only the model rigorous_flash will be created. To emphasize, the SELECT statement provides the capability of conditional compilation. An application of this capability may be the selection of the thermodynamic model to be used for equilibrium calculations: Wilson or UNIFAC or NRTL, etc.

separate the two logical terms of the relation. In each of the two terms, logical operators such as AND, OR, and NOT are allowed. Equality in a logical equation can also be interpreted as an *if and only if* implication between two logical terms expressed in clausal form. 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. Example (a) is a more complete version of the model presented in Figure 1 In this case it can be noted that the value of a boolean variable (laminar) depends on the truth value of a real expression (condition). In such a case, the real expression is defined and labeled inside the CONDITIONAL statement and then the logical operator SATISFIED gives the truth value of the expression. That is, for example, if the truth value of the real expression is TRUE, the result of SATISFIED(label of expression) will be TRUE. The objective of the CONDITIONAL statement is threefold: 1) it makes the life of the compiler easier, since the logical expressions are decoupled from the expressions on which they depend, avoiding relations containing implicit relations, 2) the use of this syntax forces the user to define a boolean variable (laminar in the example) which will have the same truth value as the real expression. That is useful for browsing purposes and, finally, 3) It is a very simple way of saying that the relations included in the statement are not going to be solved, they are used only as expressions with a truth value associated to them.

It is important to emphasize that, by checking the value of the appropriate boolean variables after each iteration, the incorporation of logical relations will allow, if required, automatic change of the structure of the problem in an iterative solution scheme.

3.3 THE SELECT STATEMENT

The syntax for the SELECT statement is very similar to that described for the WHEN statement:

```
SELECT (list_of_constants)
      CASE list_of_values_1:
                              list1_of_declarative_statements;
      CASE list_of_values_2:
                              list2_of_declarative_statements;
      CASE list_of_values_nminus1:
                              listnminus1_of_declarative_statements;
     OTHERWISE:
                              listn_of_declarative_statements;
END;
```

3.2 INCORPORATION OF LOGICAL RELATIONS

Motivated by some recent applications of logic in the solution of conditional models (Raman and Grossmann, 1994; Turkay and Grossmann, 1996) and trying to expand the ASCEND capabilities for modeling, an implementation of logical relations as a modeling tool is presented. To our knowledge, no general purpose equation-based modeling environment provides this capability.

In the implementation of logical relations, the equation based approach is maintained. 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. Therefore, this incorporation also means that we need to provide a solver which knows how to deal with logical relations. 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 (CNF or DNF).

(a)

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

CONDITIONAL
      condition: Re <= 2100;
END;

laminar == SATISFIED(condition);

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

(b)

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

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

FIGURE 2 Examples of the syntax for the implementation of logical relations

Figure 2 illustrates two examples of the syntax proposed for the implementation of logical relations. The symbol double equality will indicate that we have a logical relation and will

understood. Figure 1 shows two incomplete ASCEND models in which the WHEN statement is used, each in a different manner.

(a)

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

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

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

(b)

```
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;
```

FIGURE 1 Two different applications of the WHEN statement.

In case (b) the value of the symbol method is used only 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). The method that is going to be used and, therefore, the value of the symbol method, is a user decision. We can expect that the user will select the simplified model looking for good initial values of the variables and then he will switch to the rigorous model by changing the value of the symbol method. Once a configuration has been select, it will be kept unless the user decides to change it. Note that the user does not have to recompile the model to make the switching, since the data structure for both of the problems is available and, therefore, the reconfiguration of the system can be done automatically by simply changing the value of the conditional variable. An example of this application would be the synthesis of process networks.

On the other hand, in case (a) 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 in the problem. Actually, the value of the variable laminar will be the truth value of the expression $Re \leq 2100$. Here we have a conditional model. In an iterative solution scheme, we will expect that the value of the boolean variable laminar will change, and so will the structure of the system of equations that we have to solve.

statement is:

```
WHEN (list_of_variables)
      CASE list_of_values_1:
                              USE name_of_equation_1;
                              USE name_of_model_1;
      CASE list_of_values_2:
                              USE name_of_equation_2;
                              USE name_of_model_2;
      CASE list_of_values_nminus1:
                              USE name_of_equation_nminu1;
                              USE name_of_model_nminus1;
     OTHERWISE:
                              USE name_of_equation_n;
                              USE name_of_model_n;
END;
```

The following are important observations about the implementation:

1 The WHEN statement does not mean conditional compilation. We create and have available the data structures for all of the variables and equations in each of the models. This is actually a requirement for the solution algorithms of conditional models. All the models and equations whose name is given in each of the cases should be declared inside the model which contains the WHEN statement. It is important to recognize this feature, since conditional compilation in ASCEND IV is explained later with the description of the SELECT statement.

2 The variables in the list of variables can be of any type among boolean, integer or symbol or any combination of them. That is, we are not limited to the use of boolean variables. Obviously, The list of values in each case must be in agreement with the list of variables in the number of elements and type of each element. In other words, order matters in the list of variables of the WHEN statement, and parentheses are enclosing this list to make clear such a feature.

3 Names of arrays of models or equations are also allowed inside the scope of each case.

The described extension (allowing the user to define the domain of validity of both models and equations inside the cases of a WHEN statement) enormously increases the scope of modeling in an equation based modeling environment. Of course, this implementation may be possible in other modeling environments which support object oriented concepts, but we have proposed it as an extension to the ASCEND modeling language because ASCEND is the best one locally

respectively. Multiple states may be described by nesting several IF statements.

Similarly, Barton (1992) and Barton and Pantelides (1994) incorporated the CASE equation to *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.

There is a major difficulty in both of the previous approaches as tools for conditional modeling. They only allow the substitution of a list of equations (or arrays of equations) for another. In an equation-based modeling environment in which concepts like hierarchy (building complex models from small models) and inheritance (adding new features to a model) are constantly in use, that represents a severe disadvantage in efficiency. Thinking of how many equations have to be written down when substituting some of the units of a superstructure makes obvious the lack of flexibility in the scope of these approaches.

3 CONTRIBUTIONS TO MODELING

One of the goals of our research group has been to improve the ability to develop and solve process models. The main result of this effort has been the development of ASCEND (Piela *et al.*, 1991). 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. In this section, we describe the tools which will allow ASCEND to represent conditional models efficiently: the WHEN statement and logical relations. Also, conditional compilation is incorporated with the implementation of the SELECT statement, to our knowledge, for the first time in an equation-based modeling environment.

3.1 THE WHEN STATEMENT

Originally, the syntax for incorporating conditional dependence of some equations of the model in an equation-based environment was suggested by Piela (1989). That syntax is very similar to the CASE equation of *gPROMS* and suffers the same limitations. Instead, we want to take advantage of the fact that ASCEND is based on object oriented concepts where model definitions can contain parts that contain parts to any level. Furthermore, in ASCEND, a simple relation is treated as an object by itself and can have a name. Based on these ideas, the syntax for the WHEN

$$
\begin{gathered}
g(x) = 0 \\
\bigvee_{i \in D_k} \begin{bmatrix} h_{ik}(x) = 0 \\ r_{ik}(x) \leq 0 \end{bmatrix} \quad k \in K \\
x \in R^n
\end{gathered}
\tag{1}
$$

where $g(x)$ and $h_{ik}(x)$ represent 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 term in each disjunction D_k.

While many currently available equation-based modeling systems have been reported in the literature, only a few of them have given attention to conditional models (Pantelides, 1988; Piela,1989; Barton, 1992). In this work, we describe the incorporation of a series of tools which enables the user of an equation-based modeling environment with the capability of representing complex conditional models. This representation is intended to be independent of any particular application, or of any solver or algorithm used for finding a solution to the system of equations. Potential applications of this capability varies 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).

2 BACKGROUND

Previous implementations of conditional statements in an equation-based modeling environment have been reported. One such mechanism is the IF-THEN-ELSE construct of *SpeedUp* described by Pantelides (1988):

```
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

Conditional Modeling in ASCEND IV

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

Technical Report
Engineering Design Research Center
Carnegie Mellon University, Pittsburgh, PA 15213

Abstract: Modeling tools for the efficient representation of conditional models in an equation-based environment are described in this paper. The practical implementation of these tools has been incorporated to the ASCEND system. Examples are presented to show their scope of application.

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

1 INTRODUCTION

In recent years, much attention has been focused on tools for the formal definition of models describing the behavior of process systems (Pantelides and Barton, 1993). At the lowest level, process models are represented by a large set of variables and a large system of linear and/or nonlinear equations that related them. This paper focuses in the representation of conditional models. A conditional model 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. Zaher(1993), and Grossmann and Turkay (1996), show that a conditional model can be represented as the system of disjunctive equations: