An Introduction to ASCEND: Its Language and Interactive Environment

P. Piela, R. McKelvey, A. Westerberg

EDRC 06-111-91

An Introduction to ASCEND: Its Language and Interactive Environment

Peter Piela, Roy McKelvey, Arthur Westerberg
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA. 15213

Abstract

Recently there has been a growing realization among researchers and practioners that current technologies do not adequately support mathematical modeling "in the large." In this paper we discuss a technology called ASCEND, which addresses this issue. We describe two aspects of the technology: a modeling language and an interactive model-building environment. The ASCEND language is structured, declarative, and strongly-typed and incorporates object-oriented extensions. The interactive environment is based on the notion of a concurrent set of tools which reflect the various phases of ASCEND modeling. These tools do not enforce a strict sequence of operations, but rather have been designed to support the flexible access implied by declaratively specified models. We claim that ASCEND offers solutions to several of the issues raised by Arthur Geqffrion and use categories introduced by him to frame this discussion.

1.0 Introduction

In a paper summarizing plenary addresses given at the IFORS 87 conference in Buenos Aires and the 1988 Canadian Operations Research Society Meeting, Arthur Geoffirion[9] addresses the shortcomings of current computer-based modeling environments. As an impetus to correcting this situation, he proposes five characteristics that should be found in any system attempting to support the full spectrum of modeling activity. He then steps back from these characteristics and discusses the three main design challenges that stand in the way of realizing such systems. In his concluding remarks, he calls for the reader to consider these issues in light of their modeling environments and to begin work on closing the gaps between his admittedly ideal system and their own.

In this paper, we take up this challenge. We are particularly motivated by the fact that the system we are developing—an equational modeling environment called ASCEND (Advanced System for (imputations in Engineering Design)—already possesses many of Geoffrion's required features and seems a promising platform in which to address many of the others. Our aim is to discuss the ASCEND system—its modeling language and interactive environment—within the Geoffrion framework, and when appropriate, discuss how the ASCEND paradigm suggests alternative approaches to modeling systems. Further, we hope that the reader, after completing the paper, will have a good sense of the current ASCEND implementation and its use.

This paper is organized as follows: Section 2 describes the dominant themes of the ASCEND approach; Section 3 describes the ASCEND language in detail and Section 4 analyzes the language with respect to Geoffrion's points; Sections 5 and 6 similarly discuss the details of the interactive ASCEND environment and their relation to Geoffrion's ideal.

2.0 The ASCEND approach

In cooperation with a group of academics and industrialists we have developed a protoype model description language and a computer system through which users can create and interact with models defined in the language. These tools form the basis of an experimental program in which observation of users solving problems and subsequent discussions are used to refine and evaluate the underlying technology. The stated goal for the ASCEND project is to create an environment in which engineers are able to produce complex equational models involving thousands of equations much more rapidly than possible with existing technology. The ASCEND system reflects certain hypotheses about how best to support large-scale mathematical modeling. These we will discuss in some detail.

2.1 Models should be highly structured.

ASCEND is a structured approach to developing and solving equational models. By astmctured approach, we mean that the user is able to define groupings of equations and variables called models, and manipulate these models using a set of language-defined operators.

Today, a majority of equational modeling is done with unstructured languages such as GAMS[4], AMPL[7], and LPL[13], which are based on algebraic notation. Although the use of these languages has led to significant improvements in productivity, we believe that the lack of a capability for defining and managing abstract data models significantly limits the complexity of tasks that can practically be attempted. This view is shared by Muhanna and Pick[16] who write

"Present modeling tools do not support combining of models. This is a serious deficiency. By using existing models as "building blocks" for new composite models, the new model is developed with less effort than would be necessary if it were built from scratch. Furthermore, this enables a kind of "structured" model building in that small models may be independently built and debugged and then used as components in larger mocfels. Traditiorally models are developed (often from scratch) as stand-alone entities. As a ©suit, model integration through direct model-to-model linkage is tedious and error-prone."

In the following sections we further support the need for a structured approach to mathematical modeling by giving a brief overview of three issues that we consider to be particularly important in the development of complex models. They are: hierarchical decomposition, evolutionary modeling, and debugging. In each, the model builder must be able to manipulate individual parts of a linked model structure.

Hierarchical(kcompositihn—Omtxpencnct[l9] and that of other workers in a number of disciplines[3, 16, 21, 25] suggest a need to support the building of hierarchically organized networks of equations. In chemical engineering, Westerberg and Benjamin[27] write "Complex models are almost always built in a hierarchical fashion. An example is a distillation column which is built up of trays, flash units, splitters, mixers, heat exchangers, pumps etc. A flash unit is in fact a hierarchical structure." In their work on synthesis of electric circuits, Sussman and Steele[24] propose a language for describing hierarchical constraint networks in which compound models are defined in terms of existing parts. They write, "In this way we can build arbitrarily complicated compound objects in a hierarchical manner. The hierarchy allows the complexity at one level to be limited." In operations research, Geoffirion[10] demonstrates that a transhipment model can be hierarchically decomposed into two transportation models with a set of constraints that define resource limitations for the warehouses. Also, Muhanna and Pick[16] contend that an effective model management system must provide support for modular and hierarchical model development, and have demonstrated a strategy fa* accomplishing this in their model development language MDL[15].

Evolutionary modeling—Model evolution has been advocated as an efficient approach to solving large problems. For example, in arguing against a batch oriented approach, Locke and Westerberg[14] write that "large attempts fail more often than not," and suggest that "a more efficient approach is to solve the large problem in stages, beginning with a few pieces of equipment and working up to a complete flowsheet"

Another type of evolution occurs when a model builder first describes his or her problem in terms of simplified models which are robust and converge quickly. Based on these calculations the model builder selectively specializes certain models to more rigorous representations, and resolves the problem with the values generated by the simplified models as initial guesses. Locke and Westerberg[14] associate this style of modeling with movement along an axis of "model complexity."

Modeling can also involve moving along an axis of "computational control"[14]. For example, a chemical engineer might initialize the flowsheet shown in Figure 1 by guessing the recycle stream (S4), and separately solve the units MIX, REACT, ami STILL in a sequence such that the outputs from one unit become the inputs to the next The engineer can then alter the degrees of freedom (i.e., which variables are specified, and which are computed), and solve the entire flowsheet simultaneously.

Debugging—Although Muhanna refers to the benefits of developing independently debugged models, model instances also need to be debugged during solving. In such situations we have found it helpful to be able to pick a troublesome part, and work on it (e.g., re-scale variables and equations) in isdation. This kind of debugging often involves working with a sequence of parts as the problem is traced back to its cause. Having corrected the problem, the model builder is able to solve the complete instance structure.

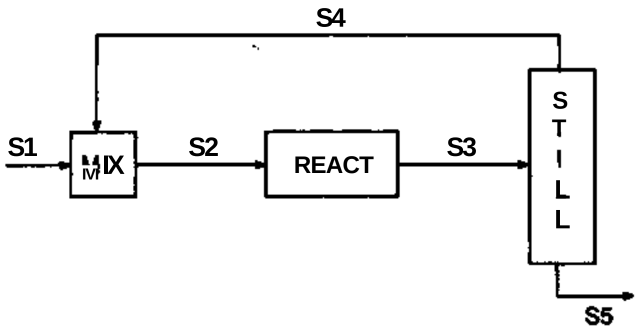


Figure 1. A simple flowsheet with a recycle stream.

22 Modeling is an inherently interactive process*

Based on our statements about evolutionary modeling and debugging, we believe that the model builder should have the choice of working interactively at any point in the modeling process. This includes both formulation of the model, and interrogation and manipulation of the model instance.

23 Simulation **is best represented** as **a** system of constraints

At present, a majority of simulation in engineering design is done *with parametric* systems. In these systems, equations are specified and ordered for procedural computation. Although this approach is useful for routine problems that have established solution procedures and do not change frequently, non-routine problems require the model builder to re write and reorder the model equations.

In response to the need for more flexible systems, there has been a growing interest in declarative equation-based techniques. Problem solving with equational models involves specifying the values of certain variables and computing the others with an independent solver. An importantfeatureofthis approach is that it allows users to examine different scenarios with one model structure by simply changing which variables are specified, and which are computed

For example, in chemical engineering process simulation there are two major types of calculation: *simulation,* and *design.* Simulation is considered to be the easier of the two calculations, and requires the user to specify all input streams and equipment parameters (e.g., size). The solver will then compute the remaining intermediate and output streams. A design calculation is more or less the inverse; the userspecifies variables in the output streams, and the solver computes either equipment parameters or input streams.

Tlie design calculation is difficult for a number of reasons including the possibility of making a specification on an output stream that is physically impossible to achieve. Locke and Westerberg[14] suggest that the correct way to approach a design problem is to start out with a sequence of simulation calculations which generate converged solutions approaching the desired solution, and then switch to the design calculation by altering which variables arefixed,andwhichare computed.

3.0 The ASCEND language

What follows is a brief overview of what we consider to be some of the most important attributes of the ASCEND language^], and how they relate to the issues raised in the [devious section. ASCEND was originally designed to support thedeclarativeand structured specificationoflarge systems of equations that arise in engineering design. The language builds on concepts used in object-oriented programming and conventional strongly-typed languages such as Pascal. We will discuss the ASCEND language using the benchmark examples of Geoffrion which include the transportation and forecasting models. (Model definitions can be fotnd in Appendices A, B, andC.)

3.1 Information hiding

There is no information hiding in ASCEND. One can gain access to any part of any model using qualified names (path). Fa* example, in the transportation model shown in Appendix A, the total cost of shipping product from plant i is accessed using the namep[i].ShipmentCost Note that qualified naming eliminates the problem of syntactically ambiguous references which may result from name clashes within two separately defined components. However, it does not completely address the broader (semantic) issue of unique name violations that need to be resolved during model integration 1].

3.2 Operators

The language has only five operators: REFINES, IS_A, IS_REFINED_TO, AREJTHEJSAME, and ARE_ALIKE which, we conjecture, simplifies learning. REFINES implements monotonic inheritance, IS_A implements incorporadonJS_REFINED_TO implements refinementofmodelparts, ARE_THE_SAME implements a way of recursively equivalencing structured objects [3,25], and ARE.ALIKE implements grouping of objects over which structural variations are to be made. Further discussion of these operators will be offered as they relate to features discussed below.

33 Arrays

Arrays of variables, relations, and models are indexed over sets of integers or symbols (or refinements of these). The contentsofthese index sets can be fixed during array declaration or computed as apart of theproblem formulation. For example, in the transportation model presented in appendix A, the set of plants from which customer i is to receiveproduct, is computed from the listofcustomersspecifiedforeachplant,c[i].plantld := [j IN plantld I i IN p(j].customerld].

3.4 Dimensional consistency

ASCEND provides dimensional checking (e.g., mass/time) foraUrdatioris (equations, inequalities, and objecdvefunctk)ns). Thus, dimensional inconsistencies among variables in an equation are readily detected. Without automatic checking of dimensionality, such errors are generally very difficult to find. Once the dimensionality of a variable is known, its value may be assigned or displayed in any set of compatible units (for example, tomes/year). In the case of the tansportationmodel, the type definition for the variable "flow" is given by:

```
ATOM flow REFINES solver_var
    DIMENSION M/T
    DEFAULT 1000 {tonnes/year};

    nominal := 1000 {tonnes/year};
END flow;
```

It should be noted that every numeric value in an ASCEND model has an associated dimensionality that is impliedby (1) a units specification(e.g., 1000 tomeVyear, 55 miles/hr), (2) a type definition (e.g., "f IS_A flow" implies that f has dimensionality mas$ftime),or (3) propagation of dimensionality through relations.

In addition, the user can define his or her own measurement units in terms of the fundamental units associated with each dimension, or any previously defined derived units. For example,

```
UNITS
        mol:=kmol/1000;
        g  := kg/1000;
        lb := kg/2.20462;
        N  :=kg*m/s^2;
        J  :=N*m;
END;
```

3.5 Object-based features

ASCEND is based on object-oriented concepts which allow models to be structured more like the systems they are meant to represent We conjecture that such decompositions are generally easier for users to understand. This argument is analogous to the suggested preference for object-oriented databases over relational databases in engineering applications.

An example of this style of modeling is given for the transportation problem (shown below) which is composed of a set of plants (p[plantld]), and a set of customers (c[customerld]). Each plant is itself a structured object containing product flow (f[customerld]) to a set of customers (custcmerld). The following sections briefly illustrate the object-oriented features of ASCEND.

```
MODEL plant;
        sup IS_A supply_capacity;
        customerld IS_A set OF integer,
        maxCustomer IS_A integer,

        CARD(customerId)<= maxGistomer,

        ffcustomerid], totalFlow IS_A flow;
        cost[customerld] IS_A unitCost;
        totalFlow=SUM(f[customerId])
        totalFlow <= sup;

        shipmentCost IS_A cost;
        shipmentCost= SUM[f[i]^1|tost[i] I i IN customerld);
END plant;

MODEL transportation;
        plantld, customerld IS_A set OF integer,
        pfplantld] IS_A plant;
        c[customerld] IS_A customer,
END transportation;
```

3.5.1 Inheritance and part refinement

Inheritance is supported through the REFINES operator. It promotes reusability and organization through the building of inheritance hierarchies, and provides a mechanism for type checking.

For example, in the integrated transportation/forecasting model shown in Appendix C, a customer_forecast model has been defined which locally inherits the attributes of the customer model, and is further specialized by adding an instance of a forecasting model and a relation which specifies that the demand (dem) will be computed using die forecasting model.

The transportation with forecasting model (trans_forecast) is then defined as a refinement of the basic transportation model with twoadditk)nalconstraints.Theseconstraints specify that the set of customers defined in the basic transportation model will be "refined" to customers whose demand will be predicted by a forecasting model (c[customerld] IS_REHNED_TO customerjbrecast), and that for each customer, demand will be predicted using an exponential forecast(c[custcmerl(l] J7 ISJIEFINEDJTO expForecast).

This refinement of parts, supported through the IS JREFINED_TO operator, permits evolutionary modeling and improves the possibilities of model reuse. An example of part refinement is shown above where the structure of a customerforecast is refined to anexponentialforecastPossible refinements are defined by the structure of the inheritance hierarchies, and the refinement process is validated by the language compiler.

3.5.2 Merging

The recursive merging of structured objects is supported through the ARE_THE_S AME operator. This facility is used to connect complex models by selecting parts (connectors) within models through which the connection is realized, and making these parts equal. Merging several connectors together results in a single equational structure that can be referenced by all naming schemes defined by the connectors. For example, the intent of making the statement p[i].f[j], customer(j].f[i] AREJTHEJS AME is that the numeric value of the flow of product from plant i to customer j is equal to the value of the flow that customer j receives from plant i. This could have been written p[i].f[j] = customer[j]i[i]; however, this would needlessly create an extra equation, and maintain a duplicate copy of the flowvariable. By using ARE_THE_SAME, no equation is created. The reduction in resources achieved by using AREJTHE_SAME is especially important in engineering applications where connectors may contain several hundred equations.

3.5.3 Grouping

Propagation of structural variations is supported through the ARE_ALIKE operator. For example, in the trans_forecast model one could write the statement c[customerId].F ARE_ALIKE which expresses the intent that all customers will use the same type of forecasting model. A structural change made to any individual forecasting model will automatically propagate to the others.

3.5.4 Strong typing

Strong typing, which requires one to indicate the type of every part in every model, reduces the debugging effort (during solving) for complex models. The base type of apart is declared using the IS_A construct Also, the type system provides a mechanism by which the user can define legitimate ways in which parts can be merged together. For example, in the case of the inheritance hierarchy shown in Figure 2, it wouklbeinvalkl to attempt tomergeaninstanceofUquid_stream with an instance of vapor_stream because the liquid.stream and vapor.stream models are net conformable. (Two models are said to be conformable if one is the ancestor of the other.) Errors that might arise in an attempt to make such a connection are detected by the language compiler. It should be noted that that statements can be incrementally compiled.

3*5.5 Procedures

ASCEND models can optionally contain procedures. These are used to compute initial values, and set degrees of freedom (e.g., the assignment xl.fixed := FALSE states that the value of the variable can be assigned by a solver). Several alternate procedures might concurrently exist (e.g., procedure init_example28a and init_example28b) which can be be invoked selectively by the model builder prior to solution. A complete description of the procedural language is outside the sc»peofthispq)er.However,asshownbebw,pitx»duiesaieable to invoke other procedures defined locally or within visible parts.

```
MODEL example28;
    xl, x2 IS_A unscaled.variable;
    xl*x2-l = 0;
    xl*xl + x2*x2-3 = 0;
    INITIALIZATION
    PROCEDURE assign.bounds;
        xl .lower_bound := 0;
        xl.upper_bound := 4.0;
        x2.1ower_bound := 0;
        x2.upper_bound := 4.0;
        xl.fixed:= FALSE;
        x2.fixed:= FALSE;
    ENDassignJxxinds;

    PROCEDURE init_example28a;
        RUN assignjxninds;
        xl:=2;
        x2:=2;
    END init_example28a;

    PROCEDURE init_example28b;
        RUN assign_bounds;
        xl:=4;
        x2:=2;
    END init_example28b;
ENDexample28;
```

4.0 A discussion of the ASCEND language

In this section we explicitly relate characteristics of the ASCEND modeling language to thecharacteristicsanddesign implications outlined by Geoffrion[9]. In some cases we directly evaluate the ASCEND language by a Geoffrion ideal, in others we question or modify the premise embodied by his ideal. We begin by focusing on the notion of "executability" proposed by Geoffrion as a necessary attribute of a flexible modeling environment

4.1 What is meant by executable?

Geoffrion writes/'theadjective 'executable' refers to functions that programs in the modeling environment should be able to perform upon receiving a model written in an executable modeling language." If one reads the previous statement literally, ASCEND is not an executable language. At present, the only ASCEND program that reads model descriptions is a compiler, which takesamodeldescriptionandgeneratesadata structure which can be interrogated using a set of procedures that we provide. External programs such as graphers, solvers, andspreadsheetsareintegratedintotheenvironmentbywriting softwarebridges thatallow values in an ASCEND datastructure to be accessed by the external program in a format that it requires, and vice versa. This approach has several benefits, (1) a single bridge can be written that will work with all models written in the ASCEND language, (2) the model builder composes and revises models using only the modeling language, the solver input is automatically regenerated by the bridge, (3) the external programs can be used "as is'* without any internal modifications, and (4) a single bridge can be constructed for a family cf programs (e.g., an MPS file generator).

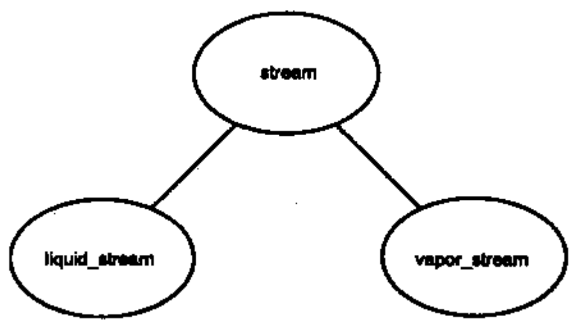


Figure 2. An Inheritance hierarchy for describing material streams.

4.2 Can one language support all users?

Geoffrion writes that the modeling language should be "sufficiently natural that non-modeling professionals can understand it with only a modest amount of training."[9] Our experience suggests that this may not be achievable. An extensive discussion of our views on this topic can be found in [20].

Our research responds to criticisms of languages such as GAMS being too low-level and too inflexible for solving real-world problems[5]. However, it should be noted that the use of structured modeling languages is currently outside the experience of many model builders.

For the past two years we have been working with academic and industrial users in an attempt to understand, evaluate, and refine the hypotheses underlying ASCEND modeling. In interviews conducted with off-campus users, the following two themes were recurrent. First, unfamiliarity with object-oriented concepts contained in the ASCEND language caused difficulties, and second, no precedent existed for taking a structured approach to the formulation of equational models (as opposed to flat lists cf algebraic specifications).

We suggest that rather than trying to make the modeling language intuitive for all users, designers should develop adequate support structires (e.g., help systems, coaching, worked examples) for different skill levels and requirements.

4.3 Evolutionary modeling

Geoffrion writes, "Flexibility is important because few modeling professionals ever get a model or model-based system 100% right the first time. Even if by some miracle they do, the requirements usually change over time and thus will soon induce the need for revision. In any case, evolution will be necessary for genuine excellence." We agree, and the current ASCEND language supports model evolution in two related ways. First, there is model inheritance which allows the model builder to define a model that locally inherits the entire structure (variables/elations, procedures, and default valics) from a single parent model. He or she can then add statements to the new model. This type of inheritance organizes models hierarchically. Second, there is part refinement which allows a model buUder to change the type of a part of a modeL The part can only be refined to a member of the set of models which inherit from the current model or any of its descendents. By adopting a strictly monotonic view of inheritance we are able to guarantee that refinement of parts will yield well-formed model structures.

4,4 Declarative and procedural

Geoffrion writes that for a modeling language to be understandable and natural it should be "declarative rather than procedural and highly mnemonic rather than cryptic." While we agree that a declarative representation is natural for equational modeling, we have decided to include procedural notions in the definition of models. An ASCEND model is divided into two sections both of which are optional. The first contains declarative statements which are used to specify the equational structure of the model. The second contains a set of procedures written in a small imperative language, that are used to compute initial values of variables, to specify which variables are fixed and which are to be computed. Whereas other modeling systems only provide mechanisms for importing externally computed values we believe that the knowledge encoded in procedures should be an explicit part of a model formulation.

4.5 A common modeling language

Geoffrion writes "in a true modeling environment, there should be a *lingua franca* (common language) for model formulation that is very broadly applicable and not biased toward any particular problem domain, or solver technology." We have adopted this approach in the development of ASCEND, and have worked with users to develop model libraries in several domains. These include, chemical engineering [22], geometric reasoning in architectural design [30], mathematics, physics, and operations research.

4.6 Consistency checking

Geoffrion writes that "an executable modeling language should be able to perform extensive consistency checking."[9] We have dealt with this issue through the use of strong typing. One of the major rationales for a strongly typed language is the problem of providing good diagnositic information in the event of solver failure[18]. Adequate diagnostic information is difficult to provide because the mathematical decomposition employed by solvers is usually different from the physical decomposition favored by model builders.

Given the difficulty in debugging during solving, we decided that the ASCEND language should be strongly typed, with the aim that problem specifications submitted to a solver should accurately reflect a user's intent both in terms of values and structure.

ASCEND's type system enables the compiler to detect errors like trying to connect (merge), group, or refine incompatible parts. By making dimensionality an explicit part of the declaration of an ATOM (variable) we are able to report equations which are dimensionally inconsistent, and to validate numeric assignments made to variables. We also use the type system to define which objects an external program can operate on. For example, plotting programs will only extract data from instances of the "plot" model or any of its refinements.

5.0 The ASCEND Environment

We now turn our attention to the interactive interface to the ASCEND system. What follows is an overview of the basic

Rgure 3. **The** toolbox Is used to control the visibility of toolkits on the desktop.

tools provided for the user to analyze and solve simulations.

Once a model has been specified with the ASCEND language, instances of those models arc displayed, solved, and evolved through an interactive graphic interface. The interface uses the metaphors of a "toolbox" (figure3) and "desktop" (figure 4). The toolbox is a permanert area of the screen vhich contains buttons symbolizing available toolkits, and buttons which organize the interface. The desktop occupies the remainder of the screen and contains tool kits currently in use. Underlying the ASCEND system is a database that stores both model definitions and any instances of model definitions created through the interface (simulations). Each tool kit implementsasemantically different view of the problem being examined (e.g., source code, structural, mathematical, etc.) and these views arc maintained concurrently with the underlying database. That is, a change made in one toolkit is immediately reflected in the others.

Our experience with ASCEND has shown that multiple views arc required to support complex problem solving, and this has been suggested by other workers in the area of mathematical modeling (e.g.,[ll]). In keeping with a direct manipulation paradigm, the user is able to share information generated in one tool by exporting references to objects within that tool directly into others. It should be noted that, unlike a conventional clipboard," onlyrcferencestoobjectsarepassed and not the data within the object There is only onecopy of any piece of data stored in the database. Following is a brief descriptionoflhecurrentlyimplemented toolkits—the Library, Sims, Browser, Solver, Probe, Units, Display, and Script

Tools in the Library Tool Kit arc used to create, view and manipulate the inheritance hierarchies in which model definitions arc organized These hierarchies arc created by loading modeldefinitionsftom text files. After loading,the user selects one of the models in the library to be compiled into a database of equations and variables called a simulation. A number of different simulations can co-exist; each is listed in the tool kit labeled Sims. Once created, a simulation can be "played with" in many ways by the other tools in the system.

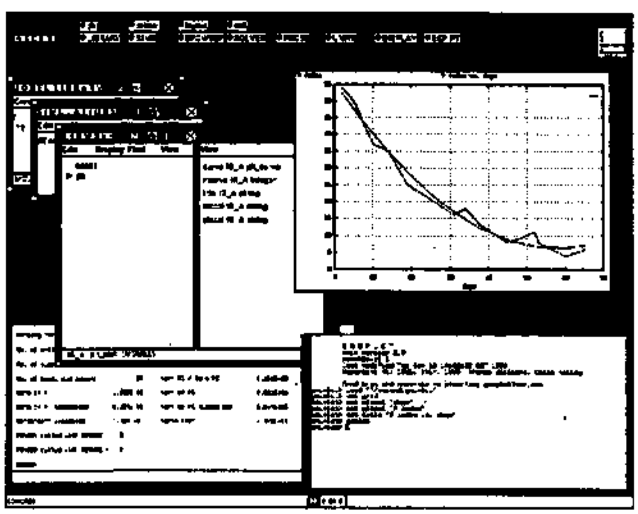
The Browser tools arc used to select objects of interest within a simulation either by incremental navigation or direct query. Other tools perform operations on these objects: for example, displaying attributes in order to verify structure, creation of new parts within the objects, and refinement of the objects in an evolutionary modeling process.

Since complex models are created by merging together parts using the ARE_THE_SAME operator, many parts of a simulation will have alternate names. One of the tools allows the user to display all the names for a part and to pick one of these as the current focus. Another tool in the Browser allows procedures defined in the INITIALIZE section to beexecuted.

The primary functions of the Solver arc to apply a chosen algorithm to the solution of the system of relations defined by the object it is viewing (the current d>ject), and to assist in tte investigation of failures that occur during solving. The current object is continually analyzed to see if it forms a well-posed problem. If it does not, the user can return to the Browser and reset some of the variable flags to indicate that some of them arctobefixedratherthancomputedIfthese flags are contained in the current object, ASCEND will immediately reanalyze and report the consequences. An effort to solve the system of equations defined by the current object can be attempted even if it is not "square." For a system of equations which has more variables than equations, our solver SLV will arbitrarily select someofthevariablesasfixedandsolvefortheremainingones.

One tool in the Solver is a debugger where one can display the incidence matrix fa the equations (rows of the matrix) versus the variables (columns) in the problem. Solving can be done by single-stepping or by executing until a maximum number of iterations or a time limit is exceeded. At present, the user can select any one of the following solving packages which is compatible with the current object. Only compatible selections are actively displayed to the user.

• SLV[28,29] is our own solver for solving *n* nonlinear algebraic equations in *n* unknowns. It is based on a modified



Rgure 4. A typical desktop configuration.

Marquardt method[26]. The variables can have bounds specified for them, which will cause the solver to restrict its search for solutions within the bounds. SLV partitions the nonlinear equations and solves the partitions in a precedence ordering.

• MINOS-Augmented[17] is a nonlinear optimization code capable of handling several thousand equality and inequality constraints. It is available from Stanford University.

• SQP is a sequential quadratic programming solver available from L. T. Bieger (Chemical Engineering, Carnegie Mellon University). The current implementation is a dense version and more appropriate for snail problems (on the order of one hundred constraints).

• LSODE[12], as used in the ASCEND system, is for solving dynamic models which involve a mixture of ordinary differential equations and algebraic equations. It integrates the model over time or space from a known initial condition. It is available from the Lawrence Livermore National Labs.

The Probe provides the user with the capability of forming collections of variables, equations, or complex parts that are of interest from disparate locations in a simulation, and to monitor their values during solving. The Probe contains tools which allow the user to detect whether any variables listed in it are poorly scaled or near one of their bounds.

The Units tool kit allows the user to specify the measurement units in which the values of variables are displayed. The user can define sets of units which can be saved in text files for later reuse.

The Script can be used in two ways. First, it can read a set of instructions from a text file specifying a sequence of actions to be taken by the system (e.g., read a model definition file, create a simulation, solve a simulation, plot a graph). The user can choose which instructions are executed. During execution, the interface is animated as if the user were actually pressing the buttons. Second, the script can be used to record commands invoked through the interface which can be written to a text file for later replay. We intend the Script to be both a convenience for expert users and an aid in teaching new users about the system.

In addition to the tool kits described above, there are number of support tools which can be invoked through the interface. For example, objects can be viewed and manipulated using a Unix spreadsheet program, plotted using a number of x-y and x-y-z plotting programs, and used to create high quality reports (e.g., equipment specification sheets) using Postscript templates generated by standard drawing programs or word processors.

6.0 A Discussion of the ASCEND environment

Geoffrion's discussion of system design issues focuses on choices of representation, language issues, system components, and the attributes of an ideal system. Little detail is provided concerning specific interface design, or issues of usability. In the following section we explore some of these questions in the context of our work on ASCEND.

6.1 Designing the ASCEND system: methodology

Before we discuss the implications of the ASCEND interactive environment as an artifact, it is important to review the methods by whkh it has come about The interface to ASCEND was developed using a iterative design approach. Our process is closely aligned with what has become known as Participatory Design[2,6], an approach to system development which emphasizes close and continuous interaction between developers and users, and techniques of rapid prototyping. An in-depth discussion of this design methodology and our interpretation of it is the subject of another paper [20]. What follows is a summary of some of this paper's major points.

The ASCEND project's primary focus is to investigate whether a design system based on a structured, declarative modeling language, and a supporting environment in which to work with the models that result, can improve modeling speed, reduce errors, expand the complexity of problems attempted, and support significant rates of model re-use. While we believed that the underlying technology had the potential to achieve these aims, we had no good way of verifying progress on these complex issues without directly capturing the experience of our intended users as they attempted to solve real problems. To this end, the ASCEND environment was created, not as an embodiment of how its developers *expected* the system to be used; but rather as an experimental apparatus to test the feasibility of the ASCEND paradigm and to provide input into its further development

Because our primary interests centered on what people could accomplish with an environment like ASCEND, we felt that this information could best be assessed through *situated* use. By *situated* we mean problem-solving in the user's workplace with the user's own problems. This is in contrast to the more common practice of evaluating a system by examining its performance on a standard set of example problems in a contrived experimental setting. Further, because our primary aim was to extend the boundaries of existing modeling practice, we consciously designed the system to support advanced modeling practice.

To study situated use, a method must be established for determining which aspects of a user's experience actually verify or contradict a project's basic hypotheses. It also implies that clear criteria for judging the relative success or failure of an encounter with the technology can be determined. We have employed three intertwined sources of data to analyze user performance. The first comes from *opinions:* our own, and those volunteered by users in informal discussions and in taped interviews. The second source comes from *observation* of people as they worked on problems, either in real-time or by use of videotape. The final source of information comes from studying the *outcomes* of modeling efforts—the partial or complete solutions toawiderangeofmodeling tasks. This last set of data gives us a **clear** sense of what types of problems model builders expect the system to handle, what fraction of the system's features are typically employed, whether there was anyre-useofcode and to what degree, where problems are typically encountered, etc.

The system has been under continual evaluation and evolution for the last three years. Its users have come from a wide range of academic dsciplines (chemical engineering, operations research, physics, architecture) and also include a set of industrial users. The development team has consisted of a faculty member oftheChemicalEngineeringDepartment who is expert in the area of mathematical modeling, a researcher whose thesis woik was directly tied to the project, two representatives ofthe Design DepartmentwithexpCTience in human factors, graphic design and user-interface issues, an expert in document design and on-line help systems, and two undergraduate programmers.

62 Interface Design Issues

Here, we focus on the particular design issues that have emerged from the development process described above. We isolate five basic features of the ASCEND environment and discuss their derivation, their implementation, and when possible, their effect on actual problem-solving behavior. These features are listed here, and are dealt with individually in subsequent sections. They are:

1. Ahighdegreeofintegrationandbehavioralconsistency among tools;
2. Supportforflexibleinteractionamongmodelingphases;
3. Support for arbitrarily fine access to models, instances, equations and variables;
4. Support for user-configurability of system organization and behavior;
5. Domain-independence;

62.1 Tool Design **and** Integration

ASCEND modeling can be conceptualized as a set of several distinct activities. These are: model formulation (coding), loading of models into the system, model instantiation, browsing and selection of instance structures, solution, and display of results. Through our analysisof system use, we have determined that these activities can vary widely in frequency, sequence, and duration. Further, these variances depend on both the type of problem being attempted, and on people's modeling sty le. As we began to evolve interactive mechanisms to support ASCEND's various modeling phases, it became quiteclear that each suggested a different view of the data with its own set of supporting operators. For example, browsing of instance structures requires some view of that structure and a series of operators which provide means for navigation through it The Solver, on the other hand, should display characteristics of the problem in terms of numbers of equations and variables, and provide operators to assist in bringing the model to convergence.

Developingasystem of this complexity isachallenge in its own right, however in the case of ASCEND, the development is complicated by the existence of a well established work practice which does not necessarily map directly to the new approach. We relied on our experience, and discussions with other experienced model builders to arrive at tool definitions which could clarify the differences between the new and the old.

In early manifestations of the ASCEND interface, we attempted to integrate all phases of modeling into a single-window environment with a static display and a large set of loosely organized commands. As we observed people using this environment and attempted to refine it, several problems kept cropping up. For example, certain operations seemed to belong to several of the modeling phases, but with a slightly different semantic—this left us with thechoiceofproducingan interface with many "modes", or creating operators with marginally different functionality and artificially different names. In our observations of users, it was clear that during a typical modeling session, it was desirable to have access to the information produced in one modelingphase while working in another. For example, it is common for a model builder to engage in browsing the instance structure while attempting to bring a simulations convergence. (This is only one of many examples.) The need to see many types of information, coupled with the large size of these information structures (e.g, hundreds of lines of computer code, deeply nested instance structures) created a crisis inmanaging screen real estate.

Our solution to these problems was to adopt the toolbox/toolkit approach as described in section 5. This approach has allowed us to isolate each modelingphase into its own context We define an ASCEND toolkit (see figire 5) as consistirg of three parts: *a frame,* a set of *menus,* and a *view. Aframe,* which defines a toolkit's size and location, includes the toolkit's name, mechanisms for repositioning and re-sizing the toolkit, and access toasetofuser-definableattributeswhich determine its meta-level behavior. For instance, the Browser can be set to display sub-items at a depth greater than one, or it can be set to display objects of a given grain size, such as showing only instances of models, and ignoring specific equations and variables. The vi^w is a display that shows objects of a relevant data-type to die toolkit in a particular format For example, in the Model Library the view shows those models loaded into the system in the form of an inheritance hierarchy. *Themenus* hold all tools whichoperate drectlyon the data elements) cirrently in the view. In creating this abstracted tool definition, we can easily bring a high level of consistency to all modeling phases, and provide wharGeofiBroncallsa^conceptual unity^M[9] to the systemJndiscussk>ns with oiff users, this toolkitapproachhas been cited as greatly reducing the time spent on learning to control the environment

622 Flexible Interaction

Because ASCEND models are declaratively specified and maintained as a dynamic system of constraints, user interaction with the modeling enviionment is similariy non-pfocedural. Although model builders will eventually encounter each of the general modeling steps mentioned in section 6.1, the sequence of different steps is not pre-ordained. For example, once a simulation has been instantiated, they may decide to solve individual parts before addressing the whole, or in debugging a simulation, they may inspect several aspects of the problem in order to make sense of diagnostic information provided by the solving algorithm.

Given the breakdown of functionality into independent toolkits, it is critical that the state of each toolkit be tightly coupled with that of others. We have frequently witnessed users employing multiple toollkits to make decisions, and require that die information within the various views of these toolkits be up-to-date and present a consistent picture of the model database. Tools which maintain this degree of communication are said to bec0ncu/ren/[8]. They are implemented so that any change to the database made by one tool is immediately broadcast to theothers. This feature is important because often while a particular tool may seem to naturally reside within one toolkit from a functional standpoint, the results may be bettter communicated through the view in another toolkit A common example in ASCEND is fixingavariable within the Browser, and seeing its effect on the block structure of the problem within the Solver.

However, projecting a notion of concurrency to our users has been difficult The notion of a set of multiple tools "hovering" over a single model representation is contrary to the more familiar "cut & paste" paradigm presented by many systems. Users often conceptualize that they are moving objects from tool to tool, rather than seeing the tool as a particular lens through which to view a single data structure.

Another aspect of ASCEND modeling that must be accommodated, is support for the user in shifting between the representations in building and solving models. These representations include the model code, the model hierarchies maintained by the Model Library, and the instance structure which results from model compilation. Where we have observed the need for quick reference between representations, we have provided specific functions that optimize this type of interaction. For example, in browsing an instance structure, it is typical to want to view the code which defines a specific object The code description of a model is normally accessed through the Model Library. In order to see it, one would have to locate the model in the inheritance hierarchy and select the "show code" function. To simplify this, we have partially automated this procedure whore a mouse-click on the current object's type indicator (in figure 5, this is the area that reads "IS_A heatjexchanger") will result in focusing the Model Library view on that type definition.

The inability to predict the exact sequence of ASCEND modeling activity makes supporting users difficult, because deviation from some "standard" sequence is not always indicative of trouble. For example, the user might decide to engage the Solver before specifying which variables are to be fixed and which are to be computed. The Solver will report that the state of the system of equations is "underspecified," whereas the desired state is "square.[1]* This may or may not be interpreted as a problem, depending on what the user does next If the next step is to execute a procedure which makes the necessary assignments, then the decision is mainly a matter of style. On the other hand, if the subsequent actions can be determined to constitute floundering, then there may be a genuine problem— i.e., what is a misstep when many alternative steps can justifiably be taken?

623 Flexible data access

The ASCEND approach is predicated on the belief that model builders require access to all parts of a model, down to specific equations and variables. Having decided what is of interest, the user may need to alter the views presented by the toolkits to reflect this interest The system provides various mechanismsforlocatingspecificobjects. These include manual navigation (browsing), search by name, and search by model type.

Once located, objects can be incorporated into toolkit views in a number of ways. For example, the Probe allows the model

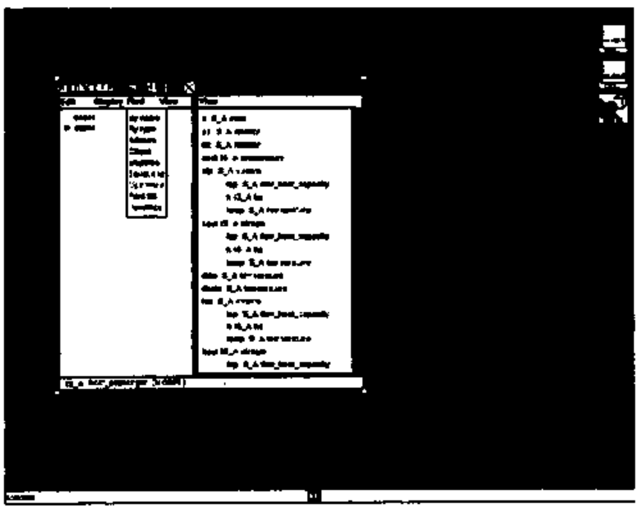


Figure 5. The Browser exemplifies the design of a prototypical ASCEND tool.

builder to create arbitrary lists of objects from disparate locations in the problem structure. This toolkit has been used extensively, and has undergone several revisions. Although it was originally conceived to support the passive observation of variables and their values, it has proved to be a convenient place to locate certain tools for analyzing the problem data. These include tods that check whether variables are properly scaled, or jammed against their bounds.

6.2.4 User-configurability

A natural outcome of the decision to cast ASCEND into a multi-tool, multi-window form, was the need to provideahigh degreeofusercontrol over the environment Given the evidence that users needed to interact with various combinations of tools and in various modeling contexts, and the fact that relevant information to a modeling activity could easily exceed the available screen space, decisions about tool size and screen layout were best made by users themselves. The ideal arrangement of tools can only be determined in the context of the current modeling situation. For example, if a simulation is being investigated to determine the details of its structure, it wouldbereasonabletowantaBrowserthatoccupiesthewhole screen, with a skeletal view of an instance structure showing only instances of models; in other situations, the Browser might simply be used to select a variable within a single model and require relatively little screen space.

Although the ASCEND interface makes no assumptions about tool size, shape, location or even presence, it has been designed to prevent catastrophic failures such as 'losing" a tool, or reshaping it to an unmanageable state (i.e., where important controls cannot be accessed). In anticipating such problems however, we have been careful not to introduce unnecessary constraints on tool management, following Suchman's advice that an interface should support "the negotiation of trouble rather than trying to preclude trouble." The overall management of tools is facilitated by the presence of the Toolbox which allows them to be easily removed from the screen and restored to their previous size, location and state. We also provide users with the means to store personally designed screen configurations for later retrieval. This allows the usernotonlytopersonalize the ASCENDenvironmenUbut to also develop specific configurations for typically encountered modeling situations.

In addition to controls on its physical properties, a tool also provides the means to modify its behavior via a set of meta-level controls. Two examples of this are the previously mentioned "filters" within the browser, and an option within the Solver which determines whether or not the incidence matrix should be partitioned into block triangular form.

We have observed a clear relationship between a model builder'sgrasp of the ASCEND approach and their useofthese configuration options. Typically, new users will create a tiled layout in which all tools can be monitored simultaneously. As they gain experience, their default layouts consist of fewer tools and usually anticipate a specific modeling task.

The ASCEND environment does not borrow the entire screen—it coexists freely with other processes and windows. Thisgivestheuseraddedflexibilitytousethesysteminalarger computing context For example, the current implementation of the Model Library organizes models with respect to an inheritance hierarchy, but does not reflect the organization of these models within the files that contain their definitions. It is not unusual for users to create a file-oriented view of models by invoking their favorite text editor and setting it along side of other ASCEND toolkits. This is an example of use that emerged completely outside of the developer's conception of the system. In Geoffrion's paper, he proposes a high degree of integration between tools and utilities for communication. In the above example, we see that it is important that such integration does not always rule out unanticipated, but helpful user innovation.

6.2.5 Domain Independence

Although ASCEND was conceived with the needs of Chemical Engineering in mind, it was developed to support expression of mathematical modeling needs in a very general way. This generality has not only made it applicable to a wide range of problems in Chemical Engineering, but also to problems in other disciplines. Because the system was viewed as an experimental apparatus, we attempted to keep the semanticsofitsinteractionclosely tied tothe ASCEND modeling approach.

The main effect this decision had on the implementation of the system was a deliberate avoidance of the "real-world" metaphor approach to interface design. In this sense, the ASCEND environment is more like a programming/debugging environment than a modeling application. That is, the only domain-specific semantics which are present in simulations are determined by the model builder in choosing names for various components of the models.

Despite this lackofsupportfor specific disciplines, we have seen significant use of ASCEND in several disciplines, as mentioned in Section4.Wehave,however,encountered some complain tsabouttheover-genexality of ASCEND. These have come particularly from industrial users, who cite significant increasesincomplexity,especiallyincomparisonwithexisting domain-specific environments. Weacknowledgethis problem, especially in the case of relativelyroutine tasks. Toaddress this issue, we are currently investigating how domain-specific layersmightbelayeredon topofthe basic ASCEND"engine."

7.0 Conclusion

In this paper we have argued for the need for a structured approach to mathematical modeling, distilling user requirements into three major categories: hierarchicaldecomposition, evolutionary modeling, and debugging. We described the syntax and semantics of the language which resulted from our attempts to support these needs. Our experience so far indicates that it supports the rapid writing of complex models. However.there is also a cost involved in learning the language, because the approach is foreign to most model-builders.

In designing the interactive environment to this language, it has been important to support the kind of flexible interaction that is implied by constraint-based models. We have argued that this means decomposing the mcxielingprocess into distinct subtasks and providing toolkits that are designed to specifically support them. Although we have reified the modelling process to this extent, we have avoided prescribing a strict order in which these tasks must be carried out

Our experience has shown that model-builders need a dynamic view of large and complex sets of data. By dynamic, we mean both changing content and changing levels of detail. By data, we refer to model code, instance values, and the structures by which they arc organized. We argue that this means allowing people a high degree of control over their environment

We have been encouraged by the success of users who have taken vastly different approaches to formulating and solving problems with ASCEND, and by the degree to which features have been utilized in actual practice. We see this as evidence for the efficacy of the ASCEND technology.

Appendix A: The Transportation Model

```
IMPORT transportation.atoms;

MODEL plant;
    sup IS_A supply_capacity;
    customerld IS_A set OF integer;
    maxCustomerIS_A integer,

    CARD(customerld) <= maxCustomer,

    f[customerld], totalFlow IS_A flow;
    cost[customerId]IS_AunitCost;
    totalFlow = SUM(f[customerId]);
    totalFlow <= sup;

    shipmentCost IS_A cost;
    shipmentCost = SUM(f[i]*cost[i] I i IN customerld);
END plant;

MODEL customer;
    dem IS_A demand;
    plantld IS_A set OF integer,
    f[plantId]IS_Aflow;
    SUM(f[plantId]) = dem;
END customer,

MODEL transportation;
    plantld, customerld IS_A set OF integer;
    p[plantld] IS_A plant;
    c[customerld] IS_A customer;
    FOR i IN customerld CREATE
        c[i].plantld := [j IN plantld I i IN p[j].customerld];
    END;
    FOR i IN plantld CREATE
        FOR j IN p[i].customerld CREATE
            p[i].f[j], customer[j].f[i] AREJTHE_SAME;
        END;
    END;
    obj: MINIMIZE
        SUM(p[i].shipmentCost I i IN plantld);
END transportation;
```

Appendix B: Forecasting Models

```
IMPORT forecast_atoms;

MODEL product;
    TfIS_A integer,
    dem[l..Tf] IS_A demand;
END product;

MODEL forecast;
    TfIS_A integer,
    D[l..Tf]IS_A demand;
    E[l..Tf] IS_A expectedValue;
    S[2..Tf] IS_A smoothedValue;
    F[2..Tf] IS_A forecastedValue;
END forecast;

MODEL expForecast REFINES forecast;
    alpha IS_A dimensionlessConstant;
    E[1] = D[1]
    FOR i IN [2..Tf] CREATE
        E[i] = alpha*D[i] + (l-alpha)*E[i-l];
        F[i] = E[i]+S[i]/alpha;
    END;
    S[2] = E[2]-E[1];
    FOR i IN [3..Tf] CREATE
        S[i] = alpha*(E[i]-E[i-l]) + (l-alpha)*S[i-l];
    END;
END expForecast;
```

Appendix C: Transportation Model with Forecasted Demand

```
IMPORT trans;

IMPORT forecast;

MODEL forecastedProduct;
    pIS_A product;
    fIS_A forecast;
    p.Tf, f.Tf ARE_THE_SAME;
    p.dem, f.D ARE_THE_SAME;
END forecastedProduct;

MODEL customer_forecast REFINES customer;
    FIS_A forecast;
    dem = F.E[F.tf];
END customer_forecast;

MODEL trans_forecast REFINES transportation;
    c[customerld] IS_REFINED_TO customerjbrecast;
    c[customerld].FIS_REFINED_TOexpForecast;
END trans_forecast;
```

References

1. Bhargava, H., Kinbrough, S.,v Krishnan, R., "Unique Names Violations: A Problem for Model Integration or You Say Tomato, I say Tomahtof" Vol. 3:2, pp. 107-121, *ORSA Journal of Computing,* 1991

2. Bjerknes, G., Ehn, P., and Kyng, M. (1987). *Computers and Democracy.* England: Avebury.

3. Borning, A. (1979). ThingLab: A Constraint Oriented Simulation Laboratory. Ph.D. Thesis, Stanford University.

4. Brooke, A., Kendrick, D., and Meeraus, A. (1988). GAMS: A User's Guide. Scientific Press, Redwood City, CA.

5. Dhar, V., and Ranganathan, N. (1990). Integer Programming vs. Expert Systems: An Experimental Comparison. *Communications of the ACM,* Vol. 33, No. 3.

6. Floyd, C, Mehl, W-M., Reisin, F-M., Schmidt, G., and Wolf, G. (1989). Out of Scandinavia: Alternative Approaches to Software Design and System Development. *Human-Computer Interaction,* Vol. 4, pp. 253-350.

7. Fourer, R., Gay, D.M., and Kemighan, B.W. (1990). A Mathematical Programming Language. *Management Science,* 36:5, pp. 519-554.

8. Garlan, D. (1987). Views for Tools in Integrated Environments. Ph.D Thesis, Deptartment of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

9. Geoffrion, A. (1989). Computer-Based Modeling Environments. *European Journal of Operations Research,* 41, pp. 33-43.

10. Geoffrion, A. (1990). Reusing Structured Models Via Model Integration. Working Paper No. 362. Western Management Science Institute, University of California, Los Angeles.

11. Greenberg, H.J., and Murphy, F.H. (1991). Views of Mathematical Programming Models and Their Instances. Technical Report Mathematics Department, University of Colorado at Denver, Denver, CO.

12. Hindmarsh, A.C. (1980). LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers. *ACM-SignumNewsletter., Vol.l5,pp.* 10-11.

13. Hurlimann,T. (1989). Reference Manual for the LPL Modeling Language (Version 3.1). Institute for Automation and Operations Research, University of Fribourg, CH-1700 Fribourg, Switzerland.

14. Locke, M.H., and Westerberg, A.W. (1983). The ASCEND-II System - A Flowsheeting Application of a Successive Quadratic Programming Methodology. *Computers and Chemical Engineering,* Vol. 7, No. 5, pp. 615-630.

15. Muhanna, W.A., and Pick, R.A. (1988). Composite Models in SYMMS. *Proceedings of the 21st Annual Hawaii International Conference on System Sciences,* pp. 418-427.

16. Muhanna, W.A., and Pick, R.A. (1991). Meta-Modeling Concepts and Tools for Model Management: A Systems Approach. Working Papers Series 91-1. College of Business, The Ohio State University.

17. Murtagh, B.A., and Saunders, M.A. (1985). MINOS User's Guide. Technical Report SOL 83-20. Systems Optimization Laboratory, Dept. of Operations Research, Stanford University, Palo Alto, CA.

18. Perkins, J.D., (1983). Equation-Based Flowsheeting. *Proceedings of the Second International Conference on Foundations of Computer-Aided Process Design.* Arthur W. Westerberg, Henry H. Chien (eds.). Snowmass, Colorado.

19. Piela, P. (1989) ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis. Ph.D. Dissertation. Carnegie Mellon University.

20. Piela, P.C.,McKelvey, R.D., Katzenberg,B., and Mehlenbacher, B. (1991). Integrating the User into Research on Engineering Design Systems. EDRC Report. Carnegie Mellon University, Pittsburgh, PA 15213.

21. Sapossnek, M. (1989) Research on Constraint-Based Design Systems. *Proceedings of the 4th International Conference on Applications ofAI in Engineering.* Cambridge, England.

22. Smith, O. (1988). Solving Optimal Control Profiles as Algebraic Equations. Technical Report Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA 15213.

23. Suchman,L. Common Sense in Interface Design. *Techne: Journal ofTechnological Studies,* June 1987.

24. Sussman, G.J., and Steele, G.L. CONSTRAINTS - A Language for Expressing Almost-Hierarchical Descriptions. *Artificial Intelligence,* 14, pp. 1-39.

25. Sutherland, I. (1963). Sketchpad: A Man-Machine Graphical Communications System. Technical Report No. 296. MIT Lincoln Laboratory.

26. Westerberg, A.W., and Director, S.W. (1978). A Modified Least Squares Algorithm for Solving Sparse n x n Sets of Nonlinear Equations. *Computers and Chemical Engineering,* Vol. 2, No. 2/3, pp. 77-81.

27. Westerberg, A.W., and Benjamin, D.R. (1985). Thoughts on a Future Equation-Oriented Flowsheeting System. *Computers and Chemical Engineering,* Vol. 9, No. 5, pp. 517-526.

28. Westerberg, K.M. (1989a). Development of Software for Solving Systems of Linear Equations. Technical Report Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA 15213.

29. Westerberg, K.M. (1989b). Development of Software for Solving Systems of Nonlinear Equations. Technical Report Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA 15213.

30. Woodbury, R.F. (1990). Variations in Solids: A Declarative Treatment *Computers and Graphics, Special Issue on Features an Geometric Reasoning,* Vol. 14, No. 2.