Building the Scientific Modeling Assistant

An interactive environment for specialized software design

RICHARD M. KELLER

AI RESEARCH BRANCH, MAIL STOP 244–17
NASA AMES RESEARCH CENTER
MOFFETT FIELD, CA 94035

NASA Ames Research Center

Artificial Intelligence Research Branch

Technical Report FIA–91–13

May 1991

To appear in *Proc. AAAI-91 Workshop on Automating Software Design*

Building the Scientific Modeling Assistant:
An interactive environment for specialized software design

Richard M. Keller *

Artificial Intelligence Research Branch
NASA Ames Research Center
Mail Stop 244-17
Moffett Field, CA 94035-1000
keller@ptolemy.arc.nasa.gov

Abstract *

The construction of scientific software models is an integral part of doing science, both within NASA and within the scientific community at large. Typically, model-building is a time-intensive and painstaking process, involving the design of very large, complex computer programs. Despite the considerable expenditure of resources involved, completed scientific models cannot easily be distributed and shared with the larger scientific community due to the low-level, idiosyncratic nature of the implemented code. To address this problem, we have initiated a research project aimed at constructing a software tool called the *Scientific Modeling Assistant*. This tool provides automated assistance to the scientist in developing, using, and sharing software models. In this paper, we describe the Scientific Modeling Assistant, and also touch on some human-machine interaction issues relevant to building a successful tool of this type.

Introduction

Although model-building is an integral part of the scientific enterprise, there is little computational support available to the scientist performing this task. Without such support, scientific model-building can be a time-intensive and painstaking process, often involving the design and development of very large, complex pieces of software. Unfortunately, this software cannot easily be distributed and shared with other scientists because the implemented code is very low- level, idiosyncratic, and difficult for anyone but the original scientist/programmer to understand. In particular, the high-level structure and content of the scientific model is not obvious from the low-level code. FORTRAN and other general-purpose programming languages do not include the scientific terms and concepts that are natural to the scientist. Furthermore, important modeling and data assumptions are typically implicit in the low-level code that implements the model. These implicit assumptions cannot be easily inspected or modified by a potential new user. This is a significant deterrent to using another scientist's model; the appropriateness of assumptions is frequently the source of scientific debate.

To address these problems, the Scientific Modeling Assistant Project at NASA Ames Research Center is developing an interactive software design environment to support the scientist in building, using, sharing, and modifying scientific models (Keller et al. 1990). The overall goal of the Scientific Modeling Assistant Project is to study modeling as a human problem-solving activity, and to provide computational support for scientists engaged in this activity. We share some goals in common with other work on automating various aspects of scientific and engineering computation, including (Kant et al. 1990, Atwood et al. 1990, Barstow 1984, 1985, Robertson et al. 1989, 1991, Abelson et al. 1989).

Scientific model-building is by its very nature experimental and incremental. The scientist begins with a particular observation he or she wishes to explain, and a set of experimental data that may be used to explain the observation. Model-building is a theory formation process in which the scientist attempts to link the data with the observation via a set of intermediary computations. Sometimes there are many different possible theories for explaining the observation, and the scientist must choose the "best" theory according to a set of goodness criteria. Other times, it may turn out that the data do not satisfactorily explain the observation, and new data must be collected. Although one of our longterm goals is to develop fully automated scientific theory formation systems, our initial goal has been more modest -- to provide intelligent assistance to support human scientists engaged in model-building and theory formation. The crucial distinction here involves degree of autonomy; a fully automated system makes all decisions without scientist intervention, whereas an "intelligent assistant" offers suggestions, but leaves the scientist firmly in control.

We view scientific model-building as a specialized type of software design activity that is inherently iterative and interactive. The scientist does not begin with a set of formal specifications for a given model. Instead, he or

* Employed under contract NAS2-13210 to Sterling Federal Systems.

she begins with a set of observations, and attempts to develop specifications for a model that will account for those observations. Whereas an initial model may account for the observations, it may fail to adequately explain subsequent observations. In this case, the initial specifications must be revised and a new model generated. This process continues as long as new observations are forthcoming.

To facilitate interactive model-building, our current software prototype includes an intelligent graphical interface, a high-level domain-specific modeling language, a library of physics equations and experimental datasets, and a suite of data display facilities. Rather than construct models using a conventional programming language, scientists will use our graphical interface to "program" visually using a more natural high-level data flow modeling language. The terms in this language involve concepts (e.g., quantities, equations, and datasets) that are familiar to the scientist. In constructing this tool, we are using a variety of advanced software techniques, including AI knowledge representation, automatic programming and truth maintenance techniques, as well as techniques from object-oriented programming, graphical interfaces, and visualization.

As a testbed for this research, we have developed a software prototype in the domain of planetary atmospheric modeling. This prototype is being used to reconstruct a model of the thermal and radiative structure of Titan's atmosphere that was originally developed in FORTRAN (McKay, Pollack & Courtin 1989). Our longterm plan is to make the software assistant a viable tool to support future NASA planetary atmospheric modeling activities, including those related to the upcoming Cassini mission to the Saturn system. Although the current prototype is tied to the planetary atmospheres modeling domain, we are also investigating the applicability of these ideas to ecological modeling, where we are studying a model of the carbon, water, and nitrogen cycles in a forest ecosystem (Running & Coughlin 1988). By studying two different application domains, we hope to gain insight that will ultimately enable us to develop a general- purpose scientific modeling "shell". The shell would be instantiated for a particular modeling task by supplying domain-specific knowledge.

A Focal Problem: Titan Atmospheric Modeling

Our initial research has focused on a small portion of the overall Titan atmospheric model -- the gas composition portion. The purpose of the gas composition portion is to develop a profile of Titan's atmosphere that describes the pressure, temperature, and density of gases at various altitudes above its surface. This problem is underconstrained due to the shortage of empirical data on Titan. The major source of relevant experimental data is the Voyager-I flyby of Titan back in November 1980. As Voyager-I reached the far side of Titan, it sent back radio waves that passed through Titan's atmosphere and then on to receiving stations on Earth. Due to the density of gases in the atmosphere, the radio waves were refracted slightly as they passed through the atmosphere. The amount of refraction was measured at different altitudes above the surface. This refractivity data serves as a starting point for inducing the desired atmospheric profile in the Titan gas composition model.

In brief, the atmospheric profile can be determined as follows (see Figure 1). First, for each atmospheric point profiled, the Voyager-I refractivity data (R) is used to compute the number-density (ND) of the gases at that altitude. The number-density of a mixture of gases is defined as the number of molecules per volume of the mixture. Assuming the identity and relative percentages of gases in a mixture is known, the number-density can be computed as a function of refractivity. Next, using the molecular weight of the various gases in that mixture, the mass-density (RHO: mass per volume of mixture) can be computed from the number-density. The hydrostatic law can then be used to determine the pressure (P) from the mass-density by numerically integrating the weight of the atmosphere above each profile point. Finally, the temperature (T) can be determined from the mass-density and the pressure by applying an equation of state, such as the ideal gas equation. Figure 1 illustrates a level of abstraction at which a physicist might describe the problem, and is far more comprehensible than the corresponding FORTRAN code. Our system's graphical interface and modeling facilities permit a physicist to construct and manipulate models at this level of abstraction.

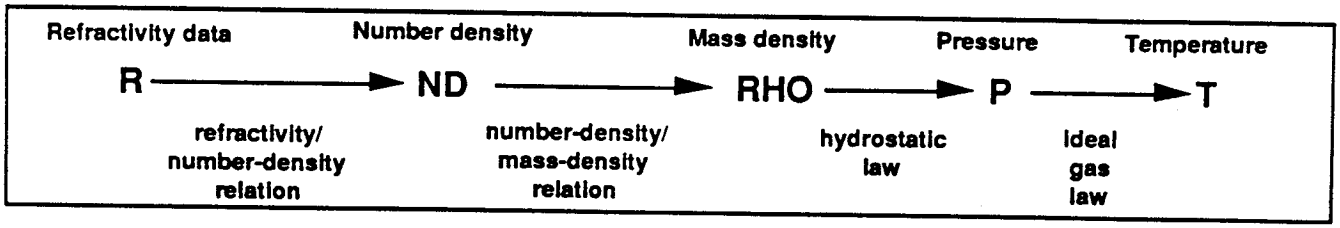

Figure 1: Determining the atmospheric profile

Initial Prototype

Our prototype enables scientists to construct simple models (such as the Titan gas composition sub-model described above), to modify parts of a model, to execute a model, and to perform analyses on the results. Our approach has been to build a visual programming tool that allows a user to construct models graphically using a visual data-flow language. The resulting model is "run" by executing the constructed data flow graph.

For the purposes of our initial prototype, we have conceptualized model-building as a process of linking uncomputed physical variables to computed variables using computational transformations. For example, the process of linking the input Voyager-I refractivity data to the output ideal gas temperature is accomplished by the sequence of transformations illustrated in Figure 1. Conceptually, each transformation takes as input a set of variables and produces a single variable as output. Physics equations and subroutines are two kinds of computational transformations supported in the current system. Physics forms the basis for atmospheric modeling, so the use of explicit physics equations is natural in this context. The introduction of subroutines is motivated by the fact that in building models, scientists often makes use of program code that has been developed elsewhere. Examples range from standard numeric integration and data interpolation packages to complex scientific models developed by other scientists. The details of these imported subroutines are assumed to be outside the scope and interest of the scientist-user, but can be incorporated into their models as 'black boxes'.

In our current prototype, the process of linking variables using transformations is accomplished via a simple backchaining procedure. In this backchaining process, the user first selects a target physical variable they wish to calculate. Then the system presents the user with a set of transformations that can be used to compute the desired variable. The user selects one of these transformations, and the system checks to see whether all the input variables required by this transformation have already been calculated. If so, the transformation fires and the desired output variable is computed; if not, the backchaining process recurses for each of the uncomputed variables. During this process, the graphical interface displays a dependency-tree visualization of the current model as it is being built. This visualization is similar to Figure 1, but in general depicts a more complex network of variables and transformations. The history of user modeling steps is recorded and maintained by the system, and can be displayed at any time.

The prototype features a Macintosh-like interface with pull-down menus and windows. The interface enables the user to visualize the model and its associated variables in a variety of formats, including plotted graphs and data tables for displaying computed variables. The interface provides various functions to manage the models stored in a scientist's workspace. For example, the user can switch the current focus of the model-building activity to a different model and/or workspace, and can retrieve old models, initiate new models, or delete existing models. Also, the interface provides a facility for applying user- defined tests of model viability to the current model under development. For atmospheric modeling, one such test is a test for atmospheric stability. If the temperature gradient predicted by a model is too steep, the atmosphere will be inherently unstable -- and this violates normal expectations. So the model testing facility provides the scientist with valuable feedback on whether the current model needs further refinement.

Sample Session

To give the flavor of how models are built and extended using our tool, this section illustrates a portion of a model construction scenario in some detail.

For the purposes of this discussion, we will assume that the user has already built an atmospheric model, and wishes to extend the model. In particular, we will assume the scientist has already constructed a model that determines pressure from refractivity data (see first 3 steps in Figure 1). Now the scientist wishes to extend the model to compute temperature, as well. This addition is accomplished with the following steps:

- Step 1: The user recalls the incomplete model stored in his or her workspace. A visualization of the current model is then displayed to the user (see Figure 2). The tree-like structure shows the transformations applied thus far in the model, as well as the associated input and output variables, and their interdependencies. For example, reading down from the root of the tree, we see that the last variable computed was *P-1*, a pressure variable. *P-1* was computed using an instance of the hydrostatic law (*hydrostatic-law-1*) based on the input values *G-1* --- the gravitational force at a particular point in the atmosphere --- and *RHO-1* --- the mass-density at that point. *RHO-1* was in turn computed from an equation relating number-density and mass-density (*density-from-nd-1*). The input number-density (*total-nd-1*) was computed using the relationship between number-density, refractivity, and the relative percentages of each of the gases in the atmosphere (e.g., *mixing-ratio-ch4-1*). The variables at the leaves of the tree are primitive inputs to the model, and are assumed to be given. (Note that the boxed variables in the tree appear more than once in different places.)

•Step 2: The user selects *temperature* from the **variable menu** (see Figure 3) indicating she or he next wishes to compute temperature within the model.

•Step 3: In response to the user's selection of a physical variable, the system displays a menu of all transformations that can be applied to compute that variable (see Figure 4). This is the beginning of a backchaining process initiated to compute *temperature*.

•Step 4: The user chooses one of the transformations -- in this case, a non-ideal gas equation of state. (This non-ideal gas equation corrects the ideal gas temperature using an empirically-derived correction factor.) The user's choice is echoed in the **history window**, and the system displays the transformation in the **equation window** for verification (see Figure 5). In the **equation window**, the complete equation is displayed along with the symbols used and their meanings. At this point, if the user has made an incorrect choice, the interaction can be aborted and different choice made. The system makes a preliminary attempt to bind the symbols in the equation to known variables in the current model, and these bindings are displayed in the **equation window** subject to user approval.

Simultaneously, the system displays the status of the backchaining process in the **backchaining status window** (Figure 5). Here the temperature variable to be computed (*T-1*) is shown as an output of an instance of the non-ideal gas equation of state (*non-ideal-eqtn-state-n2-1*). The "?" indicates that the value of the *T-1* variable has yet to be computed.

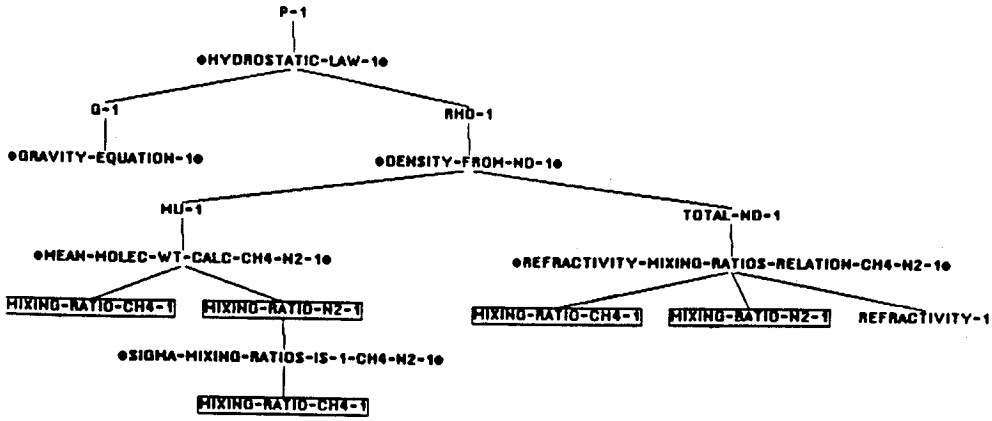

Figure 2: Dependency-tree visualization of incomplete atmospheric model

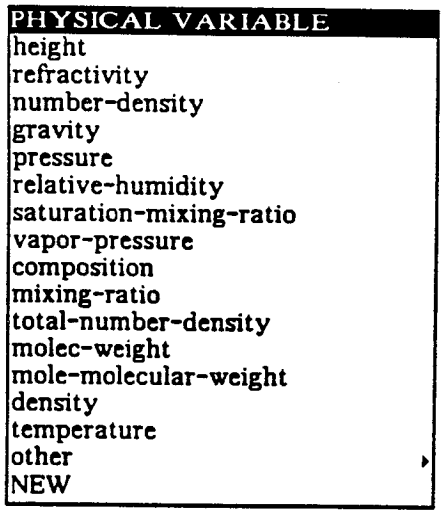

Figure 3: Variable menu

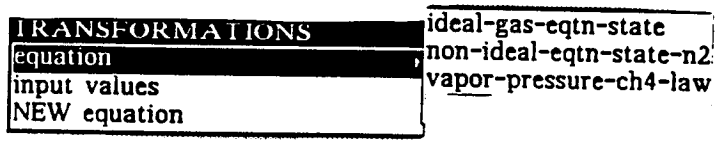

Figure 4: Transformation menu

•Step 5: Because there are two temperature symbols in the non-ideal gas equation (*TI* , an ideal gas temperature, and *T*, the corrected non-ideal gas temperature), the user is asked to select one of these symbols as the output to the equation. The system will manipulate the equation so that the symbol to be computed is on the left hand side of the equal sign in the equation.

•Step 6: After the bindings have been verified and the output variable has been selected, the system updates the **backchaining status window** to display the selected bindings (Figure 6). Note that at this point, the new *temperature* variable (*T-1*) has been connected to the original incomplete model (shown in Figure 2) by means of the instantiated pressure variable *P-1*, which was introduced by the non-ideal gas equation of state. Because the ideal gas temperature (*TI-1*) has yet to be computed, the system now begins recursively backchaining on this variable.

•Step 7: As in Step 3, the system prompts the user to select the transformation to be used in computing the ideal gas temperature. This time the user picks the ideal gas equation, rather than the non-ideal gas equation. The inputs to this equation are number-density and pressure, which have already been computed as part of the original incomplete model. The system binds these variables and the backchaining process terminates. The system now percolates the data values from the leaves of the tree up to the root to compute first the ideal gas temperature (*TI-1*), and finally the non-ideal gas temperature (*T-1*).

•Step 8: At this point, the user may choose to plot the value of non-ideal temperature as a function of altitude (Figure 7) by making an appropriate selection in the **display menu**. Or the user may decide to apply a test of model viability from the **test menu**. For example, the *saturation law* specifies that the mixing ratio of a given gas in the atmosphere must be less than the saturation mixing ratio for that gas. In other words, gases are not expected to supersaturate under normal circumstances. The test selected by the user appears in the equation window (Figure 8a). The system applies the test and reports whether it is satisfied by the results produced by the current model. In this case, the test fails, as is illustrated by the plot of mixing ratio and saturation mixing ratio versus altitude (Figure 8b), which the user has requested. Note in the region between 5 and 50 kilometers above the surface, the saturation mixing ratio (dark line) is less than the 5% fixed mixing ratio (light vertical line) that the user has defined in the model.

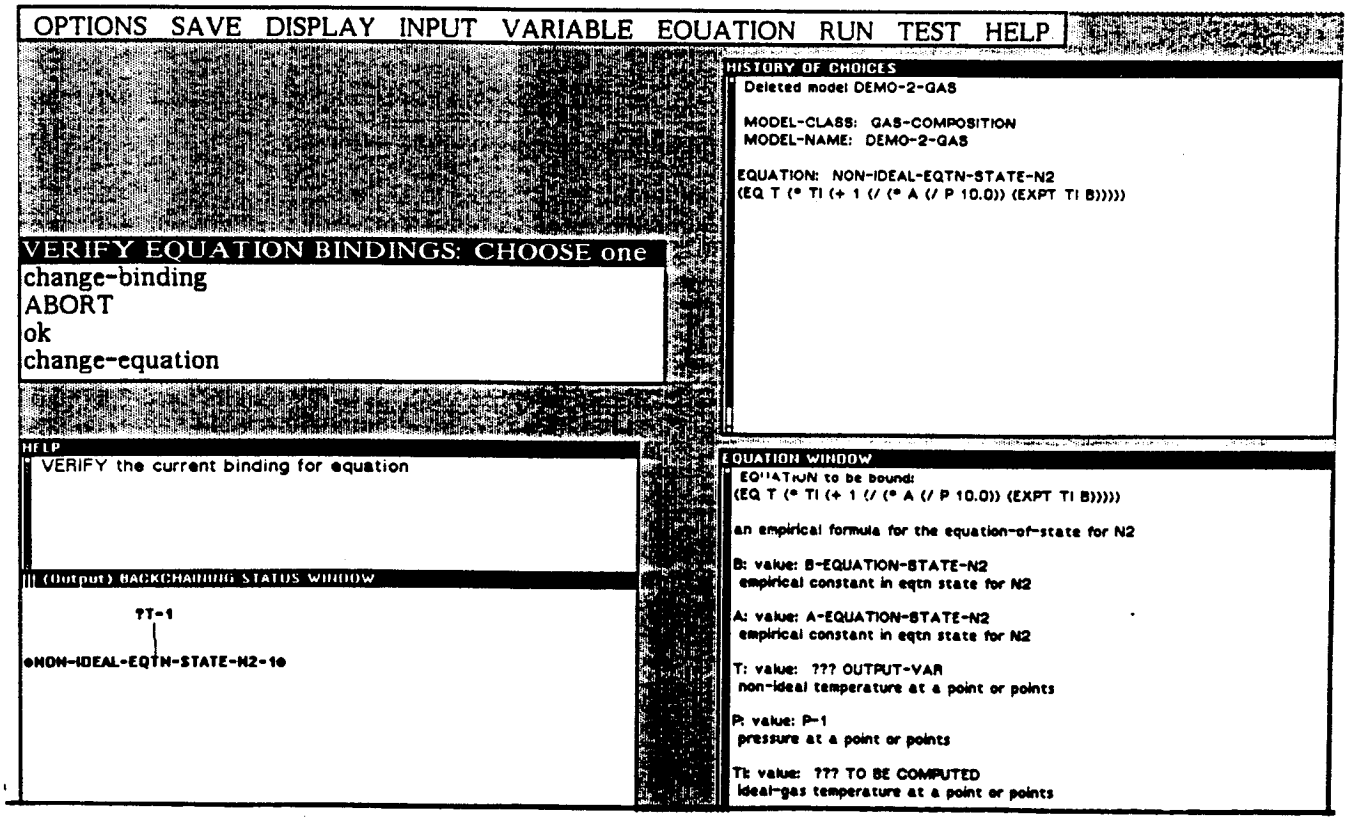

Figure 5: Tool display screen at beginning of interaction

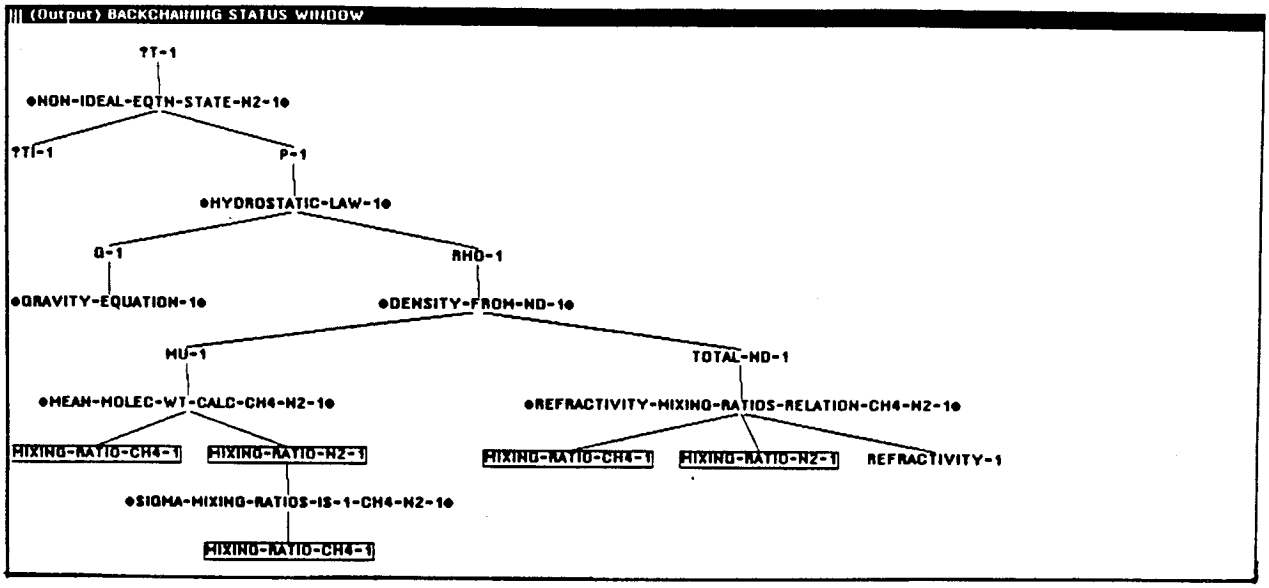

Figure 6: Updated backchaining status window

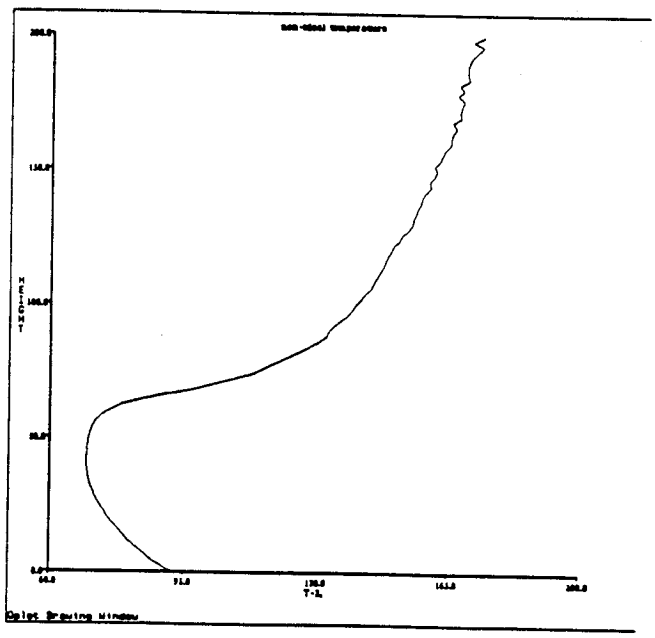

Figure 7: Plot of non-ideal temperature versus altitude

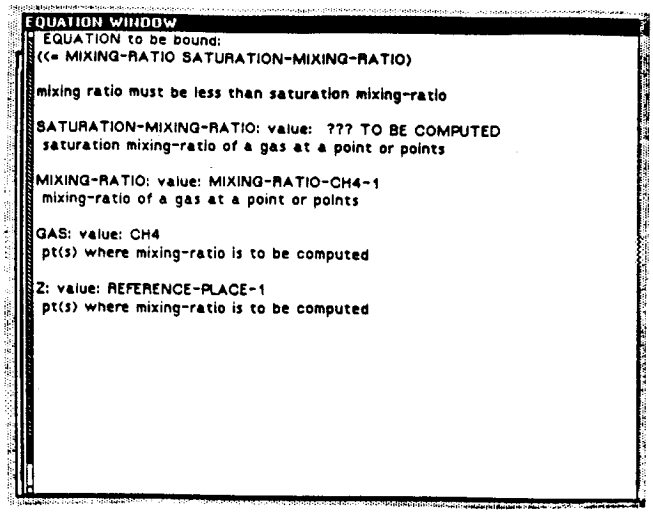

Figure 8a: Saturation law

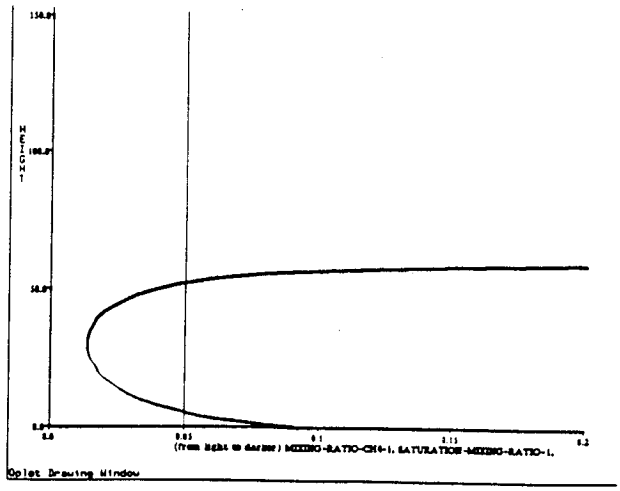

Figure 8b: Plot of mixing ratio (light line) and saturation mixing ratio (dark line) versus altitude

Status and Evaluation

To date, we have focused on model construction rather than model modification, although the tool supports simple types of changes to existing models. We have begun testing the prototype with atmospheric scientist users, and are in the process of evaluating its strengths and weaknesses.

The preliminary results of this evaluation indicate that overall, users find the system's dataflow modeling paradigm to be extremely useful both in conceptualizing models and in speeding their implementation. However, we need to significantly improve certain system limitations before the tool can be "field tested". In particular, our evaluation has highlighted the following three limitations, which we are addressing in a new implementation:

Limitation 1: *The system's ability to make intelligent choices and suggestions is hampered by its lack of sufficient background physics and atmospheric sciences knowledge.*

Fundamentally, the quality of the tool's interaction with a planetary scientist is determined by the extent of the system's knowledge about the modeling task and the related atmospheric physics. If the current system had a greater understanding of atmospheric modeling, it would be capable of providing an additional level of support for the modeling task. Specifically, additional knowledge would enable the system to present the user with more intelligent choices at decision points. By making "intelligent choices", we mean the system should provide better system defaults, more intelligent selection of data structures, better ordering of user options, and a reduced (i.e., filtered) set of options.

There are several places where more intelligent choices are desirable in the current system. For example, the system currently displays a nearly exhaustive list of physical variables and transformations when the user clicks on the variable or equation menus, respectively. As the number of transformations and variables increases, this exhaustive display will cease to be a viable option. The system must intelligently filter the set of transformations or variables that are relevant in the current modeling context.[1]

Another place where the system must make intelligent choices is in binding equation symbols to physical variables. For example, when the user chooses to apply the ideal gas equation to compute temperature, the system must bind the symbols for pressure and number-density to previously-computed physical variables. If a number of different pressure and number-density variables have been computed in the current model, the system must determine which of these variables the user intends to bind to the equation symbols. If the system "understands" the ideal gas equation, it can apply certain constraints to ensure necessary consistency among its choices. For instance, to apply the ideal gas equation properly, the selected pressure and number-density variables must pertain to the same gas mixture at the same location. Any attempt to apply the equation to a pressure at the Titan's surface and a number-density at 200 meters above the surface should be recognized and rejected as nonsensical. This type of constraint is normally implicit in the computer encoding of the

[1] Actually, the system currently performs some intelligent transformation and variable filtering. For example, during backchaining, the system filters the set of transformations displayed so that only those involving the backchained variable are shown.

model. It is precisely this implicit encoding that makes the creation, modification, and sharing of these programs with colleagues so difficult. Explicit representation of domain knowledge and reasoning processes is an approach to solving these problems.

An atmospheric scientist brings a variety of different kinds of knowledge to bear in approaching the modeling task. This includes both domain knowledge, such as knowledge about physical variables, physics equations, and the structure of atmospheres, and modeling knowledge, such as various strategies and heuristics for atmospheric modeling, and various constraints and assumptions pertaining to the modeling process. We believe that representing knowledge of this type is the key to developing a useful modeling assistant.

Limitation 2: *The current interface is too rigid, stylized, and confusing for users, who desire a more fluid, open-ended interaction with the system.*

Rather than be forced into a backward-chaining mode of interaction, where the user is locked into a strict goal-directed design mode, we need to provide for more flexibility and fluidity. Our scientists expressed a desire to construct models piecemeal, possibly working on unrelated portions of a model in succession, and then piecing together the results. We are currently designing a Macintosh-like wiring-diagram interface that will facilitate this mode of design.

Limitation 3:*The current system lacks portability.*

To be successful, our system must be used by a community of scientists in a variety of settings. Our prototype system is implemented using a proprietary piece of software (Intellicorp's KEE) and runs on an expensive, specialized Lisp machine. Our new system is being written in CommonLisp/CLOS/CLIM, and will run on a variety of platforms, including a Sun Sparcstation and a Macintosh II, as well as a Symbolics workstation.

Human-Machine Interaction Issues

In developing our prototype system, we encountered a number of critical issues relevant to human-machine communication. Some of these issues are associated with acquiring, maintaining, and explaining the system's domain knowledge; others stem from the need to communicate with specialized domain experts who may be either non- programmers or relatively unsophisticated programmers; still others can be attributed to the inherently incremental nature of the specification acquisition process in this design domain. Although we provide no definitive answers, we raise some of the issues here, and look forward to addressing them in future versions of our Scientific Modeling Assistant.

Complexity

Scientific models can be very complex pieces of software. Both the original designer and those who subsequently inspect, use, and modify the model will need assistance in understanding how the model functions and in dealing with its complexity. We intend to make use of several methods for managing complexity:

- Our high-level data flow language provide a level of abstraction that helps the scientist understand the essence of the model without focusing on implementation details involving data structures and control.

- Our interface will support hierarchical structuring of the data flow graphs, allowing the scientist to "package" portions of the model into larger chunks and hide detail.

- The interface should present multiple, selective views of the software, depending on the user's interests. A scientist interested in hydrodynamic aspects of an ecosystem model should not have to view those parts of the model dealing with thermodynamic computations.

Information Access

During interactive design, the user needs access to information necessary for completing the design. For example, during the model-building process the scientist frequently requires access to the following type of information:

- Data Transformations (equations, subroutines, and auxiliary computational models): Transformations are the building blocks of scientific models. They perform computation on the input data to derive the primary observation of interest, creating intermediate variables along the way. The number of available transformations in a full-blown operational scientific modeling environment will be significant, and any such system will have to provide methods for indexing and retrieving these transformations for the user.

- Domain knowledge: A system will have to provide the user a means of inspecting the system's domain knowledge base -- both to access information during the course of design ("What is the atomic weight of nitrogen.?") and to verify and change the domain knowledge.

- Symbol manipulation packages: Aside from accessing declarative knowledge, the scientist requires access to auxiliary computational support, such as the kind

provided by symbolic "scratchpads", such as Mathematica.

• Data visualization: The scientist frequently plots the outcome of the model in order to ascertain its quality. In general, the scientist needs access to facilities for visualizing the data that is input to, and generated by, the model under design.

Knowledge Acquisition

Knowledge acquisition is a crucial function of any domain-specific software design system. Unless the domain knowledge is completely circumscribed and static, it will be necessary for the user to have the capability of updating and extending the system's domain knowledge. In our case, we use an object-oriented, frame-base knowledge representation system to encode domain knowledge. If a scientist wants to add a new scientific equation to the system, he or she must define the semantics of that equation in terms of the domain knowledge by identifying the relevant objects, attributes, and constraints pertaining to the equation. Thus the user must be able to easily browse the domain knowledge. As a more difficult knowledge acquisition problem, consider that the user may want to change the very underpinnings of a given model, for example going from a static to a dynamic model or from a two-dimentional to three-dimentional spatial model. Enabling the user to effect these types of fundamental representational changes is a very difficult knowledge restructuring problem.

Design Rationale

In building a scientific model, it is particularly important to record the design rationale and assumptions underlying the model. This is crucial for reuse; other scientists will not use a given model unless they understand the appropriateness of those assumptions. For example, the Titan atmospheric model assumes a well-mixed composition of gases in the atmosphere. Many of the model's calculations are based on this assumption, and it is not possible to use the model properly without an awareness of this assumption. In particular, this assumption may not be appropriate for a scientist studying atmospheric inversion processes. Access to assumptions is also crucial during the design process when they may be violated. Our system should record and maintain modeling assumptions and notify the user if underlying assumptions are violated by their design actions.

Interaction Vocabulary

To encourage the use of specialized software design environments, such as the Scientific Modeling Assistant, it is important to communicate with the user in their own vocabulary and to avoid relying on unfamiliar formal constructs (such as predicate calculus, set theory) from outside their domain of expertise. This serves to reduce the overhead in learning and feeling comfortable with a new type of software environment.

Related Research

There are several pieces of relevant research on the construction of scientific models and modeling and analysis tools in various domains:

• Barstow's work on the Fnix system (Barstow 1984, 1985) is quite close in spirit to our work. Fnix is a domain-specific automatic programming system constructed to assist in generating oil well log interpretation software. The system was designed for direct use by petroleum scientists, who use it to construct geological models in the form of a set of quantitative equations relating geological parameters of interest. Like our Scientific Modeling Assistant, Fnix makes extensive use of scientific domain knowledge, in addition to programming knowledge. Fnix, however, can generate a much wider class of programs due to its less restrictive specification language coupled with general transformational methods.

• In a similar vein, (Kant *et al.* 1990) describes the SINAPSE system under development at Schlumberger. SINAPSE helps scientist perform mathematical model-building in the context of data interpretation tasks such as seismic interpretation. The system's domain of expertise is quite narrow, and specifically covers the generation of code for seismic models phrased as finite difference equations.

• A group at M.I.T. describes a suite of intelligent programs developed for automatic preparation, execution, and control of numerical experiments (Abelson *et al.* 1989). These programs have been combined in a package called the "Kineticist's Workbench" to produce a sophisticated tool for modeling and analysis of dynamical systems.

• The Reason system supports sophisticated analysis of high energy physics data using a dataflow language and an interactive graphical interface (Atwood et al. 1990).

• Finally, (Robertson *et al.* 1989, 1990) report on the ECO system developed at the University of Edinburgh to facilitate the construction of ecological simulation models using a sorted logic representation.

Summary

This paper has described a new research effort underway at NASA Ames Research Center to construct a Scientific Modeling Assistant. Scientific model-building is an inherently interactive, incremental software design process that constitutes a kind of theory-formation activity. We described several human-machine interaction issues related to the development of this interactive scientific software design and development environment.

Acknowledgments

This research was co-funded by NASA's Office of Space Science and Applications, and the Office of Aeronautics and Exploration Technology. I wish to thank the other members of the Scientific Modeling Assistant Project for their intellectual contribution to the ideas expressed in this paper. These members include Michael Sims, Christopher McKay, Esther Podolak, Michal Rimon, and David Thompson.

References

[Abelson et al. 1989] H. Abelson, M. Eisenberg, M. Halfant, J.Katzenelson, E.Sacks, G.J.Sussman, J.Wisdom, and K.Yip, "Intelligence in Scientific Computing", *Comm. ACM*, 32(5):546-562, May 1989.

[Atwood et al. 1990] W.Atwood, R.Blankenbecler, P.F.Kunz, B.Mours & A.Weir, "The Reason Project", Stanford Linear Accelerator Center technical report #SLAC-PUB-5242, April 1990.

[Barstow 1984] D.Barstow, "A Perspective on Automatic Programming", *AI Magazine*, Spring 1984.

[Barstow 1985] D.R.Barstow, "Domain-Specific Automatic Programming", *IEEE Transactions on Software Engineering*, Vol. SE-11 (11), November 1985.

[Kant et al. 1990] E.Kant, F.Daube, W.MacGregor, J.Wald,"Synthesis of Mathematical Modeling Programs", Schlumberger Lab for Computer Science tech report # TR-90-6, February 1990.

[Keller et al. 1990] R.M.Keller, M.H.Sims, E.Podolak, C.P.McKay & D.E.Thompson, "Proposal for Constructing an Advanced Software Tool for Planetary Atmospheric Modeling", AI Research Branch tech report #RIA-90-03-20-1, NASA Ames Research Center, March 1990.

[McKay, Pollack, & Courtin 1989] C.P.McKay, J.B.Pollack & R.Courtin, "The Thermal Structure of Titan's Atmosphere", *Icarus*, Vol.80:23-53, 1989.

[Robertson et al. 1989] D.Robertson, A.Bundy, M.Uschold, and R.Muetzelfeldt, "The ECO Program Construction System: ways of increasing its representational power and their effects on the user interface", *Intl.J. Man- Machine Studies*, Vol. 31:1-26, 1989.

[Robertson et al. 1991] D.Robinson, A.Bundy, R.Muetzelfeldt, M.Haggith & M. Uschold, *Eco-Logic: Logic-Based Approaches to Ecological Modelling*, MIT Press, 1991.

[Running & Coughlan 1988] S.W.Running & J.C.Coughlan, "A General Model of Forest Ecosystem Processes for Regional Applications: I. Hydrologic Balance, Canopy Gas Exchange and Primary Production Processes", *Ecological Modelling*, Vol. 42: 125-154, 1988.