N95-23716

A Toolbox and Record for Scientific Models

Thomas Ellman
Department of Computer Science, Hill Center for Mathematical Sciences
Rutgers University, Piscataway, New Jersey, 08855, USA
Phone: (908) 445-4184, FAX: (908) 445-5530
Email: ellman@cs.rutgers.edu

Key Words and Phrases

Artificial intelligence, scientific computation, automated software design, knowledge representation, approximation.

Difficulties of Scientific Programming

Computational science presents a host of challenges for the field of knowledge-based software design. Scientific computation models are difficult to construct. Models constructed by one scientist are easily misapplied by other scientists to problems for which they are not well-suited. Finally, models constructed by one scientist are difficult for others to modify or extend to handle new types of problems. Existing knowledge-based scientific software design tools, such as SIGMA (Keller & Rimon 1992), provide only limited means of overcoming these difficulties. For example, SIGMA facilitates model construction by providing scientists with high-level data-flow language for expressing models in domain-specific terms. Although SIGMA represents an advance over conventional methods of scientific programming, it supports only certain aspects of the model development process. In particular, SIGMA focuses mainly on automating the process of assembling equations and compiling them into an executable program. Construction of scientific models actually involves much more than the mechanics of building a single computational model. In the course of developing a model, a scientist will often test a candidate model against experimental data or against a priori expectations. Test results often lead to revisions of the model and a consequent need for additional testing. During a single model development session, a scientist typically examines a whole series of alternative models, each using different simplifying assumptions or modeling techniques. A useful scientific software design tool must support these aspects of the model development process as well. In particular, it should propose and carry out tests of candidate models. It should analyze test results and identify models and parts of models that must be changed. It should determine what types of changes can potentially cure a given negative test result. It should organize candidate models, test data and test results into a coherent record of the development process. Finally, it should exploit the development record for two purposes: (1) automatically determining the applicability of a scientific model to a given problem; (2) supporting revision of a scientific model to handle a new type of problem. Existing knowledge-based software design tools must be extended in order to provide these facilities.

An Artificial Intelligence Approach

We are attacking this problem using two related ideas: First, we are building a "Model Development Toolbox". The toolbox will support a set of generic model development steps that are taken by most scientists in the course of developing scientific computational models: Examples of such generic model building steps include: (1) mapping equations onto physical situations; (2) fitting models against experimental data; (3) testing models against experimental data; (4) testing applicability of models to given inputs; and (5) modification of models in response to test results. Second, we are designing a "Model Development Record". The record will contain machine readable documentation of the entire model development process. To begin with, the record will describe the goals the model is intended to fulfill. For example, this might include a representation of the questions the model is (and is not) intended to answer. The record will also describe the sequence of candidate models that were constructed in the course of developing the final model. For each candidate model, the record might describe: (1) the equations encoded in the model; (2) assumptions underlying the model; (3) fitting techniques used to instantiate free parameters of the model; and (4) tests against empirical data that were performed on the model. The record must also describe (5) the temporal sequence of candidate models as well as (6) logical dependencies between test results on early models and modeling choices made in constructing subsequent, more refined models.

Tools for checking applicability of scientific models to new problems will rely heavily on the model development record. Important applicability checks include: determining whether a proposed use of a model is consistent with the goals the model was originally intended to fulfill; determining if a new problem lies within the range of inputs for which the model was tested; and testing assumptions underlying the equations that were incorporated into the model. Each of these checks requires access to various aspects of the model development record. Likewise, tools that support model revision will also rely heavily on the model development record. Important types of model revision include: extending/modifying the model to handle a wider/different range of input parameters; re-fitting free parameters of the model to new empirical data; changing the assumptions used to model a physical process; adding/deleting physical processes to/from the model; and changing the overall purpose of the model. A model revision tool should automatically determine when a revision is needed (e.g., by determining that a new problem falls outside the range of problems handled by the original model, or by detecting discrepancies between empirical data and outputs of the model). It should suggest changes to the model that have the potential to cure the problem (e.g., by reasoning about sensitivities of outputs with respect to changes in intermediate results, or by reasoning about the effects of potential changes in assumptions on the outputs of the model). Finally the system should assist in re-validating the new model, (e.g., by suggesting new tests of validity, and carrying out and evaluating such tests.) In many cases, models may be revised by "replaying" a portion of the development record that led to the original model. Replay will require access to logical dependencies among test results and modeling choices found in the development record, using techniques similar to derivational analogy (Mostow 1989) and transformational implementation (Balzer 1985).

System Architecture

The overall architecture of our envisioned system is shown in Figure 1. The model development toolbox serves as a front end to the whole system. The toolbox interacts with a human user to build an initial model in some scientific domain. It also interacts with a user in order to revise an existing model to handle a new situation. Finally, the toolbox also includes facilities for controlling the application of scientific models. As the toolbox guides the user through a series of model building, testing and revision steps, it interacts with several data bases. The model fragment data base contains the basic building blocks of scientific models. The toolbox uses techniques embodied in

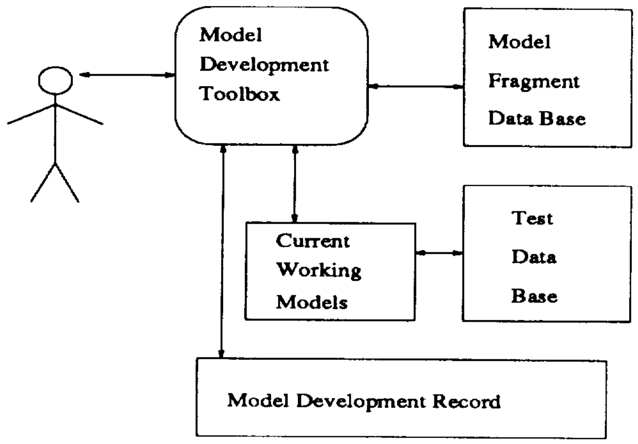

Figure 1: Model Development System Architecture

the SIGMA system to combine model fragments into one or more "current working models". As working models are constructed, they are tested against test data drawn from a test data base. Likewise, as tests are run, results are incorporated back into the test data base. As the initial model development process unfolds, the toolbox leaves a structured trace of the process in the model development record. Later on, the scientist will apply the model to specific problems in which he is interested. As the model is applied to each problem, the system consults the model development record to determine whether the model is valid for the current problem. If the model fails to apply, the scientist may use the toolbox to revise the model. During the revision process, the toolbox is also guided by the model development record. The toolbox and record is being implemented as an extension to the SIGMA scientific model building system (Keller & Rimon 1992). Testbed domains for this research include planetary atmosphere modeling and ecosystem modeling problems used in development of SIGMA. Additional testbed domains include two problems under investigation in computer-aided design research at Rutgers University: modeling of jet engine nozzle performance and modeling the motion of sailing yachts.

Controlled Application of Models

Implementation of the model development toolbox and record is initially focusing on methods for controlling the application of scientific computation models. To this end, we have developed a collection of techniques that prevent users from applying scientific models to situations which violate their implicit assumptions and lead to erroneous or meaningless results. Some of these tests can be applied to virtually any scientific model. Such generic test include: (1) comparing inputs, outputs or intermediate results to fixed bounds; (2) verifying expectations about monotonicity or uni/multi-modality of computed functions; (3) validating results in comparison to simplified models. We have also defined a collection of more specialized tests, whose relevance depends on the specific idealizations, approximations or abstractions that were used to construct the model. Examples include: (4) checking nearness to the fitting point of a linear approximation and (5) verifying self-consistency of solutions obtained by decomposing systems of equations, among others.

Our applicability testing techniques require that models be represented in a manner that makes explicit what tests are required and how the tests should be applied. For this reason, we have developed and implemented a model representation language that contains applicability checking information. Our representation is an extension of the dataflow graphs used in SIGMA (Keller & Rimon 1992). The representation includes annotations that describe what applicability tests should be carried out at model execution time. The annotations are linked to the dataflow graphs in a manner that allows the system to determine the stage of the computation at which each applicability test should be carried out. We have also defined and implemented a general model execution procedure that refers to the annotations to perform the required applicability tests during the course of model execution. We have implemented and tested several versions of a jet engine nozzle performance model and a yacht velocity prediction model in the new representation along with applicability tests suitable to each.

An example of a scientific model represented as a dataflow graph is shown in Figure 2. This graph represents a model for computing the steady state velocity of a sailing yacht as a function of several geometric and physical parameters of the yacht, (e.g., vertical center of gravity (VCG), wetted surface area (WSA), longitudinal second moment (LSM), effective draft (T_{eff})), as well as inputs describing the sailing conditions, (wind-speed (V_{tw}) and heading angle (B_{tw})). The model describes a computation that proceeds in two stages. The first stage is to solve the torque balance equation $NetTorque(\phi) = 0$ which asserts that "heeling" torques (causing the yacht to heel over in the wind) are equal to "righting" torques (causing the yacht to remain upright). The solution value of ϕ is the heel angle at which the yacht will sail. The second stage is to solve the force balance equation $NetForce(v, \phi) = 0$ which asserts that "thrust" (due to the wind acting on the sails) is equal to "drag" (due to the friction caused by water). The solution value of v is the steady state velocity of the yacht. The "Torque Balance" and "Force Balance" nodes of this graph each describe submodels of the overall yacht velocity prediction model. Each submodel is itself represented by a dataflow graph that describes a process of solving an equation using a numerical root-finding algorithm (Brent's method). This yacht velocity model is only an approximation of a more accurate model of the yacht's motion. The more accurate model solves for velocity v and heel angle ϕ simultaneously using a pair of coupled torque balance and force balance equations. The coupling is due to the fact that $NetTorque$ actually depends on both ϕ and v, just as $NetForce$ depends on ϕ and v. The coupled model is generally more accurate; however, it takes longer to run. It is also more brittle than the uncoupled model since it uses a two-variable equation solver (Newton-Raphson) which more often fails to find a root than the two one-variable (Brent) equation solvers used in the uncoupled model.

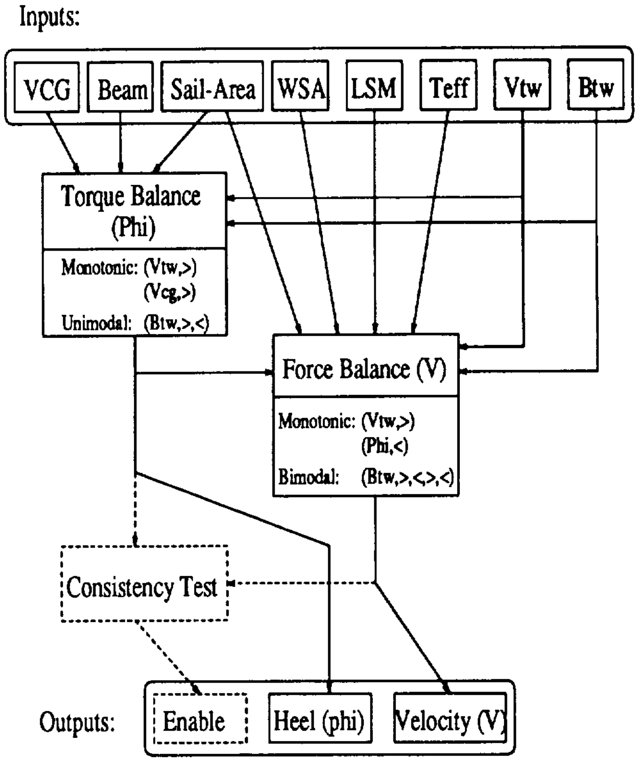

Figure 2: Yacht Velocity Model Dataflow Graph

The yacht model dataflow graph illustrates our approach to representing applicability tests as annotations to dataflow graphs. Some types of tests are general enough to apply to virtually any numerical model of a physical system. Examples of this type include testing whether a model exhibits a qualitative behavior that can be described in terms of monotonicity, unimodality or multimodality of the function computed by the model. These qualitative tests are represented as special slots appearing in each dataflow node object. For example, in the "Monotonicity" slot, an entry of the form $(Input, Sign)$ (where sign is one of $\{\geq, >, \leq, <\}$ indicates that the model's output is expected to be monotonic (increasing, strictly increasing, decreasing, strictly decreasing) in the named output. For example, the monotonicity slot in the "Force-Balance" object includes the entries $(V_{tw}, >)$ asserting that the velocity output v is expected to be a strictly increasing function of the wind speed V_{tw}. Whenever a model is executed, the execution procedure examines the monotonicity and modality slots, extracts descriptors of the expected qualitative behavior, and tests whether the current execution of the model is consistent with that behavior. The current execution is checked by examining a database of results of previous model executions and verifying that the current results bear the correct qualitative relationship to previous results.

Some types of tests are highly specialized, and apply only to a small number of models, perhaps only one model. We represent these tests as special "applicability checking nodes" that are directly wired into the dataflow graph. An example of this type is the "Consistency Test" node in the yacht model dataflow graph. The consistency test checks whether the decoupling of the torque balance and force balance equations is a good or bad approximation. It does so by evaluating the solution values of ϕ and v in the inequality $NetTorque(v, \phi) \leq K$. This test measures whether the approximate solution brings the net torque close enough to zero. By representing applicability tests as additional nodes in a dataflow graph, our system allows arbitrary computations to be used for applicability tests.

Although applicability checking nodes are represented in the same manner as the main stream of the computation, they are not handled in the same fashion by our model execution procedure. To begin with, applicability nodes are kept separate from ordinary nodes. Under the control of the user, the system can execute the entire graph, include applicability checks, (running in "restricted mode") or the system can execute only the subgraph representing the main stream of the computation (running in "unrestricted mode"). Furthermore, our execution procedure allows the applicability checking nodes to determine whether or not execution should be aborted in the event of an applicabilty failure. Outputs of applicability tests are typically routed to a special "Enable" input of other nodes. When an applicability test disables another node in the graph, all computations downstream of the test are aborted.

Initial tests of our system for controlling application of models have demonstrated two types of benefits. When provided with inputs that would previously have caused models to return erroneous results, our system returns an error condition indicating that the model is not applicable to the current input. The system thus avoids misleading the user with erroneous results. In addition, our system informs the user of which applicability tests failed and thus makes him aware of the reason the model does not apply to the current input. In the future, we plan develop tools for using such diagnostic information to support revision of scientific models to change or extend their ranges of applicability.

Summary

The model development toolbox and record is intended to support a variety of activities that occur in the course of developing scientific computation models. These activities include construction and testing of new models; controlled application of models to specific problems, and revision of models to handle new situations. The system is also expected to promote rapid development of new scientific computational models, more reliable use of scientific models among computational scientists; wider sharing of scientific models within communities of scientists; and deeper understanding among scientists of the assumptions and modeling techniques incorporated in the models they use. A more detailed description of this research is found in (Ellman 1993).

Acknowledgments

The research presented in this document is supported by NASA Grant NCC-2-802. It has also benefited from discussions with Richard Keller, Saul Amarel, Haym Hirsh, Lou Steinberg, Andrew Gelsey, John Keane and Mark Schwabacher.

References

Balzer, R. 1985. A 15 year perspective on automatic programming. *IEEE Transactions on Software Engineering* SE-11(11):1257–1268.

Ellman, T. 1993. A toolbox and record for scientific model development. In *Proceedings of the Symposium on Space Operations and Research.*

Keller, R., and Rimon, M. 1992. A knowledge-based software development environment for scientific model-building. In *Proceedings of the Seventh Knowledge-Based Software-Engineering Conference.*

Mostow, J. 1989. Design by derivational analogy: Issues in the automated replay of design plans. *Artificial Intelligence* 40:119 – 184.