1993

The Third International Conference on Expert Systems for Numerical Computing

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Robert Vichnevetsky

Report Number:
93-028

THE THIRD INTERNATIONAL CONFERENCE ON EXPERT SYSTEMS FOR NUMERICAL COMPUTING

Elias N. Houstis
John R. Rice
Robert Vichnevetsky

CSD-TR-93-028
May 1993

TABLE OF CONTENTS

SESSION 1: Expert Systems and Methodologies

SESSION 2: Knowledge Based User Interfaces

SESSION 3: Knowledge Based User Interfaces II

SESSION 4: Integration of Numeric/Symbolic/Geometric Computing

Session 5: Problem Solving Environment

A Development Shell for Cooperative Problem-Solving Environments

Francois Chevenet
URA CNRS 243
Universit Claude Bernard - Lyon 1
F-69622 Villeurbanne CEDEX, France

Francois Jean-Marie
Cap Gemini Innovation
7, Chemin de Vieut Chne
Zirst 4206

Jutta Willamowski
IRIMAG - LIFIA 46, av. Flix Viallet
F-38031 Grenoble CEDEX, France

Key-words: problem solving environment, object-centered knowledge representation, task representation, simple linear ordination.

Introduction

The concept of cooperative problem-solving environments is proposed here, in order to assist the user in the many choices that have to be made when using computers to solve problems in scientific applications. In the following, we define a problem solving environment as a knowledge-based system. Its knowledge base integrates all the knowledge necessary to solve problems in an application domain: knowledge about the types of problems to be solved, knowledge about problem-solving strategies, knowledge about the available scientific computer programs, and knowledge about the entities that are manipulated in the domain. Based on this knowledge, the problem-solving environment supports the user in choosing the elementary scientific computer programs that are necessary to solve his particular problem, and in chaining them one after another in an appropriate way. So far, this definition corresponds to the concepts presented in [Rech92]. The system might then solve automatically, on his own, complex problems, [Rech91]. But sub- problems might exist for which no solving strategy is known, or which can be solved only by the user, e.g. interpretation of graphics. Besides the solution for this kind of problems, parameter values are needed during

the problem-solving process, and to be obtained from the user. Therefore the system must be interactive. Especially in complex domains it is furthermore a fact that a knowledge base is never exhaustive; strategies might exist that are not integrated in the knowledge base, being not commonly used or not well defined. Therefore the system needs to be cooperative. The user must always have the possibility to escape the system s control, to modify the problem-solving strategy, or even to replace it by a completely new strategy. In the same way, the user should be able to reconsider at any moment his own decisions, e.g. concerning parameter values. Similar ideas are also presented in [Kant88]. The cooperative problem-solving environment should be able to manage the different versions corresponding to the modifications. Moreover the system should be open to different kinds of users, more or less competent in the application domain. Not every user has the same needs concerning the functions offered by the system. So the functioning of the system has to be easily adaptable to the user s needs, [Fisc90]. SCARP, a shell for developing such cooperative problem-solving environments has been developed. In SCARP, problem solving is based on successive decomposition of complex problems in more elementary subproblems. Its general architecture is presented here. Its components, the knowledge handler, the task engine, the cooperative dialogue handler, and the user interface are hereafter detailed. In the final version of this article an example of one of the applications of SCARP, SLOT, a cooperative problem-solving environment in simple linear ordination will be further described.

The Architecture

SCARP is structured in different layers. The whole architecture is domain independent. The core, the knowledge handler gives access to the knowledge base. The knowledge handler provides functionalities to exploit and to modify the knowledge contained in the domain specific knowledge base. The next layer, the task engine, uses the functionalities provided by the knowledge handler to extract the knowledge necessary for problem-solving (problem description, solving strategies, ...). It then manages the problem solving process according to the strategies defined in the knowledge base. Whenever the problem-solving process requires communications with the outside world, it activates the cooperative dialogue handler.

The system is structured in layers, the knowledge handler (KH), the task engine, the cooperative dialogue handler, the user interface and the system interface. Communication is necessary towards two directions, represented by the external layers user interface and system interface (the interface to the operating system). The user interface allows the system to cooperate with the user, whereas the system interface allows to control the execution of elementary programs.

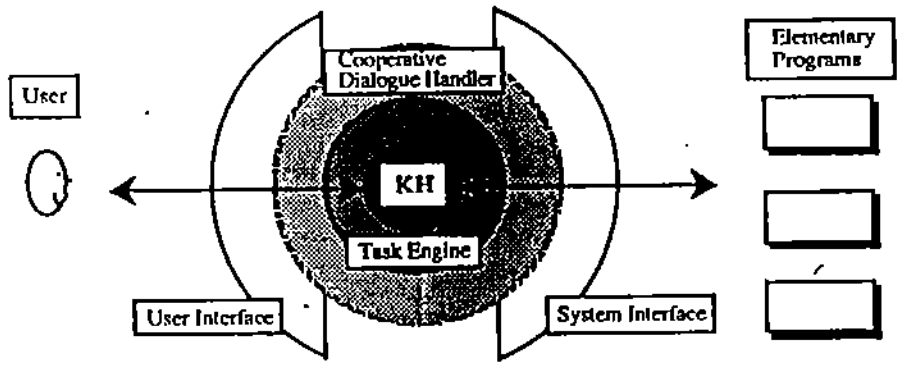

Figure 1: The system architecture

The Knowledge Handler

The knowledge handler is the center of the system. It provides access to the knowledge base. Three different types of knowledge are distinguished: knowledge concerning tasks, methods and entities. Tasks describe problems and associate problem-solving strategies if possible (in SLOT for example correspondence analysis or principal component analysis). Methods allow to integrate elementary programs and somehow their instructions of use into the system (in SLOT for example calculus on matrices or of graphical representations). Entities represent the objects manipulated in the domain (in SLOT matrices and graphical objects). In SCARP the complete knowledge is represented as objects. The knowledge concerning entities is represented using SHIRKA, [Rech89], an object-centered knowledge base management system, based on a notion similar to frames [Fike85]. To represent the knowledge concerning methods and tasks a specific model has been defined for SCARP. For such a problem solving environment the essential part of the knowledge concerns the tasks. Different approaches to task-oriented modeling are discussed in [Chan92] and the importance of a declarative task description was advocated in [Smit84]. We define a task as a model of a problem and its solution, defining a specific set of input and output entities. One problem might be defined at different abstraction levels by different, more or less specific task descriptions. At a low abstraction level, the problem defined by a task is sufficiently precise, so that a problem solving strategy can be associated to the task description. The set of input and output entities and the problem solving strategy constitute the most essential

parts of a task definition. Within SCARP, the following types of attributes are defined for a task: global / strategic input / output entities, pre-tasks, post-tasks, visualization tasks for input and output, preconditions, postconditions, control flow and subtasks. The control flow describes how the subtasks are to be chained to solve the complex task itself. Different operators are available to combine subtasks: sequence, parallel, iteration, choice and user-task. A subtask is a user-task if the system has no strategy to solve it, and if the user indicates the result of the task. For each subtask a specific execution-context describes how its execution is integrated into the execution to the complex task (data flow, wether the user has to validate input or output or not, wether the task is optional, ...). The definite version of this article will show how the domain knowledge for simple linear ordination was formalized in SLOT.

The Task Engine

The task engine exploits the knowledge base via the knowledge handler for automatically solving complex problems. Different phases are distinguished in the functioning of the task engine. First, the global input entities are determined; then, according to the characteristics of these input entities, the solving strategy is chosen; afterwards, the missing strategic input entities or parameters are obtained; if required by the execution-context, a general view of all the input entities is shown to the user; then, the solving strategy associated to the problem description is executed; next, the global output entities are synthesized; finally, if required by the execution-context, a general view of all the output entities is shown to the user. Executing the solving strategy means to decompose the complex task into more elementary subtasks, according to the control flow. Each of the subtasks is integrated in the global problem solving process as defined in the execution-context that links the subtask to the complex task. Each subtask is in turn and in the same manner executed by the task engine. Executing a task might necessitate communication with the outside, either because it indicates an elementary program to execute (a communication via the system interface), or because parameters have to be fixed by the user, or in order to ask the user to execute a user-task (a communication via the user interface). Then the task engine interrupts itself and activates the cooperative dialogue handler. The information necessary to reactivate the task engine at the end of the required communication is stored.

The Cooperative Dialogue Handler

The cooperative dialogue handler manages all the communications between the task engine and the outside world. It is usually activated by the task engine that needs information from the user or to execute an elementary program. The dialogue handler then executes this request; it initiates a specific dialogue with the user or activates the required program. In the end of the dialogue or the program execution it sends the result of the communication to the task engine, and reactivates it in order to continue the problem solving process. This is the normal case : the dialogue handler is activated by the task engine, that is itself interrupted during the dialogue, expecting the result. But the cooperative dialogue handler can also be activated by the user, while the task engine is working and not aware of the dialogue. This is where cooperative abilities are required. Via the user interface the user always follows the problem solving process. He might at any moment decide to modify its characteristics, either parameter values or strategic choices made previously. The user then activates the dialogue handler, engaging a modification dialogue. In the end he validates the modifications. The dialogue handler then interrupts the task engine. It creates a new version of the current problem solving process, copying unmodified parts of the precedent version. Finally it reactivates the task engine with this new, modified version. The task engine will then reexecute the part of the problem solving process which is in relation with the modification.

The User Interface

To enable the user to follow the problem solving process and to interact with the dialogue handler an appropriate user interface is essential. The user interface of our system always shows graphically the current state of the problem solving process, i.e. the current task decomposition tree. It indicates which subtasks are already executed, and if they were successfully executed. Furthermore any decision point is marked and can be inspected by the user. Via the interface, the user has access to all information necessary to evaluate the correctness of the problem solving process. These are for example the input / output entities of a task and its existing versions. To engage in a modification dialogue with the dialogue handler the user then simply selects the relevant information with the mouse and modifies it.

The user interface is structured in three main parts, showing the problem definition (upper left), the existing versions (upper right), and the actual state of the problem solving process (lower part).

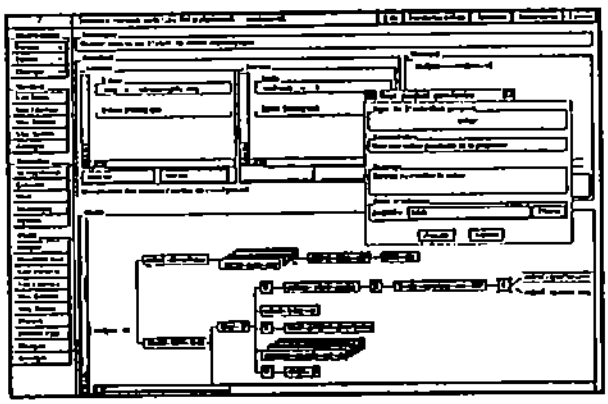

Figure 2: The user interface

Problem Solving With SLOT, an Example

This chapter will show the cooperative problem solving process using an example session with SLOT. It will detail on one hand the autonomous work of the task engine, and on the other hand the cooperation with the user.

Conclusion

The main characteristics of our system are that it is generic and mixed-initiative. It is generic because the problem solving environment shell is not domain dependent. To develop a domain specific problem solving environment, the domain specific knowledge has only to be formalized in the system-specific way, in the same way as it was done for SLOT. The rest of the system s architecture is domain independent. The system is mixed-initiative because not only the system but also the user can at any moment activate a dialogue process. This is the great difference between our system and traditional expert system shells. Problem solving environments developed with the presented shell are not dedicated to a specific user group. Beginners might limit the use of the system s cooperative facilities, while experts might fully exploit these facilities and explore new problem solving strategies. Another important point, not yet detailed in this article, is that these newly explored strategies may be stored as macros , i.e. new tasks, in the system s knowledge base.

References

[Chan92] Chandrasekaran B., Johnson T.R., and Smith J. W., "Task structure analysis for knowledge modeling," *Communications of the ACM*, September 92, pp. 124-137.

[Fike85] Fikes R. and Kehler T., "The role of frame-based representation in reasoning," *Communications of the ACM*, September 1985, Vol. 28, No 9.

[Fisc90] Fischer G., "Communication requirements for cooperative problem solving systems, Information Systems," Vol. 15, No. 1, pp. 21-36, 1990.

kKant88] Kant E., "Interactive problem solving," *IEEE Expert*, 1988, IEEE Expert, Vol. 3, No. 4, pp. 36-49.

[Rech89] Rechenmann F. and Uvietta P., "Shirka: An object-centered knowledge bases management system," *Workshop Intelligence Artificielle en Simulation Num rique et Symbolique* , SCS Europe et Laboratoire de Biom trie, G n tique et Biologie des Populations, Ly March, pp. 22-24, 1989.

[Rech91] Rechenmann F., "Modeling tasks and methods in a knowledge-based PDE solver," *IMACS '91*, June 22-26, 1991, Dublin.

[Rech92] Rechenmann F. and Rousseau B., "A development shell for knowledge-based systems in scientific computing," Houstis E.N., Rice J.R. (eds), in *Expert Systems for Numerical Computing*, Elsevier Science Publishers, 1992.

[Smit84] Smith R.G., Lafue G.M.E., Schoen E., and Vestal S.C., "Declarative task description as a user-interface structuring mechanism," *IEEE Computer*, September, 1984.

PYTHIA: An Expert System for the Support of "Smart" Parallel PDE Solvers

Elias N. Housits, Shahani M. Weerawarana
John R. Rice and Sanjiva Weerawarana[1]
Department of Computer Sciences
Purdue University
West Lafayette, IN 47906

C.E. Houstis
University of Crete
Department of Computer Science
Heraklion, Greece.

Introduction

The computing scenario of the 90's will be dominated by the following computer organizations:

(i) Parallel MIMD systems with hundreds to thousands processors executing k Million Instructions Per Second (k MIPS).

(ii) Parallel MIMD systems with tens to hundreds processors executing Billions of Instructions Per Second (BIPS).

(iii) Heterogeneous networks of supercomputers.

Unlike for sequential vector processors, algorithm implementation on these complex high performance architectures is not logically consistent with traditional programming approaches. Thus, the scientists/engineers need new tools and algorithms capable of exploring the high potential performance of these systems.

In every algorithmic strategy one can identify three types of statements: the declarative, procedural and logical. They describe the "what", "how", and "when" of an algorithm's execution in their computational environment. The description of most existing algorithms

[1]Work supported in part by AFOSR grant 91-F49620, NSF grant CCR 86-19817.

includes a set of parameters whose determination is the responsibility of the user. These parameters are usually related to storage, computational behavior, and parallel processing issues. The efficiency and speed up of these algorithms depends very much on the choice of these parameters. The determination of most of these parameters is a "hard" problem and requires extensive knowledge of the algorithm and machine combination. Its solution can be viewed as a search process through a solution space, usually a high dimensional one.

In the case of PDE (Partial Differential Equations) computations the principal user parameters relate to the selection of (grid, method (discretization-indexing-solver)) and (configuration, machine) pairs so that certain *a priori* defined computational objectives (i.e., accuracy, time, memory) are satisfied. The speed–up and efficiency of most of the existing and proposed parallel PDE solvers cannot be predicted without run-time information. Furthermore, due to the large number of parameters, the size of the solution space is sufficiently large to require the use of AI techniques to search for the "optimal" combination. In this paper we report on the structure and methodology of an expert system called PYTHIA that attempts to select optimal (solver, machine) pairs and their parameters so as to optimize the user's computational objectives transparently to the user. PYTHIA attempts to use the concept of *meta–algorithms* or *smart–algorithms* to create a simpler logical equivalent of the algorithm and the associated knowledge. PYTHIA's unique characteristic is the way it acquires knowledge. It primarily relies on the collection of a set of run time performance data to which a stochastic process is applied to derive rules and possible profiles for efficiency, time and memory. This is done for each algorithm involved.

In this paper, we provide an overview of PYTHIA's architecture, describe briefly some of its components, and present some preliminary results.

The Architecture of PYTHIA

The PYTHIA environment consists of two distinct components; the knowledge engineering environment where rules are generated and the application user environment where an end–user uses PYTHIA in an advisory capacity.

The knowledge engineering component uses the ELLPACK Performance Evaluation System to execute test problems, generate performance profiles and to execute the FORTRAN statistical analyzer. These profiles are then converted to knowledge base facts by PERL scripts. The facts themselves are represented as Common Lisp structures and/or OPS5 rules and stored in a knowledge base for later use. Figure 1 shows the structure of this component. The advisory expert system component of PYTHIA is being implemented as an editor within //ELLPACK in OPS5, Common Lisp and C. The user interface is an X–based

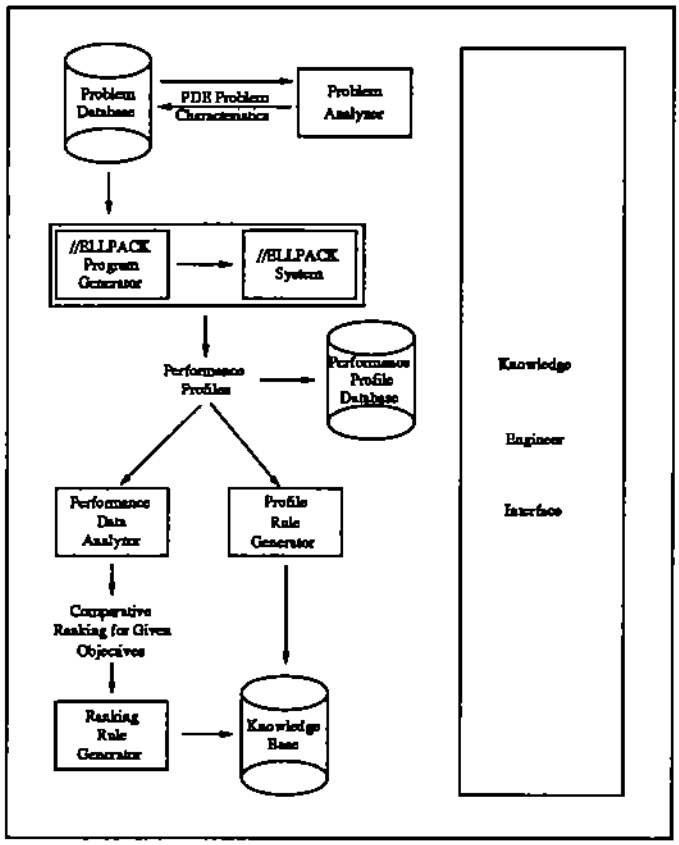

Figure 1: The knowledge engineer view of PYTHIA

tool that provides the user with convenient mechanisms for specifying the problem features (either manually or via the problem analyzer) and performance objectives and integrates PYTHIA's predictions into the //ELLPACK program being developed by the user. Figure 2 shows the structure of this component.

Problem Features

We classify problem features in terms of the following components: (i) PDE operator, (ii) PDE equation right hand side, (iii) solution, (iv) boundary conditions, (v) initial conditions, and (vi) domain of definition. Thus, the set of characteristics of a PDE problem includes

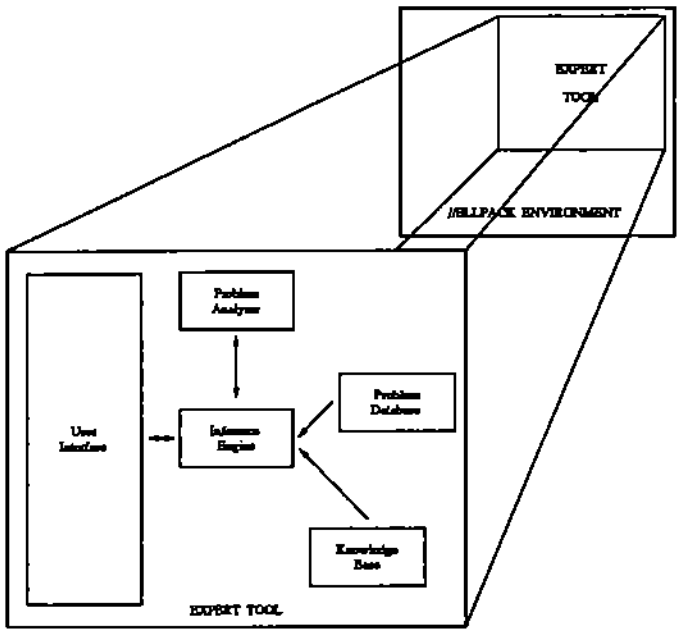

Figure 2: The application user view of PYTHIA

some classification information for the above components and some quantitative information about the behavior (i.e., smoothness and local variation) of the functions (i.e., coefficients of the operators, right side of the operator equations, and the solution) of the PDE problem. In the PYTHIA environment, a characteristic vector v is created for each user's problem which collects the values associated with these features. It is determined automatically through symbolic processing of the PDE problem specification and by symbolic/numeric processing of its functions. Features that cannot be ascertained automatically must be provided by the user or from simple defaults.

Problem features in the characteristic vector v are represented by a data structure that contains slots for boolean quantities (for example, whether the operator is self–adjoint or not) as well as numerical quantities (for example, the smoothness and local variation of the boundary conditions). Also, if any performance data is available for the problem, then that information is kept in another slot as a list of (method, performance profile) tuples. The characteristic vector of a problem is then a vector that lists the values of all the boolean and numerical slots of a problem structure.

The Rules

The knowledge base contains four classes of rules.

The first are a priori rules provided by a human expert and the rest are the rules generated by PYTHIA. We expect that the number of human–given rules will be small as many hard to measure quantities (such as the efficiency of the implementation) play a significant role in the performance of algorithms.

The second class of rules refer to uniprocessors (sequential and vector) and they are generated by deliberate actions of the knowledge engineer. The knowledge engineer "trains" PYTHIA by executing many test problems then allowing PYTHIA to gather interesting performance data from them. These performance profiles are stored as facts in the knowledge base. A statistical analyzer is also used to analyze performance data in order to rank the performance of various methods on one problem and also the performance of one method on many problems of belonging to one "class." The basic variable in these rules is the *error level* requested by the user. For example, if the user requests 2% accuracy, then these rules can relate this error level to the fineness of a grid, the number of iterations for a particular method, etc.. The knowledge engineer identifies a class of problems and requests PYTHIA to generate rules for that class using applicable performance profiles. Thus, there are two sub–types of rules in this class generated by PYTHIA: The first type of rules indicate which method works best for a given class of problems at each error level while the second type of rules indicate with method works best for a given problem at each error level.

The third class of rules refer to each individual multiprocessor machine and these are generated in a manner similar to the above actions of the knowledge engineer. Their objective is to assist in the selection of "optimal" (grid, configuration, method) triples for a particular target parallel machine. These rules are based on the solver performance data collected by the the //ELLPACK system.

The fourth class of rules assists in the selection of the most efficient machine out of a specified set of machines for which the library of PDE solvers is available. The resources and the ranking of these machines are matched with some of the user specified computational objectives and a selection is made to best satisfy the user's objectives.

The Inference Algorithm

The primary approach is to use the exemplar or case based methodology. Some general rules are used which aid the efficiency of the process. For example, there are some common simple problems where the best choices are immediately known.

Given the user's problem, PYTHIA first attempts to find the class of a priori defined problems that is "closest" by matching the properties of the problem with properties of the class. Once the closest class has been found, if class rules are available, then those can be used to provide information to the user. Otherwise, PYTHIA finds the closest exemplar within that class and then uses the rules for that problem to provide the necessary information to the user.

Preliminary Studies

We used the class of problems studied in [HR82] to evaluate the PYTHIA. Let C be that class of problems. First, we used the first 10 problems in C (or, 25% of C) as a "training set" to make performance predictions for the remaining problems in C. Then we used the first 20 problems in C (or, 50% of C) as the training set and as before, made predictions for the remaining problems. The Table below shows the results of this test. Each percentage indicates the percentage of correct predictions for each (training set, relative error level) pair. The "Number Predicted" is the number of valid predictions made by PYTHIA.[1]

Training Set	Number Predicted	Relative Error		
		0.1%	0.01%	0.001%
$\frac{1}{4}C$	37	76%	73%	57%
$\frac{1}{2}C$	19	95%	89%	58%

In the next test, we wish to observe the effect of a larger training set on the correctness of predictions on the same set of problems. Hence, we used the same training sets as above, but, instead of making predictions for all remaining problems for each training set (i.e., from problems 11 and 21 onwards, respectively), in both cases, we made predictions for just problems 21 onwards. The table below shows these results.

Training Set	Number Predicted	Relative Error		
		0.1%	0.01%	0.001%
$\frac{1}{4}C$	7	86%	86%	100%
$\frac{1}{2}C$	12	100%	92%	33%

[1] A prediction is not valid if other factors, such as method applicability, are not satisfied.

Conclusions

Although the above tests are extremely preliminary, we observe that PYTHIA does behave quite well. We hope to closely couple PYTHIA with the //ELLPACK environment in order to allow it to learn each time a problem is solved by a user. This will allow PYTHIA to improve with usage and make its predictions even more accurate.

References

[HR82] E.N. Houstis and J.R. Rice, "High order methods for elliptic partial differential equations with singularities," *International Journal for Numerical Methods in Engineering*, 18:737-754, 1982.

[HRC+90] E.N. Houstis, J.R. Rice, N.P. Chrisochoides, H.C. Karathanasis, P.N. Papachiou, M.K. Samartzis, E.A. Vavalis, Ko-Yang Wang, and S. Weerawarana, "//ELLPACK: A numerical simulation programming environment for parallel MIMD machines," J. Sopka (ed.), in *Proceedings of Supercomputing '90*, pages 96-107. ACM Press, 1990.

Artificial Intelligence Support for Scientific Model-Building

Richard M. Keller
Artificial Intelligence Research Branch
NASA Ames Research Center

Although computer models play a crucial role in the conduct of science today, scientists lack adequate software engineering tools to facilitate the construction, maintenance, and reuse of modeling software. Usually, scientific models are implemented using a general-purpose computer programming language, such as Fortran. Because this type of general-purpose language is not specifically customized for scientific modeling problems, the scientist is forced to translate scientific constructs into general-purpose programming constructs in order to implement a model. This manual translation process can be very complicated, labor-intensive, and error-prone. Furthermore, the translation process obfuscates the original scientific intent behind the model, and buries important assumptions in the program code that should remain explicit. The resulting software is typically complex, idiosyncratic, and difficult for anyone but the original model developers to understand.

At NASA Ames Research Center, we are building a knowledge-based software environment that makes it easier for scientists to construct, modify, and share scientific models of physical systems. Examples of such models include planetary atmosphere models, ecosystem models, and biochemical process models. The SIGMA (Scientists' Intelligent Graphical Modeling Assistant) system functions as an intelligent assistant to the scientist. Rather than construct models using a conventional programming language, scientists use SIGMA's graphical interface to "program" visually using a high-level data flow modeling language. The terms in this modeling language involve scientific concepts (e.g., physical quantities, scientific equations, and data sets) rather than general programming concepts (e.g., arrays, loops, counters). SIGMA assists the scientist during the model-building process and checks the model for consistency and coherency as it is being constructed. Then SIGMA automatically translates the conceptual model into an executable program, freeing the scientists from error-prone implementation details.

To provide this level of automation, the system must be given a significant amount of knowledge about the scientific domain, as well as general knowledge about programming. In our system, knowledge is represented and stored in a large knowledge base that contains information about scientific equations, physical quantities and constants, scientific units, numerical programming methods, and scientific domain concepts and relations. We have

invested considerable time and energy into representing scientific knowledge in a form that is reusable across a variety of scientific disciplines.

To ground our work, we have been conducting a case study of a planetary atmosphere model developed (in Fortran) by physicist and collaborator Christopher McKay. His model uses scientific data from Voyager I's flyby of Titan to infer the thermal and radiative structure of Titan's atmosphere. Our methodology has been to "reverse-engineer" the Fortran model in order to reconstruct the knowledge and methods used by McKay to build such a model. Based on our analysis, we are developing a general tool that will ultimately enable atmospheric scientists to construct an entire class of atmospheric models – including McKay's Titan model – using our automated modeling assistant.

To avoid developing a system that is too narrowly scoped, we also have been studying models in other scientific domains. In this way, we increase confidence that our efforts are general and transferable to other scientific disciplines. For example, we have recently extended our knowledge base to support modeling activities associated with the Forest-BGC ecosystem model. This forest carbon-cycle and water-cycle model was developed by ecologists S. Running and J. Coughlin at the University of Montana, and is used by collaborator Jennifer Dungan and other researchers at NASA. Dungan's goal is to use SIGMA to build a generalized version of Forest-BGC that can be used to investigate regional-scale forest ecosystem phenomena.

In the current version of SIGMA, we conceptualize model-building as a kind of derivation process. Given some initial configuration state of the physical system being modeled, model-building can be viewed as a process of deriving a set of unknown physical variables from the known variables in the initial state. The process of deriving unknown quantities is accomplished via a series of computational transformations that bridge the gap between the known and unknown variables. Two kinds of computational transformations currently supported in the system are equations (algebraic and ordinary differential) and Fortran subroutines.

The process of deriving unknown variables is accomplished via a simple backchaining procedure. In this backchaining process, the user first selects an unknown physical variable they wish to calculate. Then the system presents the user with a limited set of transformations that can be used to derive the desired variable. This set is determined based on the system's semantic understanding of when each transformation is applicable, and on its ability to match the applicability conditions to the current state. The user next selects one of the suggested transformations, and the system checks to see whether all the input variables required by this transformation are already known. If so, the desired output variable is computed using the selected transformation; if not, the backchaining process recurses for each of the unknown variables. During this process, the graphical interface displays a visualization

of the current model as it is being built.

Our long term plan is to make SIGMA a viable tool to support future NASA planetary and earth-based activities, including those related to the upcoming Huygens probe of Titan's atmosphere, as well as the modeling activities associated with data collected by the Earth Observing System (EOS) network of satellites. SIGMA is implemented in CommonLisp on a Sun SPARC station 2, and features a Motif-based graphical user interface.

References

[1] R.M. Keller, "Artificial intelligence support for scientific model-building," *Proc. AAAI Fall Symposium on Intelligent Scientific Computation*, Cambridge, MA, Oct. 1992.

[2] R.M. Keller and M. Rimon, "A knowledge-based software development environment for scientific model-building," *7th Knowledge- Based Software Engineering Conference (KBSE-92)*, Tysons Corner, VA, September 1992.

[3] R.M. Keller, "Knowledge-intensive software design: can too much knowledge be a burden?," *Proc. AAAI-92 Workshop on Automating Software Design* San Jose, CA, July 1992.

[4] J.L. Dungan and R. Keller, "Development of an advanced software tool for ecosystem simulation modelling," *Abstracts supplement of the Bulletin of the Ecological Society of America*, 72(2) p. 104, 1991.

[5] R.M Keller, "The scientific modeling assistant: an interactive scientific model-building tool," *Proc. AAAI-91 Workshop on Automating Software Design*, Anaheim, CA, July 1991.

[6] R.M. Keller, M.H. Sims, E. Podolak, and C.P. McKay, "Constructing an advanced software tool for planetary atmospheric modeling," *Proc. i- SAIRASU90 (International Symposium on Artificial Intelligence*, Robotics and Automation in Space), Kobe, Japan, November 1990.

* This abstract excerpted from paper published in the AAAI Fall Symposium on Intelligent Scientific Computation, Cambridge, MA.

A Case based Reasoning Approach to Mesh Specification for Adaptive Finite Element Analysis

Neil Hurley
Hitachi Dublin Laboratory
O'Reilly Institute
Trinity College
Dublin 2, Ireland

Introduction

One of the most difficult tasks for users of finite element software is to set the spatial mesh over the geometrical domain. Specifying mesh densities which capture the behaviour of local phenomena can typically become a process of trial and error. Much current research in finite element analysis is being concentrated on the development of error estimators and adaptive mesh strategies to alleviate this problem. However, the form of the initial mesh can greatly influence the rate of convergence, and hence the choosing of this mesh is highly important. *A priori* strategies are often heuristic in nature, depending largely on the skill and experience of the analyst. Also, while theoretical analysis of mesh adaptation strategies has had some success in model problems, the issue of which strategy to employ in a given real engineering problem is still unresolved. Again experience and heuristics are required to formulate the best strategy. The usefulness of knowledge-based techniques and tools to assist this process has been recognised ([Andrews, 1988], [Shepherd, 1988]).

This paper describes the application of case based reasoning techniques to mesh generation and adaptation. Given a differential equation problem to solve, the system forms a solution strategy by accessing a case base of previously solved problems, and matching the current problem with similar solved problems. Numerical modules then simulate, test and, if necessary, adapt the solution until a sufficient accuracy is obtained. The final result is added as a new case to the case base (see Figure 1). The case base serves to augment *a priori* knowledge of a given problem, by making available knowledge gained through *a posteriori* analysis of previously solved, similar problems.

In the current implementation, we are considering mesh generation for two-dimensional, steady state, second order partial differential equation problems.

Beyond Rule-based Expert Systems

While the rule-based expert system has been very successfully applied to many domains, in recent years AI researchers have recognised certain limitations : large production systems tend to become unmanageable, and maintenance of a consistent rule set increasingly difficult; rules of combination of uncertain information using confidence factors or other similar measures, are difficult to justify and can produce questionable results. More sophisticated AI techniques are being researched, which attempt to model human reasoning more closely. One of the most promising of these is *case based reasoning* [Shank, 1990], and this is now being applied to complex application domains. Case based reasoning systems model human reasoning by forming solutions through the retrieval and adaptation of successful strategies used in the past. Thus, they use past experience, rather than working always from first principles. The result is a dynamic system, which is constantly up-dating its knowledge from experience gained through solving problems.

We believe that the essential advantage of case based reasoning is its flexibility. The case base is not tightly bound to the compiled knowledge of rules; rather, it is free to postulate solutions based on less restrictive matching routines. These search only for similarities without requiring these similarities to be grounded in domain knowledge. There is therefore more potential for unexpected associations to be found, and, if we accept that similar situations produce similar results, these associations should not generally be irrelevant. Furthermore, by coupling the system with an adaptive simulation engine, we are including a module which can reject any bad assertions made. It must be accepted however, that reasoning by example alone can lead at least to inefficiencies, when commonly encountered situations could be better expressed as a rule. The ideal may be to integrate a case based and rule based approach, with the case base serving as a rule generator, or additional knowledge source, when inferencing fails.

Related Work

Knowledge-based mesh generation has been considered by a number of researchers, although typically research has concentrated on the development of rule-based systems, for example, [Andrews, 1988], [Rank, 1987]. CAD applications research, for example, [Vorozhtsov, 1989], has focused almost exclusively on how the geometrical shape of the domain affects the phenomena being analysed over the domain, without taking full account of the mathematical model which describes the phenomena. [Kang & Haghighi, 1992] presents an expert system based on a blackboard architecture which makes use of boundary and loading conditions as

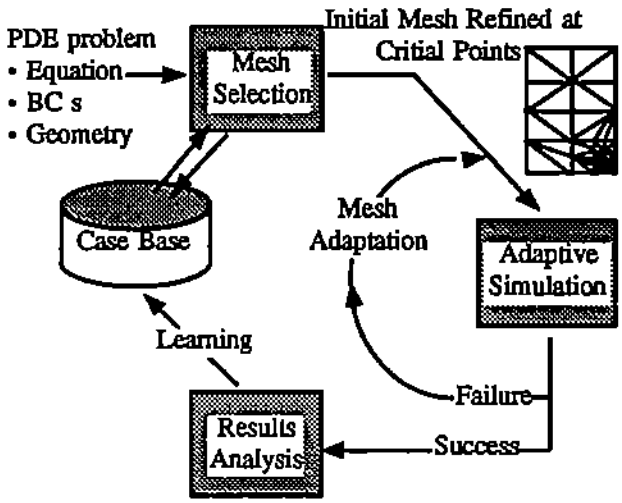

Figure 1:

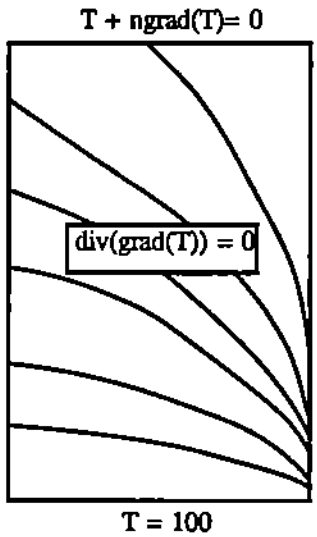

Figure 2:

well as object geometry. Finally, [Macedo et al., 1990] presents a methodology for the selection of activity indicators to be used by local mesh refinement schemes. We would argue that our approach is fundamentally different from any of these, since we are using an AI strategy which includes learning, and we are integrating this strategy with sophisticated numerical mesh adaptation techniques.

System Overview

Figure 1 shows the overall system. The AI preprocessor accesses a case base to generate an initial mesh, which has been refined in areas of expected high error concentration. The adaptive simulation engine uses *h-adaptation* , based on an element-wise error estimate generated using the Zienkiewicz-Zhu post-processing technique [Zienkiewicz & Zhu, 1987]. The results analysis module locates the important features in the solution, links these with the input features which caused them, to form an indexed case which is then placed in the case base for use in future problem solving.

Although the geometrical domain greatly influences the form of the final solution, it is not the only influence. Boundary and load conditions, for example, may produce affects which are harder to predict *a priori*. This research is concentrating on the interactions between such features in the mathematical model. Features are categorised into three subcategories : geometrical features, equation features and boundary condition features and matches between cases are considered on the basis of each. For example, Figure 2 shows a simple diffusion problem, the solution of which exhibits a high gradient in the lower right hand corner. Analysis of the problem results in indexing this case by the problem features of two boundary conditions of different kinds, connected at the corner. The index is a *relationship* between features, rather than just the features themselves.

A fundamental issue tackled in this research is case representation. A case consists of a problem together with its solution in terms of appropriate mesh densities. The representation must make similarities between problems explicit, since large amounts of computation to find these similarities is undesirable. Most solution behaviours result not from one specific problem feature, but rather from the interaction of several; so relationships between features must also be explicit. A frame based representation is used. Each case consists of a network of feature attributes, linked through geometric connectives. Thus structural mappings between cases are enabled, which match not just individual features, but networks of related features. Features and feature relationships are stored in frame taxonomies. Case retrieval is achieved by a process of generalizing up the hierarchy and specialising to match the target.

References

Andrews 1988 A.E. Andrews, "Progress and challenges in the application of artificial intelligence to computational fluid dynamics," *AIAA Journal*, Vol. 26, No. 1, 1988.

Kang&Haghighi 1992 Kang and K. Haghighi, "A knowledge-based a-priori approach to Mesh Generation in Thermal Problems," *International Journal for Numerical Methods in Engineering*, Vol. 35, pp. 915-937, 1992.

Macedo et al 1990 C.G. Macedo,Jr., J.C. Diaz, and R.E. Ewing, "A knowledge-based system for the determination of activity indicators for self-adaptive grid methods," E.N. Houstis et al,(eds.), in *Intelligent Mathematical Software Systems*, pp. 133-141, North-Holland, 1990.

Rank 1987 E. Rank and I. Babuska, "An expert system for the optimal mesh design in the hp-version of the finite element method," *International Journal for Numerical Methods in Engineering*, Vol. 24, pp. 2087-2107, 1987.

Shank 1990 "Inside case based reasoning," Lawrence, Erlbaum Associates.

Shepherd 1988 Shepherd, "Approaches to the automatic generation and control of finite element meshes," *Applied Mechanics Reviews Vol*, 41 No. 4, 1988.

Vorozhtsov 1989 E.V. Vorozhtsov, "On the classification of discontinuities by the pattern recognition methods," *Computers and Fluids*, Vol. 18, No. 1, pp. 35-74, 1990.

Zienkiewicz&Zhu 1987 O.C. Zienkiewiez and J.Z. Zhu, "A simple error estimator and adaptive procedure for practical engineering analysis," *International Journal for Numerical Methods in Engineering*, Vol. 24, pp. 337-357, 1987.

The Architecture of an Intelligent Virtual Mathematical Software Repository System

Ronald F. Boisvert
Computing and Applied Mathematics Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

Introduction

Users of modern scientific computing facilities have substantial collections of mathematical and statistical problem-solving tools at their disposal. These may be installed for use on a central mainframe, installed for access on a personal workstation from a local file server, or accessible from an external repository via anonymous FTP, a network server, or electronic mail. The average user with a particular mathematical problem to solve has a difficult time dealing with this bounty. The user must first be familiar with available packages, where they are located, and how they are accessed, and then must engage in a tedious and error-prone search for the subset that may be applicable to the problem at hand. Users have a greater chance for success if they appeal to a central information resource for help in locating software.

Even if the user knows where to look for candidate software there are still significant difficulties to overcome. Existing software repository systems have at best only primitive directory services. Potential users often lack the mathematical knowledge to be able to translate vague problem descriptions into the precise jargon associated with mathematical software abstracts. The terminology barrier also makes it difficult for users to discriminate among software for solving similar problems. Users need state-of-the-art directory services supported by readily-available expert consultants.

The Guide to Available Mathematical Software (GAMS) is a centralized mathematical software resource maintained for scientists at the National Institute of Standards and Technology (NIST). The GAMS system contains information about more than 50 mathematical and statistical software packages available from a four physically-distributed software repositories, including *netlib* [2]. Users access GAMS anonymously by logging in to a server system on the Internet. Searches based upon problem class, keyword, or name may be used to locate appropriate software, after which items such as abstracts, documentation and source code can be retrieved. Both generic and graphical user interfaces are available. GAMS functions like a management and retrieval system for a large central software repository. However,

instead of maintaining the cataloged software itself, it provides central access to multiple repositories managed by others. We use the term *virtual software repository* to describe such systems.

In this paper we outline the evolving architecture for the GAMS system. Both its current status as well as plans for the incorporation of automated domain consultants and restructing based upon a client-server approach will be described.

The GAMS Software Reuse Model

The GAMS virtual software repository is a collection of software modules. A *module* is a reusable piece of software that solves a well-defined problem. Modules are grouped together in a natural way to form *packages*. Associated with each module and package is a set of objects such as abstracts, documentation, source, examples, and tests. Object types are not predefined, so that objects unique to a particular module or package are easily accommodated. Packages are made available at one or more *repository sites*. Each site may have a different access mechanisms. For example, GAMS obtains *netlib* software using an *xnetlib* client program that downloads software from Oak Ridge National Laboratory on demand.

Support for Directory Services

An important component of our model is a taxonomy of mathematical and statistical problems which is used to classify software modules. The taxonomy we use is the GAMS Problem Classification System [1]. This taxonomy is now in widespread use, having been adopted by both library developers and scientific computing centers worldwide. In addition, we maintain a keyword index to the problem taxonomy. The index functions as a thesaurus, allowing one to map alternate terminology into the taxonomy.

The User Interface

Both a simple ASCII interface and a network-based X11 graphical user interface are supported. (See Figure 1 for a sample X11 window.)

Both are available for anonymous use on the Internet. During its first seven months of operation more than 1400 uses were logged from well over 100 Internet hosts. We encourage external use of GAMS on an experimental basis in order to get additional user feedback on the system's features and user interface. To try the system, telnet to linden.cam.nist.gov (129.6.80.116) and present username **gams** (for the generic interface) or **xgams** (for the X11 interface).

Expert Extensions

The GAMS Problem Classification System provides a rudimentary method of codifying mathematical knowledge necessary for software selection. However, it is not detailed enough to deal with situations in which many similar software modules are available for a particular problem. In this case much more detailed knowledge of the particular subdomain and the candidate software is necessary in order to provide users with guidance. We will describe plans for incorporating such domain expert subsystems into GAMS.

Client-Server Approach

The GAMS system already functions as a client of other repository systems such as *netlib*. GAMS itself is currently being restructured so that it can function as a network server. The system has already been split so that the user interface is logically separate from the GAMS core, and we are currently defining a simple ASCII protocol through which these layers can communicate. We are also experimenting with a modified version of WAIS to provide the network transport for this client-server approach.

References

[1] R.F. Boisvert, S.E. Howe, and D.K. Kahaner, "The Guide to Available Mathematical Software problem classification system," *Comm. Stat. - Simul. Comp.*, 20(4):811-842, 1991.

[2] J.J. Dongarra, and E.Grosse, "Distribution of mathematical software via electronic mail," *Comm. ACM*, 30:403-407, 1987.

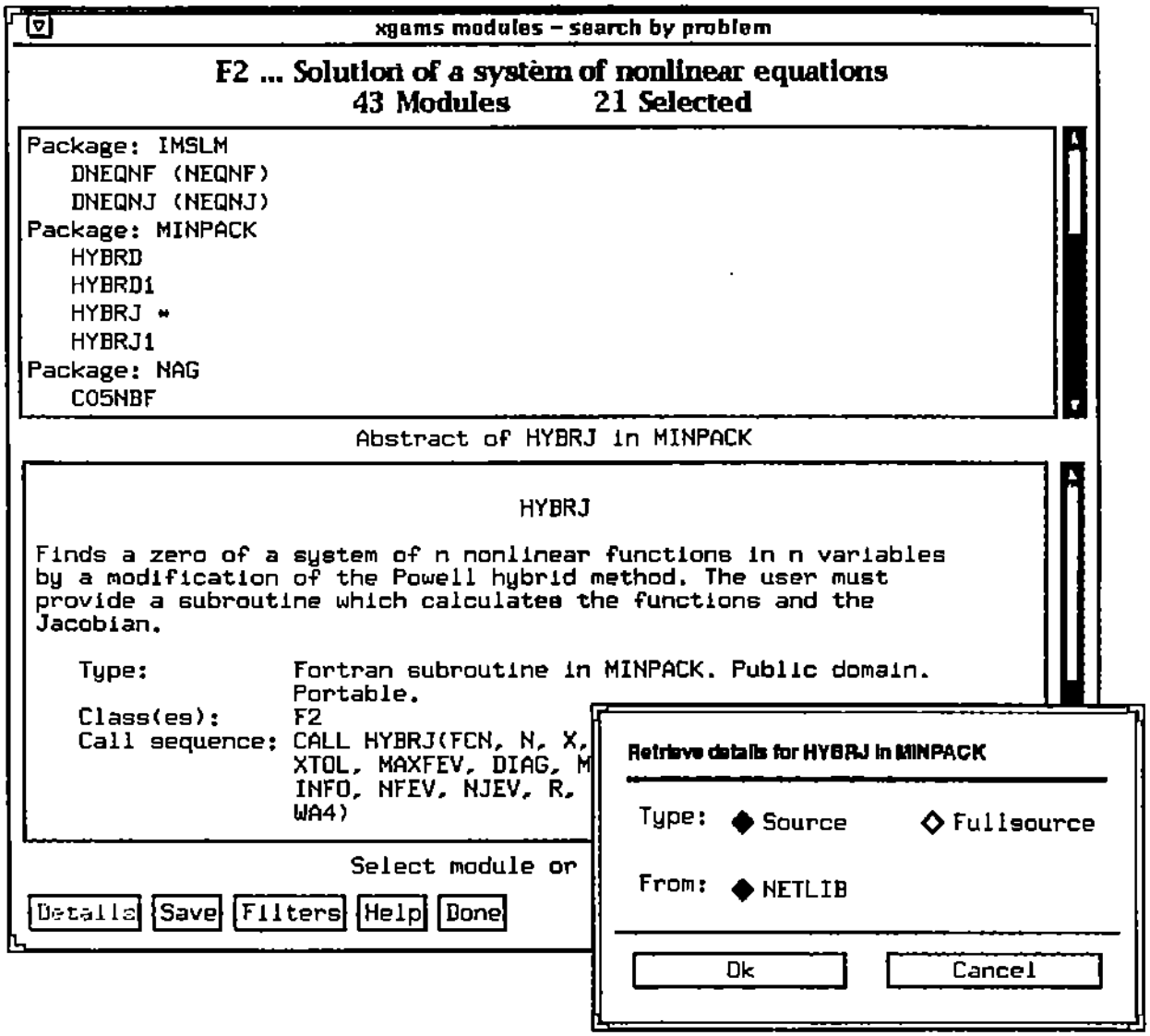

Figure 1: The xgams modules window

DOMINO: A Knowledge-Based System for the Users of a Finite Element Library

Patrick Laug
INRIA Rocquencourt
BP 105
78153 Le Chesnay Cedex, France

Introduction

Traditionally, scientific packages are considered only of components written in a procedural language (FORTRAN or C). But this architecture is unsatisfactory for users who are confused by the great number of functionalities of the software. One solution is to integrate some non-algorithmic knowledge into a knowledge-based system, as has been done for instance in ELLIPTIC EXPERT [2], EVE [1] or FOCUS [3]. In this context, several studies have been carried out at INRIA focusing on the finite element library MODULEF. One of the resulting realizations is a knowledge-based system called DOMINO, which has been coupled with MODULEF without changing anything in the library itself.

The MODULEF Finite Element Library

The MODULEF library [6] has been designed to solve different types of problems, in terms of partial differential equations, using finite element methods. Amongst the domains studied, the following can be mentioned:

- steady state or time-dependent, linear or non-linear, two- or three-dimensional heat conduction problems,
- two- or three-dimensional elasticity problems, with large and small deformations,
- two-dimensional fluid flow,
- characterization of composite materials reinforced by unidirectional fibers.

It now contains more than 400,000 lines written in FORTRAN. It is constituted of contributions (programs, expertise, material means, etc.) from all the members of the MODULEF club. These contributions are subsequently redistributed for the benefit of each member.

Usually, in order to solve a physical problem, the user must:

- model the problem mathematically as a partial differential equation system,
- find a variational formulation and discretize the problem,
- solve the discrete problem by executing a succession of algorithms; a very large number of these algorithms have already been implemented in the software (direct or dual finite element methods, several solution techniques for linear systems by direct or iterative methods, etc.),
- analyze the results.

The architecture of MODULEF (a library written in a procedural language) enabled us to code a large amount of the algorithmic knowledge of the finite element domain. But this architecture is inappropriate to represent other knowledge that the user must find in the documentation: how to model the physical problem, how to solve the mathematical problem by chaining software components, the most efficient components, their restrictions of use, how to activate them, how to interpret the results,... We then decided to represent this know-how in a knowledge-based system. This system, called DOMINO [4,5], is now operational and used by several laboratories.

The DOMINO Knowledge-Based System

The main feature of DOMINO is its ability to communicate not only with the user, but also with the scientific software (Figure 1). Instead of just advising the user to choose a certain component, DOMINO itself creates the necessary numerical data, activates the component and analyzes the numerical results. If these results are sufficient to select the next component, the latter is automatically activated. On the other hand, if some data are missing, the user is asked to supply them. With this method, the user is guided in the specification of the physical problem and gives a minimum of data.

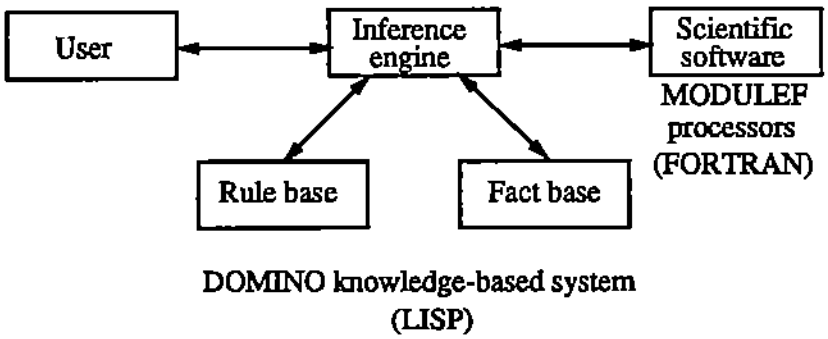

Figure 1: Global architecture of DOMINO

The knowledge is represented in a rule base and a fact base which is itself split into three parts (the *finite element base*, the *action base* and the *physical problem base*):

- The rule base represent the strategy to select the algorithms. from known facts, rules specify what actions have to be done, using a forward chaining mechanism.
- The *finite element base* contains a description of each element available in the MODULEF library: its name, geometry, formulation, interpolation, etc.
- The *action base* specifies the implementation of each algorithm: generic form of the data file, auxiliary subroutines, system commands, etc.
- The *physical problem base* is a temporary fact base that contains the data given by the user for the current problem.

The inference engine ensures communication between the previous components. It was written in LISP especially for this purpose. The main two advantages of this global architecture are to simplify the management of the knowledge involved and to possibly give justification of reasoning.

Conclusion

DOMINO has proved to be very useful for *nonexpert users*: thanks to this system, they easily use more efficient, although less familiar, methods. But this experiment has also shown us that other tools are necessary for the *expert users* and for the *developers* of finite element software:

- The *expert user* prefers to choose the methods him/herself and to create manually a set of numerical data. In order to satisfy this requirement, we are currently designing a system based on Smalltalk, which is both a prototyping language and a powerful graphical environment.
- The *developer* also needs different tools, because of the difficulties of integrating new components in a large software system, following rules and conventions that are often hard to learn and apply. One solution which appears to be very promising is a program synthesizer including a computer algebra system and a knowledge-based system.

Bringing together these different tools into a homogeneous environment raises several difficulties, but we believe it must remain a long term objective.

References

[1] P. Baras, J. Blum, J.C. Paumier, P. Witomski, and F. Rechenmann, "EVE: an object-centered knowledge-based PDE solver," *Expert Systems for Scientific Computing*, E.N. Houstis, J.R. Rice and R. Vichnevetsky (eds.), Elsevier Science Publishers B.V., North-Holland, © IMACS, 1992.

[2] W.R. Dyksen and C.R. Gritter, "Elliptic expert: an expert system for elliptic partial differential equations," *Mathematics and Computers in Simulation*, 31, Special Issue on Expert Systems for Numerical Computing, J.R. Rice (ed.), 1989.

[3] S. Hague, *FOCUS — First Annual Report – Summary of Progress*, NAG Technical Report TR2/90, ESPRIT II Project: 2620, 1990.

[4] A. Hassin and P. Laug, "Conception des matériaux composites à l'aide d'un système expert," *Tenth International Workshop on Expert Systems and Their Applications*, Avignon, France, May 28 – June 1st, 1990.

[5] P. Laug, "An expert system for the finite element method," Artificial Intelligence, *Expert Systems and Languages in Modelling and Simulation*, C.A. Kulikowski, R.M. Huber and G.A. Ferrate (eds.), Elsevier Science Publishers B.V., North-Holland, © IMACS, 1988.

[6] P. Laug and M. Vidrascu, "The MODULEF finite element library," *Problem Solving Environments for Scientific Computing*, B. Ford and F. Chatelin (eds.), Elsevier Science Publishers B.V., North-Holland, © IFIP, 1987.

A Knowledge-Based System for Eigenvalue Problems

Jean-François Carpraux and Jocelyne Erhel
IRISA/INRIA - Campus de Beaulieu
35042 Rennes Cedex - France

This paper presents a knowledge-based system devoted to the LAPACK library [1] to help the user in selecting a numerical procedure and in validating a result. Many front-end systems such as GLIMPSE and KASTLE developed by NAG [2] help the user in selecting the right routine of a scientific library. Our system is also concerned with the numerical quality of the result. It can either select a routine or check the validity of a given routine to solve a user's problem. The accuracy of the result depends upon the error on the data, the numerical stability and the condition number. Our goal is to provide an error estimation thanks to the knowledge of these three components. It should be noted that, contrarily to other expert systems such as [3], we do not intend to generate code, only to advise the user. All the knowledge base is modelized by objects, with no explicit inference rules. This approach is well-suited to describe a scientific expertise and allows to extend or modify easily the base.

We have developed a first prototype at IRISA, to demonstrate the feasibility of our goals. We do not presently deal with the complete LAPACK library but with a subset of it which aims at solving eigenvalue problems and which contains about hundred routines. We claim however that this domain is large and complex enough to show the capabilities of our system. Moreover our knowledge base can evolve easily thanks to the object modelization.

To solve an eigenvalue problem with the LAPACK library, the user can either select a driver routine or apply sequentially three methods : the first one to transform the matrix into a tridiagonal or Hessenberg form, the second one to compute the eigenvalues (if wanted) and the third one to compute the eigenvectors (if wanted). All the routines depend upon the nature of the matrix and the nature of the computation. The methods to compute the eigenvalues and the eigenvectors are also related to the number of values and vectors required. Each problem, described by the matrix and the requirements, can be solved by one or several methods. However, only one is the most suitable. For instance, it is better to compute few eigenvalues by bisection rather than by the QR method though both methods can do it. Therefore, the system must know all the possible methods when validating a user's choice and know the best one when selecting a routine. The accuracy of the computations depends upon the numerical stability, the condition number of the problem, the error on the data and the computer precision. Since we deal with the LAPACK library which is numerically stable, we can roughly estimate the rounding error by the machine precision. The library

provides routines to estimate the condition number of eigenvalues and eigenvectors. These estimations can be used to derive a simplified error estimation on the results.

Our system is based on the development shell SHIRKA [4], written in Lisp, which provides means to describe knowledge bases and to apply a classification mechanism upon them. All the knowledge is contained into the objects which are organized into hierarchies. Multiple inheritance allows to describe complex objects and is well suited to mathematical entities. The inference rules are implicitly embedded into the knowledge base thanks to the organization in hierarchies and to default values or attached procedures. No explicit rule such that "if-then-else" has to be written. The basic inference mechanism is the classification process which can take into account a given subset of attributes and which can infer values of another given subset of attributes. The process can explain any result when wanted. SHIRKA is easy to use thanks to a user interface which visualizes the objects and the actions taken on them.

The objects used in our system can be divided into three groups :

- description of the problem : the matrix, the requirements (number of eigenvalues and eigenvectors), the environment (precision).
- procedures : the driver routine (when it does exist), the procedure to transform the matrix (into a tridiagonal or an Hessenberg form), the procedure to compute the eigenvalues (by a QR-type method or by bisections), the procedure to compute the eigenvectors (by back-substitution or by inverse iterations).
- validation of the result : the routine to estimate the condition number, the formula to estimate the error, the error analysis.

To select a routine, the user provides the problem description and the system infers the procedure name after classification on a part of the attributes. On the other hand, to check a routine, the user provides also the procedure name and the system detects if the method is suitable or not after classification without inference on all the attributes. By comparing with its selection, the system can also find if the method is the right one.

To validate a result, the user provides an estimation of the error on the data which may come from a previous estimation by the system and of the condition number which can be computed by the routine proposed by the system. Then the system gives an error estimation thanks to a simple formula.

The current interface is provided by SHIRKA. We plan to develop a more specific interface in order to mask to the user the expert system aspects. The interface should also combine the different steps more automatically, for example in the selection of a method

which requires to combine several classifications corresponding to each procedure used. The validation here is simplified thanks to the knowledge on the LAPACK library. A more general system could propose tools to analyze for example the convergence behaviour or the roundoff errors. We currently investigate such numerical tools [5]. In our opinion, our system could easily be extended to the whole LAPACK library by creating new objects and possibly by updating some of them.

Acknowledgments : the authors are indebted to F. Rechenmann for providing the system SHIRKA and for his technical assistance.

References

[1] E. Anderson and al., *LAPACK Users' guide* SIAM, 1992.

[2] GLIMPSE and KASTLE, NAG reports, 1990.

[3] S. Konig, "An expert system shell for validated computation and its application to solving linear equations," in *SCAN 91*, Oldenburg, 1991.

[4] F. Rechenmann and B. Rousseau, "A development shell for knowledge-based systems in scientific computing," E.N. Houstis, J.R. Rice (eds), in *Expert Systems for Numerical Computing*, Elsevier Science Publishers, Amsterdam, 1992.

[5] J. Erhel and B. Philippe, "Design of a toolbox to control arithmetic reliability," in *SCAN'91*, Oldenburg, 1991.

An Approach to Expert Systems for Image Processing Software Libraries

(Extended Abstract)

Felix Grimm, Horst Bunke and Jürg Hählen
Institut für Informatik, Universität Bern,
Länggasstrasse 51, CH-3012 Bern, Switzerland

Image processing [1, 2] is a fast growing subfield of computer science with many applications in science and engineering. A large number of standard algorithms have been proposed, and there exist many image processing software libraries, each typically including between a few dozens and several hundreds of standard algorithms. An example of such a library is SPIDER [3] which contains more than 300 algorithms covering a large part of the field of image processing. The application of the SPIDER subroutines to a given problem, however, may be difficult even for an expert in image processing as the complete documentation consists of more then 700 printed pages, describing subroutines that may need up to 17 parameters.

In order to successfully solve an image processing problem by means of a program library like SPIDER, knowledge about both image processing in general and the underlying software package is needed. Additionally, general knowledge about the application and use of subroutines from a program library is needed. Conceptually, solving an image processing problem by means of a software package can be modeled as a process that involves the cooperation of three experts: a problem solving expert who knows how to solve problems by means of software packages independently of the particular application area and software library; a domain expert, in our case an expert in image processing; and a software package expert, e.g. an expert for the SPIDER package. This approach can be directly implemented and results in an expert system for problem solving by means of a software package. The three experts are represented by three more or less independent parts of the knowledge base.

Using the concept of the three independent experts described above, we have developed an expert system for the application of the SPIDER package. The aim of the system is to support users of image processing in solving a variety of image processing problems. The system provides the following functions:

- Classification of the underlying image processing problem (e.g. contour based or region based processing).
- Selection of a sequence of methods of image processing and corresponding SPIDER subroutines which can be used to infer the desired results from an input picture.

- Parametrization of the subroutines to be applied.
- Automatic execution of the selected subroutines.
- Interpretation of the intermediate results in order to decide whether the actual processing is to be continued or whether alternative ways of processing are preferable.

The operations of the system are driven by a dialog with the user. In such a dialog, the user specifies the problem to be solved and can give his/her preference if the system finds more than one possible way of processing. There are two dialog modes, one for experienced users and one for novices. Background information and recommendations are available for any question asked by the system, and default answers are proposed whenever suitable. Furthermore, as it is often necessary in image processing to test various parameter combinations, the system supports systematic tests without the need of repeated specification of constant parameters.

Pictures generated by the system as intermediate results are displayed to the user who has to decide whether they are useful for further processing or to be rejected. In the latter case, the system tries to find alternative subroutines. There are some cases where the system automatically recognizes that the result of a subroutine has to be rejected. An example is the case where an error code has been generated by a subroutine.

The full version of our paper will describe in more detail the three parts of the knowledge base consisting of (1) a general strategy for problem solving by means of software packages, (2) knowledge about image processing, and (3) knowledge about the SPIDER package. As the three experts are represented by independent parts of the knowledge base, it is possible to adapt our system to other image processing software packages by changing only the part of the knowledge base that represents knowledge specific to the underlying software package.

In order to demonstrate the generality of our approach, we have developed a second similar expert system for image processing based on the GIPSY package [4]. In contrast to SPIDER which is a collection of independent and self-contained subroutines, GIPSY also provides an image processing environment. For practical reasons it uses global variables and other concepts at different levels of a sophisticated calling hierarchy. Thus it is not a collection of modules with clean interfaces. Despite the rather complicated structure of GIPSY, which differs significantly from SPIDER, most parts of the knowledge base of the original system, except the part that is specific to SPIDER, i.e. the SPIDER-expert, could be adopted without changes.

Both the system for SPIDER and GIPSY have been implemented by means of Siemens-Nixdorf's expert system shell TWAICE on a UNIX machine and are in the stage of a pro-

totype. The systems have been tested on several different image processing problems and shown good performance.

In conclusion, our general approach of independent experts, i.e. parts of the knowledge base, together with a general problem solving strategy leads to a general-purpose shell for expert systems supporting the use of software modules from program libraries. We expect that such general shells will be a significant contribution to a more successful and profitable use of scientific subroutine packages in future.

References

[1] D.H. Ballard and C.M. Brown, *Computer Vision*, Prentice Hall, New Jersey, 1982.

[2] A. Rosenfeld and A.C. Kak, *Digital Picture Processing* Academic Press, 1982.

[3] H. Tamura, S. Sakaue, F. Tomita, N. Yokoya, M. Kaneko, and K. Sakaue, "Design and Implementation of SPIDER - a transportable image processing software package," *Computer Vision, Graphics, and Image Processing*, 23, 1983, pp. 273-294.

[4] S. Krusemark, R.M. Haralick, V. Ramesh, and C. Lee, *UNIX GIPSY Users Manual*, Intelligent Systems Laboratory, Dept. of Electrical Engineering, FT-10, University of Washington, Seattle WA 98195, USA, 1988.

Knowledge Based Algorithm Construction for Real-World Engineering PDEs

Eamonn Cahill
Hitachi Dublin Laboratory, O'Reilly Institute
Trinity College, Dublin 2, Ireland

Introduction

There has been considerable interest in recent years in the application of Artificial Intelligence techniques to the numerical solution of PDEs. The work to date on such Knowledge-Based (KB) PDE Solvers has tended to focus on the architecture and A1 paradigms employed, rather than on the fundamental issue of the knowledge to be encoded within the system. Therefore, the application domains considered have been simplified and idealized mathematical models which are not representative of the types of models that arise in real-world engineering problems. In this work we consider a realistic application domain from the outset, rather than try to scale up a system developed for a reduced domain to a more complex domain.

This work is part of an on-going, long-term project to develop a sophisticated knowledge-based (KB) numerical simulation environment known as the *Scientific Simulation Workbench* (SSW). This system will be built upon PDEQSOL, a high level language for the solution of PDEs. However with this system, along with other systems of this type, e.g., ELLPACK, a familiarity with PDEs and the construction of numerical algorithms is assumed on the part of the user. A prototype system, A1-DEQSOL, incorporating a knowledge-based front end to overcome this restriction, has already been completed and reported upon. A practical system based on this experience is now being developed. This paper reports on the development of a robust knowledge base for this system. The key points of the paper are:

- A realistic application domain is being focused on from the outset.
- Detailed consideration is being given to the contents of the Knowledge Base.
- Indices for characterizing the key features of a given problem are being investigated.
- The results of knowledge acquisition for this domain are presented.
- A library of algorithm sub-components construction problem is being developed.
- The knowledge base is implementation independent.

Knowledge-Based Algorithm Construction

There are two distinct approaches to Knowledge-Based algorithm construction in a PDE solver. The first is to select from a library of complete and self-contained "black-box" type algorithms. The other is to construct an algorithm from basic principles without the help of a *priori* knowledge for a particular problem. In between these extremes is the approach adopted here which is the construction of an algorithm from a library of pre-existing sub-components. Individual engineers tend to work within fairly specialized and clearly defined domains. Within any such domain the number of PDEs of interest is fairly limited and are well known, e.g., Navier-Stokes, advection-diffusion, Maxwell's equations, etc. Therefore, there exists considerable prior knowledge about the problem and possible solution algorithms. Therefore, apart from the difficulty of developing a general purpose KB-PDE Solver working from first principles, the reality is that very few users require such dexterity. In order to construct an efficient and robust algorithm, the system must characterize the given problem and map this problem onto a suitable algorithm using the following prior knowledge:

- Phenomena-centered Knowledge (advection, inflow, sources, etc.). This makes use of high-level, compiled knowledge acquired through experience with specific problems. It also facilitates interaction with an engineering end-user. Examples are: for Rayleigh numbers of $> 10^3$ upwinding is required for natural advection. In addition, an under-relaxation factor of 0.6 is required for such flows. For non-linear diffusion, an under-relaxation factor of 0.8 is optimal.

- Algorithm-centered Knowledge. This knowledge enables different possible algorithms to be assessed. Examples are scaling properties, stability (Crank-Nicholson prone to oscillations, etc.).

The knowledge acquisition approach being pursued is a combination of systematic numerical experimentation with representative problems and theoretical study of the issues involved. Representative problems have been chosen for which either analytic solutions exist or for which there are published numerical results, e.g., the *Driven Cavity* and the *Thermal Cavity*. It is believed that such an approach will contribute to the robustness of the system. A typical issue being investigated is the critical Rayleigh Number above, which upwinding is required.

The Application Domain and its Characterization

We must bound the application domain in such a manner that the construction of a KB-PDE Solver is tractable from an AI perspective, but still interesting from an engineer one. For the purposes of this work, we have focused on *Finite Element* analysis of 2D incompressible fluid and heat flow including the effects of conduction and advection (forced and natural). The indices being developed to characterize a given engineering problem include:

- Physical phenomena (heat, fluid flow)
- Transient/Stationary analysis
- Geometry/Topology of domain
- Presence of sources
- Boundary conditions (Neumann, Dirichlet)
- Ratio of viscous to inertial effects
- Ratio of advection to diffusion effects
- Strength of non-linearity
- Stiffness

Stiffness of a system is particularly difficult to assess a priori, however, heuristics are being developed to estimate it by studying the material parameters (e.g., ratio of largest to smallest diffusivity) and the magnitudes of boundary fluxes, etc. The knowledge acquisition is focused on relating these indices to the optimal combination of algorithm sub-components as outlined in the next section.

Modularization and Assessment of Algorithms

The "degrees of freedom" within an algorithm which the KB system has under its control are determined by the flexibility of the underlying solver. They can be summarized as:

- Element Type (linear, quadratic, etc.).
- Non-Linear Expansion (Picard, Newton-Raphson).
- Time Expansion (implicit, explicit, Θ method).
- Coupled Equations (simultaneous, sequential solution).
- Matrix Solver (Gaussian, Conjugate Gradient (CG)).

There are also a number of "of the shelf" algorithms for specific problems, e.g., SMAC, penalty method for pure fluid flow. These can be combined with other components for coupled problems.

The knowledge base encodes knowledge to assess the performance of the different options with regard to specified criteria (accuracy, stability, computer resources, etc.). Typical issues being investigated include:

- Trade-off between use of a fine linear mesh and a coarse quadratic one.
- Trade-off between reduced number of iterations and increased amount of CPU time per iteration for non-linear problems when quadratic elements are employed instead of linear ones.
- Relative accuracy of Picard and Newton-Raphson iterations for non-linear problems. It has been found that Newton-Raphson requires half the number of iterations as Picard to achieve the same level of accuracy for diffusion problems.
- Sensitivity to round-off error. Bi-conjugate gradient has been found to be particularly susceptible.
- The condition number of the system matrix is very sensitive to the aspect ratio of the elements. For CG methods the number of iterations required to achieve convergence is O (condition number$^{1/2}$). A polynomial relationship has been found between the aspect ratio and the number of iterations.

Conclusions

The development of a robust, high-level knowledge base for a practical KB-PDE Solver has been described. This system has been targeted from the outset of real-world engineering problems, rather than idealized or abstract mathematical problems. Great emphasis is being placed on the quality of the kB. It is believed that this is vital to produce a genuinely practical system. We have chosen a bounded but realistic domain and have proposed indices to characterize problems within this domain. Extensive use is made of *a priori* knowledge to relate these indices to appropriate algorithms and to assess their performance. The required knowledge is being elucidated in a systematic manner from numerical and theoretical investigations.

Knowledge Based Support of User of Numerical Programs

Henk J. van Zuylen
Delft Hydraulics, The Netherlands

Abstract

Numerical programs are difficult to use for people who were not involved in their development. Knowledge necessary for the use of the program can be included in a user interface. Often, better approaches are possible when the user interface has the appropriate support. The user task can be modified such, that less knowledge is necessary. The remaining knowledge can be inserted in a knowledge base, coupled to the user interface, but the real improvement comes often from the fact that a good user interface makes it possible to work in a way which requires less knowledge. The role of knowledge analysis and (cognitive) task analysis is indispensable.

Introduction

Many programs for numerical analysis have been developed initially by scientists for their own use. The natural evolution of such programs is, that they disappear silently if they are not successful or if they are usable only for a very limited domain. In other cases the programs are enhanced and at a certain moment they become attractive for other users: students, colleagues or clients of the company. This evolution is determined by the functionality of the program, not by its usability. The term usability is used here in the sense

> *The capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfill the specified range of (...) scenarios. Usability is the result of objective and subjective factors: usability has a multivariate causality; it is a user response to the total context in which he is operating, to the benefits he hopes for and to the costs he perceives [5].*

From this description it is clear that a natural evolution of a program normally leads to programs with a low usability. When the range of users changes, it would be a coincidence when the new users would have the same tasks, knowledge and experience as the original ones. It is characteristic for good programs, that people try to apply them for purposes which tend to be just outside or at the boundary of the original application domain. Further new

users have other experience and knowledge so that they cannot use the programs without training and support.

People tend to forget how they have to work with a tool, when they use the tool infrequently. The more one has to remember for the correct operation of the tool, the more complex the knowledge is, the less the chance that the user will easily work with the tool after some time.

New users of a program have often a different view of the program than the developers. They have different problems in mind. The developers may have thought in terms of mathematical equations, numerical algorithms, data files and programming languages. The next generation of users work, for instance, on physical or statistical problems. For instance, a computer program which solves the Navier Stokes Equation for a incompressible fluid can be considered as a specification of a physical model, a numerical algorithm, a 'black box' which converts input files to output files. This is typically a view of a program developer. Another view is to consider the program as a tool to predict flows under certain circumstances or to convert a description of a physical situation and boundary condition to a flow field.

Although in both cases the utility of the program will be appropriate, the second situation imposes higher requirements to the usability. The usability is most important in the user interface and in the supporting functions. While the specifications for a program are derived from a problem definition and analysis of functional requirements, the specification of the user interface and supporting functions have to be derived from:

- the user profiles
- the user tasks
- the knowledge and mental model of the users
- the experience and expectation of the users
- the knowledge necessary to use the program
- the knowledge necessary to obtain a description of the problem in terms which can be handled by the program.

A good guideline in the development of user interfaces of programs is, that the user interface should match the problem domain of the user. If the user thinks in terms of a geographical area, the user interface should present the information in the format of a map and the users should interact with the map on the screen when they enter geographical data. The user interface should not ask for e.g. coordinates or names of towns. So the objective

should be that the 'cognitive distance' between the problem domain of the user and the user interface should be minimized.

Another objective is, that the amount of information presented to the users and entered by the user should be as small as possible. Redundant information which can be inferred from other data should be asked by the user interface. Information should be asked such in a representation which is easy for the user to understand and to enter. Also the presentation of the data should be so, that it can be easily understood.

In this paper some examples are considered of user interfaces which have been built for numerical programs. The first one is a program which executes tidal analysis and builds a mathematical model for the tide. The second one is a system which simulates the transport of pollution in a river or in the sea. In both cases a certain development process had to be followed in order to build the user interface. The next section gives a short description of this process.

The Development Process of Knowledge Based User Interfaces

The development of a user interface can be done in the following process [6]:

1. Analysis of the requirements (preferably with some mock-up of the user interface to illustrate the ideas)
    - definition of requirements
    - feasibility study
    - validation of the requirements
    - decision to start the development process or to try other solutions
2. Detailed analysis
    - knowledge analysis of expert users
    - determination of user groups and user profiles
    - task analysis of all user groups
    - functional specification and cost estimation
    - test plan

- decision to continue or stop

3. Restructuring
   - restructuring of the application
   - redesign of the user task and dialogue
   - redesign of the user interface
4. Implementation
   - reengineering of the application
   - generation of the user interface
   - coding of support functions and data (including rule bases)
   - integration
   - test
5. Evaluation

The development of user interfaces is nearly always expensive. The decision to go on or to stop should be made explicitly, as soon as the necessary information about feasibility and costs are available. Knowledge analysis, in the same way as is done for expert systems, appears to be very effective as a first step in the cognitive task analysis [3]. A good knowledge acquisition phase has the secondary effect that the (expert) users involved have to think in another way about their program. This may lead to a better way of working [7]. An important point is that the implementation of a new user interface for an existing program often makes it necessary to modify the programs. In some cases the old user interface has to be extracted from the program with reverse engineering techniques [9].

GETIJSYS

At Delft Hydraulics the program GETIJSYS has been developed for the analysis of the tide. The objective of the program is to build a mathematical model which describes the time variation of the water level. The mathematical model of the tide is used afterwards to calculate tidal tables, to define boundary conditions for water flow models in sea and to find extreme water levels for e.g. the design of dikes, drilling platforms or harbors. This mathematical model is calibrated and validated with measured data of water levels over

some time period. The preparation of input, the use of the program and the interpretation of the output and subsequently to improvement of the mathematical model, were rather complicated. It was very difficult for a novice user to use the program in a sensible way and obtain useful results, without an extensive support of an expert. The objectives of the user interface was to bring all necessary knowledge about the program in some kind of expert system. During the task analysis it appeared that a more efficient task execution would become possible, when the appropriate support was available. Instead of error-prevention (which required much knowledge) the emphasis of the task is now on error-correction. This requires the visibility of errors and a fast turn around time of the error correction cycle.

The new user interface makes a different task execution possible. The user task for the original program has become obsolete. The structure of the knowledge needed to execute the task has not changed much. The most important change is that some knowledge which was necessary in the old situation, has become obsolete: the user interface does some analysis and provides the user with information which makes it unnecessary to remember all kind of rules and to make all kinds of checks. This means that the task execution changed rather radically but that the main knowledge structure remained stable. The way in which this knowledge is applied may be different in the new user interface. This support the approach to start the development of such user interfaces with a *knowledge analysis* instead of a *task analysis*, as is done in most methods for user interface development [2].

The most important issues in the new user interface of GETIJSYS are now the visibility of weaknesses in the model of the tide, the recommendations for improvement and support to modify the tide model efficiently.

Other User Interfaces for Numerical Programs

Other programs for which user interfaces have been developed are

- grid generators
- water quality programs.

In the first case the emphasis is on the possibilities for the user to manipulate grid points and to visualize the grid. In the case of the water quality programs a set of several programs and data files (databases) had to be made available for clients of the company. The different programs had to be integrated, such that they can work with the data or that one programs produces data which is suited for another program. Further the operation of the different programs had to be integrated.

An analysis of the user profiles and tasks appeared to be very effective to make sure, that the future users would accept the system and to give them confidence in the developers. The knowledge analysis appeared useful to identify logical discrepancies between the different components of the system, such as differences in terminology. Also the logical structuring of the data was one of the products of the knowledge acquisition phase.

Conclusion

Working with complex (numerical) programs can be improved with a good knowledge-based user interface. The fact that the user interface is 'knowledge based' does not necessarily mean that it is some kind of expert system, but that it is developed form a knowledge analysis. Cognitive task analysis as used in the domain of ergonomy [2] and knowledge analysis techniques such described by Hickman [3] or Nijssen and Halpin [4] are suitable for the analysis phase of the user interface development. The emphasis should be on task restructuring and the reduction of knowledge necessary to execute the task. The analysis of the knowledge needed for the use of the program gives a suitable starting point for the specification of the new user task.

References

[1] Diaper D., Gilmore D., Cockton G., and Shakel B. (eds.), "Human computer interaction," *Proceedings IFIP TC 13 Interact'90.*

[2] Diaper D., "Task analysis for human-computer interaction," *Ellis Horwood*, 1989.

[3] Hickman F.R., Killin J.L., Land L., Mulhall T., Porter D., and Taylor, R.M., "Analysis for knowledge based systems," *Ellis Horwood*, 1989.

[4] Nijssen, and G.M. Halpin, T.A. "Conceptual schema and relational database design," *Prentice Hall*, 1989.

[5] Novara F., Allamano N., Bertaggia N., and Olphert, W., "Factors governing usability and criteria for their assessment," *ESPRIT project 385 HUFIT, Deliverable A5.1b*, Stuttgart, Fraunhofer-Institut für Arbeitswirtschaft und Organization, 1986.

[6] Smit E.Y.M., "ANDES, Method for user interface analysis and redevelopment," *REDO report 2487-TN-WL-1071*, Delft Hydraulics, 1992.

[7] van Zuylen H.J. and Gerritsen H., "Knowledge based user interfaces for scientific programs," in *[1]*, 1990

[8] van Zuylen H.J., "From scientific computation to decision support," *Knowledge Based Systems* Vol. 6(1), March 1993.

[9] van Zuylen H.J. (ed), "The REDO compendium of reverse engineering for software maintenance," *Wiley*, 1993.

An Expert Assistant to Monitor Finite Element Simulations

C. Thilloy, R. Labrie and P.A. Tanguy
Department of Chemical Engineering
Laval University, Quebec, G1K 7P4, Canada

The finite element method is a numerical method well-adapted to the solution of complex, large scale engineering problems which involve various kinds of data, very different in nature (physical properties, geometrical data, numerical parameters, etc.) . In using this method, experience (often empirically acquired) plays an important role, which is seldom coded in programs or expressed in algorithms. Consequently, for practical computations performed in a conventional manner (i.e. with declarative language like C or FORTRAN), the complete resolution process may easily involve several simulations on a trial-and-error basis.

It is well-known that for generic problem solving method with difficult-to-express problems, large scale dynamic data and uncertain knowledge, knowledge-based system is an efficient alternative to conventional data representation (Martin and Oxman 1988). Such an approach has already been used successfully in the early stage of product/process design (Grierson and Cameron 1988, Beltramini and Motard 1988). It is possible to go a step further than the simple design definition stage and use a knowledge-based approach for the simulation itself. The aim of this paper is to report on our experience in developing an expert system prototype to monitor the complete finite element simulation procedure (preparation of data structure and diagnostics). This prototype was built around our 2D finite element simulation package POLY2DTM and tested in the context of 2D incompressible viscous fluid flow in complex geometry.

In a finite element simulation, the human operator or expert, is usually in charge of all steps of the simulation process, namely:

- the mesh generation;
- the selection of an element (choice of approximations);
- the imposition of the boundary conditions;
- the specification of the rheology and material properties;
- the choice of the resolution method;

- the prescription of the numerical parameters;

- the analysis of the algorithm numerical behavior (convergence);

- and finally, the analysis of the results.

This involves simultaneously the use of well-established algorithms and a good deal of practical experience. In the present work, a mixed approach based on traditional declarative computation and expert system, was adopted to develop the prototype expert assistant shell.

¿From a practical standpoint, the NEURON DATA Nexpert ObjectTM expert system shell was selected as the development environment and over 200 rules were entered in the knowledge base. Two computer platforms were used for the work, namely a 386 microcomputer for Nexpert and an HP/Apollo DN3500 workstation for the simulation. The transfer of data between the two platforms was ensured through an interface written in C. The expert assistant control was divided in several steps. Due to the knowledge structure used by the human expert, the initial knowledge base was also segmented in several smaller ones, each refering to the knowledge of a particular step (ex: mesh generation). This strategy was found more efficient for the two following reasons:

- it follows closely the usual way of doing, and therefore is easier to use by a non-

- it reduces the set of rules used at any time by the prototype.

This assistant system has successfully been tested with the finite element code POLY2D. It acts as a consultant, giving advice to the user on the best way to define the problem (mesh, boundary conditions, numerical parameters, etc.) and analyzing the algorithm behavior during the resolution. The expert assistant showed good skills in detecting possible simulation parameter definition errors and in results analysis. Due to its satisfactory performance, it also proved useful in a tutoring mode.

The presentation will focus on the structure, implementation and capabilities of the expert assistant. Some ideas about possible future developments will also be given.

References

[1] Beltramini L. and R.L. Motard, "A knowledge-based approach for process design, computer in chemical engineering," 12, pp. 939-958, 1988.

[2] Grierson D.E. and G.E. Cameron, "A knowledge-based expert system for computer-automated structural design," *Computer and Structures*, 30, pp. 741-745, 1988.

[3] Martin J. and S. Oxman, "Building expert systems," *Prentice-Hall*, 1988.

Intelligent Object-Oriented Scientific Computation

A. Cuyt
Research Director NFWO, University of Antwerp
(UIA), Universiteitsplein 1, B–2610 Antwerp, Belgium

B. Verdonk
Post-Doc Researcher NFWO, University of Antwerp
(UIA), Universiteitsplein 1, B–2610 Antwerp, Belgium

J. Verelst
Assistant Faculty TEW, University of Antwerp
(RUCA), Middelheimlaan 1, B–2020 Antwerp, Belgium

Our goal is the integration of scientific computation and intelligent object-oriented techniques to obtain powerful tools which can be used for the successful development of scientific expert systems, i.e. systems which provide expertise in the choice of algorithms within a particular problem domain and generate reliable, robust and maximally accurate results of full integrity. The need for such scientific expert systems is increasing significantly [ABEL 89, HUBE 89]. Through this study we want to identify general principles which must be taken into account both concerning the computational aspects, such as automatic validation of the results (see 1.), and aspects of programming methodology, especially with respect to classification and inheritance (see 2.). The usefulness and power of such intelligent object-oriented scientific expert systems will be tested on a prototype which will be developed for the problem domain of multivariate rational data fitting [CUVE 92]. A more detailed description of this prototype is given in 3.

Scientific Computation

Although computers were historically developed for scientific computation, extensive interest in the subject of computer arithmetic has mostly developed in the early eighties, yielding some important improvements in the accuracy of floating-point computations. Two IEEE floating-point standards were agreed upon [GOLD 91] in order to formalize the representation of the floating-point numbers $z \subset z$ and the implementation of the basic floating-point operations. However, many computations in numerical algorithms are carried out in product spaces (vectors, matrices, ...). The arithmetic operations in these spaces are traditionally

performed in terms of the given elementary floating-point operations in z. It is well-known that the computational error due to the accumulation of rounding errors of each of the basic floating-point operations can become quite large. Therefore, a new definition of the operations in the product spaces which overcomes this shortcoming, is given in [KUMI 81]. It is proven there that to implement the operations in all the product spaces according to the new definition, the set of basic operations in the IEEE standard has to be extended with the scalar product for vectors.

The advantages of highly accurate computations can be combined with interval arithmetic to obtain self-validating numerical methods. Interval arithmetic is the only computational tool so far available that incorporates guarantees as part of the basic computational process. Interval arithmetic which has been around for a long time has often been criticized since its naive use may deliver bounds which are unreasonably large and thus do not contain much information about the solution of the problem. If one combines interval computation, the process of defect correction [ULWO 89] and the optimal scalar product this criticism is superseded. In this approach the role of the optimal scalar product is again crucial.

Intelligent Object-Oriented Techniques

There are several concepts and principles underlying object-oriented programming: the concept of class or abstract data type, inheritance, overloading which is closely linked with the principle of dynamic binding and many others. We have mentioned these three since it is mainly these which will be put to good use when implementing a scientific expert system [WOLF 92]. For a specific problem domain, usually different problem types can be identified, and with each problem type appropriate algorithms can be associated. The translation of this classification in an object-oriented programming language can thus easily be done. Indeed, each problem type can be implemented as a class, where the algorithms to solve that type of problem are encapsulated within the class. As such, each class certainly defines a method to "solve". The implementation of the *solve*-method in the class C_1 is the algorithm to solve the problem of type C_1, whereas the implementation of the *solve*-method in class C_2 will be the algorithm to solve problems of type C_2. This classification and the partial order between the different problem classes is closely related to the particular problem. The overloading of the method name *solve* can be resolved through dynamic binding. However, for numeric computations the default dynamic binding mechanism must be carefully examined. Indeed, it may be the case that although a particular problem belongs to class C_1 the *solve*-method in the superclass of C_1, instead of in C_1 itself, should be applied for reasons of stability and reliability. In traditional object-oriented programming languages all

local methods have a priority higher than that of methods inherited from superclasses. The overloading of methods should be conditional on the performance of the algorithms for the particular data. The principles of object-orientation also allow to improve the programming methodology. Indeed, if more optimal algorithms or data structures can be developed for a particular class, only the implementation within that class needs to be modified, while the rest of the program remains unchanged.

Prototype Scientific Expert System

In order to demonstrate the power of intelligent object-oriented scientific expert systems, a prototype will be developed for the problem domain of multivariate rational data fitting [CUVE 92].

It is our goal to combine in the prototype the philosophy of the XSC-languages (eXtensions for Scientific Computation) with the power of object-orientation, including an interface to logic or functional programming languages to allow the implementation of intelligent reasoning. Moreover, the prototype should exhibit a certain "fault tolerance" in the sense that the brain of the expert system should be able to deal with faults without producing nonsense as output [BEJA 90, p.12]. A kind of uncertainty or fuzzyness can probably be dealt with by using interval arithmetic. Allow us to remark that also in the human brain a neuron is only activated when the input to the neuron body exceeds a certain threshold [BEJA 90, p. 7]. As long as the activating input remains within a certain interval, the neuron is inactive. This programming style exhibits a natural gracefulness that is lacking in traditional programs.

The practical realization of the prototype intelligent object-oriented scientific expert system can be pictured as follows. The expertise is presented to the expert system in the form of a collection of problem-domain specific numerical routines, e.g. in C^{++}, which are structured in a class hierarchy. Starting from these routines, the prototype should be able to

1. extract the class hierarchy from the source files and depict it graphically on the screen with some additional information;

2. build a decision tree equiped with the necessary tests for the classification of real-life data (creating an instance of a class in the above hierarchy);

3. construct a knowledge base and an inference engine to conclude for the correct numerical procedure; note that, because of the way the expertise is presented to the expert system, this knowledge base is constructed dynamically; the expert system has

to learn from the expertise presented to it and will adapt its knowledge base whenever the expertise is upgraded;

4. take into account possible unstable behaviour of the suggested numerical procedure or large perturbations in the input data ; as a result choose a numerical technique from a superclass instead of a more specific class, overruling the classical inheritance mechanism;

5. build the call to the selected numerical procedure and test for availability and consistency of all the input data;

6. provide a (sort of natural language) user interface prompting for lacking info or input and explaining decisions upon request;

References

[ABEL 89] H. Abelson et al., "Intelligence in scientific computing," *Communications of the ACM*, Vol 32(5), pp. 546-561, 1989.

[BEJA 90] R. Beale and T. Jackson, "Neural computing, an introduction," J.W. Arrowsmith Ltd, Bristol, 1990.

[CUVE 92] A. Cuyt and B. Verdonk, "Multivariate rational data fitting: general data structure, maximal accuracy and object orientation," *Numerical Algorithms*, to appear.

[GOLD 91] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comp. Surv.*, Vol 23, pp. 5-48, 1988.

[HUBE 89] E. Huber (Ed.), "Artificial intelligence in scientific computing," *IMACS Annals on Computing in Applied Mathematics*, Vol. 2, Baltzer AG, Basel, 1989.

[KUMI 81] U. Kulisch and W. Miranker, "Computer arithmetic in theory and practice," *Academic Press*, New York, 1981.

[ULWO 89] Ch. Ullrich and J. Wolff von Gudenberg (eds), "Accurate numerical algorithms: A collection of research papers," *Springer Verlag*, Berlin, 1989.

[WOLF 92] J. Wolff von Gudenberg, "Object-oriented concepts for scientific computation," *IMACS Annals of Computing and Applied Mathematics*, Vol. 12, 1992.

Qualitative Rule Formation Through Numeric Data Analysis

(Extended Abstract)

Z. Chen
Dept. of Math and Computer Science
University of Nebraska at Omaha
Omaha, NE 68182-0243

Introduction

In order to find the relationship in complex data involving multiple variables, we may apply the idea of qualitative reasoning as discussed in artificial intelligence. The based idea here is to convert continuous data into discrete data (eg., through clustering) so that qualitative rules can be constructed. In this paper we propose a method of compressing data to establish relationship between the dependent variable and other variables. Qualitative rules can be constructed and then stored in a knowledge base. This also facilitates automated knowledge acquisition.

In this paper we consider complex data which involve a variable Y and other variables $x_1, x_2, ..., x_n$ (n may a relative large number, eg. 100). The variables are associated through parameter t:

$$Y = y(t), x_1 = x_1(t), \ldots, x_n = x_n(t).$$

Observational data have been collected. We will call Y a variable *dependent* on $x_1, x_2, ..., x_n$, but this does not necessarily imply causal relationship. Another note should be made here is that we do not assume all x_i's ($i = 1, 2, ..., n$) are independent to each other.

Lebesque Discretization for Dependent Variable

For the function

$$Y = y(t),$$

instead of discretize t into subsections, we discretize Y into subsections

$$y = (Y_1, Y_2, ..., Y_n).$$

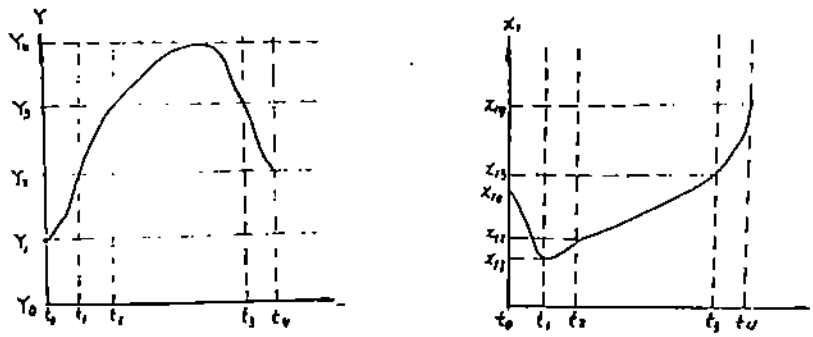

Figure 1:

All x_i's are also dissected, but they are dissected not in the same way as for Y; instead, they are dissected according to the intervals of t determined by Lebesque discretization on Y.

An intuitive explanation of discretization in this manner is that, *starting from the result of the dependent variable,* we are looking for the reasons which contribute to the result by examining the values of other variables.

The discretization process can be depicted by an example in Figure 1, where the left figure depicts Lebesque discretization on Y, and the right figure depicts the discretization on a variable x_1. To simplify our discussion, for the time being, we assume that the values of Y are divided into four intervals, each has the same length, namely, $[Y_0, Y_1) = [Y_1, Y_2) = [Y_2, Y_3) = [Y_3, Y_4]$. In reality, the interval of Y is determined by how to cluster the values of Y in terms of other variables (as to be briefly mentioned in a later section).

The way of dissection on Y is called Lebesque discretization due to the similar way of doing dissection in integral:

$$Y = \int f(x)dx,$$

instead of dividing x axis into subsections (as in classical Riemann integral), y axis is dissected into subsections.

Note that either Y or any of x_i is not necessarily a continuous function of t; but for convenience, we will treat them as continuous functions, so that they can be represented using curves.

For all observations collected in kth interval obtained through Lebesque discretization,

our task is to determine the relationship between variables

$$Y_k, x_{1_k}, x_{2_k}, ..., x_{i_k}$$

and summarize the result in terms of "if ... then" type production rules.

Back to the example in Figure 1, comparing the figure in the left and the one in the right in Figure 1, we notice that when Y falls in $[Y_2, Y_3)$, t falls in $[t_1, t_2)$ and $[t_3, t_4)$; that is, we find values of t from the values of Y. Furthermore, from intervals of t determined in this manner, we can determine the correspondent ranges for each variable $x_i (i = 1, 2, ..., n)$. For instance, for variable x_1, interval $[t_1, t_2)$ determines the range $[x_{1_1}, x_{1_2})$, while interval $[t_3, t_4)$ determines the range $[x_{1_3}, x_{1_4})$. Associating the intervals for variable x_1 and the dependent variable Y, we may end up with the following rule:

> *If Y is in [Y2, Y3], then the associated values for variable x1 is in [x11, x12] or [x13, x14].*

Forming Qualitative Rules

There may be many sets of observation data; what we are interested is the general relationship between Y and a variable x_i, rather than the relationship on any particular set of data between them. Also note that a variable x_i may or may not have significant contribution to Y. A rule of thumb for determining that a variable x_i may have a significant contribution for Y is that there is consistent relationship between Y and x_i over all sets of observation data. On the other hand, if such kind of consistency does not exist, that particular variable x_i should be discarded in the further analysis.

The above discussion also explains why we are in favor of capturing result of data analysis in terms of qualitative reasoning.

Qualitative reasoning was first successfully used for qualitative physics (Kuipers 1986), but its application goes beyond that, because in many areas of natural and social sciences, we have only qualitative knowledge about behavior (Iwasaki, 1989).

In fact, observational data usually fluctuate, while what we are really interested is the intrinsic association (a kind of invariance) among the variables. A qualitative description for such kind of this relationship is more stable and useful than a quantitative (numeric) one. Note that this is not to say that numeric analysis is not important; on the contrary, a good numeric analysis using some mathematical or statistical software is the base for any valid qualitative reasoning.

Back to our original example in Figure 1, we notice that $[Y_2, Y_3)$ is the middle cluster for Y, $[x_{1_1}, x_{1_2})$ is the low cluster of x_1, and $[x_{1_3}, x_{1_4})$ is the high cluster of x_1. The production rule obtained in the previous section can then be rewritten in qualitative reasoning form:

If Y is in the middle, then the associated values of x1 are either low or high.

This rule implies that the middle value of Y is associated with two extremes of variable x_1 (that is, both low or high values of x_1 may contribute to the middle value of Y).

Qualitative rules constructed in this manner can be added to a domain knowledge base, thus automating knowledge acquisition process. There may be a need to check whether such acquired rules are consistent to the overall knowledge base; but this is not an issue to be addressed in this paper.

Entropy Data Analysis: A Case Study

The qualitative rules can only be constructed in combination with numeric data analysis over the observational data sets. This includes to determine the best way of clustering the values of dependent variable Y in regard to other variables (namely, x_i's). The data analysis can be carried out by using existing mathematical or statistical software, or developing your own.

The software we have experienced is based on entropy data analysis (sometimes called K-system analysis) (Jones 1985, 1986). It finds the factors that control and describe the behavior of the data (a factor, also called a substate, is a subset of variables each having its own values), thus reconstructing systems at the factor level.

Summary

The approach we have taken can be summarized below.

1. Perform Lebesque discretization on dependent variable, and dissect values of other variables using mathematical/statistical software (or computer program).

2. Form qualitative reasoning rules and add them into knowledge base.

Although there have been more and more discussion on the issue of coupling numeric computation and symbolic reasoning (eg. Kowalik 1985, Kitzmiller and Kowalik 1987, Fitch 1990), in many cases, the coupling is domain-dependent. In this paper, we have introduced an approach general enough to deal with a class of complex data which may fall in many different application domains. The paper by Abelson *et al.* discussed the development of

intelligent techniques appropriate for the automatic preparation, execution, and control of numerical experiments, and the tools that autonomously prepare simulation experiments from high-level specifications of physical models. In contrast to that paper, the purpose of our work is how to capture, explain and incorporate the *results* of scientific computing (by developing math or statistic programs or using existing software) into knowledge bases.

The contribution (or significance) of this approach can be justified from the consideration of knowledge acquisition and automated knowledge base construction. The numerical computation process can thus be coupled with symbolic reasoning.

References

[1] Abelson H., M. Eisenberg, M. Halfant, J. Katzenelson, E. Sacks, G.J. Sussman, J. Wisdom, and K. Yip (1989), "Intelligence in scientific computing," *Communications of the ACM*, 32(5), 546-562.

[2] Fitch J. (1990), "The symbolic numeric interface," *Computer Physics Communications*, 61(1), 22-33.

[3] Iwasaki Y. (1989), "Qualitative physics," in *Handbook of Artificial Intelligence*, Vol. IV, 323-414.

[4] Jones B. (1985), "Reconstructability considerations with arbitrary data," *Int. J. General Sys.* **11**, 143-151.

[5] Jones B. (1986), "K-systems versus classical multivariate systems," *Int. J. General Sys.* **12**, 1-6.

[6] Kowalik J.S. (1985), "Coupling Symbolic and Numeric Computing in Expert Systems," Amsterdam - North Holland.

[7] Kitzmiller C.T. and J.S. Kowalik (1987), "Coupling symbolic and numeric computing in knowledge-based systems," *AI Magazine*, Summer, 85-90.

[8] Kuipers B. (1986), "Qualitative simulation." *Artificial Intelligence*, 29: 289-338.

Automated Synthesis of FEA Programs

(An Extended Abstract)

Naveen Sharma and Paul S. Wang[1]
Institute for Computational Mathematics
Department of Mathematics and Computer Science
Kent State University
Kent, OH 44240

A combined symbolic-numeric approach to solve mathematical models on high performance computers is presented. Boundary and initial value partial differential equations (PDEs) frequently arise in many engineering and scientific applications. The solution of such models, however, is not trivial and sophisticated numerical methods are often required. Traditionally large software systems have been developed in `Fortran` to solve such models, such systems provide facilities for frequently used cases. These "canned" packages can not be used where new formulations, new solution procedures, or parallel execution is required.

We describe the design and development of **PIER** problem solving environment for mathematical modeling. Rather than programming in a conventional programming language, the modeler expresses the solution steps directly in textbook-style and the system automatically creates numerical solution procedures. Our emphasis is on the *finite element analysis* (FEA) method of solution although the techniques developed here are applicable for other methods of solution. Existing FEA packages are typically very large volumes of `Fortran` code. Not all sections of the FEA package need to be changed as one adapts the package to solve new models or to use a different element approximation. Also, some sections of the code are not amenable to useful parallelism. For these reasons, PIER handles only those FEA solution steps that are *computation-intensive* and are *problem-formulation* dependent. Such steps are referred to as the *key* FEA solution steps. Automatically generated codes for key steps are interfaced with other fixed parts of the FEA package. Figure 1 overviews our approach.

The major design goals for PIER are the following.

1. Provide a set of architecture-independent input specifications.

2. Automatically generate sequential and parallel programs to compile and execute on high-performance computers.

[1]Work reported herein has been supported in part by the National Science Foundation under Grant NSF CDA-9211137

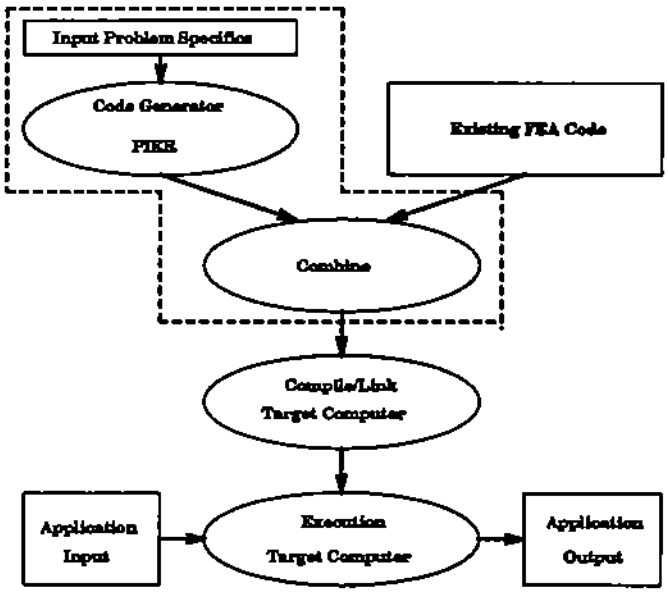

Figure 1: Overview of approach

PIER offers very high-level input specifications that can be used to describe FEA solution steps as well as numerical algorithms. The input specifications support high-level statements for discretized mesh, element, nodal properties, symbolic derivations for element shape functions and element equations, compact storage schemes for data arrays, and generation of sequential or parallel code. The numerical algorithms are specified in terms of numerical components (known as p-OPERATIONs) available in PIER knowledge-base. A p-OPERATION represents a well-defined symbolic or numerical computations required in a FEA solution step. p-OPERATIONs and regular F77 syntax can be intermixed the express numerical algorithms for FEA solution steps. The programming model is shown in Figure 2.

A p-MODULE is a straight-line[1] sequence of p-OPERATIONs. This mechanism helps identify the p-OPERATION-level parallelism. A p-METHOD represents a numerical algorithm and it is a composition of F77 statements, calls to p-MODULEs, and p-OPERATIONs.

PIER code generation comprises — *parsing*, *definition*, and *program synthesis* phases. The parsing phase recognizes and translates input statements into internal `Common Lisp` functions. Symbolic derivation of element formulation and definition of input/output data structures, mesh, etc. take place in the definition phase. Element shape functions and other quantities needed for element equations are derived here. PIER employs MAIXMA

[1]No control sequence is involved.

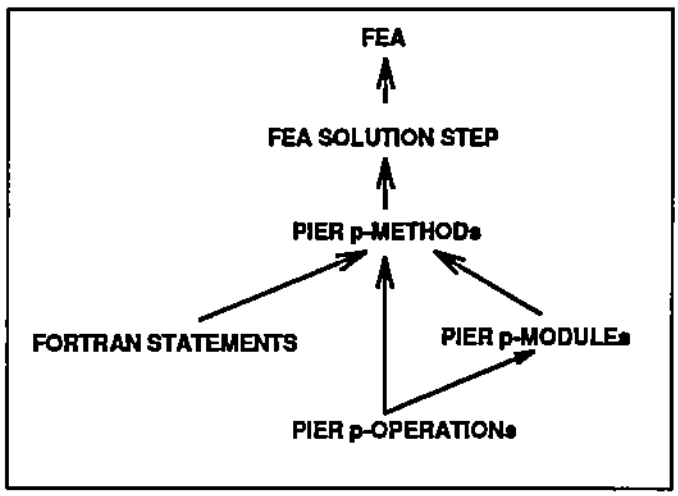

Figure 2: The programming model

for symbolic mathematical computations.

In PIER the FEA programs are synthesized by the method of *composition of program components*. p-OPERATIONs (in PIER knowledge-base), p-MODULEs (user-defined), and the p-METHOD (user-defined) represent program components in a numerical algorithm. The code generator incrementally transforms these statements into semantically equivalent pieces of code and combines these pieces to form FEA programs. Internally p-MODULEs and p-OPERATIONs are represented as *flowgraph*. The graph-based approach makes it easier to generate code for multiple target architectures. A p-OPERATION produces a piece of F77 code. Such pieces are composed to form F77 subroutine(s) representing a p-MODULE. The composition obeys the partial order depicted by p-MODULE definition. All the generated code at this stage is in `Common Lisp` forms, the **Gencray** translator is called to translate the `Common Lisp` forms into equivalent F77 statements.

Currently, we are applying the techniques developed in our research to numerical solution of the problems in liquid crystal physics. The overall problem is to compute the equilibrium configurations of liquid crystal materials in a finite containment subject to boundary conditions and the effect of applied electric and magnetic fields. The equilibrium stage is achieved by minimizing the free energy of the system. The free energy is approximated by a PDE-based mathematical model known as *Landau-de Gennes Free Energy* model. This minimization problem lends itself naturally to the finite element formulation. We are using various 2-D and 3-D elements (like tetrahedrals) to discretize the solution region and iterative numerical algorithms to solve the system of equations. Experiments with with two different FEA

computational schemes (element-by-element and traditional assembly approaches) are being carried out. So far, the target computer is Sequent Balance shared-memory multiprocessor.

Domain Decomposition Method Based on AI and Perturbation for PDEs

(Abstract)

Peiyuan Li and Richard L. Peskin[1]
CAIP Parallel Computing Laboratory
Rutgers University
Piscataway, NJ 08855-139

Domain decomposition for partial differential equations(PDEs) presented in this paper is implemented with singular perturbation analysis and artificial intelligence techniques. The programming work is accomplished in a symbolic programming environment.

Based on singular perturbation analysis of nonlinear PDEs, we can get an approximate solution of the equation, constructed from a solution of the reduced equation associated with the given nonlinear PDE, and correction terms. By applying the above methods, the domain decomposition method for PDEs can be developed. In this paper only elliptic equations are discussed.

The first step is solving the reduced equation associated with an elliptic equation. A general elliptic equation in 2-D can be described as follows,

$$a_1(x,y,u)\frac{\partial u}{\partial x} + a_2(x,y,u)\frac{\partial u}{\partial x} + h(x,y,u) = \varepsilon\nabla^2 u, \quad (x,y) \quad in \quad \Omega,$$

$$u(x,y,\varepsilon) = \varphi \quad (x,y) \quad on \quad \Gamma = \partial\Omega.$$

The associated reduced equation is,

$$a_1(x,y,u)\frac{\partial u}{\partial x} + a_2(x,y,u)\frac{\partial u}{\partial x} + h(x,y,u) = 0, \quad (x,y) \quad in \quad \Omega,$$

which is a quasilinear first order PDE. For the reduced equation, we cannot have a solution satisfying the complete boundary condition, only a solution which satisfies the given boundary condition along portions of Γ. With the method of characteristics for solving the first order PDE, we obtain the first order system,

$$\frac{dx}{ds} = a_1(x,y,u),$$

[1]This research was sponsored by National Science Foundation grants ECS-9110424 and IRI-9116588.

$$\frac{dy}{ds} = a_2(x, y, u),$$
$$\frac{du}{ds} = -h(x, y, u),$$

with initial conditions at $s = 0$ given as $x = x(\tau)$, $y = y(\tau)$ and $u = u(\tau)$. This first order system is solved in MAPLE, a symbolic computation system. In MAPLE, with the method of elimination, a third order ordinary differential equation(ODE) results from the first order system. If this ODE is linear, it can be directly solved in MAPLE. However if it is nonlinear it may be solved only under certain circumstances. That is, the nonlinear ordinary equation is of the form $\mathcal{F}(x', x'', x''') = 0$ which can be transformed to a lower order ODE then solved in the symbolic computation system. If the coefficient terms in a reduced equation are not functions of u, the reduced equation become a linear first order PDE. Solving the linear PDE in MAPLE is much simpler than a quasilinear one.

With u_0, the solutions of the reduced equation, we can partition the domain into a set of subdomains. Among all possible partitions we are only interested in the stable partitions with which u, the solution of the elliptic PDE is approximated in every subdomains by u_0 and is convergent to u_0 on ε.

Certain conditions required by convergence have to be imposed on such solutions. If we assume the region Ω in 2-D is described by a function $F(x, y)$ in the sense that $F(x, y) < 0$. Then we set

$$\gamma(x, y) = a_1(x, y, u_0(x, y))\frac{\partial F(x, y)}{x} + a_2(x, y, u_0(x, y))\frac{\partial F(x, y)}{y}.$$

The main aspect of these conditions can be simply described:

If $\gamma(x, y) \geq 0$ in subsets of Γ, a solution u exhibits boundary layer behavior along the portions of Γ.

If $\gamma(x, y) < 0$ in subsets of Γ, u_0 must satisfy the given boundary data along the portions of Γ.

An analysis system based on artificial intelligence techniques is developed to search stable partitions of domain. In the system, a set of rules is built based on conditions of stable partition. In the system we can search all the possible stable partitions as well as determine the associated boundary and interior layers.

A New Search Method in Domain Decomposition for ODEs

(Abstract)

Peiyuan Li and Richard L. Peskin[1]
CAIP Parallel Computing Laboratory
Rutgers University
Piscataway, NJ 08855-139

Using singular perturbation and artificial intelligence techniques in domain decomposition for ordinary differential equations(ODEs) has proven to be a successful method. In this method the analysis is accomplished in two steps.

A second order ODE can be described as follows,

$$\varepsilon \frac{d^2y}{dt^2} = h(t,y), \quad a < t < b, \tag{1}$$

$$y(a,\varepsilon) = A, \tag{2}$$

$$y(b,\varepsilon) = B. \tag{3}$$

The associated reduced equation is defined by setting $\varepsilon = 0$.

$$0 = h(t,u), \quad a < t < b, \tag{4}$$

and solutions of the reduced equation will be denoted as u(t).

The first step is solving the reduced equation and gaining stable ranges. In each stable range, $y(t,\varepsilon)$, the solution of the given ODE is approximated by $u(t)$, the solutions of the associated reduced equation and correction terms and is convergent to $u(t)$ on ε. This step is implemented in MAPLE, a symbolic computation system.

The second step is searching for a stable partition from all possible partitions of the domain in which all subdomains are stable, as well as determining certain connectors(layers), between subdomains or boundary and subdomain, that satisfy some conditions required by convergence. Russo has developed a system which uses artificial intelligence planing techniques to generate a stable partition. In this paper we present a more efficient and direct search method.

In the new method we define a search space which is stretched on the reduced solutions. The space is represented by a B-tree. This tree consists of three parts:

[1]This research was sponsored by National Science Foundation grants ECS–9110424 and IRI – 9116558.

node A node in the tree has $n-1$ children except the root, if the corresponding reduced equation has n reduced solutions. In general a node in the tree contains six fields, each of which stores some useful information respectively: a parent number, a connector, a stable range, a parameter of local criterion, a parameter of reduced solution and a marker.

root The top node of the tree corresponds to an end of boundary, and the starting position of a search. The root has n children if the corresponding reduced equation has n reduced solutions.

edge The edge between a parent node and a child node corresponds to a parameter of the reduced solution. So a part of branch of the tree composed of an edge incident upon a node , the node and an edge exiting from the node, indicates that the connector in the node joins the two reduced solutions labeled by the two parameters on the two edges.

The object of the search is to construct an acceptable path that traverses the space from an end of a boundary to another end of the boundary. In the program the path is expressed in terms of a linked list which consists of a sequence of nodes.

Searching a path in a reduced space, that is searching a branch in a tree, has to use a set of rules. Russo designed F-rules for a planner based on conditions of stable partition. A lightly modified set of rules is used in the tree search. Two search strategies, DFS search and backtrack, are used in the search process.

The second aspect is accomplished in Smalltalk, an object oriented programming environment.

Object Oriented Mathematical Programming and Symbolic/Numeric Interface

(Extended Abstract)

Larry Lambe[1] and Richard Luczak
Department of Mathematics and Computer Science
Kent State University
Kent, OH 44240

Many problems of interest to engineers and scientists involve the use of numeric libraries. It could be helpful and convenient if the output of numerical routines could be processed directly in a symbolic computational environment. In addition, it could be convenient or revealing if systems could be prepared for numeric processing in a symbolic computational environment.

Axiom (formerly called Scratchpad) is a language based on the notions of "domains" and "categories". Using the Axiom System, a very high level of consistent mathematical representation can be quickly obtained. The language supports object oriented paradigms and polymorphism and has a compiler. While the compiler provides a way to produce efficient procedures, there is still a desire to take advantage of the work done by others (e.g., the NAG Numeric Library, or personal libraries, etc.).

The main concepts (categories, domains, and packages) of the Axiom language will be described and a few simple examples of Axiom programming will be presented.

It will be shown how one can interface with external programs in Axiom using a variety of simple methods. For example, an interface to the NAG numeric library and graphics facilities will be covered in detail. Once done, the interface becomes invisible and convenient to the user.

Specifically, one can use the NAG FORTRAN library function D01AKF for numerical integration to produce an Axiom package with "signature"

nagNumericLibraryIntegrate : (**EXPR SF, L L SF, SF**) → **L SF**

where **SF** denotes the domain SmallFloat (all double precision floating point numbers), **EXPR SF** denotes the domain "Expression SmallFloat" (all expressions with SmallFloat coefficients), **L SF** denotes the domain of "List SmallFloat" (all lists of SmallFloats), and **L L SF** denotes all lists of lists of SmallFloats. By this we mean that nagNumericLibraryIntegrate is a function of three variables, ex, listIntervals, and to which returns a list of double precision numbers. The expression ex represents some function $f(x)$ to be integrated, listIntervals is

[1]Partially supported by grant No. CCF-9207241.

a list of two element lists $[a_i, b_i]$ representing intervals over which ex is to be integrated and tol is a tolerance for the precision. The value returned is

$$\left[\int_{a_1}^{b_1} f(x)dx, \ldots, \int_{a_n}^{b_n} f(x)dx\right]$$

Using a sequence of such lists of numbers (based on refinements of a partition of the interval $[a, b]$, for example), we can use the Axiom system to produce a sequence of polynomial approximations to the integral. Using graphical facilities, we can get a fair idea of how the approximations are behaving as we increase the mesh.

Task Structure: A Vocabulary for Integrating Numerical Methods and Knowledge-based Systems

Ananthapadmanaban Sundaram, James K. McDowell and Martin C. Hawley
Composite Materials and Structures Center and AI/KBS Lab
Research Complex - Engineering
Michigan State University
East Lansing, MI 48824-1326

Often Expert Systems (ES) or Knowledge-based Systems (KBS) are described in terms of their implementation details, namely rules, frames or objects. Though important, implementation details are not at the right level of description appropriate for systems design. If the complex problem solving activities captured in such systems can be viewed as a type of information processing task then an alternative vocabulary is possible (and useful) for such descriptions. This alternative vocabulary is called Task Analysis and the resulting description of a KBS is referred to as a Task Structure.

The task structure for a problem solving system illustrates what tasks the KBS performs and how such tasks are broken down into subtasks. For each subtask, a method is identified that when performed will satisfy the goals of the subtask. The method represents a miniature knowledge-based problem solver. Each method contains a specific knowledge representation and inference strategy for performing the necessary problem solving. The ubiquity of certain combinations of knowledge representation and inference strategy packages appearing across activities and across domains gave rise to the Generic Task (GT) Theory of KBS. Task analysis, with it's task/sub-task/knowledge-based method/knowledge+inference organization offers a descriptive language for KBS that facilitates design of such systems and their subsequent understanding.

Rarely are problem solving activities in engineering exclusively symbolic. Often sophisticated numerical computations are necessary for engineering design and operations. The integration of numerical computation and KBS continues to be an important area of research. Many integration efforts have focused on implementation aspects between KBS and other software, for example, can the rule-base be called from the computation code or can the KBS access a properties database? This type of integration does not emphasize problem solving system design. If both numerical methods and KBS could be described using the same vocabulary then a medium for integration at the problem solving system design level would be possible.

This raises the issue as to whether Task Analysis can be used to describe numerical computational systems in the same manner as a KBS? Our initial explorations indicate that

Task Analysis vocabulary can be slightly modified and be used to represent numeric computations. For numeric computations, the description takes on the form task/sub-task/numerical method/numerical information+numerical computation. Since the same vocabulary can be used to describe both knowledge-based problem solving and numerical problem solving, the Task Structure represents a medium for integrating knowledge-based systems and numerical computation.

Many physical systems in engineering both static and dynamic can be described quantitatively in terms of a set of differential equations which express implicitly the nature of the system variables in terms of independent variables of space and time. Such a description most often forms an essential part of a larger problem solving architecture (possibly involving KBS). This type of computational scenario will be used as a demonstration of Task Analysis' descriptive power and it's utility as a integration medium.

The solution of any physical system that is being defined by a set of differential equations can involve the following problem solving operations: 1). A domain of definition or the physical boundaries of validity of a differential equation description. This involves the definition of the geometry in spatial domains. 2). Evolving elements for the domain (finite elements or grids) depending on the nature of the problem and solution approach. 3). Application of appropriate boundary conditions on the equations. 4). Iterative solution for the variables involved and the gradients of the variables. 5). Refining the elements or the grid size based on the gradients of variables.

A brief task analysis is presented here for the above steps. Steps 1 and 2 could be condensed into the single subtask of mesh generation. This involves information concerning the coordinates of the surface or the volume and the boundaries of the domain. The subtask itself has a number of methods depending on the nature of the domain and the discontinuities involved and each of these methods involves a different information requirement. Steps 3 and 4 are part of the subtask of evaluating the system variables at every grid point or element of the domain defined by the mesh generation subtask. Once again there are many methods for the evaluation of the variables depending upon the dimensions involved and the type of boundary conditions to be applied. Step 5 is a refinement subtask. The meshes and the grids are refined to provide for finer size in the regions of the domain where large gradients are found.

The brief task structure outlined above is presented in detail in a specific example of a simulation of a composite curing process which is done using a traditional differential equation approach. The integration of this numerical computation with other knowledge-based problem solving (design and fabrication) is demonstrated using the task analysis vocabulary.

Scientific Problem Solving in a Distributed and Collaborative Geometric Environment[1]

Vinod Anupam, Chandrajit Bajaj, Steve Cutchin, Susan Evans, Insung Ihm,
Jindon Chen, Andrew Royappa, Daniel Schikore and Guoliang Xu
Department of Computer Sciences
Purdue University
West Lafayette, IN, 47907

Introduction

We present a distributed and collaborative environment called SHASTRA for cooperative scientific problem solving. At its core SHASTRA has a powerful collaboration substrate – to support synchronous multi-user applications, and a distribution substrate – which emphasizes distributed problem solving. At the application level, SHASTRA provides collaboration and multimedia communication facilities together with graphic interfaces based on X-11 windows that let users cooperate in a "as if in the same room" environment. A synergistic union of these two elements yields a sophisticated problem solving environment.

In this extended abstract we describe how SHASTRA toolkits, GANITH, a surface modeler, SHILP, a solid modeler, VAIDAK, a medical imaging toolkit, and BHAUTIK, a mesh generation and physical analysis toolkit, are used in the custom design of artificial hip implants, the optimization of the connection of a jet engine to an airplane wing, and the efficient computation of molecular "docking" for drug screening.

SHASTRA: An Overview

The SHASTRA [1] environment consists of a group of interacting applications. Some applications are responsible for managing the collaborative environment (the Kernel applications), whereas others provide specific services (the Service Applications), while yet others provide scientific design and manipulation functionality (the SHASTRA Toolkits). Service applications are special purpose tools for multimedia support – providing textual, graphical, audio and video conferencing. See Figure 1 which shows the video support interface in SHASTRA.

[1]Supported in part by NSF grants CCR 90-02228, DMS 91-01424, AFOSR contract 91-0276 and a grant from AT & T

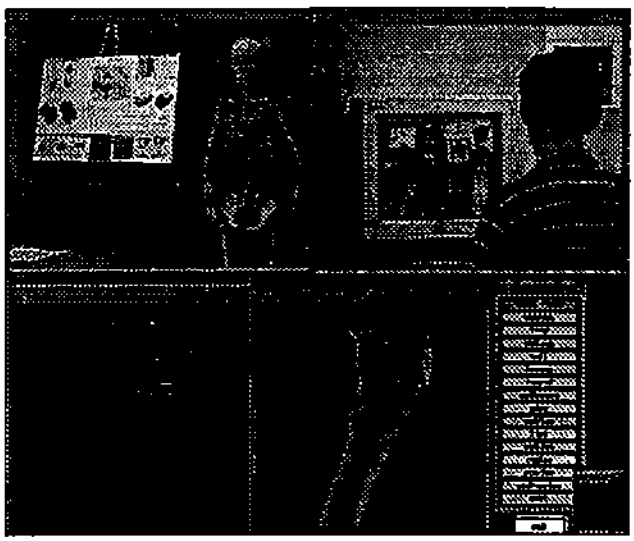

Figure 1: Video support for analysis in SHASTRA

A researcher uses a live video image in SHASTRA to verify femoral topology of a reconstructed femur model. Different tools register with the environment at startup, providing information about what kind of services they offer (Directory), and how and where they can be contacted for those services (Location). The environment provides mechanisms to create remote instances of applications and connect to them in client-server mode (Distribution). In addition, the environment provides support for a variety of multi-user interactions spanning the range from master-slave electronic blackboarding to simultaneous multiple-user interaction (Collaboration). It provides mechanisms for starting and terminating collaborative sessions, and joining or leaving them.

Custom Hip Prosthesis Design

BHAUTIK and the other related SHASTRA toolkits provide a distributed environment for the interactive custom design and analysis of a hip prosthesis [4,5]. VAIDAK accurately reconstructs a solid model of a patient's femoral bone [3], which SHILP can use as a guide for the design of an artificial hip replacement. Using these two models, BHAUTIK can analyze the interaction of the prosthesis with the original bone of the patient, and provide feedback for possible modifications for the bone and the prosthesis design. Figure 2 shows a BHAUTIK problem session in which 3D structural analysis is being performed on part of

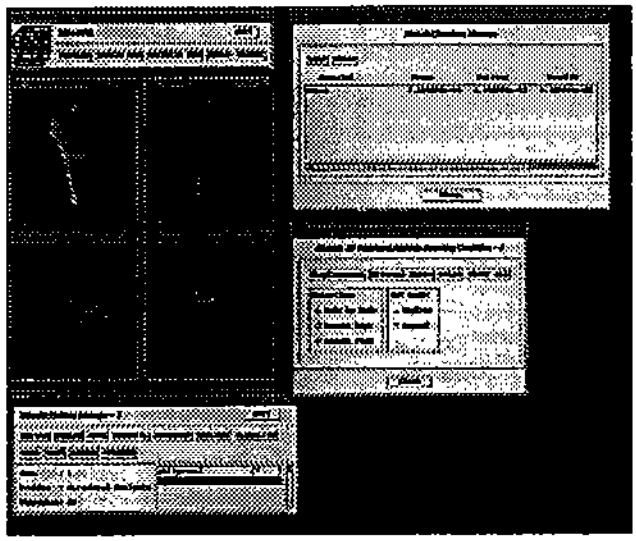

Figure 2: Stress transfer modeling from prosthesis to femur in BHAUTIK

the femur.

Shape Optimization of Engine and Wing Connections

We are also using the SHASTRA environment to study the effect of geometry of connections between engines and aircraft wings my simulating 3D viscous flow around the geometry [7,9]. Figure 3 shows a volume mesh, generated in BHAUTIK, of the fitted 3D model of the engine and a bounded surrounding region. The geometric design task of for engine model was accomplished via SHILP in conjunction with remote calls to GANITH for surface fitting operations. We are currently trying to link to available CFD (computational fluid dynamics) solvers (like P/FLOTRAN of the PATRAN package), while handling all the geometry, mesh generation and graphics visualization tasks in SHASTRA.

Molecular Docking for Drug Screening

An interactive application is being developed to compute and visualize the "docking" of drug and protein molecules [8] under molecular Brownian motion. The application reads in a description of the atom locations of a molecule from a file, computes the bonding information and then displays the molecule through SHASTRA. Both the analysis and visualization of

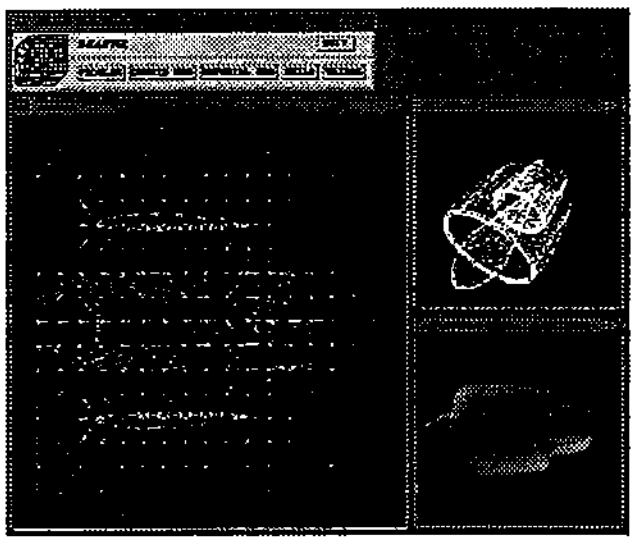

Figure 3: Volume mesh surrounding a jet engine and computed in BHAUTIK

the potential energy surfaces of the molecules and the stationary points on these surfaces require sophisticated surface fitting (interpolation) techniques. New algorithms are being developed to characterize the molecular structure [6] and the potential energy surfaces [2]. For specific polymers and compounds, one would produce customized optimization searches to localize stationary points and intrinsic curves on the potential energy surfaces to guide the docking solution.

References

[1] V. Anupam, C. Bajaj, and A. Royappa, "The SHASTRA distributed and collaborative geometric design environment," Computer Science Technical Report, CAPO-91-38, Purdue University, Department of Computer Sciences, 1991.

[2] C. Bajaj, "Surface fitting with implicit algebraic surface patches," H. Hagen (ed.), in *Types in Surface Modeling*, pp. 23-52, SIAM Publications, 1992.

[3] C. Bajaj and M. Fields, "The VAIDAK medical image model reconstruction toolkit," in *Proceedings of the 1993 Symposium on Applied Computing*, pp. x-y, Indianapolis, Indiana, 1993.

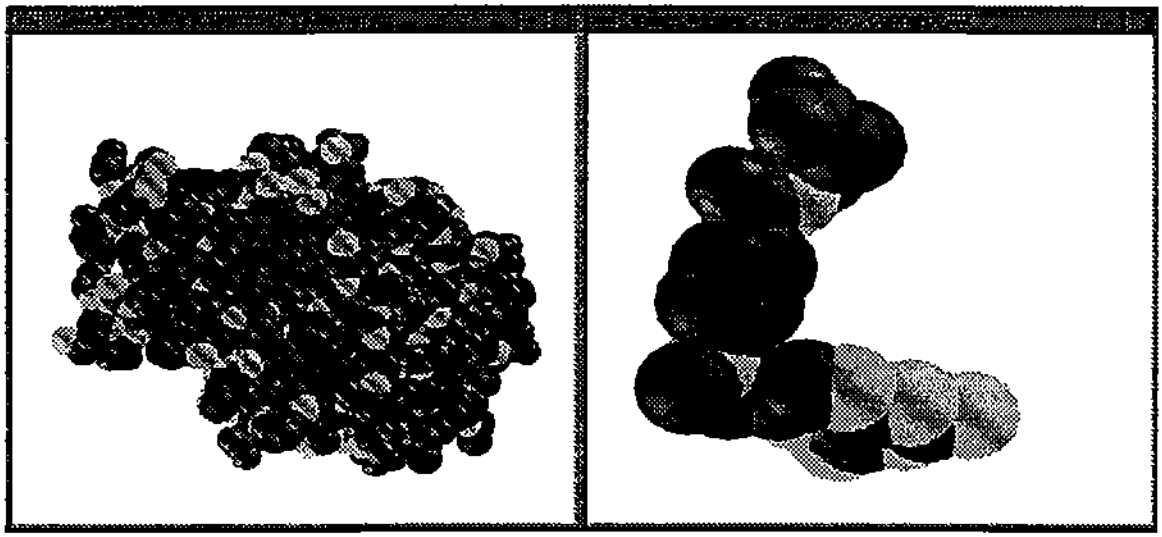

Figure 4: Modeling of protein and drug molecules

[4] C. Bajaj and D. Schikore, "The distributed design of hip protheses using BHAUTIK," in *Proceedings of the 1993 Symposium on Applied Computing*, pp. a-b, Indianapolis, Indiana, 1993.

[5] W. Bargar, "Shape the implant to the patient," *Clinical Orthopaedics and Related Research*, 10(249):73-78, 1989.

[6] H. Edelsbrunner, "Weighted alpha shapes," Computer Science Technical Report, UIUCDS-R-92-1760, University of Illlinois, Urbana-Champaign, 1992.

[7] O. Pendergraft, A. Ingraldi, R. Re, and T. Kariya, "Installation effects of wing-mounted turbofan nacelle-pylons on a 1/17-scale, twin-engine, low-wing transport model," NASA Technical Paper 3168, NASA Langley Research, 1992.

[8] B. Shoichet, D. Bodian, and I. Kuntz, "Molecular docking using shape descriptors," *The Journal of Computational Chemistry*, 13(3):380-397, 1992.

[9] R. Smith and P. Kerr, "Geometric requirements for multidisciplinary analysis of aerospace-vehicle design," NASA Technical Report, AIAA 92-4773, NASA Langley Research, 1992.

ParPDELab: A Problem Solving Environment for the Development of Partial Differential Equation Based Applications on Parallel Machines

Sanjiva Weerawarana, Elias N. Houstis and John R. Rice [1]
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Introduction

Partial differential equations (PDEs) are the fundamental mathematical tools for describing the physical behavior of many application processes in science and engineering. Hence, a reasonably large software base for solving PDEs exists today. However, most existing PDE software systems deal primarily with the solution of specific classes of PDE problems on sequential and vector machines. Even amongst these systems, only a handful provide support for any aspects of the problem other than the numerical solution phase. Important tasks such as problem specification, method selection, solution visualization, and performance evaluation are completely left to the user of the package. Furthermore, most PDE systems have not been designed with the intention of being used as a component of a larger process, and hence have esoteric input/output mechanisms that are difficult to adapt to a given application.

This limitation has forced most application scientists with PDE problems to implement their own PDE solving code for the problem at hand. These codes are often impossible to modify for similar analyses and make very limited use of the vast amount of freely available support software, resulting in large scale waste of human resources. This effort to develop special–purpose PDE solvers is a distraction to the application scientist as his/her interest is in the larger problem, not just the solution of a partial differential equation. The goal of software for solving PDE problems should be to provide an environment that allows the application scientist to concentrate on the science/engineering aspect of his/her work and not on the mathematics and computer science problem of writing programs to solve PDE problems.

The concept of problem solving environments (PSEs) evolved to address exactly the issue mentioned in the previous sentence. The task of a PSE, for, say partial differential

[1]This work was supported by NSF grants 9123502–CDA and 9202536–CCR, AFOSR F49620–92–J–0069 and by the Intel Foundation.

equations, is to provide an environment that allows its user to concentrate on specifying the partial differential equation problem at a very high (mathematical/physical) level using terminology familiar to him/her and to visualize and understand its solution at the same abstract level. Low level details of exactly how the the solution is obtained is often not of interest to the user.

ParPDELab is our attempt at developing a comprehensive, interactive problem solving environment for solving PDE problems and for building custom software for PDE based applications. As a problem solving environment, ParPDELab will provide mechanisms for specifying the PDE problem, for specifying a solution scheme, and for visualizing and analyzing the solution. As an environment supporting PDE based application building, it will allow one to apply ParPDELab in the application–context by using its functionally equivalent application programming interface.

The rest of this extended abstract is organized as follows: Section provides an overview of our current //ELLPACK[1] system. Section considers the design goals for ParPDELab. Section provides a very broad overview of the proposed architecture of ParPDELab.

The //ELLPACK System

//ELLPACK is a problem solving environment for solving PDE problems using parallel machines. Its domain of applicability is two and three dimensional second order systems of (coupled) PDEs. //ELLPACK consists of three components– a high level language for specifying PDE problems and their solution, a graphical environment for developing //ELLPACK programs and visualizing solutions, and libraries of PDE solver components for solving PDEs on various parallel machines.

The //ELLPACK language is a superset of the ELLPACK language. The main extension is of course in the direction of parallelism where transparent, high level support for the parallel solution of PDE problems has been added.

The //ELLPACK user interface has been built using the X Window System and consists of a program editor along with visual editors for specifying the domain, mesh, decomposition, etc.. Each of these editors assists in the specification of one aspect of the PDE problem or the solution process and interact with one another via the program editor.

The numerical capabilities of //ELLPACK is also a superset of the ELLPACK capabilities. Several parallel discretization methods and parallel linear solvers have been added. Parallel modules are implemented using a portable communication library (PARMACS) so

[1] //ELLPACK is read "parallel ELLPACK."

as to have them available on multiple machines. The numerical capabilities of //ELLPACK have been greatly improved by integrating other's PDE solving packages as "foreign systems" that are transparently accessible from within the //ELLPACK environment.

The //ELLPACK environment is driven by a top–level tool that provides several "editors" for performing various editing tasks. A //ELLPACK program is executed by translating it to FORTRAN and then compiling it using the native FORTRAN compiler. The back–end component of //ELLPACK (solution visualization/analysis and performance evaluation) is accessible after the problem is solved.

Requirements and Design Choices

In this section, we discuss some of the requirements of a PDE PSE and the choices that are available to us for providing them.

A fundamental limitation of PDE solving systems such as //ELLPACK is that they are geared towards human end–users only. Such systems were not designed to be used in a larger application context and hence are not readily adaptable for such purposes. The requirements of ParPDELab are then to be a usable PDE solving PSE while still being applicable as a component of a larger application. To this end, we list the following design goals for ParPDELab:

- high level language for specifying PDE problems
- graphical editors for specifying PDE problems
- domain decomposition tools for solving the problem using parallel machines
- support for many PDE solving packages
- support for many linear system solvers
- expert system tools for solution method selection and other assistance
- solution visualization/analysis and performance evaluation facilities
- functionally equivalent application programming interface for use in application context

We briefly consider the choices available for the realization of the above goals in ParPDE-Lab. The guiding principle behind our design is to avoid reinventing or reimplementing the wheel. Hence, an important aspect of ParPDELab will be the abstraction of the essence

of PDE solving components/tools/utilities and the coercion of existing implementations to meet this abstraction. We will develop a uniform interface for attaching such components to ParPDELab in order to allow its continual updating and also so that users can install their own special components. In this extended abstract, we will not address issues in the realization of each task separately. Instead, let us address the task of integrating a wide variety of software components into a cohesive whole.

The most direct approach to software integration would be to implement an inter–process communication mechanism using low level facilities such as Berkeley sockets, Sun Microsystem's Open Network Computing (ONC) platform or the Open Software Foundation's Distributed Computing Environment (DCE). However, this is a time–consuming task and may not be necessary as several higher level integration platforms are now available. We are currently evaluating ABE,ITHACA, REXX, PCTE, and for use as the software integration platform for ParPDELab. Some of the features that we need are: data communication between heterogeneous architectures, data compression, synchronous and asynchronous communication facilities, global communication mechanisms, process management mechanisms, tool configuration facilities and directory services. In addition, we need the ability to integrate the communication/integration system with existing clients written in several languages and also user interface clients implemented using the X Window System.

The Architecture of ParPDELab

Let us briefly look at the architecture of ParPDELab, which is based on our view of how problem solving environment for PDE based applications are or should be built (see Figure 1. At the heart of this layered architecture is the generic PSE layer, the component that is common to all problem solving environments, not just the ones based on or using PDEs. The next layer to be provided is functionality specific to PDE problems. Finally, the outermost layer must provide functionality that is specific to PDE based applications.

Generic PSE layer

A PSE roughly consists of several specialized agents that communicate and collaborate with each other to solve the problem at hand. Hence, the needed infrastructure consists of mechanisms for a collection of tools or editors to communicate and to share information with each other. We call the component that provides the first set of mechanisms a *software bus* and the latter an *object manager*. Hence, the software bus is the component that provides an inter–editor communication platform and editor management tools (for example, process

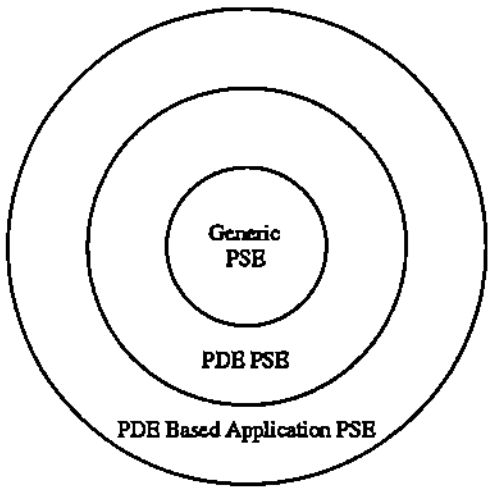

Figure 1: Layered architecture of problem solving environments for PDE based applications

management and editor configuration). The object manager is the component that provides a shared storage area for inter–editor communication and mechanisms for assigning and monitoring object attributes.

Software bus

As mentioned above, the software bus component provides two types of tools: tools for managing editors and tools for editors to communicate with each other. Management tools include process management (for example, (local and remote) process initiation, interruption,

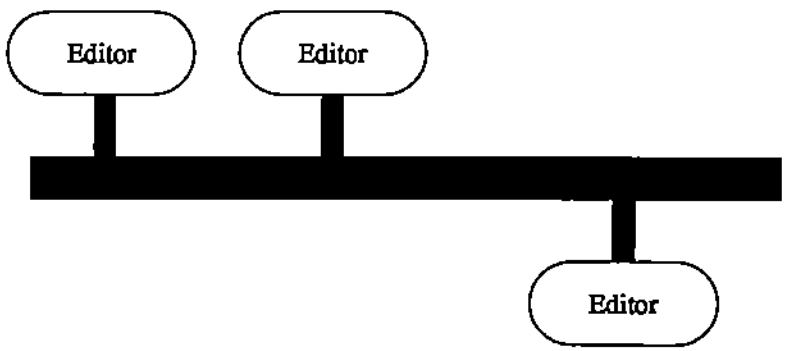

Figure 2: Architecture of a generic problem solving environment

and termination), editor configuration (for example, starting a preconfigured set of editors interconnected in a predefined manner), and various lookup tables (for example, finding the location of a named service).

The communication tools that are needed in a PSE include data representation encapsulation (to hide data representation differences on heterogeneous architectures, for example, XDR, data compression (to allow transmission of very large messages, for example, meshes), both synchronous and asynchronous communication capabilities, and mechanisms for global communication (for example, broadcasts and multicasts).

Object manager

The main function of the object manager is to support indirect inter–editor communication by providing a shared workspace. The workspace is accessed via requests to the object manager while editors are notified of changes to the workspace via asynchronous events generated by the object manager. Furthermore, the object manager provides file system interaction services to editors in the PSE and also a mechanism for transparently converting the types of objects.

Objects are immutable, have opaque types, have attributes (for example, name and owner) and may be persistent. As the object space is shared by multiple editors, the object manager provides a simplistic security model via access control lists.

The immutability of objects requires that the object manager provide garbage collection. This is implemented via a reference count scheme. The reference count of an object is incremented when an editor indicates interest in it or when another object depends on it. The reference count is decremented when such an interest or dependency is removed and is tagged for removal when the reference count becomes zero.

The set of requests supported by the object manager includes requests for creating and accessing objects, requests for listing objects of a given type, requests for expressing interest in objects, and requests for expressing inter–object dependencies. The object manager also provides file system interaction services for all the editors via a specific set of requests for this purpose. These requests support standard file system operations such as create, delete, save, and locate. Type conversion for objects is provided by another component of the object manager. This is implemented by maintaining a type lattice and by providing means to register type converters that convert from one type to another. This scheme allows us to provide transparent type conversion facilities for editors.

PDE PSE layer

This layer augments the generic problem solving environment layer with functionality that is specifically needed by partial differential equation problem solving environments. The functionality provided by ParPDELab therefore represents an instance of this layer.

This layer defines a specific set of object types that are specific to PDE problems and their solution. These include Operators, Domains, Meshes, and Solutions. Specific type relationships and conversion facilities are also defined at this level; for example, the relationship between domains and meshes and how to convert from one to another. A set of clients that manipulate these object types is the main component of this layer. For example, a mesh generator client creates/edits mesh objects and a domain editor client creates/edits domain objects. An abstraction for implementing PDE solvers using parallel machines is also supported at this layer.

PDE based application PSE layer

This layer adds the functionality that is specific to the application and constitutes the user–level for PDE based application PSEs. Typically, a PDE based application would have one fixed PDE model that is solved using different parameters at each instance. Hence, the application layer would basically consist of a user interface for accessing those parameters and, of course, specialized visualization tools.

Conclusions

This paper presents the design of the ParPDELab PSE for developing PDE based applications on parallel machines. In a nutshell, ParPDELab consists of a set of editors that collaborate with each other via a shared workspace to solve the PDE problem at hand. An important aspect of this design is the abstraction of the essence of various PDE components in order to develop a system that supports these abstracts and provides services by interfacing with various existing packages. Furthermore, by attempting to design a stable system that can solve "real world" problems, we are forced to address some issues that are often ignored (for example, security and robustness).

Intelligent Simulation of Physical Systems

Feng Zhao
Dept. of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Computationally simulating complex physical systems in engineering design has become a common practice. Yet most of today's simulations rely entirely on extensive numerical computations and laborious human analysis. Human engineers have to shoulder the burden of translating physics and constraints into models, preparing numerical simulations, interpreting consequences of the experiments, and performing design tasks. Moreover, nonlinear systems can exhibit extremely complicated behaviors that defy human analysis and pure numerical simulations. The complexities of these systems are largely due to nonlinearities, high dimensionality, and uncertainties of the systems and environments the systems operate in.

The difficulties in the traditional engineering simulation arise from the lack of (1) parsimonious representations capturing the essence of physical systems and amenable to efficient computations, (2) efficient modeling algorithms for constructing the representations, and (3) effective reasoning methods that can use the representations to compute and synthesize useful properties for the systems. The lack of computable representations for physical dynamical systems hinders the exploitation of the special properties of the systems and the attainment of the maximum performance for the design. The simulation and design of the dynamical systems are limited by the available computational power and the complexities of the systems.

Hybrid Simulation

We address the difficulties of traditional numerical computing by developing computer representation and simulation technologies necessary for enabling programs to autonomously perform and interpret numerical simulations. The domain of control system simulation and design has been investigated. The design of complex control systems requires powerful simulation technologies to represent, reason about, and manipulate the dynamics of nonlinear systems. We demonstrate that difficult control synthesis tasks can be automated, using a computational workbench consisting of programs that actively exploit knowledge of nonlinear dynamics and phase space. We employ hybrid computation integrating symbolic and

numerical methods with AI reasoning and representation techniques in (1) manipulating, visualizing, and animating the dynamics of physical systems based on a phase-space representation, (2) guiding the execution of numerical computations, and (3) codifying sophisticated mathematical knowledge in engineering problem solving.

More specifically, we have developed a qualitative representation for complex behaviors of dynamical systems in phase space and a design language for computationally expressing and manipulating these behaviors [8, 9]. We are interested in representing the qualitative behaviors of dynamical systems for control analysis and design. One useful qualitative representation of the phase space of a dynamical system is in terms of equilibrium points and limit cycles, stability regions, trajectory flows exhibiting the same qualitative features, and the spatial arrangement of these geometric objects. The qualitative representation captures the gross aspects of dynamics in a relational graph of phase-space structure and a set of discrete objects called *flow pipes*—the equivalence classes of behaviors. The design language describes a control design task in terms of well-defined geometric, combinatorial operations on the flow pipes. This language helps formalize aspects of implicit expert reasoning of control engineers in solving control design problems. The representation and the language are developed independently of the orders of systems, *i.e.*, the dimensionality of phase spaces.

An Environment: The Control Engineer's Workbench

We have constructed a computational environment, the Control Engineer's Workbench, integrating a suite of programs that automatically analyze and design high-performance, global controllers for a large class of nonlinear systems using the qualitative phase-space representation [11]. Given a model of a physical system and a control objective, the Control Engineer's Workbench analyzes the system and designs a control law achieving the control objective. A user typically interacts with the Workbench in the following way.

The user first tells the Workbench about the system: he inputs a system model in terms of a differential equation, parameter values, and bounds on state variables for analysis in the form of a phase-space region. The user also tells the Workbench about the requirements on the control design: he specifies the desired state for the system to settle in, the initial states of the system, the allowable control parameter values, and the constraints on the control responses.

The user then asks the Workbench to analyze the system within the parameter ranges of the model. The Workbench visualizes the totality of the behaviors of the system over the parameter ranges; it represents the qualitative aspects of the system in a data structure and reports to the user a high-level, symbolic summary of the system behaviors and, if necessary,

a graphic visualization of the phase-space qualitative features.

Next, the user instructs the Workbench to synthesize a control law for the system, subject to the specified design requirements. The Workbench searches for the global control paths that connect the initial states of the system and the desired state, using the qualitative description about the system. More specifically, the search is conducted in a collection of discrete entities representing trajectory flows in phase space. After the global control paths are established, the Workbench determines the controllable region of the system and the switching surfaces where control parameters should change values. A synthesized control reference trajectory consists of a sequence of trajectory segments, each of which is under a constant control. For a point-to-point control design, the Workbench further constructs a smoothed trajectory connecting an initial state and the desired state.

The Workbench serves as an intelligent assistant to control engineers. The components of the Workbench are shown in Figure 1:

- the MAPS program for simulation and interpretation
- the Phase Space Navigator for control synthesis
- a graphic program for visualizing the design
- a user interface for communication with the system.

The component for model building is not in the Workbench yet. The programs in the Workbench implement various algorithms: symbolic differentiation, numerical algorithms on differential equations, modeling of geometric structures, clustering of equivalence classes, graph algorithms, etc. The inference mechanism of the Workbench uses these programs to construct a qualitative phase-space structure for representing a system, to check for the consistency of the structure, and to reason about and manipulate the representation through a graph of flow pipes.

Conclusion

The Control Engineer's Workbench complements and enhances human design activities. By providing manipulation and visualization mechanisms for the design, the Workbench relieves engineers from routine, tedious low-level tasks of simulation and interpretation, allows the engineers to focus on higher-level design issues, and enlarges the design space the engineers can explore.

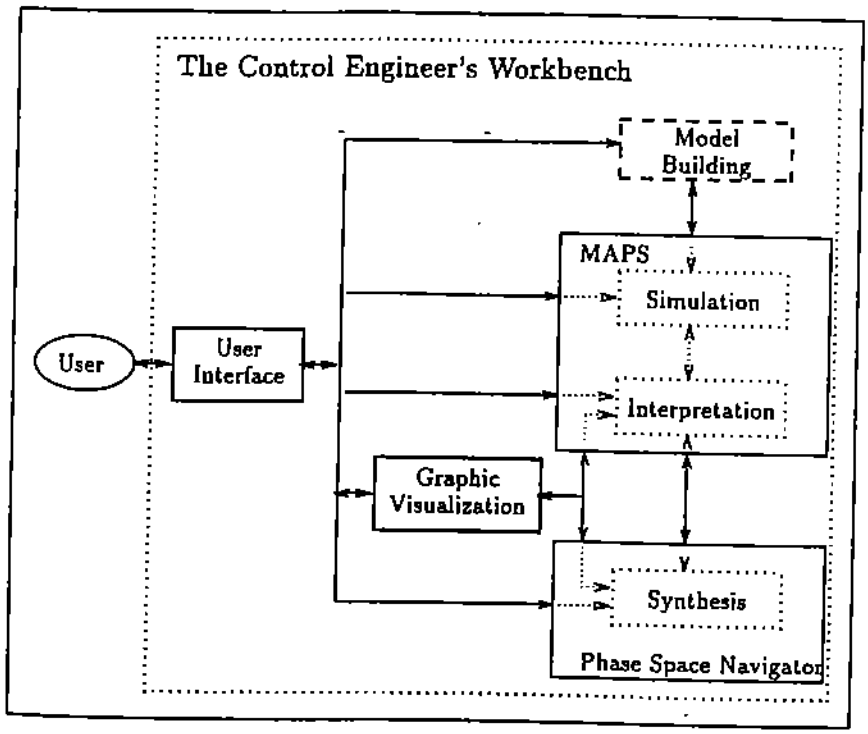

Figure 1: The control engineer's workbench

While the traditional numerical computing has successfully attacked many practical problems, we can greatly enhance its effectiveness and significantly expand the scope of what can be done with the traditional simulation by integrating the numerical computation with advanced artificial intelligence technology and symbolic computing methods. Programs equipped with AI reasoning techniques and deep domain knowledge have already helped engineers solve an open problem in hydrodynamics [7] and given new insights into behaviors of a heart model in cardiology [6]. In our own work, we have demonstrated the utility of the Workbench in an application of great practical interest: the Workbench helped design a high-quality controller for a magnetic levitation system—the German Transrapid system. The control system synthesized by the Workbench outperforms a previous linear design on the same system by a factor of 20 [10]. The future for intelligent simulation is very bright.

References

[1] H. Abelson and G.J. Sussman, "The dynamicist's workbench I: automatic preparation of numerical experiments," in *Symbolic Computation: Applications to Scientific Computing*, R. Grossman (ed.), SIAM, Philadelphia, PA, 1989.

[2] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E. Sacks, G.J. Sussman, J. Wisdom, and K. Yip, "Intelligence in scientific computing," *CACM*, 32(5), May 1989.

[3] J. Guckenheimer and S. Kim, "Kaos: dynamical system toolkit with interactive graphic interface," *Technical Report (Draft)*, Cornell University, 1990.

[4] E. Kant et al. (eds.) *Intelligent Scientific Computation.* Working notes of AAAI Fall Symposium Series, 1992.

[5] M.F. Russo and R.L. Peskin, "Automatically identifying the asymptotic behaviors of nonlinear singularly perturbed boundary value problems," *J. Automated Reasoning,* 8(3), June 1992.

[6] E. Sacks and L. Widman, "Nonlinear heart model predicts the range of heart rates for electrical alternans in pericardial effusion," Technical Report *CS-TR-365-92*, Princeton University, 1992.

[7] K.M. Yip, *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer.* MIT Press, 1991.

[8] F. Zhao, "Extracting and representing qualitative behaviors of complex systems in phase spaces," *Proc. of the 12th Int'l Joint Conf. on Artificial Intelligence*, Morgan Kaufmann, 1991. To appear in *Artificial Intelligence* journal.

[9] F. Zhao, "Phase space navigator: towards automating control synthesis in phase spaces for nonlinear control systems," *Proc. of the 3rd IFAC International Workshop on Artificial Intelligence in Real Time Control*, Pergamon Press, 1991.

[10] F. Zhao and R. Thornton, "Automatic design of a maglev controller in state space," *Proc. of the 31st IEEE Conf. on Decision and Control*, Tucson, Arizona, December 1992.

[11] F. Zhao, "Automatic analysis and synthesis of controllers for dynamical systems based on phase space knowledge." *Technical Report AI-TR-1385*, MIT Artificial Intelligence Lab, 1992.

[12] F. Zhao, "Computation and Reasoning in Automating Control Design." *Working Notes of AAAI 1992 Fall Symp. on Intelligent Scientific Computation,* Cambridge, Mass., October 1992.

[13] F. Zhao, "Computational dynamics: modeling and visualizing trajectory flows in phase space," *Annals of Mathematics and Artificial Intelligence*, to appear. Also in *Proc. of the 2nd International Symp. on Artificial Intelligence & Mathematics*, Florida, Jan. 1992.

Qualitative Flow Descriptors for Unstructured Triangular Grids

A.M. Froncioni and R.L. Peskin[1]
Department of Mechanical and Aerospace Engineering
Rutgers University
Piscataway, NJ 08855-139

Abstract

Flow visualization techniques are rapidly evolving to keep pace with improved flow solution methods. Critical point flow analysis has emerged as an important visualization and data compression method. In addition, flow topology can be described qualitatively using such methods. We propose a merger of this type of analysis with unstructured triangular-grid solution techniques. The role of symbolic computation in developing such a system is discussed.

Flow Solutions

Finite volume schemes [1] have existed for some time. The basic aim of the method is to make use of conservation properties of certain flow quantities to generate stable numerical schemes. Different variants of this method have been developed for many different types of grids. The unstructured triangular grid is gaining favor among many researchers [2]. Conservation laws exist for both primitive and streamfunction-vorticity formulations of the incompressible Navier-Stokes equations [3]. We have chosen the streamfunction-vorticity method on a triangular unstructured grid as a transient flow solver. The governing equations are written for the flow vorticity and streamfunction:

$$\begin{aligned} \frac{\partial \omega}{\partial t} &= -\nabla \cdot (\mathbf{u}\,\omega) + \frac{1}{R_e}\nabla^2\omega \\ \nabla^2\psi &= \omega \end{aligned}$$

The flow variables are interpolated on the triangular grid using a piecewise linear approximation within each triangle. The flow equations are then discretized and solved using a time-explicit scheme. The velocity field can be recovered from the streamfunction by using the streamfunction definition, $u = \frac{\partial \psi}{\partial y}$ and $v = -\frac{\partial \psi}{\partial x}$.

[1]This research was sponsored by National Science Foundation grants ECS-9110424 and IRI-9116588.

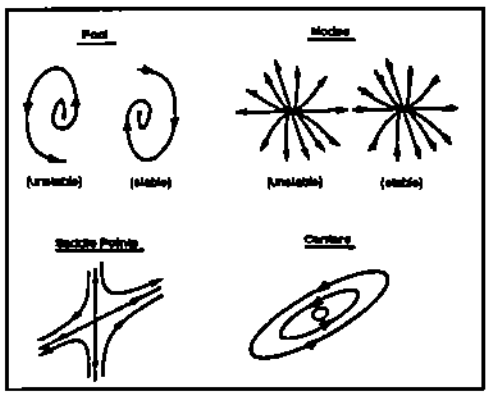

Figure 1: Some common critical points found in fluid flows.

Flow Post-Processing

Critical points are points in the flow where the velocity vector has zero magnitude. These points provide a framework in which to qualitatively describe the flow [4, 5]. Critical point analysis relies on the expansion of the velocity field in terms of its Taylor series in the spatial coordinates:

$$u_i = A_i + A_{ij}x_j + A_{ijk}x_jx_k + A_{ijkl}x_jx_kx_l + \cdots$$

At a stagnation point, the constant term disappears, leaving $A_{ij}x_j$ as the next highest term. Examination of the trace $(-p)$ and determinant (q) of $\mathbf{A}$ leads to a phase representation of the points in question. Examples of some common types of critical points is given in Figure 1.

Phase-space classification of critical points allows a description of the fundamental nature of the flow in the vicinity of a given critical point. Streamlines connect the critical points together to form a set of directed graphs within the flow. The critical points are computationally extracted through the use of two grids, a computational grid and a post-processing grid. Each time-step of the simulation is generated on the computational grid. The critical point analysis is performed by superimposing the post-processing grid onto the computational grid. The interpolation currently consists of either a second-order (6 nodes) or third-order (10 nodes) triangle pattern.

The Role of Symbolic Math Packages

The generation of higher-order Lagrange interpolations for triangular finite-element-type patches has been given in Hughes [6]. It is possible to write the generation algorithms for arbitrary orders, although the algebraic terms soon become unwieldy and numerical evaluation becomes extremely time-consuming. However high-order derivations are performed using both MAPLE and AXIOM. C code is automatically generated for the analysis on the grids mentioned above. The code is inserted into a *for*-loop which runs through all the higher-order triangles in the grid and determines the location and type of critical point in each triangle.

The symbolic math packages allow us to investigate the properties of the Jacobian matrix in a very short time. It is also possible to make use of optimizing code generators, such as those found in MAPLE, to efficiently evaluate the flow descriptor information.

Conclusions

A two-dimensional unstructured time-varying code has been adapted for use with critical point analysis techniques. This allows a natural, qualitative description of the flow, amenable to graph rewriting algebra. Symbolic math packages proved to be invaluable aids for prototyping this code.

References

[1] Peyret R. and Taylor, T.D., "Computational methods for fluid flow," *Springer-Verlag Series in Computational Physics*, Springer-Verlag, New York, N.Y., 1983.

[2] Barth T.J., "Numerical aspects of computing viscous high reynolds number flows on unstructured meshes", *AIAA Proceedings of the 29th Aerospace Sciences Meeting*, AIAA No. 91-0721, 1991.

[3] Roache P.J., "Computational fluid dynamics," *Hermosa Publishing, Inc.*, Albuquerque, N.M., 1982.

[4] Perry A.E. and Chong M.S., "A description of eddying motions and flow patterns using critical point concepts," *Ann. Rev. Fluid Mech.*, **19**, pp. 125-155, 1987.

[5] Helman J. and Hesselink L., "Automated analysis of fluid flow topology," *SPIE Proc.*, **1083**, pp. 23-32, 1989.

[6] Hughes, J.R., "The finite element method: linear static and dynamic finite element analysis," *Prentice Hall, Inc.*, Englewood Cliffs, N.J., 1987.

Interactive User Filters for the Visualization of Large Data Sets

Toufic I. Boubez and Richard L. Peskin[1]
CAIP Parallel Computing Laboratory
Rutgers University
Piscataway, New Jersey 08855-139

Introduction

During the exploration and visualization of large data sets, it is often required to compute local quantities that were not part of the original large-scale computation. This is an integral part of the process of computational steering. It is customary in these cases to recompute the problem, rewriting the program to include new code for the desired expression. This obviously creates a significant bottleneck in the analysis loop. A more severe problem occurs, however, if the code for the original computation is not available. While the handling of large data sets is in itself an issue that has been discussed in several previous publications [1, 2, 3], we present here a a recent addition to our scientific database toolbox that allows scientists to enter an expression for the desired quantity in symbolic format (using MAPLE syntax) and to automatically generate a program to extract the required data from the database, compute the desired quantity, and append the results to the existing database.

The Data Warehouse

Both the SCENE environment and the object-oriented data warehouse have been described extensively in [1, 2, 4]. The SCENE concept establishes a set of requirements for a system designed to handle large scientific datasets:

1. **User configurable:** The researcher should be able to configure the system to the problem being investigated in a clear and simple fashion.
2. **Automatic programming:** The SCENE user should not be required to perform any programming. Required objects and methods should be generated automatically.
3. **Computational steering:** The user should be able to dynamically modify the problem definition without loss of context.

[1]This research was sponsored by National Science Foundation grants ECS–9110424 and IRI 9116558.

4. **Parallel and distributed processing:** A typical computational fluid dynamics (CFD) computation will involving a $50 \times 50 \times 50$ three-dimensional mesh will usually produce over 11 Mbytes of data for each time step. For a truly interactive system, this data has to be accessed and searched, and computations performed on it in a reasonable amount of time. This requires the use of parallel computers with large amounts of memory.

Within the SCENE environment, the data warehouse is an object-oriented database system capable of handling large scientific datasets while providing all the required graphical and visualization tools. Through the use of an Object Editor [5], the user is allowed to completely define the data being visualized. The data is then represented on a logical mesh, with each point on the mesh being a computational object that knows about its location in real space, and the values of all the defined fields at its location. For example, assuming we have defined a database containing one scalar pressure field and one vector velocity field, each computational object will contain seven information fields: three coordinates, one pressure value and three velocity values. All the fields are treated equally, with vector fields being indexed onto three scalar fields each. This allows a large degree of flexibility in selecting components for visualization or for use in user entered expressions.

The User Filter

By taking advantage of the logical mesh organization of the data warehouse, we can now automatically generate interpolation polynomials over the mesh triangles, enabling us to compute user-entered mathematical expressions. A user defined expression is entered in symbolic MAPLE format. For example, and assuming the existence of a vector velocity field labeled `v`, the expression for the vorticity magnitude, $|\nabla \times \mathbf{v}|$ can be entered as `norm(curl(v,[x,y,z]),2)`. The entered expression is sent to a running MAPLE process. The expression is analyzed and, using predefined interpolations of the appropriate order over an implied mesh, MAPLE then generates the equivalent C code to compute the desired quantity. This C code is then integrated within a larger program, generated using various templates, to form the source code for the user defined filter process. This program, when compiled and run, opens communication channels to the warehouse process, requests the data for the fields needed in the computation, performs the computation, and sends the resulting field(s) to the warehouse process to be registered according to the correct database protocol. This ensures that these newly created fields conform to the database specifications and can thus be visualized and manipulated by the user with any of the standard tools.

Conclusions and Future Work

The user defined filter facility has allowed researchers to analyze several CFD data sets and to compute additional quantities, such as vorticity magnitude and laplacians that were not part of the original data sets, without any additional programming or recomputation of the whole problem.

Currently, inter-process communication is achieved through the use of socket calls. This situation will be improved in the future by implementing a shared-memory environment more suitable to the kind of heterogeneous distributed processing tasks required by such a method.

References

[1] R.L. Peskin, S.S. Walther, and T.I. Boubez, "Computational steering in a distributed computer based user interface system," *Artificial Intelligence, Expert Systems and Symbolic Computing*, E.N. Houstis and J.R. Rice (eds.), Elsevier Science Publishers, North-Holland,, IMACS 1992.

[2] S.S. Walther and R.L. Peskin, "Strategies for scientific prototyping," *OOPSLA '89*, New Orleans, LA, ACM, 1989.

[3] R.L. Peskin, S.S. Walther, A.M. Froncioni, and T.I. Boubez, "Interactive quantitative visualization," *IBM J. of Research and Development*, **35**:1/2, pp. 205-226, Jan/March 1991.

[4] T.I. Boubez, A.M. Froncioni, and R.L. Peskin, "A prototyping environment for differential equations," *Second International Conf. on Expert Systems for Numerical Computing*, West Lafayette, IN, North-Holland, 1990.

[5] S.S. Walther, T.I. Boubez, and K. Ghazi, "Object-oriented data management for distributed scientific simulations," *Soc. for Computer Simulation*, Proc. Western Multiconference, Jan. 1993.

Invariance and Calibration in Image Processing Problems

(Extended Abstract)

Svetlana A. Filatova and Peter V. Golubtsov
Department of Physics
Moscow State University
119899 Moscow Russia

Introduction

A lot of modern Measurement Systems (MSs) produce a large amount of data (results of measurements). First of all we keep in mind MSs used for formation of images (visual, infrared, etc.) in tomography, 3-d Microscopy, in SAR (synthetic aperture RADAR). Usually results obtained on such systems are not useful for direct interpretation and demand for additional processing. In such signal processing problems, one often has to deal with a large (potentially infinite) amount of data (field of vision, for example), a fact that causes appreciable difficulties in obtaining an optimum processing algorithm. As often as not, the measurement in situations like that is invariant with respect to a certain group of transformations. Proper allowance for this invariance can appreciably cut down the computational costs involved both for constructing of an optimum processing algorithm and for its subsequent functioning [1], [2].

The problem becomes crucially more complicated when the model of signal formation (e.g., point spread function in image formation systems) is unknown or the information about it is not precise enough. In such cases it is useful to involve calibration data (results of measurements of known signals) for constructing of an optimum processing algorithm. It turns out, that the allowance for invariance significantly simplifies the solution of calibration problem as well.

Invariance of measurement system is reflected on the structure of processing algorithm. In fact it will have a high level of uniformity. This is particularly attractive for implementation on parallel architecture, such as systolic arrays, etc.

Processing for Invariant Measurement Systems

Invariant measurement system related to measuring of signal f:

$$\xi = Af + \mu$$

is described by $\mathcal{G}$-invariant linear transformation A and $\mathcal{G}$-symmetric correlation function of noise μ, where f and ξ are functions on some spaces with an action of a given group $\mathcal{G}$ defined on them. Let the processing algorithm performs some transformation R, i.e., transforms ξ to $R\xi$ – functions on $\mathcal{G}$-spaces $\mathcal{H}$ and $\mathcal{U}$ respectively. Then the problem of construction of an optimal algorithm is stated as a problem of construction of map R from a class of maps $\mathcal{R}$ with a given influence function support Δ (e.g., a given shape of scanning mask). The support $\Delta \subseteq \mathcal{H} \times \mathcal{U}$ is the relation between spaces $\mathcal{H}$ and $\mathcal{U}$; $(h, v) \in \Delta$ if point $h \in \mathcal{H}$ can influence on point $h \in \mathcal{U}$ via R. An optimum R provides maximal accuracy to the whole measurement computer system (MCS), i.e., minimal mean error of estimation of vector Uf (function on $\mathcal{U}$) in all points v:

$$R : \{\mathrm{E}\|(R(Af + \mu) - Uf)_v\|^2 \mid R \in \mathcal{R}\} \sim \min \forall v \in \mathcal{U}$$

We want to stress that due to the linking of measurement and processing the new measurement (to be more precise measurement computer) system emerges which maximally fits our requirements. Therefore when developing a measurement system it makes sense to develop appropriate algorithm for it implemented, for example, as a dedicated processor and to produce MCS as an integrated unit.

Problem Solution for Invariant MS

The problem stated above appears to be highly complicated due to large dimensions of signal spaces. Significant simplification of the problem solution can be achieved due to allowance of the high level of MS's invariance. It turns out, that if measurement system is invariant an optimum transformation R will also be invariant with respect to the same group. This aspect leads to considerable restriction of a class, in which an optimum transformation R is produced, thus significantly simplifying the development of such map and its subsequent implementation as a computational algorithm or dedicated processor.

Note that the wider the group of transformations with respect to which a measurement system is invariant, the narrower the class of invariant maps and the more the synthesis problem is simplified. For instance, in the case of processing of images on hexagonal lattice allowance for invariance with respect to reflections and rotations (when the point spread function of a system depends solely on distance) in additional to rotations cuts down the requirements (in developing a map) for memory by a factor of more than 100, and for time, by a factor of more than 1500 [2].

Uncertainty of MS and Calibration Problem

Up to this we suggested that the MS (i.e., operator A) is known precisely. But the problem rises to a qualitatively new level if an information about measurement system is not precise enough. In such cases for revision of information about the system it is conventional to organize the sequence of measurements of known test signals φ_i: $\xi_i = A\varphi_i + \mu_i$, and to use its results for construction of an approximate measurement model. We will formulate the calibration problem as a problem of an optimum usage of calibration data directly for construction of map R (an optimum processing algorithm).

Let $\Psi = \{(x_i, \varphi_i)\}$ is calibration data series. Calibration problem consists in constructing calibration map β, converting measured calibration data Ψ to transformation $R = \beta(\Psi)$ (and therefore data processing algorithm) in an optimum way. To be precise, we can introduce function $H(\beta)$, describing average estimation error for a given map β. Then an optimum calibration map β, delivering minimal value to function $H(\beta)$ can be constructed. Notice that invariance allowance in calibration problem also leads to its significant simplification.

Such approach doesn't demand for exotic test signals (such as a point source) and also automatically accounts a noise in calibration measurements and what is more, using the test objects of general structure results in more qualitative MCS. It is proved that under increasing an amount of calibration data (e.g., number of calibration measurements or sizes of fields of vision) the synthesized processing algorithm converges to an ideal one, which is associated with the precisely known model of measurement system.

Implementation Considerations

As to algorithms involved in processing in all these cases the existing signal processing technique (e.g., fast convolution, FFT, number-theoretic transforms) can be used. It is true even in such cases when it doesn't follows directly from the problem statement, say in case of different lattices' organization.

In the simplest case, i.e., when the lattices for source images and results of measurements are identical in structure and the group of transformations is simple enough, processing algorithm may be represented as a convolution of an image with spatial mask. It is known, that convolution allows effective parallel implementations. It turns out that the same technique can be applied in the case of different lattices and for vast class of groups of transformations. The higher level of measurement invariance the more uniform will be the algorithm and the more effective implementations on parallel architecture it will admit.

Conclusions

The use of a space-invariant map permits processing to be done on a computer or a dedicated processor in real time, thus saving a good deal of computational resources. Note that a procession algorithm and the degree of complexity in producing it do not depend on the amount of data to be processed (say, the size of the image being processed), because the problem may be oriented towards a potentially infinite collection of measurements. Processing time is proportional to the amount of data and can be cut down through the use of special numerical algorithms or dedicated processors. And what is more, in many cases, signal can be processed directly after its inflow, concurrently to the measurement process.

The "larger" group, under which an optimum MCS synthesis problem is invariant, the higher quality of the whole MCS can be achieved under fixed computational resources (memory and time usage). As a result certain rearranging of measuring component of MCS can lead to significant "expanding" of related group. In particular, these considerations can determine the scheme of scanning for optical or RADAR image formation MCS, in tomography, etc., providing the highest level of measurement system invariance. Also notice that different arrangement of lattices for results of measurements and results of processing doesn't effect on the synthesis problem statement and it's solving.

References

[1] Filatova S.A. and Golubtsov P.V., "Invariant measurement computer systems pattern recognition and image analysis," Vol.1, No. 2, pp. 224-235, 1991.

[2] Filatova S.A. and Golubtsov P.V., "Invariance and synthesis of optimum image formation measurement computer systems "Artificial Intelligence", Expert Systems and Symbolic Computing," E.N. Houstis and J.R. Rice (eds.), Elsevier Science Publishers B.V. (North-Holland), pp. 243-252, 1992.

Knowledge Based Front-End to NAG Library — KASTLE

Dr. Venkat V.S.S. Sastry
Applied & Computational Mathematics Group
Royal Military College of Science
Shrivenham, Wilts., SN6 8LA, U.K.

Abstract

This paper describes a knowledge based front-end to a subset of NAG FORTRAN library (called the foundation Library) hereafter referred to as KASTLE (Knowledge Assisted Selection Tool for Library Environments). The development of KASTLE has gone through several changes to accommodate technological advances. In this paper, we describe how the knowledge of individual routines is represented and give a brief overview of the selection mechanism which enables the user to select the desired routine.

Introduction

Customarily library software is accessed with the help of a hard copy of its documentation. With the increase in the complexity of library software consulting the software through a hardcopy is not only impractical, but also error prone. For example, Mark 15 of FORTRAN library has 36 Chapters and has 950 routines. A systematic way of accessing the desired routines is not only desirable, but also expected by the modern user with the advent of the window-based systems.

Overview of KASTLE

KASTLE consists of three main components – knowledge base (Selector), hypertext help and example program environment. The user interaction is through WIMP based interface. The user is offered a typical window (see Fig. 1) to initiate the interrogation process. Help on the usage of KASTLE system itself is available during the process.

Selection of a desired routine is achieved with the help of Selector, which when initiated offers a selection of keywords in a separate window (see Fig. 1) along with the available chapters. The definition of these keywords in their context is offered optionally. The user can start the selection either via an individual chapter (*chapter level selection*) or via a

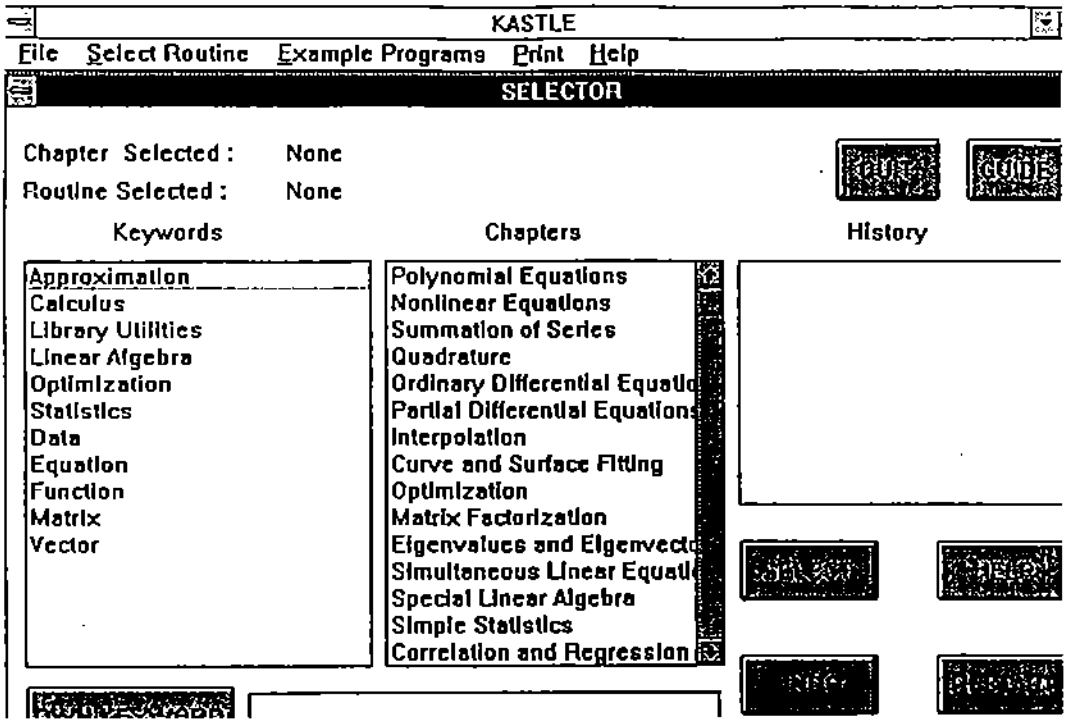

Figure 1: Initial screen layout for the selector

sequence of keywords (*top level selection*). At every stage only those chapters that are deemed relevant to the selected keywords are retained for further selection. Upon successful completion of the interrogation, full details of the selected routine are displayed in the help window.

At this stage, the user will have a choice to execute the pre-packaged example program or alternatively edit the relevant source file and rerun the program.

Knowledge Representation in KASTLE

Knowledge in KASTLE consists of *two levels* – Library level and Chapter level. Library level knowledge consists of keywords dealing with *major* activities of the NAG Chapters. Several chapters may share a keyword. Chapter level knowledge consists of keywords which

are predominantly exclusive to that chapter.

For example, Library level knowledge for Quadrature chapter is represented as

```
chapter_keys(D01,[Abscissa,Adaptive Rule,Calculus,Data Values,Dimension
Discrete Integrand,Finite Difference,Finite Region,Function,
Infinite Region,Integral,Integrand,Inverse Weight,Known Function,
Many Dimensions,Nonadaptive Rule,Numerical Integral,One Dimension,
Oscillatory Integrand,Quadrature,Return Weights,Semi-Infinite Region,
Sine Weight,Singular Integrand,Singular Weight,Smooth Integrand,
Weight Function]).
```

while Chapter level knowledge for the same chapter is represented as

```
chapter_routines(D01,[D01AJF,D01AKF,D01ALF,D01AMF,D01ANF,D01APF,D01AQF,
D01ASF,D01BBF,D01FCF,D01GAF,D01GBF,E01BHF,E02AJF,E02BDF]).
routine_keys(D01AJF,[Adaptive Rule,Finite Region,Known Function,
One Dimension,Smooth Integrand,Unit Weight,Weight Function]).

                                .
                                .
                                .

routine_keys(E01BHF,[Data Values,Discrete Integrand,Finite Region,
Fitted Cubic Hermite,Nonadaptive Rule,One Dimension]).

initialkwds(D01,([Return Weights],[One Dimension,Many Dimensions,
Finite Region,Semi-infinite Region,Infinite Region,Abscissa,
Weight Function, Discrete Integrand,Smooth Integrand,
Oscillatory Integrand,Singular Integrand],[Data Values,Known Function,
Adaptive Rule,Nonadaptive Rule])).
```

The Library level knowledge base also consists of additional facts such as *keyword dependencies, aliases* and *exclusives.*

In view of this two tier system, the notion of the *chapter* is extended to include knowledge of routines from other NAG-Chapters wherever appropriate. For example, E01BHF computes the integral of a fitted-cubic-hermite polynomial to discrete data; and this knowledge is also included in the knowledge base dealing with quadrature.

Selection Mechanism

The selection of an individual routine is achieved essentially in one of the following two ways – *top level selection* or *chapter level selection.*

- *Top Level Selection*

  When a keyword is selected from those initially displayed, first any keyword dependencies are added to the search list, and the relevant chapters containing the keyword are retained for further consideration and displayed. During this process, if there is only one chapter, then the Chapter Level Selection is initiated. If there are no further keywords left, then the differences (disjoint union) of list of initial keywords of the remaining chapters are presented.

- *Chapter Level Selection*

  When a specific chapter is selected the list of initial keywords for that chapter are offered together with a list of available routines. Upon selection of a keyword, the selection works in exactly the same fashion, *mutanis mutandis* to that of Top Level Selection

Acknowledgements

KASTLE is developed under NAG/RMCS Teaching Company Scheme. NAG is registered trademark of Numerical Algorithms Group Limited. The author wishes to acknowledge the kind help of Mr. Mark Pengelly for various discussions regarding selection algorithms and Mr. Ian Reid for making this possible.

Geometry Modeling Problem Solving Environments

Christoph M. Hoffmann[1]
Jörg Peters[2]
Computer Science Department
Purdue University

Introduction

A common definition of Problem Solving Environments (PSEs) is that these environments should assist scientists and engineers to solve problems by offering a wide variety of component systems and tools that *communicate with the user in his or her own terms* [1]. This suggests that a PSE ought to federate subsystems, whether they be written especially for that common purpose or are pre-existent. It suggests that the aggregated capability ought to be accessed through an intuitive user interface that removes the idiosyncracies of the individual components of the environment and presents all capabilities with a common view. We consider both objectives, federating component systems and exercising them through a good user interface, in the context of geometric modeling and its applications.

Federating and Interoperability

Problems considered by engineers and scientists are rarely confined to the precise and often narrowly delimited scope of a specific subsystem such as a linear algebra package, a differential equations solver, a symbolic computation system, a geometric modeling system, a graphics package, and so on. Even when a system is expanded to incorporate other submodules, say a graphics module to display the zero set of a polynomial, this package must be tailored to the originating system and thus is saddled with the legacies of the parent system. To give the human problem solver full access to the combined power of these tools requires federating very diverse systems. Indeed, each subsystem manipulates objects that are, if at all related semantically, certainly not conceptualized in the same way. This is not a deficiency because it enables different approaches to the same problem and does not force an unnatural representation on the user. However, the desirable diversity does require

[1]Supported in part by ONR Contract N00014-90-J-1599 NSF Grant CDA-92-23502, and by NSF Grant ECD-88-03017.
[2]Supported in part by NSF grant CCR-92-11322

solving communication problems between foreign subsystems and reinterpretation of data. For example, it is possible to describe a triangulation within a symbolic solver as the zero set of polynomials, but clearly this is not a very intuitive representation.

Federating diverse subsystems can be done following one of two different strategies:

1. The subsystems, knowing about each other's existence, send messages directly to each other, converting data and reinterpreting them as needed.

2. A global manager is the user's proxy and accesses each subsystem by translating a global operation or request into one or more component operations translating them into the local language of the subsystem and interpreting the local results.

The first strategy requires that each subsystem understand the topology of the federation and the formats native to the members of the federation. This stringent requirement can be loosened somewhat by wrapping federation members into a communication substrate or software bus written in a module interconnection language. The second strategy, in contrast, synthesizes the functionalities of each subsystem and assigns them to a global conceptual schema. No specific changes to the subsystems are required, but an additional software layer is needed to accomplish the integration.

Unlike differential equations, geometry does not have an accepted, canonical symbolic language, primarily because geometry representations currently in use are very diverse and usually low-level, because of efficiency considerations. For example, a three-dimensional shape is typically represented as a collection of vertices, edges and faces, along with their adjacencies. Each face is a subset of a surface, and is represented by the geometry of the surface plus a structure that defines the face by describing a loop of edges on the surface that delimit the face. In turn, an edge is represented by defining a space curve of which the edge is a segment, and delimiting the edge by two vertices. The entire data structure of vertices, edges and faces can describe a closed composite surface in 3-space, and therby, implicitly, the enclosed solid volume. The data structure is complex, specialized, and cumbersome to compute with, and conversions between this and other geometry representations is not normally possible [6,7]. Hence we cannot expect to devise an efficient, universal representation schema that is understood by every geometric modeling system.

The second strategy to federate subsystems through a common global view is therefore required when complex geometric objects are to be manipulated. We sketch some of the elements required to develop such a unifying global view.

Levels of Abstraction

While it appears futile to devise a universal shape representation for specific, three-dimensional shapes, it seems possible to devise a representation schema that is *generative*. That is, instead of specifying a particular shape we specify a sequence of operations that construct the shape. This approach to geometry can be further abstracted by adding the ability to parameterize the generation rules, e.g. by adding geometric constraints such as incidence, perpendicularity, concentricity, tangency, and so on, as well as adding dimensional constraints of distance and angle.

Since humans comprehend two-dimensional geometric constructs better than three-dimensional ones, it is advantageous to conceptualize the shape generation at the inception level in terms of sweeping two-dimensional cross sections along trajectories. In the simplest case, the trajectories are straight lines or circles, and generate extruded or revolved shapes. Clearly, sweeps along more complex trajectories are also possible. Primitive shapes so generated can then be combined, possibly using operations such as set union or difference, or using more complex operations. Unfortunately, there are in addition a number of intuitive shape definition and modification operations that are quintessentially visual and thus not easily formalized using a symbolic representation.

The high level of abstraction effecting the federation must be supported by a lower level on which the details of a shape are evaluated as required by the parameter values. At this lower level of abstraction reside the traditional geometric modeling systems operating with the detail representations such as the one sketched before, and manipulating strictly specific instances. The transition from higher to lower level is effected by a geometry compiler that translates, with the help of a constraint solver, the generative rules into specific operation sequences executed by the geometric modeling subsystem. In [3] we have discussed such an architecture.

System Components

Solving geometric constraints is an activity that naturally involves solving systems of nonlinear algebraic equations. This suggests that a symbolic algebraic computation system ought to be an integral part of a geometry PSE. The other end of the tool spectrum is an explicit representation of geometry in terms of meshes of points; e.g., a surface or volume triangulation. While the algebraic equations embody the abstract formulation of problems and may be complemented by a geometry theorem proving system, the meshes provide direct examples of the theory ready to be animated by a graphics module. Intelligent display routines

are needed to query and analyze the computed geometry and to validate it. Easy conversion between the representations, in particular of their properties relevant to constraint solving, requires that the highest level of abstraction represents objects in such a way that a translation to geometry is as easy as an encoding into variables and equations.

PSE Applications

Rather than giving a laundry list of subsystems and components, we sketch some of the functionalities a geometry PSE should support by considering some very simple example scenarios.

The derivative matching problem for parametric surfaces

Complex surfaces are built from pieces. To join these pieces with the required smoothness, it is necessary to determine an appropriate parametrization for each piece. The corresponding constraint equations are most elegantly formulated in the language of differential geometry, [2,5]. For example, given two surfaces pieces p and q, and a map ϕ from R^2 to R^2 that maps the domain of p to the domain of q, the differential formulation of continuity requires matching the derivatives of p and $q \circ \phi$ up some order across the joint boundary.

To make the differential algebraic formulation concrete and useful one needs to translate it into a particular representation for the surface pieces. This is a tiresome and error-prone process. Then one must ascertain whether the choice of representation allows a solution of the constraints and if not choose a new representation and restart. For example, we might represent p, q and ϕ as polynomials of a fixed degree, constraining them to match certain boundary curves. Thereafter, one has to determine whether the polynomials have the required degrees of freedom to join smoothly without violating the interpolation constraint. Since the coefficients of q and ϕ are in general not prescribed by the application, the problem is nonlinear and one has to make several simplifying assumptions in order to reduce the solution space and arrive at a tractable problem [4].

For C^1 surfaces manual translation and subsequent verification is still manageable. However, for curvature continuous surfaces and surfaces of higher smoothness, there is to date no generally useful algorithm, largely because the number of constraints in a particular representation makes it difficult to reason about the nonlinear solution space. Automating the translation and verification process would allow an exploration of this space.

The geometry coding problem

When a particular representation for the surfaces has been chosen and the constraint equations have been determined and shown to be solvable, the programmer now has has the unglamorous task of translating patch indices into corresponding entries of the data structure. For example, the coefficients of the polynomial pieces that occur in a constraint equation are typically assembled from a half-edge representation, sent to a mathematical package to solve a minimization problem, and the output is then returned and distributed into the data structure. The mapping of the coefficients in and out of the data structures is a common source of errors. The mapping can and should be automated.

A second issue is the sequencing of the computations. Generically, the original nonlinear problem is transformed into a sequence of linear problems to make it tractable. That is, some coefficients of the polynomials that define the surface pieces have to be determined before others. Instead of depending on the cleverness and patience of the algorithm designer, topological sorting may be used to serialize the constraint elaboration.

The geometry analysis problem

It is often necessary to study algebraic properties of a surface in addition to those that explicitly enter into their derivation. For example, generating a surface that is guaranteed to be convex and matches given data is, in general, too hard a problem to solve or even to verify algebraically. However, one can use iterative techniques to improve the surface quality and heuristically check the surface quality by sampling the surface and displaying a curvature map. This requires interfacing a display program capable of understanding high level information like a B-spline representation. Other quality checks may be done better with the help of a symbolic solver. For example, one might so determine how well a set of equations defining an offset surface are satisfied by a single polynomial equation.

The geometry editing problem

Mesh generation in three dimensions is a notoriously hard problem and different applications require different properties of the mesh to be optimized beyond a certain threshhold. For example, the output of a solid modeler may contain faces that have to be subdivided for use in a surface modeler. Since the objective of subdividing faces into smaller pieces does not completely define the algorithm, one would like to experiment with alternative strategies. This motivates access to intuitive geometry operations, say Euler operators, that allow a local editing of the mesh to gain a better understanding of its properties.

References

[1] E. Gallopoulos, e. Houstis, and J. Rice. Future research directions in problem-solving environments for computational science. Technical Report 1259, University of Illinois at Urbana-Champaign, Center for Supercomputing Research and Development, 1992.

[2] J.A. Gregory. Smooth parametric surfaces and n-sided patches. In W. Dahmen, M. Gasca, and C. Micchelli, editors, *Computation of Curves and Surfaces*, pages 457–498. Kluwer Academic Publishers, Dordrecht, 1990.

[3] C.M. Hoffmann and R. Juan. Erep, a editable, high-level representation for geometric design and analysis. Technical Report cER-92-24, Comp. Sci., Purdue Univ., 1992.

[4] J. Peters. Smooth interpolation of a mesh of curves. *Constructive Approximation*, 7:221–247, 1991.

[5] J. Peters. Joining smooth patches around a vertex to form a c^k surface. *Computer Aided Geometric Design*, 9:387–411, 1992.

[6] V. Shapiro and D. Vossler. Construction and optimization of CSG representations. *Computer-Aided Design*, 23:4–20, 1991.

[7] V. Shapiro and D. Vossler. Efficient representations of two-dimensional solids. *Transactions of ASME, Journal of Mechanical Deisgn*, 113:292–305, 1991.

Towards an Intelligent Problem-Solving Environment for Signal Processing

Chandra Shekhar, Sabine Moisan and Monique Thonnat

Keywords: Signal Processing (**SP**), problem solving, Expert Systems, intelligent environment

Introduction

Signal Processing (SP) is a rapidly growing area. A vast number of methods are now available for various problems in SP, and new methods are constantly emerging. This rapid evolution makes it difficult for the non-specialist in SP, who needs to apply its methods to resolve problems in his or her field of interest (Fig. 1(a)), to select and use the best methods.

In response to this problem, specialists in SP are developing expert systems which help the user in selecting appropriate algorithms, setting parameters, and offer the possibility of automated control of execution and failure handling (Fig. 1(b)). A detailed review of the state of the art in developing such systems is presented in the paper.

While such expert systems are certain to be of great help to non-specialists in SP, they are usually based on general purpose artificial intelligence (AI) tools that do not incorporate any *deep knowledge* of the SP domain. Considerable experience with AI tools is required in adapting them to the problem. It is difficult to use the experience gained in building an expert system for one application, for other applications in SP of a similar nature. Further, the resulting expert systems are too dependent on a particular collection of SP algorithms or a specific SP environment.

The goal of this paper is to propose a new approach to problem-solving in the domain of SP, based on the development of an AI environment *dedicated* to solving SP problems. This environment should make it easier for the specialist in SP to build expert systems, by providing mechanisms for knowledge representation and reasoning suitable for their domain. This approach is contrasted with traditional and current approaches in Fig. 1.

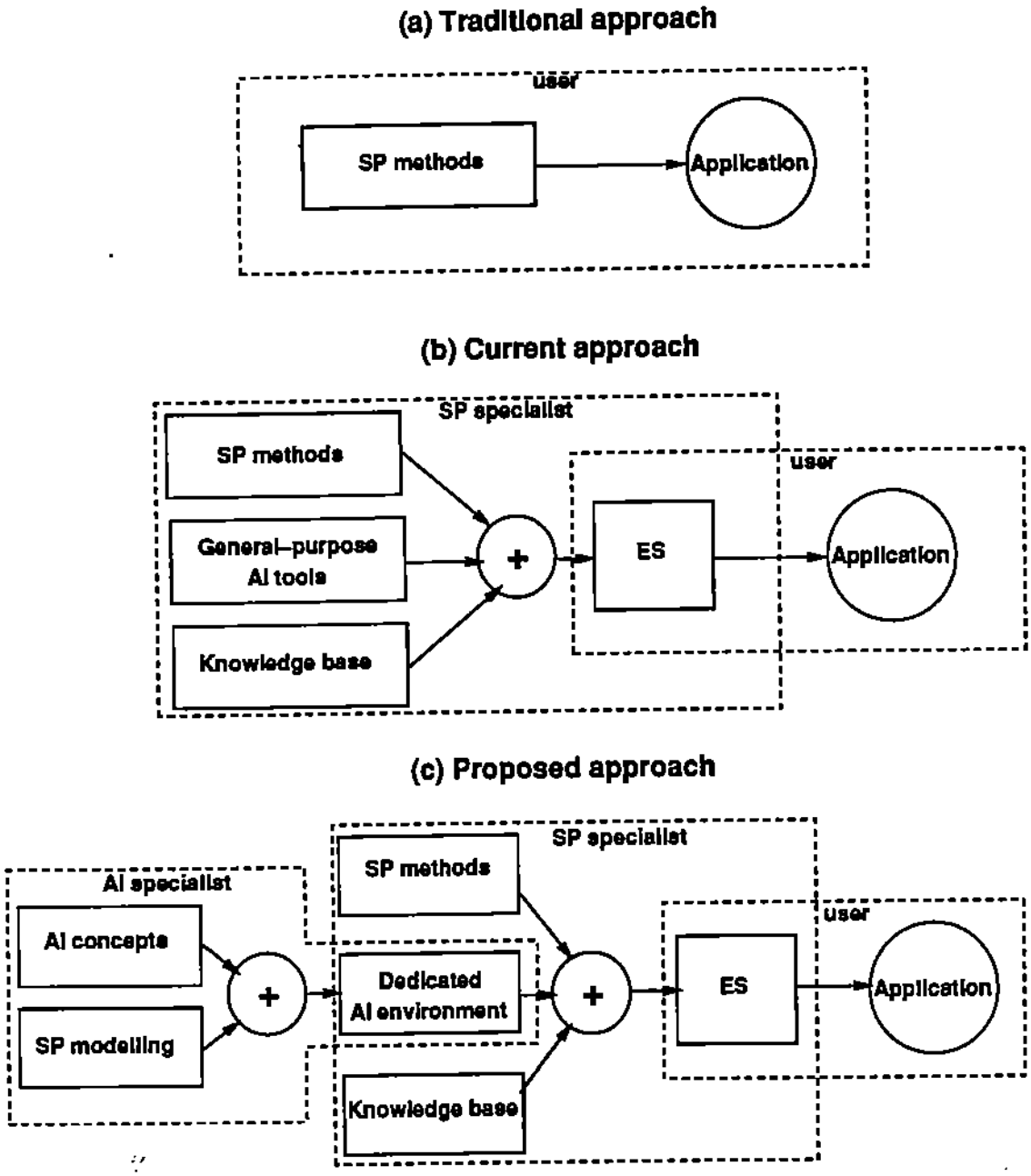

Figure 1: Approaches to using SP methods in an application

Here we try to answer the following question: What are the types of knowledge and reasoning used in solving SP problems?

Firstly, we examine how SP problems are represented. There are two fundamental concepts in SP, that of a *signal*, and that of a *system*. A signal is a time-flow of data. Thus the *temporal* aspect is fundamental to SP. A system takes as input a signal, called the *input signal*, and transforms it into another, *the output signal*. Many problems in SP, such as filtering and identification, can be interpreted in terms of a numeric input signal S_i, a system H, and a numeric output signal S_0, among which some are fully or partially unknown, and have to be determined based on the others. In some problems, there may be a need for *real-time* processing.

Nest, we look at the approach used by the SP specialist in solving a particular problem.

The specialist employs two main forms of reasoning, planning and supervision. Planning is done before the execution of the problem, and supervision is done during the execution.

The planning stage first involves choosing an *abstract configuration* of signals and systems for solving the problem. Each system or module at this point is chosen for performing a certain SP functionality, and is not related to any physical method. Later in the planning stage, each of these modules may be linked to a definite *method*, which may have one or more physical *implementations*. Thus the knowledge of the specialist is present at several levels of abstraction.

The main supervision tasks are *parameter setting, performance analysis* and *failure handling*. A typical SP application involves a number of interconnected systems, each of which performs certain basic SP operations. Each of these operations may involve a number of *parameters*. In order to obtain the desired results, the specialist selects suitable values for the parameters, as well as strategies for modifying them if the need arises. The specialist decides the criteria for evaluating the performance of each module. These criteria are often based on numerical measures of the module's output. In the event of unsatisfactory performance (failure), he decides what steps should be taken.

The Proposed Environment

Based on the preceding discussion, we propose a software architecture for intelligent problem solving environment for SP. This builds on our experience with OCAPI [1], an AI environment originally developed for the automatic choice and supervision of image-processing programmes. SP modules to be expressed at two levels, as *goals* and as *operators*. A goal is the abstract form of an SP functionality. This functionality may be realized in a more concrete form by operators. An operator may be either a simple one, corresponding to an executable

program, or may be a complex one, consisting up of several steps to be executed in sequence or in parallel. These steps themselves have goals corresponding to them, and are specific instances of these goals, called *requests*. A request is made up of a goal, and instance-specific information about its data and parameters. The initial problem to be solved is also stated in the form of a request.

Data and functions; The environment recognizes a basic data type, called "signal", which is a time-sequence of variables. Sub-types of signal, such as integer sequences and image sequences can be defined. The environment provides for the definition of a class of functions, called "system", which operate on a data type "signal". These functions may be defined by any of the commonly used forms, such as dynamical equations, impulse response, or frequency response.

Planning: The basic planning mechanism provided is *hierarchical script-based planning*. In order to solve a problem, a skeletal plan is first created. This is refined during the execution process, based on the results of the current and previous stages of execution. A goal in the plan may be realized by more than one operator, in which case *choice rules* can be defined to select the most suitable one. This operator may have several parameters, which are set using *initialization rules*.

Supervision: The reasoning during supervision is accomplished using rules for *evaluation* and *adjustment*. The results of a goal are verified using evaluation rules, and if the results are unsatisfactory, either another operator is selected, or the same operator is re-executed with its parameters modified by adjustment rules.

Reference

[1] V. Clément and M. Thonnat, "Supervision of perception tasks for autonomous systems: the OCAPI approach," in *AIS'92 3rd Annual Conference on Artificial Intelligence, Simulation and Planning in High Autonomy Systems*, (Perth, Western Australia), pp. 203–212, July 1992.