Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

AN APPROACH TO INTEGRATING AND CREATING FLEXIBLE SOFTWARE ENVIRONMENTS SUPPORTING THE DESIGN OF COMPLEX SYSTEMS

Kirstie L. Bellman

Computer Science and Technology SubDivision
The Aerospace Corporation
Los Angeles, California, 90009-2957

ABSTRACT

Engineers and scientists are attempting to represent, design, analyze, and reason about increasingly complex systems. Because of the complexity of these systems, no single analysis, model, approach, or viewpoint is sufficient. Such complex systems require not only the availability of a variety of analysis tools, knowledge bases, databases, and programs of all sorts, but also a framework within which these different programs, types of information, and viewpoints can be brought together. Software developers have responded to these needs by introducing the concept of a software "environment." In an environment, the user has access not only to a large number of different "tools" (e.g. analyses, editors, other programs), models, and databases, but often a number of "utilities" and features in the environment that make it easier to go from one tool or model to another. Often these environments have a diversity of knowledge representations (procedural code, equations, text, rules) and languages. Many environments are extendable in at least a limited manner to the languages and information styles already available in the system. However, new languages and representations are being developed continuously for very good reasons: as with mathematical formalisms, a good language can make certain problems easy to do.

Many researchers have been developing new ways of creating increasingly open environments (See Purtilo et al., as one example). In our research on *VEHICLES*, a conceptual design environment for space systems, we have been developing an approach to flexibility and integration based on the collection and then processing of explicit qualitative descriptions of all the software resources in the environment (Bellman and Gillam, 1990; Landauer, 1990). The detailed descriptions (or metaknowledge) of the resources are used by the system to help partially automate the combination, selection, and adaptation of tools and models to the particular requirements of the user and the type of problem being solved.

1 INTRODUCTION

The present challenge in much of science and engineering is to learn how to represent (model), design, develop, analyze, manage, and operate complex systems. Complex systems have some of the following characteristics: a large number of parts; different kinds of parts; different behavioral levels (different levels at which the system can be meaningfully analyzed or controlled); and emergent qualities (the behavior of the system globally is not fully derived from knowing the behavior of individual components). Examples abound: managing a product over its lifecycle; developing a new space system; reengineering a car; improving the quality in a service organization; designing a new building; and conducting a scientific study.

Because computer models and software support are essential to the ability to analyze and utilize complex models, it is no surprise that computer science and related fields have become the focal point for attempts to understand and handle complex systems. One of the first things we learned about supporting the design and analysis of complex systems is that no single viewpoint, approach, model, or analysis (however broad or good) will be sufficient to design, develop, understand, or manage a complex system. A design environment must provide a variety of models, analyses, software tools, and types of information. This imposes on such software environments the dual requirements of providing the *flexibility* necessary to handle a diversity of tools and information types and the *coordination* necessary to integrate, monitor the interactions, and interpret a diversity of software resources in order to obtain both desired and consistent system behavior. In this paper, we examine the types of flexibilities and integrative processes necessary to a

design environment, especially the services available in an environment to help the user select, integrate, adapt, and explain the software resources in that environment(*intelligent user support functions*), our approach to providing these services by the processing of explicit descriptions of the software resources (*wrapping*) and *VSIM*, a simulation we built to study the nature of the wrappings, wrapping processors, and different software architectures.

2 COORDINATION AND FLEXIBILITY

The *VEHICLES* environment is composed of both conventional and artificial intelligence methods and programs. It is a distributed, multilingual environment that is largely written in Prolog, C and C++, but also supports external programs written in several languages. It supports many types and sources of information and knowledge, multiple models, and a broad toolchest of analyses, graphics, and other types of software programs. Although it is a prototype environment, in its four years of development it has been used to provide some of the analyses supporting several space programs.

The capabilities and types of information required in *VEHICLES* are numerous, varied, and complex. Designers and planners of space systems take many different approaches and may work at different levels. Mission requirements, technological advances, and constraints on cost or other resources all influence design decisions. Consequently, design tools must be flexible enough to allow the designer to focus on different levels of detail and on different aspects of the problem; an important part of design is knowing what to consider a factor and what to consider a constraint at any given time in the design process. *VEHICLES* allows designers to address many different design problems, from the definition of mission requirements to scheduling issues related to the effectiveness of environmental testing. As sets of possible design solutions are generated, the user can compare designs and evaluate their relative merits. *VEHICLES* has taken a significant step toward speeding up the creation of *families* of viable designs, rather than primarily focussing on the analysis and performance of single-point solutions.

We can summarize these flexibilities into three broad areas. First there are analyses that support the user in performing trade-offs, in deciding what the critical factors are in designing a system, and in relaxing the requirements and studying the impact of different sets of requirements and constraints. In other words, rather than just providing analyses and models that operate on the basis of a fixed set of constraints, we want to support the user's ability to understand and characterize the space of possible solutions.

Second, we want to make each feature of a model a variable. That way, the users can explicitly examine the impact on their design or study of using different models, different sets of constraints, different numerical routines, different approaches or design strategies, and even different user models. We support this in Vehicles by providing a library of models, often with several alternative versions that can be compared, and a toolchest composed of many different types of analyses. We allow the user to start anywhere in the design. For example, a user can adapt a previous design, start top-down by interpreting broad mission requirements for a new satellite, focus initially on sizing a subsystem and assessing its impact on other subsystems, or start with a new technology for a component and study its effect on different systems. Also, the system will process with partial results and missing information, and will provide intelligent defaults for initial factors to consider, parameter values and options. Some of the continuing research challenges in providing this type of flexibility is to process with incomplete information, to provide models at different levels of detail and precision and with alternative assumptions, and to combine qualitative and quantitative information.

Third, all of these tools and models were not developed in the same language, on the same platform, or by the same authors. Hence, at a network level, we have found that the system needs to be multilingual, distributable over different platforms, and both extendable (can add new instances of known types) and open (can add new types of programs, functions, and operations).

However, each of these flexibilities noted above requires some active coordinative processes to manage the diversity of features provided. By *coordination*, we mean the organized activities of parts in relationship to global goals. In order to help coordinate the models of a complex system and the supporting software resources in a software system, we need to provide: (1) perspective (providing both representations that present different overviews of the whole system and representations that permit a number of different subjective views); (2) interface (providing low-level protocols); (3) integration (providing information and processes to help determine the conditions under which tools or models should be interfaced); (4) overhead (providing sufficient information in a processible form so that the system can monitor interactions and interpret and evaluate its results and resources.) The distinction we want to make between

"interface" and "integrate" is essentially the difference between providing the permissive and essentially static capability that establishes that two tools can exchange bytes (interface) and providing the capability for the system to dynamically determine when two tools should exchange information (integrate).

Intelligence in a complex system costs - both in terms of the types of information that must be available and the types of processes necessary to make use of it. In a biological system, adaptive, intelligent behavior includes the ability of the system to somehow see itself within a context, monitor and evaluate its own and others' behaviors, and adjust and refine its behavior as required by criteria (or constraints) imposed from both the external environment and itself. A biological system does this by having a variety of different mechanisms that support different kinds of flexibilities at many levels. For example, biological systems have many ways of producing graded responses - at a cellular and chemical level, at a neuronal level, and at a behavioral level. They can address several goals at once and combine the information from a variety of senses. Yet all of this diversity is gracefully merged into one fluid, continuous, and coordinated pattern of behavior (See Bellman and Walter, 1986). However, even in the simplest organism, the flexibilities and the coordination does not come simply or cheaply. [1]

In a computer system, at least for the present, the emphasis must largely be on explicit knowledge that is processed by other programs and less on generative processes that by their actions give rise to coordination. For example, one key aspect of coordinating a complex system is to have one program monitor the results and behavior of other programs. This requires much more than simply having the monitoring program triggered to do some action by the values sent to it by the monitored program; instead it requires explicit representations of the monitored program, including its failure modes, its usual output, and a record of its results and so forth. Hence, to provide such coordination requires a tremendous amount of bookkeeping (documentation, tracing, design histories, shared databases and knowledge bases).

Many researchers have been developing new ways of creating increasingly open environments (See Purtilo et al., 1985; Erman et al., 1986; Bond and Gasser, 1988 for examples). In our research on *VEHICLES*, a conceptual design environment for space systems, we have been developing an approach (called *wrapping*) to flexibility and integration based on the collection and then processing of explicit qualitative descriptions of all the software resources in the environment (Bellman and Gillam, 1990; Landauer, 1990). The detailed descriptions (or metaknowledge) of the resources are used by the system to help partially automate the combination, selection, and adaptation of tools and models to the particular requirements of the user and the type of problem being solved. This approach also allows for a great diversity of information types and languages. At the current time, we have a simulation, *VSIM*, used to study both the types of wrapping descriptions and the wrapping processes.

3 INTELLIGENT USER SUPPORT

As noted above, supporting the design and analysis of complex systems requires a diversity of models and tools; the result is often a software environment that becomes itself a complex system. Hence, we feel it is important to provide *intelligent user support functions*; that is some means of supporting the user (be it human or another computer program) in the selection, assemblage, integration, adaptation, and explanation of the software resources.

By *selection*, we mean that the system helps the user to select which software resources are appropriate given the current problem or task. For example, in *VSIM*, we have experimented with two simple scenarios involving selection: in the first case, a human user has selected *optimize* from a menu containing a number of analyses in *VEHICLES* and the system uses the wrappings of three optimization programs and the wrapping for the set of equations to be optimized to determine which optimization program is most appropriate; when the system finds no basis for distinguishing between two of the optimization programs, it uses wrappings again to select an appropriate user screen for presenting the user with the remaining candidate optimization programs from which to select. In the second case, a *VEHICLES* solver has bombed on a set of equations and the system itself poses the problem of selecting another solver, which is done automatically on the basis of the wrappings, with a record kept of the choice and use of the selected solvers.

To us *integration* is more than simply allowing tools to "talk" (we prefer to use the term *assemblage* for this permissive hooking-together of tools); rather, it is providing some means for deciding when tools should talk. For example, when should a given model send its output to another model; when should a given

[1] One current debate in theoretical neuroscience is whether such adaptive, intelligent behavior is best modelled by making use of knowledge or information concepts (a software analogy) and/or as physical dynamical systems (an analogy relying on chaotic dynamics, a mathematics describing stabilities and changes of stabilities). In a software system, we have far fewer interesting and generative capabilities.

database provide the information for a given analysis. In the wrappings, we have conditionals (implemented as rules, but there could be other implementations) which help define the context for integrating tools and models.

By *adaptation*, we mean the modification of the software resource depending upon the problem or task and the information currently available. This adaptation could be changing the input file or control parameters to a simulation or changing the queries to a database or changing the default values in a model and so forth. The last critical intelligent user support function is *explanation*, that is, at a minimum, providing the means to record and document how the software resources were selected, integrated, and modified during the use of the software environment. Eventually, we would like a more interesting form of explanation, where the explanation is adjusted depending upon the user and the problem or task.

4 STUDYING WRAPPING WITH VSIM

Using *VSIM*, we have learned a number of things about the knowledge necessary in wrapping, which we summarize below. First, in order to perform the five intelligent user support functions listed above, we need to represent and utilize three types of knowledge: metaknowledge (e.g. knowledge about a given method or tool or about the use of knowledge in a knowledge base), user models (knowledge about the types and activities of the user), and domain knowledge (especially knowledge about the types of problems in that domain and the types of contexts that constrain the choice and use of given methods and information.) In *VSIM*, we have been experimenting on how to utilize each type of knowledge; currently *VSIM* is composed of a *planner knowledge base (PKB)*, a *wrapping database (WDB)*, and a set of wrapping processors and other software resources, which are all wrapped. The PKB contains triplets of the form:

{ Problem Definition
Information Available
Resource Name }

The "Problem Definition" has been simplified to be a list of keywords corresponding to the activities that the system can provide to the user, such as "optimize", "solve", "parametric study"; or at a higher level, they could be such activities as "design a new satellite" or "tailor an existing satellite". Eventually, we can incorporate more interesting problem decomposition methods; we have simplified the problem definition in order to study how to relate the problem descriptions to the software resources, and how to specify the minimal information required by the software resource to be used for a given problem. In the WDB, each wrapping contains a name of a software resource, input and output requirements and restrictions, and then we have been experimenting with many different ways of expressing additional information about the appropriate use of the resource under different conditions. One important point to note is that in *VSIM* all the software resources are wrapped, including all programs processing the wrappings. Hence, *VSIM* selects the "matcher" program used to match the wrappings of the model and the optimization programs, in the example scenario described above.

In the PKB, the problem definition reflects knowledge about: the resources provided by the software environment (metaknowledge); the desired activities of the user (user models); and the methods and requirements of solving problems in a given domain. In the WDB, the knowledge is largely metaknowledge about the use and type of software resource, but it crosses any neat lines and includes in any conditionals, references to domain knowledge and user models.

One of the problems we encountered when we started to write the wrappings was what we call the "library problem." That is, we tried to formulate a description of a software resource that would be suitable for all wrapping purposes for all time. We soon learned that, at least for the purposes of formulating these descriptions, we need to start with five different descriptions, each containing the semantics corresponding to the five different intelligent user support functions described above. In addition, for a large software resource (such as the large simulations we deal with in *VEHICLES*), we need to develop several wrappings, each corresponding to a major mode of use for that resource. Lastly, an issue we have not yet addressed in *VSIM*, we can not consider the wrappings as a static description. This becomes particularly clear when we attempt to include the designs that are being generated by the *VEHICLES* system as part of the wrapped resources in the system. Thus the system must do more than just process existing descriptions; it must have the processes that permit the development of these descriptions as the system develops its "products" (designs, reports, and so forth). This has led to the realization that in fact the wrappings will always be evolving as either the system generates new resources (its "products"), or either the system or the user discovers additional information about a given resource under some given context or usage.

Although we have focussed on the flexibility and integration provided by utilizing wrappings, it is impor-

tant to emphasize that flexibility and integration in a software environment occur at several different levels. Hence, in addition to the use of wrappings, we have also experimented with how best to use network services and message-passing kernels to take advantage of different programming languages and platforms.

Lastly, the wrappings represent a self-description of a software environment that is processible by that environment. In Maes' terminology (1987), such a system is "computationally reflective" and her everyday examples of reflection range from the now commonplace, e.g. keeping performance statistics and debugging information to the exciting possibilities for autonomous systems and programs with self-optimization, self-modification, and self-activation. We are excited by the recent realization that *VSIM* can eventually be considered just another resource in the *VEHICLES* environment; one with the rather special property of being a simulation of itself. Hence when we add a new resource to *VEHICLES*, we would eventually be able to immediately simulate its integration into the system. With wrappings, we hope to make software architectures more testable, maintainable, and open. The hope is that eventually we will have computer systems in which the means to test and evaluate the system are not peripheral, but rather an integral part of the software system.

REFERENCES

Bellman, Kirstie L. and A. Gillam. Achieving Openness and Flexibility in Vehicles. In AI and SIMULATION Theory and Applications. Proceedings of the SCS Eastern Multiconference, 23-26 April, 1990, Nashville, Tennessee. Simulation Series Vol. 22(3), April 1990. pp 255-260.

Bellman, K. and D. Walter, Biological Processing. *Am. J. Physiol.: Reg., Int., Comp. Physiol.* 15(6), 860-867, 1984.

Bond, A.H. and L. Gasser, editors. Readings in Distributed Artificial Intelligence. Los Altos, Ca: Morgan Kaufmann, 1988.

Erman, Lee D., Jay S. Lark, Frederick Hayes-Roth. Engineering Intelligent Systems: Progress Report on ABE. Teknowledge Inc TTR-ISE-86-102. In Proceedings: Expert System Workshop, April 1986. SAIC Report Number SAIC-86/1701.

Landauer, Christopher. Wrapping Mathematical Tools. In AI and SIMULATION Theory and Applications. Proceedings of the SCS Eastern Multiconference, 23-26 April, 1990, Nashville, Tennessee. Simulation Series Vol. 22(3), April 1990. pp 261-266.

Maes, Pattie. Concepts and Experiments in Computational Reflection. OOPSLA '87 Proceedings, 1987. pp 147-155.

Purtilo, James. POLYLITH: An Environment to Support Management of Tool Interfaces, ACM 0-89791-165-2/85/006/0012. 1985.

Purtilo, James M. "POLYLITH and Environments for Mathematical Computation", University of Illinois Dept of Computer Science, Report No. UIUCDCS-R-84-1135. 1984.

Walter, Donald O. and Kirstie L. Bellman. Some Issues in Model Integration. In AI and SIMULATION Theory and Applications. Proceedings of the SCS Eastern Multiconference, 23-26 April, 1990, Nashville, Tennessee. Simulation Series Vol. 22(3), April 1990. pp 249-254.

AUTHOR BIOGRAPHY

KIRSTIE L. BELLMAN is a senior scientist in the Computer Science and Technology Subdivision of The Aerospace Corporation. Dr. Bellman has over twenty-five years of academic, industry, and consulting experience in the development of both conventional computer models and artificial intelligence applications. Her published research spans a wide range of topics in the cognitive, neurophysiological, and information-processing sciences. Five years ago, she started the Vehicles project.