N93-11946

AN APPROACH TO INTEGRATING AND CREATING FLEXIBLE SOFTWARE ENVIRONMENTS

Kirstie L. Bellman, Ph.D.
Computer Science and Technology Subdivision
The Aerospace Corporation
El Segundo, California

Engineers and scientists are attempting to represent, 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., 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 processes necessary to use the metaknowledge to combine, select, adapt, and explain some of the software resources used in *VEHICLES*. Below, we briefly describe what we have learned about the types of knowledge necessary for our wrapping approach and the implications of wrapping for several key software engineering issues.

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 a diversity of languages. It supports a variety of information and knowledge types, 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. 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 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.

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. Rather, we need to devise wrapping processes such that the descriptions continue to build, as the resources are used. Similarly, a human user must be able to browse, edit, and add to the descriptions.

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

The wrapping approach also advances software engineering in several significant ways: 1) it provides explicit descriptions (and documentation) about each software resource, including what is in essence both a specification for that resource and practical advice on its acceptable and appropriate use; 2) it provides traceability during dynamic testing, and an easy way to insert probes; 3) it allows standard structural testing of the wrappings, when these are stored as a knowledge base/database; 4) it allows the possibility of incorporating on-line software checkers.

The wrappings are descriptions in a database/knowledge base. Hence, a number of standard static testing and analysis strategies that have been applied to knowledge bases (see Landauer, 1990; Bellman, 1990), can be applied to the wrapping database. For example, static analyses can check to see if a given resource is used by any other resource; standard type checking can pick up not only lower-level information about data types, but also new higher-level information about the type of resource and uses that were made explicit in the wrappings. With a simulation such as *VSIM*, one can dynamically test the interactions among different software resources. Wrappings can provide an interesting means of seeding errors, adding special programs to provide intermediary values or other types of debugging information, or altering the combination of resources. When combined with 'user log'(in *VSIM*) and self-documentation (in *VEHICLES*) programs, this approach offers the ability to perform and record a large variety of software engineering experiments.

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. The Modeling Issues Inherent in Testing and Evaluating Knowledge-Based Systems. Expert Systems With Applications, Vol. 1, 1990. (Pergamon Press)

Bellman, Kirstie L. and A. Gillam. Achieving Openess 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.

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. Correctness Principles for Rule-Based Expert Systems. Expert Systems With Applications, Vol. 1, 1990. (Pergamon Press)

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.