THE KNOWLEDGE-BASED SOFTWARE ASSISTANT

Lt Kevin M. Benner, USAF and Douglas A. White
Command and Control Technology Division
Rome Air Development Center
Griffiss AFB, NY 13441-5700

ABSTRACT

In 1983, Rome Air Development Center (RADC) published "Report on a Knowledge-Based Software Assistant" [Green, et al 83]. This document brought together key ideas on how artificial intelligence (AI) could be used in the software development process. Since then RADC has embarked on the first of three contract iterations to develop both a Knowledge-Based Software Assistant (KBSA) and the enabling supporting technologies which are necessary. KBSA is a formalized computer-assisted paradigm for the development, evolution, and long-term maintenance of computer software. KBSA captures the history of system evolution. It provides a corporate memory of: how parts interact, what assumptions were made and why, the rationale behind choices, how requirements are satisfied, and explanation of the development process. KBSA accomplishes this through a collection of integrated dedicated facets. Their areas of expertise are: project management, requirements, specifications, implementation, performance, testing, and documentation.

RADC is currently in the midst of the first iteration. Facets which are now under contract include: Requirements Assistant with Sanders Associates, Specifications Assistant with Information Sciences Institute (ISI) at USC, Performance Assistant and Project Management Assistant with Kestrel Institute, and the KBSA Framework with Honeywell. This paper will first describe where the KBSA program is now, four years after the initial report; secondly, describe what RADC expects at the end of the first contract iteration; and finally, characterize what the second and third contract iterations will look like.

INTRODUCTION

In 1983 Rome Air Development Center (RADC) published "Report on a Knowledge-Based Software Assistant" [3]. This document brought together key ideas on how artificial intelligence (AI) could be used in the software development process. Since then RADC has embarked on the first of three contract iterations to develop both a Knowledge-Based Software Assistant (KBSA) and the enabling supporting technologies. This paper will describe: 1) History leading up to KBSA, 2) What is KBSA, 3) Development strategy and current status of KBSA, and 4) Concluding remarks.

HISTORY

The KBSA research program is a natural progression of research and development undertaken by RADC in its continuing pursuit of a solution to the well known "software life cycle problem". This application of AI technology to the problem of software development was a predictable outgrowth of RADC's longstanding commitment to research and development directed at enhancing productivity. From the early 1970's when RADC was championing the cause of "modern" high level languages and "structured" implementation methodologies, a less publicized but important track of research was being addressed on a smaller scale for the

development of greater formalism and abstraction for the objects and activities belonging to the technology of software development. The publicity given to the AI community in the late 1970's and early 1980's due to successes in building workable expert systems and the announcement of the Fifth Generation Computer Program of Japan, resulted in the emphasis of AI technology within RADC's research program. This in turn led to the examination of the possibility of applying the technology that worked so well in areas such as geological analysis and locomotive maintenance to the problems of software development and maintenance. This atmosphere, when coupled with the growing compilation of results from earlier research, encouraged the selection of software development as an application to drive and demonstrate AI technology research developments. The particular paradigm selected and eventually identified as the KBSA was the result of careful consideration of the state of technology as demonstrated by prior work, and the goals and practical requirements of a system to support software development.

Throughout the 1970's, concurrent with the major RADC projects in language/compiler systems and programming methodologies, efforts were undertaken which explored formalisms with which to better describe the objects and algorithms comprising software. It was recognized that there were many flaws with the existing manner in which programs were created and the languages in which they were expressed. A few of the outstanding problems that were addressed during this time included: programs could be syntactically correct without providing the desired solution or being logically correct; and, even with "better" high level languages, programs were incomprehensible, even to the author with the passing of time. Each of these problems and the research efforts addressing them had a part in exposing the members of the Command and Control Software Technology Division to the work being performed in the world of AI, and while simultaneously providing necessary support for some of the important AI research ideas of the time.

The problem of logical correctness of programs was attacked in many ways. However, the two that are important to the KBSA (even though not receiving widespread adoption in the world of software engineering) are formal proofs of correctness and formal specification languages. In late 1974 an initial effort was undertaken to explore the applicability of formal verification methods to existing programming languages. This process of "verifying" a program's correctness consists of establishing by mathematical proof that whenever a program is executed with specified input data and execution environment the execution will terminate and upon termination the values of program variables will meet output specifications. The foundations of formal program verification are identical with those of a significant body of the work in automatic programming. As might be expected, many of the same individuals are involved in both areas of research. The need for formal specification languages was emphasized by the difficulty of verification of programs in existing computer languages. In 1976, research was initiated to develop a "language" that could be used to provide formal descriptions of programs. The specification languages resulting from this research were found to be unwieldy for extensive use by humans and as is the case with formal program verification technology are not known to have achieved widespread use. However, formal specifications are particularly important to the KBSA because they provide the formalism needed to enable reasoning about programs. Additionally, these particular research efforts caused RADC to become involved in a progression of efforts addressing automatic programming which continue today.

The need for a more natural and abstract method of expressing a problem solution to a computer led to the exploration of the concept of a Very High Level Language (VHLL). The goal of this research was to provide a language system combining the capabilities of conventional languages with those of logic programming systems enabling the user to program not only computational processes in the conventional sense, but also "reasoning" processes. From this research at Syracuse has emerged an evolving family of languages which exhibit many of the characteristics that will be needed in the Wide Spectrum Language (WSL) of the KBSA. This research was facilitated by the early theoretical work of Robinson [14] which has provided much of the foundation for logic programming. Through this work at Syracuse University, which began in the mid 1970's, RADC has been cognizant of the potential for a software development paradigm unlike that existing for conventional programming languages.

The arrival of practical diagnostic systems based on AI technology in the late 1970's led to a project to investigate the possibility of creating a knowledge-based system that would diagnose software systems and assist in their maintenance. The conclusion of this investigation was that this type of diagnostic expert system would be impossible for software because of the inadequacy of the knowledge about software in general and any software system in particular. This initial negative result, along with the dim prospect for immediate relief from automatic programming caused a serious consideration of the alternatives. The alternative perceived to have the greatest promise was that of a knowledge-based system that did not provide total automation of the software synthesis process, but did maintain a total record of all decisions and activities which occurred in the creation of a software system. This system would possess the expertise to automatically perform many of the tedious tasks of program development, but would be guided in the application of transformations by the human user. Communications among members of a development organization would also be enhanced by the monitoring and reporting capabilities provided by the knowledge-based system. These are the concepts that were further developed and described in the 1983 report considered to be the "defining document" of the KBSA.

WHAT IS KBSA?

The KBSA approach is a departure from the existing software engineering paradigm in that it attempts to formalize all activities as well as products of the software life cycle. It is a formalized computer-assisted paradigm for the development, evolution, and long-term maintenance of computer software. KBSA captures the history of system evolution. It provides a corporate memory of: how parts interact, what assumptions were made and why, the rationale behind choices, how requirements are satisfied, and explanation of the development process. KBSA accomplishes this through a collection of integrated dedicated facets and an underlying common framework.

KBSA has four main distinguishing features. First, the specification is incremental, executable, and formal. Incremental means that the specifier may gradually add more detail to the specification and is not forced to initially describe the system in complete detail. Executable means that the specification is "runnable" like a prototype. This allows the specifier to validate the specification against user intent by actually showing him/her the "running" specification. Finally, formal means that the specification is expressed in a language with precise semantics, avoiding the ambiguity of natural language.

Second, the implementation is formal, that is, all decisions made during the implementation are captured and justified. Typically, implementation will be done via correctness preserving transformations, thus guaranteeing by default a verified implementation.

Third, project management policies will be formally stated and enforced by KBSA. That is, project policy will define the relationship between various software development objects (eg. requirements, specifications, code, test cases, bug reports, etc.) and then be enforced by KBSA throughout the software development process.

Fourth, and finally, maintenance will be done at the requirements and specification level, rather than via patches to the code. That is, since maintenance activities are normally a result of new or better defined user requirements, it makes sense to reflect this in the requirement/specification.

In order to build a KBSA, the authors of the initial report point to the need for specific supporting technologies. These supporting technologies fall into four main categories: a wide spectrum language, general inferential systems, domain specific inferential systems, and system integration.

A wide spectrum language (WSL) is a single language which provides the user with the ability to capture the formal semantics of the system under development regardless of the level of detail (or the step in the development cycle). A wide spectrum language is both a language and an environment. It must provide uniform expressibility, regardless of what is being described (ie.

requirements, specifications, code, test cases, project management policy, etc). Not only must a WSL be able to express these objects, it must do so in a way which is consistent at all levels, both syntactically and semantically.

A general inferential system is a system which supports reasoning. In particular, we are concerned with the overall efficiency of this reasoning, how to capture such things as logic in inference rules and data structures, the quality of explanation generated by the system, and the ability to apply this inferencing power to specific domains.

Domain specific inferential systems extend general inferential systems to include aspects unique to software development. This topic focuses on the knowledge representation of software development objects and inference rules and, in particular, how they can be formally represented and used for further reasoning.

System integration deals with the inherent competition between facets and how a technology base can be put together such that all phases in the software development process are supported sufficiently.

DEVELOPMENT STRATEGY AND CURRENT STATUS OF KBSA

When the KBSA report first came out, it was clear that the supporting technologies were not adequately developed. To address this shortfall a KBSA was to be developed in three iterations. The first iteration was aimed at designing the individual facets and seeing where the supporting technologies could be pulled along. Additionally, there was a desire for an advancement of the understanding of the software development process, particularly within this new KBSA development paradigm. In line with this concept, work began on a Framework (FW) and five (5) facets: Project Management Assistant (PMA), Requirements Assistant (RA), Specification Assistant (SA), Performance Assistant (PA), and Development Assistant (DA) [DA will not begin until FY 88]. Though boundaries between facets may appear in the first iteration, they will blur in the second iteration and disappear completely in the third iteration.

First Iteration

The results of this have been twofold. First, each facet has pulled at the supporting technologies such that they have advanced the state of these technologies. Universal solutions were not sought, rather solutions unique for each facet have been found. Secondly, each facet developer has made progress in formalizing their particular life cycle phase. This formalization has focused both on the products of individual phases (eg. requirements, specification, and code), but more importantly on the process of how these products came about. In this section the basic approach and milestones of each contractor will be described. Included in this description will be the formalization of the particular life cycle phase and the impact on supporting technologies.

Work on the definition of a PMA formalism and construction of a prototype began in 1984 [9]. Kestrel Institute was the developer. The life cycle goals of PMA were to provide knowledge-based help to users and managers in project communication, coordination, and task management.

The capabilities of PMA fall into three categories: project definition, project monitoring, and user interface. Project definition consists of structuring the project into individual tasks and then scheduling and assigning these tasks. Once the project has been decomposed into manageable tasks, it must be monitored. This monitoring is in the form of cost and schedule constraints. Also included in monitoring is the enforcement of specific management policies (eg.DoD-Std-2167, rapid prototyping, KBSA, etc.). In addition, PMA provides a good user interface for project monitoring and project definition. This interaction is in the form of direct queries/updates, Pert Charts, and Gantt Charts.

The above capabilities were important, but would be expected of any project management tool. What sets PMA apart from its predecessors is the

expressibility and flexibility of the PMA architecture. Not only does PMA handle user defined tasks, but it also understands their products and the implicit relationships between them (eg. components, tasks, requirements, specification, source code, test cases, test results, and milestones). Also present in PMA are objects unique to programming-in-the-large (ie. versions, configurations, derivations, releases, and people).

From a technical perspective, the advances made in PMA include: the formalization of the software development objects enumerated above, the development of a powerful time calculus for representing temporal relationships between software development objects [10], and a mechanism for directly expressing and enforcing project policies.

The work on the Requirements Assistant [2, 15] began in 1985 by Sanders Associates. The main task of RA was to deal with the informal nature of the requirements process. Sanders' intention was to allow the user to enter requirements in any desired order or desired level of detail. It would be the responsibility of RA to: 1) do the necessary bookkeeping to allow user manipulation of requirements and 2) maintain consistency among requirements as they become known.

RA capabilities include: support for multiple viewpoints (eg. data flow, control flow, state transition, and functional flow diagrams), management and smart editing tools to organize the requirements, and the ability to support free form annotations to requirements. In addition to this, RA's underlying knowledge representation, Structured Object and Constraint Language Environment (SOCLE), enables RA to identify contradictions and generate explanations.

The main technical thrust has been on how to handle informality when trying to build an underlying representation of the requirements. This is done by supporting incomplete graphical descriptions at the user interface level, but maintaining a consistent, though not necessarily complete, internal representation. This is done via SOCLE that provides a truth maintenance system which supports default reasoning, dependency tracing, and local propagation of constraints. RA provides application specific automatic classification which is used to identify missing requirements by comparing the current requirements against "typical requirements" of a generic system (already represented within RA's knowledge base). This comparison is then used to generate questions of the specifier to either be sure something vital has not been left out or to gather a justification for the difference.

The main goal of the Specification Assistant [1, 8] is to develop a formal specification of the system under development and then to validate it against user intent. The development of the formal specification must be supported in an incremental fashion, modeling the way developers typically construct specifications. The validation must be done by exposing the specification to the user at the earliest opportunity and continued throughout the construction process. The effort to develop a KBSA Specification Assistant began in 1985. This work is being done at the University of Southern California- Information Sciences Institute (ISI).

SA capabilities include: an incremental specification language which is executable and a natural language paraphraser which will translate a given specification into English. These capabilities have been built on top of ISI's Wide Spectrum Language AP5, and the development environment CLF. SA can currently handle specifications of a couple pages.

The main technical issues concern: 1) identification of specification statements as requirements or goals and the transformation of these from a high level specification into a low level specification and 2) extracting and assembling system views (ie. reusing specifications and parts of specifications).

The distinction that ISI makes between requirements and goals is that requirements are inviolable constraints, while goals describe general behavior which may have exceptional cases not currently covered by the goal. With this distinction in mind SA provides high level editing commands to further transform the specification into a low level specification. Requirements are transformed in a correctness preserving manner to maintain satisfaction of the requirements.

Goals, on the other hand, may be "compromised" in order to handle exceptional cases. That is, after a transformation, the meaning of a goal specification may change.

Extracting and assembling system views deals with building up a specification from smaller specifications (ie. reuse of other previously defined specifications) as opposed to the top down refinement presented in the previous paragraph. SA can combine disjoint specifications, but tools are needed to aid in combining specifications which share common terms.

The Performance Assistant [4, 5, 6] work began at Kestrel Institute in 1985 and is expected to run until 1990. Long term goals for a performance facet are to guide software performance decisions at many levels in the software development cycle, from requirements specifications in very high level programs to low level code. The approach is to combine heuristic, symbolic, and statistical approaches which will provide capabilities for: symbolic evaluation, data structure analysis and advice, and algorithm design analysis and advice. This effort is focusing on data structure selection, performance annotations of a specification, analysis and propagation of performance information, and control structure performance analysis.

Technical issues that have been addressed thus far are data structure selection (DSS) using symbolic and heuristic techniques and the development of PERFORMO, a functional specification language with set theoretic data types. PERFORMO is similar to VAL [12], developed at MIT, and SISAL [13], developed at Lawrence Livermore Laboratory. PERFORMO is intended primarily for DSS work, but is sufficiently expressive to be a good initial specification language for the next two research issues: subroutine decomposition and control flow optimization.

The basic strategy employed in DSS is to supply refinement decisions when they are needed by the implementation generator (ideally this would be the DA). When a refinement decision is needed, PA determines the relevant program properties necessary to make a satisfactory selection. The relevant properties would vary on where in the implementation the generator is. Properties refer to how a specific variable will be used and some characteristics of it. These properties could include: whether the variable is random access, ordered, enumerated, dynamic, and/or possibly empty. Based on these properties, specific implementation decisions can be made.

Development Assistant is the most recent facet undertaken, although contract work has not yet begun. RADC will award the DA contract in early FY 88. The basic thrust of this effort will be to derive an implementation from a completed specification, automating (via automatic transformation) where possible and capturing user supplied design decisions when needed.

The Framework [7, 11] was considered to be necessary to bring a global perspective to KBSA. Initial work on the FW began at Honeywell Systems and Research Center in early 1986. The goal of the Framework is twofold: 1) to develop an integrated KBSA demonstration and 2) to propose the specification of a KBSA framework through which all facets must interact and communicate. The purpose of the former is to provide a concept definition that would be intuitively obvious to the most casual observer. The purpose of the latter is to facilitate a tightly coupled interaction between facets. The framework will provide a common reference for each facet developer allowing the sharing of information. Interacting with the framework will be a requirement for all second iteration contracts. The result in the future will be a more tightly integrated KBSA.

The main technical issues are 1) define minimum functionality which the framework must provide to all facets, 2) define a common interface to the framework, 3) extend the framework to a distributed environment, 4) support programming-in-the-large concepts like configuration control, and 5) provide a consistent user interface.

The overall results of the first iteration will be a KBSA concept demonstration consisting of mostly loosely coupled facets with the exception of PMA and the framework which will be tightly coupled. Each facet will exist on separate machines and communicate via the framework. The framework will be responsible

for maintaining traceability between software development objects and keeping facets updated. Establishing the initial traceability (eg. the relationship between requirements objects and specification objects) is the responsibility of the involved facets.

In the past, each facet developer has had their own problem domain in which to work. In general these domains have been small or toy-like. For the KBSA demonstration the problem domain will be the air traffic control problem (ATC). This domain has the advantage of being a substantial problem with a variety of real world issues (ie. real time requirements, data base management, user interaction, interaction with the outside world, and changing or not well defined requirements). The intention of the demonstration is not to solve the ATC problem, but rather to have the ATC requirements be a driver for KBSA. The demonstration will most likely focus on some portion of the overall ATC problem.

Second And Third Iterations

The second iteration of KBSA will begin at the completion of the framework effort. All facets in the second iteration will interact with the framework and thus each other. This will be done by either building individual facets in Honeywell's framework, or more likely, individual facet developers will extend their own frameworks (eg. REFINE, AP5, SOCLE, etc) to adhere to the framework specification. Both are acceptable from a RADC perspective. Prospective developers must convince an RADC that either 1) they will use the Honeywell framework or 2) that their framework does or will soon adhere to the standard. There will be a mechanism to allow for some deviations from the framework specification. This is important since at the beginning of the second iteration the framework described in the specification may not be sufficiently powerful to implement all facets or some specific feature of the framework may preclude functionality necessary for a particular facet. For the third iteration there will be no exceptions.

During the second iteration, work will continue on individual life cycle phases, while also focusing on the interaction between facets and the framework.

For the framework contract, work will address raising the functional level of the framework. The goal is for individual facets to be concerned only with activities unique to their respective facets, while the framework will be responsible for all generic tasks (eg. knowledge representation, knowledge base maintenance, communication between facets, policy enforcement, and general inferencing capabilities).

CONCLUSION

In conclusion, there has been progress over the last four years. The question now is, how close are we to a workable KBSA? The answer is greatly dependent upon the framework specification which will come out of this first iteration. If it is sufficiently powerful, we could have a workable KBSA at the end of the second iteration. On the other hand, if most second iteration contractors have to make generic extensions to the framework, we can not expect a workable KBSA until the third iteration.

REFERENCES

[1] Balzer, R. et al., "Knowledge-Based Specification Assistant", Interim Technical Report, RADC, Griffiss AFB, NY, Dec., 1986.

[2] Czuchry, A. J. Jr., "Where's the Intelligence in the Intelligent Assistant for Requirements Analysis?", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[3] Green, C. et al., " Report on a Knowledge-Based Software Assistant," RADC Tech. Report TR-83-195, RADC, Griffiss AFB, NY, Aug, 1983.

[4] Goldberg, A. and Smith, D., "Towards a Performance Assistant", Interim Technical Report, RADC, Griffiss AFB, NY, Nov., 1986.

[5] Goldberg, A. and Smith, D., "Performance Estimation for a Knowledge-Based Software Assistant", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[6] Goldberg, A., "Technical Issues for Performance Estimation", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[7] Huseth, S. and King, T., "A Common Framework for Knowledge-Based Programming", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[8] Johnson, W. J., "Turning Ideas into Specifications", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[9] Jullig, R., et al., "KBSA-PMA Technical Report", Final Technical Report, RADC, Griffiss, NY, Nov., 1986.

[10] Ladkin, P., "Primitives and Units for Time Specification", AAAI-86, Philadelphia, PA, Aug. 11- 15, 1986.

[11] Larson, A. and Huseth, S., "KBSA Common Framework Implementation", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[12] Robinson, J. A., "A Machine Oriented Logic Based on the Resolution Principle", Journal of the ACM, Jan., 1965.

[13] McGraw, J. R., "The VAL Language: Description and Analysis", ACM Transactions on Programming Languages and Systems, Jan., 1982.

[14] McGraw, J. R., et. al, "SISAL: Streams and Iteration in a Single Assignment Language", Technical Report M-146, Lawrence Livermore Laboratory, Mar., 1985.

[15] Sanders Associates, "Knowledge Based Requirements Assistant", Interim Technical Report, RADC, Griffiss AFB, NY, Mar., 1986.