The Knowledge-Based Software Assistant: A Formal, Object Oriented Software Development Environment

Capt Mark J. Gerken, Nancy A. Roberts, and Douglas A. White
Rome Laboratory
Knowledge Engineering Branch/C3CA
525 Brooks Road
Rome, NY 13441-4505

Abstract — The Knowledge-Based Software Assistant (KBSA) program was initiated in 1984 following the publication of the Rome Laboratory technical report, "Report on a Knowledge-Based Software Assistant." This report described a new paradigm of software development where the tedious and mundane tasks of system development were automated. We have striven to implement and improve the paradigm described in that report. Specifically, we have concentrated on the definition and implementation of an integrated suite of tools designed to support the management, development, and evolution of complex software-intensive systems. This paper describes the KBSA paradigm through an exploration of four of its key facets: Project Management, Requirements Acquisition, Specification Elaboration, and Formal Development.

I. INTRODUCTION

Decreasing cost and the growing availability and power of computers have enabled them to invade practically every aspect of life. Computers in one form or another are familiar to everyone as activities once requiring humans are now automated. Appliances and facilities that formerly were controlled through simple mechanical or electromechanical mechanisms are now controlled by embedded computers. This widespread use of computers combined with an explosion of information has created an insatiable demand for software. The demand for software has increased more rapidly than the ability to produce it. Software technology’s failings are familiar. Costs have grown to where needed systems are not affordable. The complexity of systems is outstripping our ability to produce and maintain them. The illusory promise of software is that it will be flexible and low in cost because it is not implemented as a physical object; it has not lived up to this promise. Software is expensive to develop and difficult to maintain.

In the early 1980’s, this rapid rate of growth in demand for software was projected to be 12 percent per year. Based upon that projection, a 20 percent improvement in productivity was estimated to be worth $45 billion to the United States in the year 1995 [Boe87]. It was realized at that time that a significant increase in productivity was needed to make modern system development affordable. In 1983, two approaches were proposed to address the software problem. The first, the DoD’s answer to this challenge, was the STARS Program [LDR83]. With Ada as a cornerstone, an improvement of the state of practice was sought through an evolutionary improvement of the existing software development paradigm. A second and less noticed approach proposed a revolutionary new paradigm where the computer handled the formidable task of documenting and maintaining an understanding of the activities and rationale behind the application systems design [GLB+83]. This knowledge would then be used to assist humans, serving in a supervisory role, as they undertook the evolution of ever more complex applications. This second approach, a research and development program initiated by Rome Laboratory and known as the Knowledge-Based Software Assistant (KBSA), is designed to address the complete software life cycle.

II. PROJECT MANAGEMENT

The Knowledge-Based Software Assistant is based upon the belief that by retaining the human in the process many of the unsolved problems encountered in automatic programming may be avoided while still achieving orders-of-magnitude improvements in productivity and quality. It proposes a new programming paradigm in which software activities are machine mediated and supported throughout the life cycle. The goal of this paradigm is to shift from informal manual development to formalized computer-assisted development while addressing the failings of conventional software technology and dramatically improving both productivity and quality. Automa-
tion, intelligent assistance and increased involvement of expert application engineers and end-users will reduce the time to develop and update software, and will ensure that the systems being developed meet the user's requirements. A formal implementation methodology, one that derives code from formal specifications, will help ensure that the implementations are correct. Decision processes as well as products will be represented and recorded to form a knowledge base that is the "corporate memory" of the system development. This knowledge base will provide the basis for various types of analysis and automation, and will be available for life time system support and evolution. Unlike today in which systems are evolved by modifying code, system evolution in a KBSA environment will carried out by modifying requirements; new versions will be derived by reusing the previously captured development process. The time required to produce new versions will be reduced, making systems much more responsive and less costly. The ability of computers to efficiently organize and manipulate large quantities of knowledge in cooperation with the creative and common sense abilities of humans [Ret83] will enable us to create systems of great complexity.

The KBSA program at Rome Laboratory has explored the formalization and automation of many of the facets of the conventional software life cycle. These facets have included Project Management, Requirements Acquisition, Specification Elaboration, and Formal Development. Underlying framework capabilities such as activity coordination (process modeling, enactment, and monitoring), configuration management, learning and documentation generation have also been addressed. KBSA facets have been used to some degree in a variety of programs, including the ARPA Rapid Object Oriented Development and Delivery Technology Reinvestment Program, third party Rome Laboratory research efforts, and the current ARPA Evolutionary Development of Complex Systems initiative. The KBSA program is now at the point where it is about to achieve an operational capability in the form of a prototype. Key facets of this prototype, called the KBSA Advanced Development Model (ADM), are described in the following sections.

The Project Management (PM) facet of the KBSA environment is the focal point of the other KBSA facets. Through the PM facet, new projects can be created and existing projects can be managed.

As shown in Figure 1, the PM interface contains course-grained information about a new or existing project, such as the project name, contract or project number, starting date, and expected or actual completion date. The PM interface also provides access to programmatic information. Specifically, the PM menu button WBS provides access a Work Breakdown Structure (WBS) of the project, and the menu button IBIS provides access to the Issue Based Information System — a hyper-media representation of programmatic issues. Eventually, the PM facet will include access to scheduling tools, and will allow access to the development facets as well. Both the IBIS tool and the WBS tool are described in more detail below.

Before proceeding, it is important to realize that each of these tools work in concert with each other. For example, issues captured by the IBIS tool can be hyper-linked with activities in a WBS. Thus the PM facet acts as an information web or gathering point.
From the PM, it will be possible to map user requirements to statements in specifications, statements in specifications to standards and design documents, and design documents to implementations and test plans, all within a hyper-media environment.

A. The IBIS Tool

As shown in Figure 2, the IBIS tool is a graphical, hyper-media tool used to capture and structure information. IBIS provides basic editing, viewing, and traversal functions.

Four classes of information can be represented in IBIS:

1. Issue. Represented by a question mark, issues are named objects used to represent concerns, questions, or alternatives. Although not shown in the figure, associated with each issue is a dialogue block containing information such as the date the issue was identified, who identified the issue, issue status, and a brief description of the issue. Currently, issue descriptions themselves do not contain hyper-links, but such links could be added.

2. Argument. Arguments — represented by an exclamation mark — are used to state positions associated with issues. Like issues, arguments are named objects containing information identifying their creator and their creation date, and they contain a brief description of or justification for the argument.

3. Position. Positions are named objects representing decisions. Like arguments, positions — represented by a hand — contain a write-up of the decision, and identify the date on which they were taken and identify the person or group who made the decision.

4. Relationship. Relationships between issues, arguments, and positions, can be represented using one of six different types of arrows: a response, support, an objection, a generalization, a replacement, a question, or a suggestion. The different types of relationships are color coded to ease diagram interpretation.

Issues, arguments, and positions can have text, pictures, audio, and/or video clips associated with them; these entities can be viewed from within the IBIS tool. Thus, IBIS defines a form of a hyper-media entity relationship diagram useful for representing and exploring various types of information, including design rationale. In addition to providing access back to the PM facet, IBIS provides access to the WBS tool as well.

B. The WBS Tool

As shown in Figure 3, the Work Breakdown Structure tool is used to create and view hierarchical decompositions of the major tasks and milestones associated with a project. Each milestone has a dialog box associated with it containing schedule and status data. Deliverables associated with each milestone are also identified. Tasks descriptions contain additional information, such as resource assignment and estimated and/or actual resource expenditures.

The WBS tool provides access to PERT chart representations of non-trivial tasks. The PERT representation can also be used to view the flow of documentation within and between tasks. The PERT representation and the WBS representation are tightly coupled in that a modification to one is reflected in the other. For example, adding a new sub-task to a PERT representation will result in the creation of a corresponding sub-task in the WBS representation.

Both the PERT and WBS tools are linked to the IBIS tool. IBIS diagrams can be used to capture and...
explore issues associated with individual WBS tasks or milestones, so issues impacting a task can be represented and investigated within a unified framework.

III. REQUIREMENTS ACQUISITION

As mentioned in Section II, the IBIS tool provides some of the hyper-media support contained within KBSA. The Hypermedia tool provides the remaining support.

The Hypermedia tool is a hyper-media documentation tool used primarily to define and traverse hyper-media links between various documents such as requirements documents, standards, or code segments[And94]. The interface to the Hypermedia tool is shown in Figure 4. As can be seen in the figure, the Hypermedia tool provides the capability to embed pictures, audio, and video within a hyper-node. The structure of the overall hyper-document is also presented, and eventually, a user will be able to jump directly to nodes of interest.

Taken together, the Hypermedia tool and IBIS provide a powerful, hyper-media requirements representation and tracing mechanism. For example, issues identified in an IBIS diagram can be hyper-linked to both objects in a WBS or PERT and to documents represented in the Hypermedia tool. A developer
will thus have hyper-media access to requirements documents, issues, scheduling information, WBS and PERT information from his or her desktop. Eventually, a requirements assistant will be added to facilitate the acquisition and representation of user requirements.

A. Summary of the Program Management Facet

The PM facet supports hypermedia access to WBS and PERT diagrams, documentation flows, and associated issues. We are planning to extend the PM facet with a more powerful scheduling facet; initially we will use a commercial scheduling tool, and later replace the commercial tool with a tool developed using the Formal Development facet.

IV. SPECIFICATION ELBAORIZATION

Argo, formerly known as Module Specification Language Environment, is a C++ based language used to represent both problem and solution space class structures [And95]. Argo supports the definition of preconditions and postconditions for individual methods, and supports the definition and use of class invariants. These elements have an execution-based semantic, and can be used to generate either guarded statements or embedded comments.

The Argo Language Environment (ALE) is a graphical environment in which Argo classes, objects, and relationships can be defined. The interface to ALE is depicted in Figure 5. After selecting a task from the WBS or PERT representation of a project, the Task List window of ALE is updated to reflect any unfinished tasks and subtasks. For each task in the Task List window, potential resolutions are given in the Task Resolution window. Task details can be viewed by pressing the Task Details button, and a task resolution option can be selected by highlighting the option and pressing the Execute button.

Task resolution within ALE is currently restricted to one of the following three options:
1. Develop package diagrams,
2. Generate package code, or
3. Create an executable prototype.

The above options are further restricted in that their availability is context dependent. For example, a user cannot attempt to generate package code without first developing package diagrams.

Package diagrams in ALE provide a means of abstraction. Entities in a package diagram represent individual object models (in the Rumbaugh sense [RBP91]). Using the Package Diagram (PD) tool, relationships between packages can be defined, and critics — which analyze the design — can be invoked. Errors discovered by the critics are converted to tasks and inserted into the Task List window for later resolution. Thus the PD tool can be used to create a high level view of the structure of a project.

Double clicking on a package identified in a package diagram brings up the Class Diagram Editor (CDE). The CDE, shown in Figure 6 is a graphical editor.
used to define object models. Like the PD, the CDE contains a collection of critics which analyze a class model for such things as incompletely or incorrectly specified associations, violations of inheritance rules, etc. As is the case with the PD critics, errors discovered by the CDE critics are converted to tasks which are then inserted into the Task List window. An editor linked to the CDE can be used to modify Argo representations of object model entities.

After creating a package diagram and associated object models, and after satisfying the critics, ALE can be used to generate an executable prototype. It is through ALE that method preconditions and post-conditions — as well as class invariants — are used to generate guarded statements or embedded comments: the choice is left to the developer.

Argo specifications have an execution-based semantic, so the generation of code from a diagram like that of Figure 6 is somewhat limited. Specifically, ALE will generate the appropriate C++ header files and will create stubs for each defined operation or method. ALE can decorate these stubs with comments describing the pre and post conditions of the stubbed method, but is unable to use the pre and post conditions to generate executable code. We are exploring the possibility of using a denotational semantic for defining method behavior. If successful, the formal specification facet could be used to generate executable code for the methods defined/referenced in the ALE object diagrams. The Formal Specification facet is described in greater detail in the following section.

V. FORMAL SPECIFICATION

The formal specification (FS) facet of KBSA consists of a number of related tools and technologies, including the following:

- Algorithm specialization as defined by Smith (e.g., [Smi83], [Smi92], [SL90]) and implemented in the KIDS system [Smi90], could be used to develop algorithms for Argo class methods.
- Process and temporal logics could be used to define safety and liveness properties, and used to define inter-class and inter-object communication and synchronization. Such logics would also permit the definition of multi-threaded objects.
- Petri-Net simulators [DAE98] could be used to investigate dynamic properties of specified applications.

We are actively pursuing the definition and implementation of a specification environment based on SPECWARE [SJ95] in which functional (stateless) specifications are used to define data types and operations, process-based specifications are used to define
communication and synchronization, and temporal specifications are used to define safety and liveness properties. We plan to integrate this specification environment into the KBSA system to provide a mechanism to formally specify and investigate application properties. For example, the communication network of an application — defined by process specifications — can be used to investigate method pre-condition satisfaction. Specifically, if the pre-conditions of a method are not satisfied by the post-conditions of the methods from which it obtains its data, then derived antecedents can be used to define data conditioning operations. The point here is that formal specification of methods and communication structures permits semantic as well and syntactic analysis of class structures.

The specification environment described above will be coupled with ALE so that changes made in one diagram will be reflected in another. For example, adding a method definition to an object in an ALE diagram will result in the creation of a new functional specification incorporating the extended signature.

Additional plans for the future of the KBSA system are discussed in the following section.

VI. SUMMARY AND FUTURE WORK

The KBSA ADM is expected to be completed in mid-1997 and will consist of the integration of enhanced versions of the above described capabilities along with commercial project management, graphical user interface and object base management components. Third party research is already underway to add “critical properties experts” in the extra-functional areas of user interface design, security, and fault tolerance (the ability to automatically generate distributed, fault tolerant implementations having already been demonstrated). Enhancements in requirements acquisition capabilities will include extension of the Hypermedia facet and the incorporation of scenario generation capabilities to increase the involvement of end users. The conclusion of the SPECWARE implementation effort, also mid-1997 will yield a specification environment capability which must be merged with the ADM to provide the high assurance and optimization capabilities originally promised by the KBSA program. This will also lead to the investigation of formal hardware/software co-design where the allocation of functionality to hardware or software is the result of a formal transformational synthesis process. In the long term, as a paradigm for evolutionary development and support of systems, the KBSA program is expected to contribute to and benefit from joint sponsorship of the ARPA Evolutionary Development of Complex Software initiative.

REFERENCES


Captain Mark J. Gerken has been working software issues within the DoD for approximately seven years. After receiving a B.S. degree in computer engineering, Capt Gerken began his career by working software acquisition and supportability issues associated with a variety of programs, including the AMRAAM, ATF, and the C-17. After working software acquisition, Capt Gerken enrolled in the Air Force Institute of Technology(AFIT) where he studied software engineering and computer graphics. After receiving his masters degree, Capt Gerken joined in the Ph.D. program at AFIT where he studied formal specification techniques. After successfully completing and defending his dissertation, “A Formal Foundation for the Specification of Software Architecture”, Capt. Gerken joined the staff at Rome Laboratory.

Nancy A. Roberts has been a member of the technical staff at Rome Laboratory for the past seven years and is presently assigned to the Knowledge Engineering Branch. Her present responsibilities include testing and evaluating software, and managing efforts related to the Knowledge Based Software Assistant. She is currently working on her masters thesis in Computer Science in the area of formal specifications at SUNY Institute of Technology.
Douglas A. White has been a member of the technical staff at Rome Laboratory since 1971 and is presently assigned to the Knowledge Engineering Branch. His present responsibilities include the role of program manager for the Knowledge Based Software Assistant program. Prior to assuming his responsibilities in the Knowledge Engineering Branch, Mr. White worked in the area of languages and compilers, serving as the DoD representative to the Vice Chairman of the ANSI Technical Committee standardizing the language PL/1, and providing technical assistance to the Air Force representative to the DoD High Order Language Working Group during the early days of DoD-1/Ada definition.