

AD-A256 254



HL-IH-92-163
Final Technical Report
June 1992



2

AN ASSESSMENT OF KBSA AND A LOOK TOWARDS THE FUTURE

The MITRE Corporation

Melissa Chase and Howard Reubenstein

DTIC
ELECTE
SEP 30 1992
S B D

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

235050

92-26108



38p4


Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

92 26 108

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-92-163 has been reviewed and is approved for publication.

APPROVED:



DOUGLAS A. WHITE
Project Engineer

FOR THE COMMANDER:



JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CA) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE June	3. REPORT TYPE AND DATES COVERED Final Oct 89 - Sep 90	
4. TITLE AND SUBTITLE AN ASSESSMENT OF KBSA AND A LOOK TOWARDS THE FUTURE			5. FUNDING NUMBERS C - F19628-90-C-0001 PE - 62702F PR - 5581 TA - 27 WU - 53	
6. AUTHOR(S) Melissa Chase and Howard Reubenstein			8. PERFORMING ORGANIZATION REPORT NUMBER KES U.91.7	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) The MITRE Corporation 202 Burlington Road Bedford MA 01730			10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-92-163	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (C3CA) Griffiss AFB NY 13441-5700			11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Douglas A. White/C3CA (315) 330-3564	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The Knowledge-Based Software Assistant (KBSA) project has concluded its first research phase and is moving on to the second phase. This report describes a review conducted of: the short-term goals (as set out in the initial 1983 KBSA report) and how well they have been met, whether the mid-term goals should be reevaluated and how to proceed towards these mid-term goals. To facilitate integration of the various KBS facets, and to make effective use of available funding, reusable capabilities and supporting technologies within the KBS project need to be identified and pursued. This report proposes that this identification can be made from a knowledgable position now that progress has been made on most of the required facets.				
14. SUBJECT TERMS Knowledge-Based Software Engineering, Software Engineering, Automatic Programming, Formal Specifications, Artificial Intelligence, Project Management			15. NUMBER OF PAGES 44	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT U/L	

TABLE OF CONTENTS

SECTION	PAGE
1 Introduction	1
2 Phase 1 Goals: Have they been met?	3
2.1 Framework	3
2.1.1 KBSA Framework	3
2.1.2 Activities Coordinator	4
2.2 Project Management	5
2.3 Requirements	6
2.4 Specification	8
2.5 Development / Performance	9
2.5.1 Development	9
2.5.2 Performance	10
2.6 Phase 1 Summary	10
3 Phase 2 Goals: Are they still appropriate?	11
3.1 Concept Demonstration	13
3.2 ARIES	15
3.2.1 Requirements	15
3.2.2 Specification	16
3.2.3 Directions for ARIES	16
3.3 Development / Performance	17
3.4 Project Management	19
3.5 Phase 2 Summary	20
4 The Final Phase	21
5 1 AK	23
6 Reusable Capabilities and Products	25
7 Missing Technologies	27
8 Future Plans	29

SECTION

List of References

31

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification _____	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

DTIC QUALITY INSPECTED 3

SECTION 1

INTRODUCTION

In the initial Knowledge-Based Software Assistant (KBSA) report [1], a three-stage development plan for constructing a KBSA is set out. In the first phase, individual facets (each corresponding to a lifecycle activity) are to be developed and an overall framework is to be defined. In the second phase, systems integrating several facets are to be demonstrated and more advanced versions of individual facets are to be developed. In the final phase, a full KBSA, integrating all facets, is to be constructed.

In addition to this overall development plan, the report envisions five stages of development for each facet:

- database stage - the contents and properties of the various artifacts involved in a software development are represented in a central repository to facilitate computer-based access and querying of the information. The initial phase of the KBSA was to be bootstrapped into the software development process by serving primarily as a documentation tool.
- inference stage - inference rules are represented and used to automatically fill-out the database and answer queries.
- action stage - development activities can be represented, recorded, manually applied and replayed.
- planning stage - the KBSA automatically invokes certain activities to achieve development goals.
- knowledge acquisition stage - tools are created to facilitate the acquisition of domain models required for intelligent KBSA processing.

The actual history of the KBSA program is close to the overall three-phase development plan, while the construction of individual facets has diverged from the five-stage development vision.

Phase one research has made the most progress on developing inference stage capabilities (e.g., KBRA constraint management, limited classification, contradiction detection) and action stage capabilities (e.g., KBSpecA high-level editing commands) while providing basic required database stage support. In retrospect, leap-frogging the robust database repository stage was a natural occurrence since blindly representing software artifacts would have been less effective without a clear understanding of the kind of processing

the database needed to support. Thus, the research has first explored techniques for intelligent process assistance.

To support programming-in-the-large and achieve more relevance to current practice, the database stage needs to be made more robust. This will facilitate KBSA integration by forcing the development of shared database meta-models. By becoming a repository for a realistic amount of software development information, the KBSA will be able to be integrated into current development climates and cause renewed attention on the *assistant* aspect of KBSA. This will also provide a forcing function to address multi-user, cooperative work issues.

SECTION 2

PHASE 1 GOALS: HAVE THEY BEEN MET?

The general goals for the first phase of the KBSA were to develop and demonstrate individual facets. Prototypes of most facets, excepting testing and documentation, have been created and demonstrated. Concurrently, a KBSA framework was to be developed. Framework design, accompanied by some prototyping, was also carried out during the first phase.

It was also expected that guidelines and standards to facilitate facet integration would be developed. Facet development has been on an individual basis and only in the second phase are these issues being considered.

We now consider the short-term goals for the framework and the individual facets.

2.1 FRAMEWORK

In the KBSA report, the framework is envisioned as the basis for the development and integration of the facets. It consists of an activities coordinator, a knowledge-base manager, a wide-spectrum language, and a set of utilities (e.g., user interface, inference engine). One of the early KBSA contracts was the KBSA Framework effort, which attempted to look at the entire framework envisioned in the report. A later contract focused on developing a formalism for activities coordination.

2.1.1 KBSA Framework

The initial thrust of the KBSA Framework was the design of a framework architecture [2] and a prototype to demonstrate these architectural ideas [3]. The proposed framework provided a loose integration of facets within a distributed object management system. The framework consisted of a central knowledge-base and activities coordinator, a client handler for each facet (to handle communication between the facet and the knowledge-base), a user interface, an inference engine (to help handle the data objects), and a wide-spectrum language. Each facet has its own framework and wide-spectrum language.

In the KBSA Framework prototype the knowledge-base and some activities coordination was handled by a "home-brew" distributed object management system developed in CLOS. The inference engine was provided by LogLisp, the wide-spectrum language by

ObLog (essentially a loose combination of CLOS and LogLisp), and the user interface by Emacs modes.

After the initial definition and prototyping, work continued on developing a graphical user interface and defining requirements for the configuration management needs of KBSA. As part of the interface work, KUIE (KBSA Use Interface Environment), an X Windows toolkit, was built on top of the CLUE and CLIO toolkits.

The initial framework design did not have a significant impact on the rest of the KBSA program, although the lessons learned during the project were important. The Framework contract helped force agreement on low levels of integration (i.e., adoption of Common LISP and X Windows), suggested areas of future effort (e.g., configuration management), helped prod the two major developers to think about integration at the level of specification languages, and suggested that loose coupling of facets would not be adequate for building an integrated KBSA.

In retrospect, it appears that it was premature to expect significant framework development so early in the KBSA program. First, underlying technology, such as commercial object-oriented database management systems and high-level LISP-based X toolkits, was not available. Second, the requirements of individual facets were poorly understood, making it difficult to develop an information model to facilitate integration. Finally, the new software development process itself was poorly understood; only towards the end of the first phase was research in formalizing activities coordination (which can provide the basis for specifying this process) carried out.

Surprisingly, much of the fundamental research in defining a KBSA framework took place outside the KBSA Framework contract. The major facet developers defined their own wide-spectrum languages and considered issues of compatibility and integration (spurred on by the needs of the Concept Demonstration). They also developed their own inference engines and knowledge-based repositories.

2.1.2 Activities Coordinator

The KBSA report singled out the Activities Coordinator as the novel component of the KBSA framework and support system. The report recognized that in a large software development effort, coordinating the activities of agents (both human and software) would be central. It called for a formalization of such coordination through developing a language to describe types of coordination and an interpreter of that language, which could then monitor project development in compliance with the policies described in that language.

The line between the Activities Coordinator and the Project Management facet was not clearly drawn in the KBSA report. As described below, this led to the Project Management facet developing its own coordination and communication protocols.

Late in the first phase a contract was awarded to develop a genuine formalism for activities coordination. The formalism developed under this contract is called "transaction graphs." Transaction graphs provide a notion for describing activities. A transaction graph is an undirected graph with largely independent computations at the nodes (activity descriptions) and transactions along the arcs (corresponding to the interactions of the activities). The visible state of computations can be controlled (which provides the basis for presenting information to the user) and there are various operations to compose, condense, and expand graphs. These graphs are executable specifications. A demonstration was developed: an election was defined using transaction graphs and this specification was executed. Although the current mechanism for defining transaction graphs is crude (there is no graphical facility), the formalism itself looks promising as the basis for KBSA coordination.

2.2 PROJECT MANAGEMENT

The short-term goals for the Project Management facet were to develop a formalism for project management knowledge, use this formalism to build a knowledge base of project tasks, develop a message handling capability for communicating about task assignments and progress, and extend the formalism to include project management procedures and inference capabilities (for task tracking).

The initial Project Management Assistant (PMA) project was one of the early KBSA contracts and ended prematurely owing to budget cuts [4]. The work was later picked up under the KBSA PMA for Ada Systems contract [5].

The PMA made significant strides in formalizing project management knowledge: an ontology of project management concepts was created (i.e., structural models for products and tasks), a formalism for temporal reasoning was developed, and steps were taken towards formalizing the construction of structural models. In the second PMA prototype, the knowledge-base of project tasks was made persistent both by a facility for saving and loading project models and through use of the Software Lifecycle Support Environment (SLCSE) relational database. A message handling capability was demonstrated in the context of problem reporting. Finally various decision support procedures were implemented: cost estimation, scheduling, resource allocation, and progress monitoring. Two of the most novel features are the PMA's ability to evolve consistent task and prod-

uct structures, and the ability for managers to define policies which are then monitored automatically by the PMA.

As with the Framework, the PMA suffered from the lack of the Activities Coordinator. According to the KBSA report, the PMA is supposed to use "the coordination and message handling capabilities of the activities coordinator to carry out its work." Since the PMA effort preceded the Activities Coordinator research, the PMA had to develop some of these low-level capabilities instead of focusing solely on the higher-level policy management issues.

2.3 REQUIREMENTS

The short-term goals for the Knowledge-Based Requirements Assistant (KBRA) included developing a formal requirements language. A simple requirements ontology whose basic concepts include: activity, event, data, transition, and constraint was developed. The thrust of the KBRA was exploring how computer-based support could be provided for the process of transforming initial informal requirements descriptions into formal representations based on both the above ontology and an underlying reasoning system that provides semantics to the ontology.

To realize the goal of acquiring informal descriptions and representing them more precisely, the KBRA put emphasis on supporting multiple presentation input types with a corresponding underlying multi-paradigm representation. The presentation types supported by the KBRA include:

- A notepad which was meant to serve as an on-line working notebook for the capture of the most informally formulated requirements. This notebook supported keyword recognition to help tie these informal statements into the other presentations.
- Various diagramming capabilities including: context (interface) diagrams, system function diagram, internal interface diagram, functional decomposition diagram, and data flow diagrams.
- A spreadsheet used in capturing non-functional requirements, e.g., performance parameters.
- A state transition diagram.
- Activation tables.
- A requirements document output presentation.

The above multiple views were integrated via a common underlying hybrid representation system called SOCLE. Among the capabilities SOCLE provides, one key integration feature is contradiction detection and explanation which is highlighted in the spreadsheet presentation. When an analyst enters parameters into a spreadsheet that are inconsistent with the underlying constraints, SOCLE detects this and assists in correcting the problem.

The multiple presentations are integrated into the SOCLE system via a recognizer and presenter architecture. Each presentation translates itself into a set of underlying SOCLE constructs. Further, each presentation type can extract (recognize) the necessary data to reconstruct an instance of the presentation from the SOCLE knowledge-base. In this way, data can be shared across multiple presentations.

The KBRA did not aggressively explore knowledge reuse. Components were generally available for reuse simply via cut and paste. Certain components were explicitly hooked into a decision table construct which permitted a form of classification to instantiate more specific component instances when appropriate (according to the rules of a selection decision table).

Another short-term goal for the requirements facet was intelligent editing and managing of requirements. The presentation interface supports this goal by providing for initial data acquisition. No strong capabilities to support a review process were provided. Document generation is one tool to assist in running a review process, however, the requirements document is a generated output presentation and not editable in its own right.

No support was developed for the short-term goal of requirements testing. Testing in general has not been strongly addressed in Phase 1.

An evaluation of KBRA [6] indicated that this first phase tool had limitations to be expected in a research prototype. One comment, however, which needs to be addressed by more than a natural evolution of the KBRA functionality is the observation that KBRA supports no methodology (or therefore all methodologies). This appears to be even more the case in KBRA because of the support for multi-paradigm representation. In addition to general methodological guidance, the KBRA will need to incorporate guidance as to which presentations are most appropriate for particular kinds of information.

2.4 SPECIFICATION

The short-term goals for the Knowledge-Based Specification Assistant (KBSpecA) begin with the creation of an executable specification language. An expressive wide-spectrum specification language GIST was created. However, GIST is not fully executable and therefore various sublanguages have been developed with more limited expressive power but better execution properties. Refine is one such language with reduced expressive power and complete compilability. Along with Refine and GIST (and SOCLE from the KBRA) other representational formalisms that have been developed and experimented with (by USC/ISI the KBSpecA contractor) include AP5 and LOOM. Consensus on a language inspired by the features of the above languages and representational formalisms has yet to be achieved. Strategies for integrating these formalisms have begun to be explored and are discussed in the context of ARIES.

The KBSpecA adopts the notion of transformational development and thus provides a set of high-level editing commands (HLEC) for transforming GIST specifications. HLECs embody different semantic actions that can be taken on a specification, e.g., adding a parameter, bundling relations onto a new type, and unfold invariants. HLECs have been classified according to their syntactic effects on various semantic representations (e.g., as splicing a link into a class hierarchy) and thus can be indexed and accessed according to this generic classification for easier application.

In concert with the development of the KBSpecA, a methodology for specification evolution by combining parallel elaborations has been developed [7, 8, 9]. This methodology encourages a separation of concerns while developing a specification allowing issues to be addressed independently. While work remains to be done on supporting the merging of these separately developed elaborations, this is a compelling methodology, one that should support reuse and specification maintenance.

The other KBSpecA short-term goals include specification wellformedness checking, specification testing, and a specification paraphraser. A GIST simulation capability is present and has been tied into a summarizer [10] that aids in understanding simulation traces. A static GIST paraphraser [11] has also proved successful. This paraphraser has experimented with the idea of annotations to capture ontological information that is important for understanding and presumably could be of use in compilation and execution.

2.5 DEVELOPMENT / PERFORMANCE

In the KBSA report, the functions of the development and performance facets are not sharply distinguished: the development facet is "to aid the creation of a production quality implementation" while the performance facet is to "help to create and maintain efficient programs that meet their performance requirements." The short-term goals for the two facets both differentiate between them and underscore the interrelationships. In terms of development history, both facets were awarded to the same contractor and the research and development of the facets show a strong synergy.

2.5.1 Development

Several of the short-term goals for the development facet were linguistic: to develop a wide-spectrum language (capable of representing a system design from formal specification to optimized implementation), a transformation language (capable of describing transformations from the abstract to the concrete constructs), and a property language (capable of describing the properties of program segments). In addition, the facet was to be capable of interactive mechanical development, i.e., apply development steps requested by the user, and be capable of proving properties as needed during development.

The prototype development assistant, KIDS [12], fulfilled these goals for a subset of programming concerns, combinatorial algorithms. The wide-spectrum language underlying KIDS is REFINE. Formal specification typically consists of developing a domain theory of types, functions, and laws (e.g., distributive laws) and then developing a specification in terms of that domain theory. The interactive development consists of applying algorithm design tactics (selected from menu) to the specification. The design tactics include simple problem reduction, divide-and-conquer, global search, and local search. This application essentially specializes an algorithm theory to create one appropriate to the given problem in the specification. Then optimization techniques can be applied, along with data structure selection. These are really a part of the performance assistant.

KIDS includes a deductive inference engine, RAINBOW II, which is able to reason about REFINE expressions. RAINBOW II is able to prove whatever properties are required during the development process (to support the application of algorithm design tactics and optimization).

2.5.2 Performance

The short-term goals of the performance facet were to conduct symbolic evaluation (to propagate efficiency estimates and perform symbolic analysis), provide data structure analysis advice, and offer program decomposition advice.

The prototype performance estimation assistant (PEA) achieved most of these goals [13]. The PEA operated on specifications in the PERFORMO language, essentially an extended subset of REFINE. PERFORMO is a single assignment functional language with a general iteration mechanism.

The PEA achieved the first two goals (and carried out others, which will be discussed later). Using the KIDS inference engine, the PEA propagates assertions about program segments and conducts a performance analysis that is useful for optimizations and data structure selection. The PEA is able to perform various optimizations: simplification, finite differencing, iteration (i.e., selecting an efficient way to realize the general iteration mechanism), and loop fusion. The PEA carries out semi-automatic data structure selection, that is, the user specifies the implementations of some objects and the PEA selects the implementations for the rest (based on the specified implementations, algorithm and data structure design knowledge, and the optimizations). The PEA does not offer modularization advice, but much of the modular structure of the program is created automatically by KIDS through refining the specification by applying tactics.

2.6 PHASE 1 SUMMARY

Two facets were not contracted out in Phase 1: the testing facet and the documentation facet. The short-term goals of the testing facet were to develop a test-case maintenance assistant. The short-term goals of the documentation facet were to get documentation on-line in a partially formalized representation. As discussed later, development of both of these facets will be necessary to achieve the goals of Phase 2.

In general, Phase 1 has succeeded in creating a research base of ideas and techniques that appear to be useful. These ideas need to be both elaborated and scaled-up. Elliot Soloway is probably on target, however, in saying that a great deal of effort must now go into inserting KBSA into an actual development cycle. Research on refining these issues should continue, but system integration, repository and domain model population, and interface issues are key to success of the next phase.

SECTION 3

PHASE 2 GOALS: ARE THEY STILL APPROPRIATE?

The general goals for the second phase of the KBSA are to continue facet development and produce a more integrated system. As discussed above, KBSA is ready to address these goals but must begin to address technology transfer issues [14].

A hallmark of the KBSA approach [15] to software development is that software maintenance is intended to be performed at the specification level. While this is a beautiful idea in theory, it needs to be closely examined. Maintenance at the specification level is an idea that seems to be initially predicated on the notion that software development can be divided up into a front-end and back-end process. The front-end process is semi-automated with large amounts of human input. The back-end process is a repeatable, automated process. Maintenance occurs by modifications made in the front-end process and an automated repetition of the back-end process.

Balzer's extended automatic programming paradigm (shown in [16] and figure 1) corresponds to the intended KBSA paradigm. In this paradigm there is a relatively clean front-end in which a specification is acquired, validated, and maintained. The back-end includes an interactive formal development of a high-level specification to a low-level specification, an automatic compilation process and a tuning process. We might also add an embedded systems integration process to the back-end. Note that this back-end process is far from automated. This would indicate that the astounding productivity gains that could be attributed to specification-level maintenance may not be forthcoming, however, dramatic gains may still be expected.

Productivity gains will be enabled due to a number of factors. Even though the back-end process is not fully automated, it can be instrumented to capture the development steps. Automated redevelopment can be replaced with semi-automated redevelopment and structured guidance to a (re)developer. Furthermore, the information captured in a tool assisted development process will provide invaluable documentation that will help alleviate the program understanding burden associated with maintenance.

Two areas of attention are indicated to realize these productivity gains. First, a clean separation of "compiler-like" processing and semi-automated design processing needs to be maintained. The goal of this is to define the frontier of compiler technology so that KBSA can work on pushing this frontier closer to processing the results of the front-end processing. Also, this could motivate looking at the area of non-standard compilation

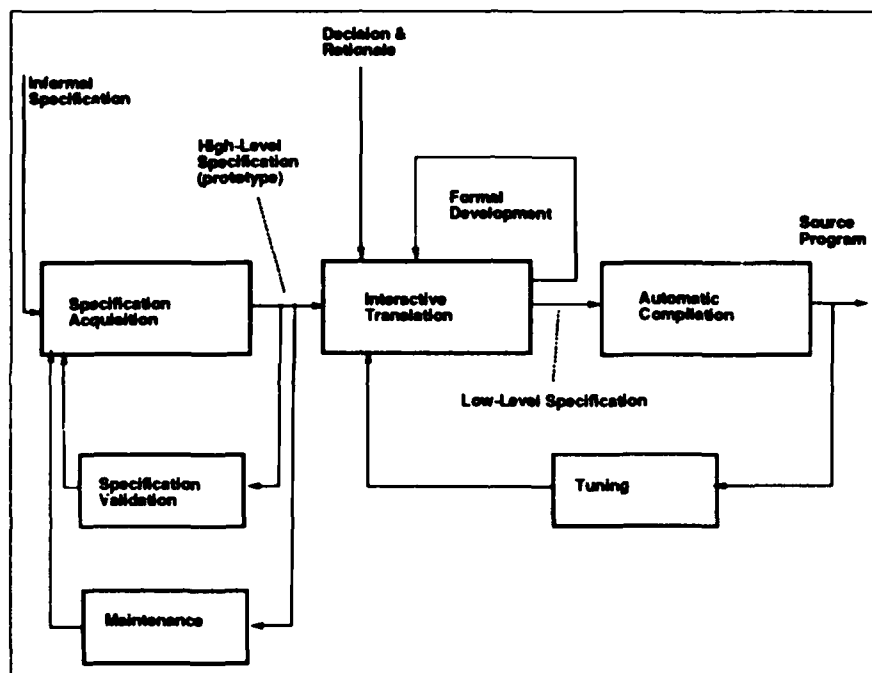


Figure 1. Balzer's Extended Automatic Programming Paradigm

processes, i.e., compilation processes that use more than the standard input of source code, e.g., design parameters such as time/space tradeoffs or degree of error checking required. Second, automated back-end rederivation strategies need to be developed in order to minimize the burden of the required redevelopment process. The difference between compilation and assisted rederivation comes in turn around time for maintenance activities. With compilation, turn around time is the time required for a system build, e.g., overnight. With assisted rederivation, turn around requires human intervention to make a potentially large number of decisions, however, the process itself is reliable in that the assistant is tracking potentially complex impact and interaction relationships.

3.1 CONCEPT DEMONSTRATION

The KBSA Concept Demonstration forces issues of facet integration, underlying knowledge representation integration, and to some extent system platform issues. The issue of common representation services is critical to the other integration issues. One approach would be to standardize on a knowledge representation and reasoning system for the entire KBSA system. This would be premature, however, especially since it is unclear that a single system could satisfy all reasoning requirements.

Currently, the successful use of meta-models to facilitate interface communication has been demonstrated [17]. This approach to hybridization of knowledge representation leads one to think about a facet server architectural model. A prerequisite for successful integration of capabilities in this fashion is publishing of ontologies and schemata that define the information captured in various software development artifacts.

Another reason for publishing ontologies and schemata is that it will permit access to the repository of information about a specific system development. This access is required for two important purposes (see figure 2). First, in keeping with the assistant notion, it is important that all artifacts under development eventually be editable by standard technology. This is a necessary requirement to permit human users to perform processing that is beyond the KBSA. Second, and perhaps more importantly, is that opening up repository interface standards will permit the use of reverse engineering tools to populate software artifacts. The use of reverse engineering will be critical in allowing the KBSA to be used on legacy software systems which in turn is critical to successful technology transfer.

In general, the KBSA program needs to think about capturing an initial vertical software development scenario of reasonable size. The data provided by this scenario will provide a test base for new tools and components. It can also provide a baseline for experimentation

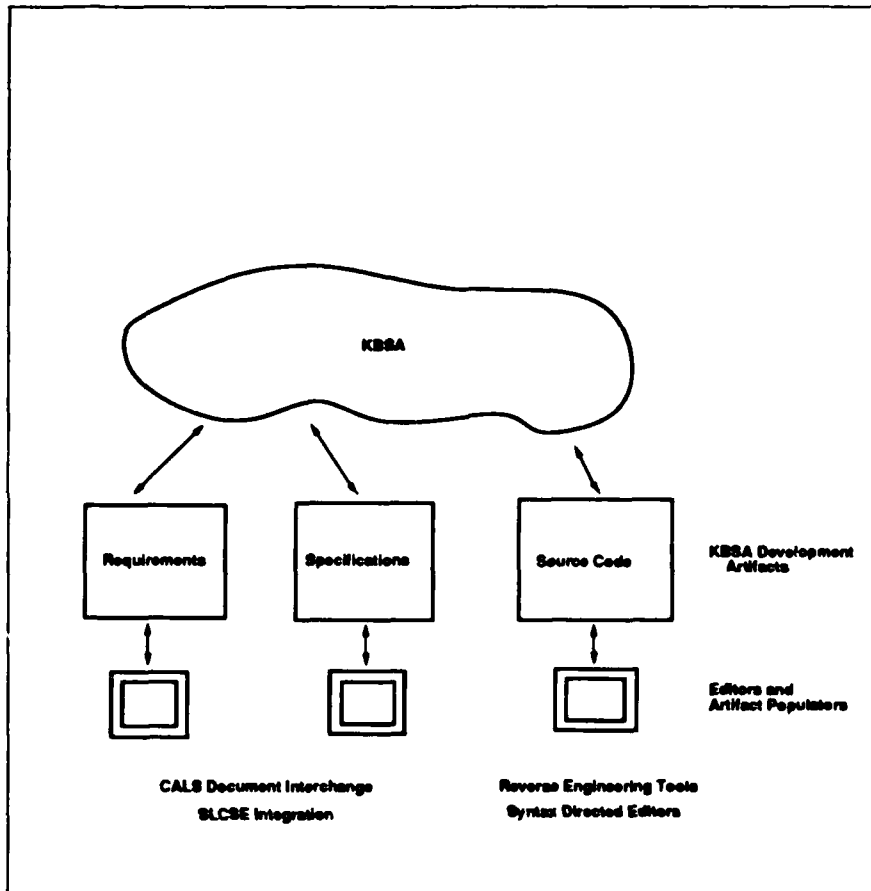


Figure 2. An Open KBSA System

among alternative facet capabilities.

Another issue which the Concept Demonstration will address relates to Abbott's critique of KBRA [6]. We believe that Abbott may have confused multi-paradigm representation with multi-methodology support. While the former is a positive (and almost certainly) necessary attribute, the latter can be confusing, especially as Abbott notes that support for many methodologies really amounts to support for no methodology. Currently, KBSA supports multi-paradigm representation and no particular methodology has been developed. The Concept Demonstration is addressing this lack of methodology problem [18] and should produce a candidate methodology for KBSA interaction. The candidate evolution transformation and refinement methodology must be explored, supported, and documented convincingly to be accepted and used in practice.

A crucial issue that needs to be addressed in the Concept Demonstration, though probably after the concept is elucidated for a single analyst/designer/programmer, is that of computer-supported cooperative work and concurrent access to development artifacts. Another advantage to opening up the KBSA repository is that this will permit addressing issues of concurrent access. Team support issues should be studied, particularly in the context of current CASE tools, however, the underlying goal of creating a coherent environment for a single user is difficult enough that it bears full attention.

3.2 ARIES

ARIES is a pair-wise integration of KBRA and KBSpecA. This section will discuss the mid-term goals for the requirements and specification facets separately, as they were laid out in the initial report. Then we will discuss implied goals and requirements for ARIES.

3.2.1 Requirements

The mid-term goals for the requirements facet included support for reuse of domain knowledge. The representation of such knowledge is being addressed in ARIES through the concept of folders. The notion of folders includes some ideas about a protocol for importing and exporting knowledge. The process for actually reusing (manually or automatically) such knowledge is in a weaker stage of development.

Another goal was support for the process of requirements transformation and refinement. By importing the notion of evolution transformations (i.e., ETs, an extension of the high-level editing command notion) from the KBSpecA, ARIES has a powerful representation formalism for requirements transformations. ETs have probably been the most successful

transfer of technology from the specification assistant. Work has progressed to produce a characterization of ETs in terms of their generic effects on a node/link representation of different kinds of semantic information. This characterization has proven useful in searching through and organizing the ET catalog. An impressive accomplishment is the integration of the presentation framework with ETs. A capability has been developed which permits a user to make graphical changes to a presentation, have ARIES translate these changes into a description of the generic kind of change made, and then have ARIES figure out an appropriate ET to apply in order to semantically achieve the graphical change [19].

Two mid-term goals which went unaddressed are the creation of an automated requirements walk-through system and of a requirements tutor. In general, KBSA needs to produce better tutorial material. The Concept Demonstration will partially address these needs.

3.2.2 Specification

The mid-term goals for the specification assistant were partially satisfied in Phase 1 and little additional work appears to have been required to further address them. Behavior explanation was implemented in the GIST behavior explainer [10]. Rapid prototyping is addressed partially by the high-level implementation language REFINER, but there is still not a good vertical connection from requirements to rapid prototype. The integration of KIDS into the Concept Demonstration will provide such a scenario. Verifying satisfaction of requirements, e.g., providing traceability from requirements to specifications, appears to have been somewhat neglected. For example, ARIES's ET library does not appear to have been augmented with a significant body of transformations that map (model) informal requirements constructs down to formal specifications.

3.2.3 Directions for ARIES

In the integration of the requirements and specification facets the process of design, specifically higher-level design, needs to be accounted for more explicitly. (Design can be divided into high-level and low-level design. KIDS assists in some of the low-level design chores.) High-level design involves: system and software architectural design, the incorporation of considerations due to non-functional requirements into the system structure, allocation of resources, and functional composition (i.e., assigning system functional components to satisfy functional requirements).

Requirements guide the evolution of specifications through a process in which the specifications become increasingly operational. The process is evolutionary and not a form of

direct compilation of requirements into specifications (and specifications into implementations). This process is also high-level design.

Part of the support for the design process requires providing a library of reusable specifications as targets for realizing system requirements. One decision of high-level design is an explicit modeling step where it is decided that a particular formal specification adequately models certain pieces of the requirement. Precise specifications are hard to write, but when written well are good candidates for reuse.

Another important piece of functionality to support this process of evolution is rationale capture. Rationale provides a context that defines why particular ETs were appropriate. Capturing this context is necessary to facilitate automated replay. One of the other pay backs for this capture is that it provides an additional source of requirements and domain knowledge.

ARIES provides some multi-user (not necessarily concurrent) support via the folder mechanism which permits capturing multiple viewpoints of the same problem. The higher level folders have also defined a common viewpoint, the ARIES metamodel, which permits integration of the KBRA and KBSpecA capabilities. Currently, folders are most like namespaces in that they provide a way to manage potentially conflicting viewpoints. More work can be done to address parameterization of folders to allow them to capture cliché/schema knowledge that has been identified by many researchers as critical to expert design/specification/requirements capabilities.

Support for the capture of requirements test scenarios for purposes of validation is another tool that might be integrated into ARIES's acquisition component. This capability should be integrated with the provisions for reuse, e.g., test scenarios associated with certain folder components can be reused when a folder is incorporated into a system description. Eventually, folders should propose important testing issues and test scenarios should be generated automatically to respond to these issues.

3.3 DEVELOPMENT / PERFORMANCE

Second stage contracts for the development and performance facets have not yet begun (although further work on performance is slated to start soon), but in some respects the initial contracts had already begun to address some of the mid-term issues.

One of the general mid-term goals is a tighter integration of facets. KIDS and the PEA are already integrated. Both are built on top of REFINE, share an inference engine

(RAINBOW II), use the same specifications, and share some of the same transformations.

Some of the more specific mid-term goals were also addressed by KIDS and the PEA during the initial contract. The mid-term goals for the development facet were to carry out automated development (create a sequence of development steps to achieve a simple goal) and to adapt previous developments through automated replay. KIDS carries out impressive semi-interactive developments, and work is proceeding on encapsulating development steps in larger strategies, which will allow the user to make a smaller number of higher-level decisions. KIDS is also able to carry out a limited form of replay (by carrying out syntactic matching) [20].

The mid-term goals of the performance facet were to augment domain models to cover performance analysis, analyze control structures and other optimizations, and provide performance advice for real-time systems. The PEA already uses KIDS's domain models to help in its performance analysis and performs some control structure optimizations.

Although KIDS and the PEA have made significant advances, they do not fully realize the KBSA mid-term goals for the development and performance facets in two significant ways. First, the programming problems that KIDS and the PEA address are relatively small. It is crucial to begin addressing larger problems in order to understand how these systems scale up, and to refine future goals if necessary. Second, KIDS and the PEAS use their own domain models (i.e., their domain theories and knowledge of algorithm design). The KBSA report appears to assume that there is a single domain model used by the entire KBSA, but in reality, the different facets have developed their own. Yet, there may be inferences that can be drawn from the specifications (and domain models) produced by ARIES that could yield the kinds of annotations that the PEA exploits in data structure selection. Merging, or at exploiting, these models is an issue to be explored.

Although KIDS and the PEA operate as a unified system, there is a separation of concern: the PEA is essentially an advanced compiler and KIDS turns specifications into algorithms. As we have discussed above, we believe this type of separation is appropriate for KBSA. To carry it out further, some of the human intervention that is needed by the PEA should be pushed further back into KIDS and its domain theories. That is, information needed to make inferences about bounds, containment, etc., should be supplied early in the specification process and the knowledge bases should be augmented to take advantage of this information (so the inferences to support optimization and data structure selection can be made). In this way, some of the initial data structure selections that are made by the programmer (and then propagated by the PEA) could be made by KIDS.

Finally, the development facet needs to begin addressing certain less algorithmic programming issues, e.g., computations with state, exception handling routines, interrupt driven programs, concurrency and security issues.

3.4 PROJECT MANAGEMENT

Work on the project management facet has not continued, so this discussion will differ from that of the other facets. The mid-term goals for this facet were to suggest simple management decisions (requiring weighing of evidence and detailed models of tasks, but still a localized decision process), to generate or modify plans and procedures, and to allow simple knowledge acquisition.

As discussed earlier, the Project Management Assistant suffered from the lack of a genuine Activities Coordinator. This meant that there was no way to develop executable specifications that could carry out the coordination activities implied by project management policies, nor was there a good formal language for specifying software process and management policies. The work on configuration management, which will use the activities coordination language described above, will provide a good test for this language. If it is successful, this language could provide the basis for specifying management policies and the automatic generation of appropriate events. For example, when a portion of code is completed, checking in a "released" version could automatically trigger updates to the project management knowledge base (that a milestone is completed) and send messages to the test team that the code is ready for testing. Handling this kind of communication in an automated way, linking the project management facet with the other KBSA activities, needs to be achieved before realization of the project management mid-term goals can be accomplished.

The first two mid-term goals essentially ask that the project management facet provide more automated decision support, and that is a reasonable goal. The last goal, knowledge acquisition, is concerned with developing project management policies and plans. When we have some sense of what is like to develop software under this new KBSA regime, we may see that there are different project management concerns than those we are used to. Managers may be concerned with new kinds of tasks, not concerned with others, have to think about different kinds of resources, and may structure projects very differently. Making it easier to define these policies, and plans for executing them and resolving difficulties, may involve developing new management primitives. Perhaps, given a specification of a software development process and various constraints, it might even be possible to automatically generate policies and plans.

3.5 PHASE 2 SUMMARY

The mid-term goals for the testing assistant involved knowledge-based test generation. Eventually, it was thought that testing would disappear as a separate activity and be redistributed into validation and development (correctness by construction). In the near-term, testing will be required as a form of validation and to verify integration of KBSA-generated code with existing systems.

The scope of Phase 2 appears to have been diminished due to funding constraints. As originally intended, Phase 2 would have broadened the capabilities of the original facets and addressed integration issues. Rather than staying in a predominately basic research mode, however, it is time to address hard issues of technology transfer. KBSA has certainly generated a wealth of ideas, the incorporation of which into current software development processes and tools could have significant impact.

SECTION 4

THE FINAL PHASE

In the initial report, the final phase of KBSA development was viewed as involving the installation of more intelligence and automatic processing into a KBSA that, at the end of Phase 2, was to be an integrated broad coverage system. Current plans for Phase 2 address concerns of integration. It is likely that most of the facets, by the end of Phase 2, will also already incorporate some sophisticated intelligent processing.

The tasks most likely to be emphasized in the final phase should concern integrating KBSA into current software development environments and developing at least one realistic system with KBSA and at the same time developing a set of domain models to support that development. In other words, KBSA is likely to be smart enough at the end of Phase 2. Technology transfer issues, breadth of knowledge, and multi-user interface will be most important.

SECTION 5

1 AK

What are the new problems that will arise in the first year after a KBSA (1 AK) becomes operational? There will be the obvious needs for continued support and adaptation, but at this point KBSA should have software product support.

The new problem introduced by the KBSA will be a shift from programming problems to knowledge acquisition problems. Knowledge acquisition problems are different than programming problems in that they are bounded by the properties and experience of the real world. In one sense then, these bounds (unlike those of general software development which is not governed or limited by any laws of nature) make the acquisition problem one which people are better equipped to deal with. It is also true that knowledge acquired regarding particular domains can become asymptotically more reliable as experience in the world is obtained and recorded. However, people will not be familiar with what is required, for example, to populate a KIDS domain theory. Training and methodology will be required to help people state the knowledge required to build systems under a KBSA approach particularly when they are used to thinking primarily about how to state solutions to the problem as opposed to describing the knowledge they used to arrive at those solutions.

As has happened before in the history of automatic programming, a successful KBSA will result in an altered definition of what it means to develop software and new tools will be required to automate this new process.

SECTION 6

REUSABLE CAPABILITIES AND PRODUCTS

In this section we discuss some of the products and technologies that have come out of the KBSA project. In the next sections we will discuss technology gaps and comment on future plans. We discuss what supporting technologies exist (or are implied by the current KBSA) that are general enough to be used in multiple facets? What intermediate products can be produced in developing the KBSA?

The clearest success of the KBSA project is the commercial spin-off from Kestrel Institute, Reasoning Systems, that produces the Refine and Refinery systems. It is interesting to note that while Refine was originally developed as a wide-spectrum high-level executable specification language, Reasoning has found its market niche by integrating Refine with parser-generator technology and producing a reverse engineering workbench and prototype reverse engineering systems for about six widely used languages. This is especially interesting in light of the fact that reverse engineering capabilities are not present in the current KBSA prototype systems or Concept Demonstration. A number of KBSA associates have undertaken work on their own to develop sophisticated reverse engineering capabilities. Reasoning has garnered many high-technology clients and it should also be noted that, as of this writing, it is highly probable that the 1912th CSG/TAC at Langley AFB will adopt Refinery as their reverse engineering platform for their Tactical Software Maintenance Environment (TSME).

The PMA has been integrated into the SLCSE environment [5] providing SLCSE with an appropriate project management schema and providing the PMA with a persistent repository.

Each facet produces a formalism, an ontology, of the part of the software development process and the artifacts it is concerned with. The Concept Demonstration has used meta-models for translation between representations. A meta-language for describing formalisms could be produced and representation of these meta-models should be encouraged. Certain formalisms could be agreed upon in a preliminary form to facilitate sharing of information between facets.

Production and release of these ontologies of process and product (e.g., like that embedded in the SLCSE database schemas) could facilitate the integration of facets and other tools into the KBSA framework. Furthermore, if tools treat these meta-models as

parameters and are data-driven from them, the resulting tools (e.g., intelligent browsers and paraphrasers) will be reusable for tasks different than that for which they are initially developed. These formalisms need to be related, however, to emerging DOD standards, e.g., CALS information exchange standards, the CIM (Corporate Information Model) technical reference model [21], and the PCTE reference model for framework of computer-assisted software engineering environments [22].

The general presentation-based interface may prove to be a separable and useful tool. The presentation interface should be abstracted and parameterized so that a new presentation (with edit, display, and knowledge-base update functionality) can be written in a modular fashion. Reasoning Systems has done just this in a more limited fashion with Dialect and Intervista.

The current KBSA architecture will require an object-oriented project database (OODB). Since the KBSA facets make strong use of knowledge representation and reasoning technology, this OODB will need to feed information into user local reasoning systems. It is neither practical or desirable to run these reasoning systems at a global project-wide level. Rather, users need to be free to work in their own local workspaces with the assistance of KBSA knowledge-based tools. As users work in their local workspace their product may be inconsistent with parts of the global state. These problems should not stop the user from developing their current product but need to be addressed when integrating local work with the global OODB. The architecture of such a distributed reasoning system would be an innovative development.

The front-end of the KBSA could stand alone as an intermediate product, a conceptual analyzer. (For use by people who, e.g., produce RFPs and review proposals.) This would provide a transferable technology short of requiring a fully integrated, vertical KBSA product. One requirement for such a tool is a fully available textual presentation for use in review. Support for commentary and editing would also be appropriate in order to hook into an actual review process.

SECTION 7

MISSING TECHNOLOGIES

It is appropriate, as integration efforts begin, to consider the requirements of multi-user KBSA support. Some of these issues begin to be addressed by work on configuration management [23]. However, as indicated above, more attention needs to be paid to defining the KBSA repository. Tools to support larger numbers of users must also address the problem that if the tool does not permit the users to do certain tasks then the users will circumvent the system, rendering the invariants under which the system is designed invalid. As indicated in figure 2 on page 14, KBSA must evolve an open repository approach so that users will be able to supplement KBSA capabilities with other tools and techniques and still be able to incorporate their work into a KBSA controlled development. This open repository approach will also facilitate facets transferring information among themselves. For example, as discussed regarding the development and performance facets, ARIES may have information that could be used to help make data structure selections decisions.

Multi-user support could very usefully include a comment consolidation and review tool. Concurrent review would be supported with PMA monitoring on top of an IBIS-like rationale model [24, 25]. The tools would support on-line negotiation and debate. This would also facilitate acquisition of decision rationale.

In order to develop realistic tools, realistic data must be available to support experimentation. This requires reverse engineering an existing system to populate the KBSA repository for an already fielded system. Acquiring this data by bootstrapping a KBSA system into an actual initial software development is going to be hard. The system must undergo a vertical integration and provide realistic tools in each facet in order to be accepted. Further, given the software backlog and maintenance problems, there may be much more demand for maintenance environments than new development environments. Work on recognition and systems to allow assisted "parsing" of existing artifacts (e.g., SRSs, design documents) will facilitate reverse engineering. KBSA needs to include a reverse engineering research component.

Documentation and testing facets should be developed, e.g., for assistance in maintenance. Humans will have to write code and integrate it with machine written code. A documentation facet can help determine where and how to perform this integration. Since the entire development is no longer correct by construction (and since blind-confidence is

not necessarily appropriate), testing of the resulting code will be required. In order to address current needs, provisions for managing test suites and analysis tools to develop test cases (perhaps based on specification information since the specification defines various input spaces) should be provided.

Knowledge acquisition tools will need to be developed to populate KIDS domain theories, ARIES folders, and other representations. The knowledge required to populate these representations, above and beyond basic problem domain knowledge, may be implicit problem solving knowledge that will be hard to elicit from KBSA users. An opportunity to develop sophisticated acquisition tools exists, however, since this knowledge is being used in the context of knowledge-based tools which can potentially provide assistance regarding the kind of knowledge they expect or that would be useful in a particular problem solving situation. For example, in an application of KIDS it may be possible to observe that if KIDS could prove a particular theory then it could finish a derivation. Explaining this need to a user and allowing them to provide information to help prove the theory would be a powerful elicitation strategy.

SECTION 8

FUTURE PLANS

In the proceedings of the 6th KBSA conference (actually KBSE-6) a number of papers by various members of Andersen Consulting's CSTaR Lab [26, 27, 28, 29] outline some visions for the future of KBSA. While many of their comments are well taken, we also wish to provide a different perspective regarding some suggestions.

One observation that is made about KBSA is the risk of being eclipsed by CASE technology (due mainly to an all or nothing vision of KBSA). To avoid this, it has been argued that KBSA technology should be inserted into CASE environments filling holes in these environments. This is called a KBSA "point-solution" approach where KBSA technology can access and store data in a CASE repository. An alternative approach is to have a KBSA repository with CASE point-solutions that can access and store data into the KBSA repository.

We would argue for the insertion of CASE into an open KBSA framework so that: CASE functionality does not have to be duplicated and so KBSA takes advantage of the familiar feel and capability of CASE but with the potential to automate and assist. If the KBSA framework was open and supported use of current tools to populate representations, then we could have incremental adoption of KBSA functionality. CASE repositories are limited and cannot accommodate the vision of KBSA program development and specification maintenance, i.e., KBSA requires a broader range of information than CASE tools. KBSA development will be limited if it is designed to fill gaps in CASE technology. Putting CASE into KBSA on the other hand means KBSA can take advantage of the advancing industry-state while continuing to push the state of the art.

Another underlying theme of commentary has been towards adapting KBSA to accommodate more traditional management information system (MIS) needs. KBSA should be careful not to lose its more general systems approach and become a system for supporting MIS development for two reasons. First, there are already huge market forces directing attention at this problem. Second, the P/D goals/needs indicate a need for help in real-time system development which requires different kinds of assistance (e.g., focus on process versus data) and which is being neglected in the marketplace partially due to the difficulty of the problem.

As we have previously discussed, scalability issues to support programming in the large (PIL) certainly need to be addressed. This implies support for group cooperative work.

When thinking about PIL, one is also led to consider system versus pure software needs and KBSA needs to be extended to take more general system problems into consideration, particularly the need to reason about system/software architectures.

Reverse engineering has been an indirect success story of the KBSA project. Building on KBSA technology, many good results in reverse engineering have been achieved. However, these results must now be integrated back into the KBSA system. For example, some amount of code will still have to be written by hand and integrated into the (semi)-automatically generated KBSA code. Therefore, both sets of code must be navigable by reverse engineering/program understanding tools. Furthermore, large budgetary outlays are going towards system maintenance activities. KBSA needs to create a gradual revolution by providing support to installed systems and gradual conversion to a KBSA paradigm based software system. In other words, KBSA should not neglect the problems of program maintenance.

Finally, the observation has been made that KBSA is incompatible with current practice. This is true but it should not be an a priori reason for dispensing with the KBSA approach. Current practice is not unassailable. KBSA is a revolutionary approach and so it should not be surprising that it might require a change in the way business is done. What needs to be balanced is the risk of the cost and acceptance criteria (e.g., retraining can be hugely expensive) versus the large potential gains.

LIST OF REFERENCES

1. C. Green et al. Report on a knowledge-based software assistant. Technical Report RADC-TR-83-195, Kestrel Institute, 1983.
2. S. Huseth et al. KBSA framework (phase 1). Technical Report RADC-TR-88-204, Honeywell Systems and Research Center, 1988.
3. S. Huseth et al. KBSA framework users manual. Technical Report , Honeywell Systems and Research Center, 1988.
4. R. Jullig et al. KBSA project management assistant. Technical Report RADC-TR-87-78, Kestrel Institute, 1987.
5. M. Daum and R. Jullig. Knowledge-based project management assistant for ADA systems. Technical Report RADC-TR-90-418, Kestrel Development Corporation, 1990.
6. D. Abbott. KBSA's requirements assistant and aerospace industry needs. In *Proceedings of the 4th Annual Knowledge-Based Software Assistant Conference*, 1989.
7. M. Feather. Constructing specifications by combining parallel elaborations. *IEEE Transactions on Software Engineering*, 15(2), 1989.
8. M. Feather. Specification evolution and program (re)transformation. In *Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference*, 1990.
9. M. Feather. Detecting interference when merging specification evolutions. In *5th International Workshop on Software Specification and Design*, 1989.
10. W. Swartout. The GIST behavior explainer. *3rd National Conference on Artificial Intelligence*, 1983.
11. W. Swartout. GIST English generator. In *2nd National Conference on Artificial Intelligence*, 1982.

12. D. Smith. KIDS: A semi-automatic program development system. *IEEE Transactions on Software Engineering*, September 1990.
13. A. Goldberg et al. KBSA performance estimation assistant: Final report. Technical report, Kestrel Institute, 1988.
14. D. Elefante. Knowledge-based software assistant technology transfer consortium: Status report 1. Technical Report RADC-TR-90-414, RADC/COES, 1990.
15. R. Balzer, T. Cheatham, and C. Green. Software technology in the 1990's: Using a new paradigm. *IEEE Computer*, November 1983.
16. W. Sasso and M. DeBellis. A software development process model for the KBSA concept demonstration system. In *Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference*, 1990.
17. G. Williams and J. Myers. Exploiting metamodel correspondences to provide paraphrasing capabilities for the KBSA concept demonstration project. *Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference*, 1990.
18. W. Sasso and M. DeBellis. A software development process model for the KBSA concept demonstration system. *Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference*, 1990.
19. W. Johnson, M. Feather, and D. Harris. The KBSA requirements/specification facet: ARIES. *Proceedings of the 6th Annual Knowledge-Based Software Engineering Conference*, 1991.
20. A. Goldberg. Reusing software developments. In *Proceedings of the 4th Annual Knowledge-Based Software Assistant Conference*, 1989.
21. Draft. Technical reference model for corporate information management. Technical report, CIM, September 1991.
22. ECMA. A reference model for frameworks of computer-assisted software engineering environments. ECMA TR/55, ECMA, December 1990.

23. J. Kimball and A. Larson. A change and configuration management model for the KBSA framework. *Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference*, 1990.
24. B. Ramesh and V. Dhar. Representation and maintenance of process knowledge for large scale systems development. *Proceedings of the 6th Annual Knowledge-Based Software Engineering Conference*, 1991.
25. M. Lubars. Representing design dependencies in an issue-based style. *IEEE Software*, July 1991.
26. M. DeBellis, W. Sasso, and G. Cabral. Directions for future KBSA research. *Proceedings of the 6th Annual Knowledge-Based Software Engineering Conference*, 1991.
27. W. Sasso. Motivating adoption of KBSA: Issues, arguments, and strategies. *Proceedings of the 6th Annual Knowledge-Based Software Engineering Conference*, 1991.
28. G. Cabral and M. DeBellis. Domain-specific representations in the KBSA concept demo. *Proceedings of the 6th Annual Knowledge-Based Software Engineering Conference*, 1991.
29. W. Sasso. Encouraging the adoption of KBSE technology: What needs to happen first? (panel). *Proceedings of the 6th Annual Knowledge-Based Software Engineering Conference*, 1991.