Knowledge-based systems in the AD/Cycle environment

by D. M. Hembry

Knowledge-based systems technology is a branch of artificial intelligence that deals with the processing of knowledge, as distinct from other branches of artificial intelligence that deal with topics such as robotics, vision systems, and speech recognition. This paper describes how, over the last decade, knowledge-based systems have evolved into a viable technology for building commercial data processing applications, and how increasing attention has been paid to incorporating these applications into commercial data processing environments. A logical conclusion of this direction is the capability to build knowledge-based applications that are full Systems Application Architecture™ (SAA™) applications. As this conclusion is approached, a requirement emerges that the knowledge-based development process be integrated with the application development environment provided by the other SAA language and service components. The integrated environment must provide high customer productivity in the development of applications that use knowledge-based technology, and must support a spectrum of development scenarios, ranging from the most basic to those involving complex applications and large development teams. This paper explores how knowledge-based products can address these requirements by integrating their development facilities with AD/Cycle™.

IBM's AD/Cycle™ is directed at improving productivity in the development of Systems Application Architecture™ (SAA™) applications and the management and control of complex application development projects.

This paper describes the movement of knowledge-based applications into the mainstream of commercial data processing. It discusses the logical conclusion of this trend as knowledge-based applications become fully conformant to SAA. The trend and SAA conformity result in requirements for advanced knowledge-based development facilities that can be addressed by the integration of those facilities with a spectrum of AD/Cycle environments.

The term *knowledge-based systems*[1,2] distinguishes artificial intelligence software products that deal with knowledge about some problem domain from those that deal with other areas of artificial intelligence, such as robotics, vision systems, and speech recognition. Knowledge-based systems are tools for building applications that draw logical inferences from their stored knowledge of the problem domain. From its early days, knowledge-based technology was recognized as being suitable for building applications that capture the knowledge of a human expert and deliver that expertise through a computer dialog in a manner that emulates the expert. These are *expert*

systems. Some well-known early examples dealt with medical diagnosis[3] and geological prospecting.[4] To-

Today, expert systems are being constructed for a wide range of application types and industries.

day, expert systems are being constructed for a wide range of application types and industries.

Increasingly, knowledge-based systems are also being embedded in conventional applications that apparently provide no clue to the inferencing component hidden within them. Many problems are more amenable to the knowledge-based inferencing style of programming than to the conventional (high-level language) procedural style. An example of this is a module that deals with tax regulations, where the multitude of individual regulations are easier to encode (and subsequently maintain) as knowledge-based rules than as procedural code. Such an inferencing module that makes tax decisions might be embedded within a COBOL application programmed for a payroll. Embedding knowledge-based systems in conventional applications has enormously increased their potential in commercial data processing, and it has generated new demands to mold knowledge-based systems into a technology that can easily be utilized in that environment.

An overview of knowledge-based technology

Most knowledge-based systems are rule-based systems. The term *rule-based systems* describes a programming paradigm in which discrete pieces of knowledge are expressed in terms of an expression that states: WHEN some condition is true, THEN some conclusion can be drawn or some action performed. These rules are utilized under control of an inference engine which traces inference chains through these rules, either by starting with some known facts and exercising the rules in a left to right manner until all possible conclusions have been drawn (termed *data-driven inferencing*), or by starting with a goal (a fact to be determined) and exercising the rules from right to left until a set of known facts are identified that allow that fact to be concluded (termed *goal-driven inferencing*). Other, more complex forms exist, such as opportunistic reasoning, but these are less prevalent.

Initially the reason for all the excitement about rule-driven inferencing may not be obvious. The secret lies in that the rules provided to the knowledge-based application are distinct and, in a sense, are independent entities. The rules can be created individually, generally without regard for the sequence in which they will be chained to accomplish a given inferencing task. New rules may be added to represent new pieces of knowledge without worrying about their positioning relative to existing rules. This situation has been compared to building a spreadsheet[5] in which cells are defined by the builder as formulae dependent on other cells. The builder is aware of the logical dependencies between the cells being defined, but there is complete freedom to physically place the cells anywhere on the spreadsheet and to define them in any convenient order. The builder does not have to be concerned with execution order of the cell formulae since the spreadsheet software automatically sequences the execution. The rule-driven inferencing paradigm shares this characteristic and is frequently referred to as *declarative programming* to highlight its sequence-insensitive nature, compared with *procedural* (or *algorithmic*) *programming* where all the paths through the program are completely defined by the programmer.

Early knowledge-based products held the facts with which they inferenced as simple data elements. More recently, systems have introduced *frames* as a more flexible and powerful way to represent the factual knowledge about a problem domain.[6] Frames typically represent entities in the problem domain. A single frame contains a number of *slots* which represent the attributes of the entity and which are referenced by the rules. A slot is made up of some number of *facets* that identify the attributes of that slot, one of which is its actual value. Other facets may identify the default values, constraints, certainty factors, etc., that apply to the slot. The list of facets is not fixed, although certain facets are generally present.

Facets may also contain a reference to a procedural routine that is automatically invoked under certain conditions such as update, deletion, or reference. These routines, often called *demons*, can be used for

such purposes as maintaining referential integrity between frames. Slots may also reference a procedural routine, called a *method*, that may be invoked by sending messages between frames.

Frames support inheritance. A hierarchy of class and subclass frames may be defined so that slots and facets, both data and procedures, are inherited by child frames from parent frames unless overridden by that slot or facet at the child frame level. Parent frames are essentially abstract data types that function as place holders for the default slots and facets of frame instances at the lower levels. Inheritance links may be defined at definition time and at execution time.

The frame structure provides a knowledge-based application with a very high degree of flexibility in the data that can be stored to describe entities, the metadata stored to describe the data, and the relationships that can be represented between the entities.

The rules and the frame definitions in a particular knowledge-based application are generally referred to collectively as the *knowledge base*. This is the knowledge about a problem domain that, together with specified goals or a set of starting facts, is operated on by the inferencing engine of the knowledge-based system.

Knowledge-based applications in commercial data processing

Early commercial knowledge-based systems. Early knowledge-based tools were predominantly built in LISP and ran on specialized LISP workstations. Application development was carried out by an artificial intelligence (AI) professional with an in-depth knowledge of the tool and AI techniques. LISP skills were typically required, since the LISP of the knowledge-based tool implementation frequently showed through to the application. After the initial years of technological advance, the most successful of these tools reached a very high level of sophistication in terms of their support of leading-edge AI techniques.

However, the tools were notably lacking in their capability to integrate into the hardware, operating systems, languages, database, and transaction processing systems typically found in commercial data processing organizations. In addition, most tools lacked those qualities loosely called "industrial strength" qualities, such as sharability, usability (particularly by large development teams), data integrity, and recoverability, and a skill requirement that did not restrict their use to a small number of extremely highly trained AI professionals.

The result was that commercial applications at this time were mainly restricted to pilot studies and a small number of specialized applications where the financial leverage was strong enough to justify the specialized hardware and the high skills needed for implementation. The poor capability to integrate into commercial environments was typically circumvented by building expert systems for problem domains that were self contained, and that did not demand tight connection to existing commercial systems.

The movement toward knowledge-based integration

By the mid-1980s, the limited acceptance of knowledge-based technology in the commercial arena was sufficiently noticeable to prompt questions in the media as to whether the early promise of the field had been exaggerated. At almost the same time, the crisis was recognized by the knowledge-based system developers, and there was a collective effort to move the technology out of the advanced technology world into the more prosaic, but much wider, commercial environment. Several key vendors announced their intention to provide versions of their products to run on commercial mainframes. Several new products were introduced that were designed from the beginning to be integrated with commercial data processing. In addition, numbers of small but high-function expert system shells for personal computers (PCs) made their appearance. The effects of this change in direction are presently being felt strongly and are probably still gathering momentum.

The trend to ever-increasing integration can be seen in IBM's knowledge-based products.

Expert System Environment (ESE)[7,8] initially provided a self-contained environment for non-data processing professionals to build and consult expert systems through a terminal interface. The expected environments for such professional users were time sharing option (TSO) and conversational monitor system (CMS), and accordingly these were the environments in which ESE operated. ESE allowed the invocation of external routines written in high-level languages, and thereby allowed access to DATABASE 2™ (DB2™) or Information Management System (IMS) databases. From its inception, ESE provided a

good degree of integration into commercial mainframe environments.

Nevertheless, there soon emerged a strong user requirement to extend access to ESE knowledge bases from terminals to other applications, including applications running under the Information Management System (IMS) or Customer Information Control System (CICS) transaction management systems. Thus, a recent release of ESE includes an application program interface (API) which provides this access.[9] In addition, Expert System Consultation Environment/PC (ESCE is the consultation-time component of ESE) addresses another requirement, to widen the range of supported environments by allowing the delivery of ESE applications on a PC running PC/DOS.

KnowledgeTool™ V2,[10] on the other hand, instead of being designed with its own terminal interface (like ESE), was structured from the beginning to integrate efficiently into a high-level commercial language, Programming Language/One (PL/I). By providing a language that can be used in conjunction with PL/I, KnowledgeTool V2 has enabled users to mix data-driven inferencing with the full procedural programming power of the PL/I language in any application on a System/370 or Application System/400® (AS/400™) platform. A KnowledgeTool V2 application can use any SAA Common Programming Interface (CPI) services that are available to PL/I, such as Structured Query Language (SQL), as well as other interfaces such as CICS command language statements, Graphical Data Display Manager (GDDM™), Interactive System Productivity Facility (ISPF), and IMS calls. Through the interlanguage communication capabilities of PL/I, KnowledgeTool V2 can call and be called by application modules written in C, COBOL, FORTRAN, Pascal, and assembler language. KnowledgeTool V2 applications will run under TSO, VM/CMS, IMS, CICS, NetView™, and Operating System/400™ (OS/400™).

The marriage of a data driven inferencing capability and a powerful commercial high-level language in KnowledgeTool V2 broke new ground in integrating knowledge-based technology into the business data processing environment. Nevertheless, there are requirements to make this function available on the Operating System/2™ (OS/2®) platform, as well as to enhance the knowledge-based function provided by KnowledgeTool V2, for example, to include goal-driven inferencing and a closer integration of rules with frames.

In December 1989, IBM released IBM Prolog for 370,[11] a replacement for the existing IBM VM Programming in Logic (VM/Prolog). This new Prolog version, which supports the Edinburgh Syntax as well as the IBM Prolog syntax with powerful new extensions and very high performance, will execute under the Virtual Machine/System Product (VM/SP) or Virtual Machine/Extended Architecture (VM/XA™) operating systems. It may call and be called from other application modules written in C/370, Restructured Extended Executor Language (REXX), VS Cobol II, VS Fortran, OS PL/I, or Assembler. In addition it provides an interface to Structured Query Language/Data System (SQL/DS™) for relational database function, and GDDM and ISPF to manage screen displays.

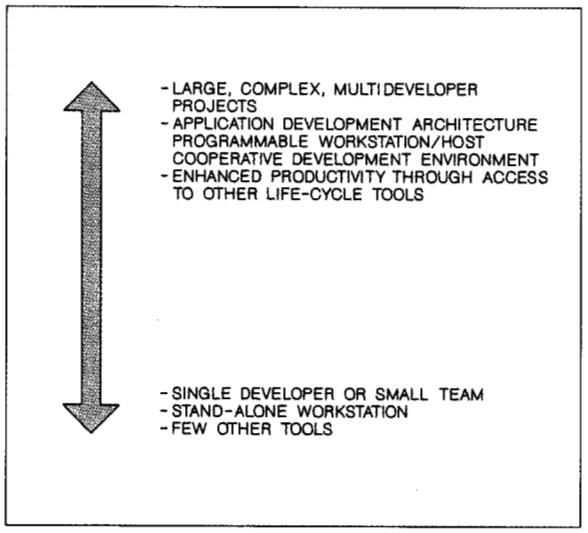
This new version of Prolog, through these interfaces and with the availability of a run-time environment, brings to the Prolog language a new capability for integration into commercial data processing. Remaining known requirements to extend this capability include the support of Prolog on other SAA platforms and further connectivity with SAA CPI components.

Finally, to bring this brief chronology to the present day, The Integrated Reasoning Shell,[12] which IBM has identified as the first release of a new strategic knowledge-based product line, provides forward, backward, and opportunistic reasoning integrated with frames. It delivers this function on MVS, VM, CICS, IMS, and OS/2 platforms of SSA.[13] Requirements associated with the delivery of applications built with The Integrated Reasoning Shell that remain to be addressed include the support of the OS/400 platform, enhanced connectivity to the SAA CPI high-level language and service components, and enhancements to the knowledge-based function of the system (for example, by providing procedural attachments to frames).

Knowledge-based applications for SAA. The ongoing direction of integrating of knowledge-based technology into commercial data processing will result in tools that build applications that conform fully to the Systems Application Architecture (SAA).[14,15] The SAA architecture requires that such applications have the following characteristics:

- The application is deliverable, with equivalent results, on each of the defined SAA platforms, namely TSO, VM/CMS, IMS, and CICS on System/370, OS/2, and OS/400.

Figure 1 The spectrum of knowledge-based development environments



- The application's end-user screen interfaces conform to the SAA Common User Access (CUA) standards.
- Where necessary, the application should use the defined SAA CPI service components, such as the SAA database (Structured Query Language), the SAA Presentation Manager™, the SAA Dialog Manager, the SAA query manager, and the SAA communications manager.
- The knowledge-based function within an application must be connectable to, or in other words can invoke and be invoked from, the SAA CPI language components such as C, COBOL, PL/I, Report Program Generator (RPG), etc., and the SAA CPI generator component implemented by Cross System Product (CSP).
- It is desirable that the knowledge-based system itself should define an SAA CPI component for defining the rules and frames and other parts of the knowledge base, and for access to the inferencing services of the knowledge base.

The intent and the net result of these SAA architectural requirements is that a knowledge-based application will finally have the same ability to integrate into a commercial SAA data processing environment as an application built using any of the other SAA components.

Knowledge-based application development for SAA

Knowledge-based technology's evolving capability to deliver applications in commercial data processing environments has greatly increased its potential in the commercial world, and the future conformance of knowledge-based applications with the SAA architecture is intended to maximize this potential. To realize the potential, IBM's knowledge basel architecture will evolve to integrate knowledge base development facilities with the facilities of AD/Cycle. The following sections of this paper describe the expected direction of this evolution.

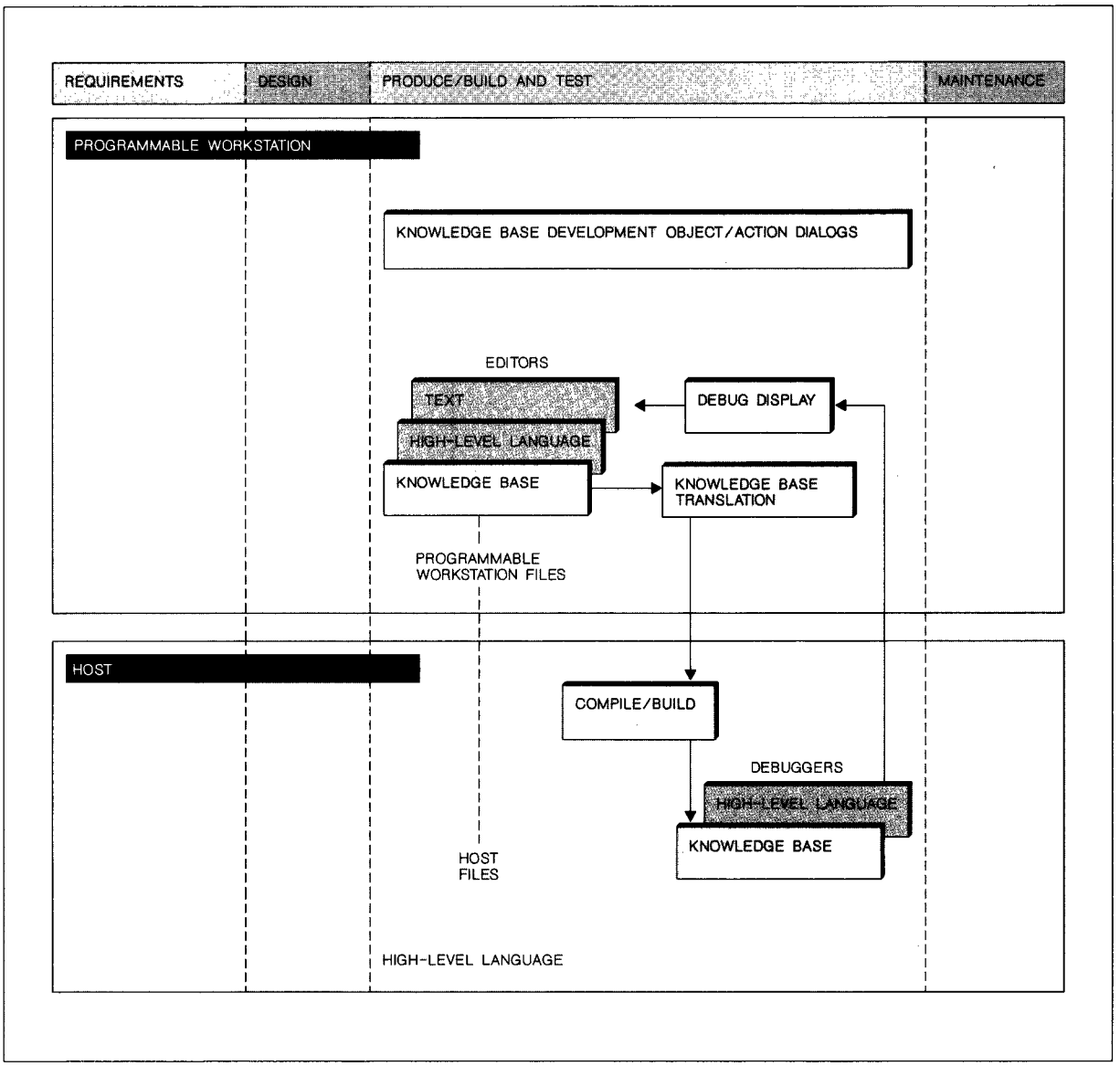
In practice, knowledge-based development facilities will address a spectrum of commercial development environments as shown in Figure 1.

At the simple end of this spectrum is the traditional knowledge-based development shell that is oriented towards the individual application developer (or knowledge engineer) or a small development team, and is focused on the life-cycle phases of building and testing the knowledge-based application. Typically, this simple environment will be used for prototyping or to develop small, stand-alone expert systems. It roughly corresponds in scope to the development environment provided today by ESE. At this end of the spectrum, only a few other application development products might be used by a developer in conjunction with the knowledge-based product.

Such a basic development environment for knowledge-based applications is shown in cooperative form in Figure 2. This development environment can be recognized as a derivative of the OS/2 development environment provided by The Integrated Reasoning Shell. It utilizes the graphic capabilities of the Personal System/2® (PS/2®) programmable workstation (PWS) to provide a highly responsive, highly usable object-oriented editor for the definition and interrelationship of rules, frames, procedural routines, and other objects[16] of the knowledge-based application.

Testing of the application is performed with a debug facility that visualizes information about rule inference chains, frames, and other knowledge base objects. Changes to the knowledge base definition are possible directly from the debug environment, and incremental compilation techniques are used to provide a fast iteration around the edit/debug/edit cycle. If a high-level language debugger is installed, then the two debuggers will work in a complementary

Figure 2 A knowledge-based development environment



fashion to present an integrated end-user interface that supports the debugging of both the inferencing and procedural parts of an application in a single debug session.

At this end of the spectrum, the knowledge-based development process is manually controlled through dialogs that are based on the object–action principle and are consistent with AD/Cycle user interface standards. The knowledge-based objects that are defined may be stored in private files either on the PWS or on a host.

This knowledge-based development environment builds applications to satisfy all the requirements for SAA applications that were described in the section

on knowledge-based applications for SAA. The development environment runs either on a stand-alone PS/2 platform to build applications for delivery on OS/2, or on a cooperative programmable workstation plus host configuration (System/370 or AS/400) as shown in Figure 2, to allow the building of applications targeted for a host environment. In the latter case, the application is uploaded to the target environment for the processes of compilation and debugging, with the programmable workstation being used for the control and viewing of this process.

At the other end of the spectrum, shown in Figure 1, is the customer with complex development projects that consist of mixed knowledge-based and high-level language or generator components, with large development teams and the need to integrate the knowledge-based development process with a range of other AD/Cycle tools.

For such complex application development environments, the knowledge-based development facilities work in conjunction with the other facilities of AD/Cycle. The specific additional items of integration with AD/Cycle follow:

- The AD information model of AD/Cycle is extended to include knowledge-based objects (rules, frames, etc.).
- The knowledge-based system development environment stores knowledge-based objects under the control of the Repository Manager™ and Library Services.
- The knowledge-based system development executes under the control of AD/Cycle's work management facilities.
- Access to and the complementary use of other AD tools in the AD/Cycle environment is enabled.

The following sections describe how these items of integration with AD/Cycle enhance the knowledge-based development environment in the areas of management and control of complex development projects involving large development teams, and productivity in the development of knowledge-based applications.

Management and control of the knowledge-based development process

Library control of knowledge-based objects. A knowledge base is comprised of a variety of objects. Object classes may include rules, frames, demons, and other constructs, or collections of these. Each object instance is defined individually, using edit facilities oriented toward each type of object.

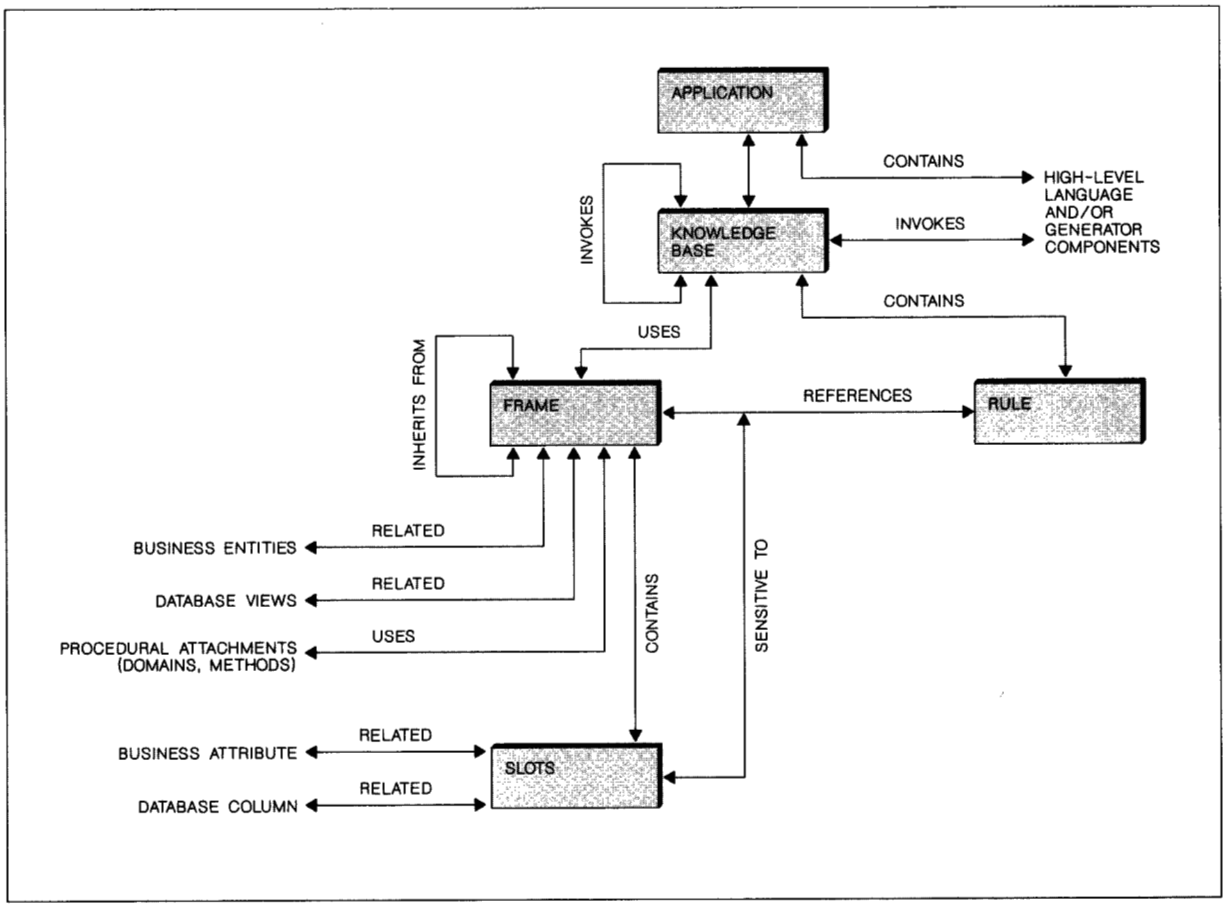
A hypothetical set of objects making up a knowledge-based application is shown in Figure 3. In many cases, the relationships between these objects are one-to-many or many-to-many. For example, multiple rules and more than one knowledge base may be sensitive to the definition of a particular frame, or more than one knowledge base may use a particular rule. In practice, this means that separate development teams developing different knowledge bases within a complex application may be sensitive to the same objects.

In large application development projects, the knowledge-based development facilities obtain library control over the objects by storing them under the control of the Repository Manager and Library Services. This control is available during development and during the subsequent production and maintenance phases of the application's life cycle. Figure 4 shows the knowledge-based development environment using the Repository Manager and Library Services for object storage. The Library Services provide functions that are familiar in high-level language development library systems. Changes to shared objects by multiple developers are serialized by locking the objects. Multiple versions of each object (for example, development, test, and production) are recognized and managed. A version of an object for development or test is obtained from a higher version of that object and subsequently promoted back to a higher level in a controlled manner. Library Services provide a build facility, that ensures that consistent objects are utilized in building an executable knowledge base, and recognizes the existence of multiple releases of the application over its lifetime, where some objects are common to the different releases and some objects are distinct.

The Library Services support applications consisting of both high-level language components (for example a COBOL main program), and the knowledge base and its constituent objects. Applications may be either fully host resident or cooperative, involving both host-resident and PS/2-resident elements.

Automated control of the knowledge-based development process. Figure 4 also shows the knowledge-based development environment with the superimposition of the AD/Cycle work management facilities over the dialogs.[17]

Figure 3 Hypothetical model of a knowledge-based application



A management-related requirement for large development projects is for automated control over the actual process of development. In AD/Cycle, this requirement will be addressed by an AD/Cycle tool that provides facilities such as those available today in the Application Development Project Support (ADPS) for System/370.[18] This AD tool uses a process model and current status information for the project to guide the user through the application development process. Knowledge-based development facilities will be invocable from the process management AD tool, and will return status and completion information at their termination.

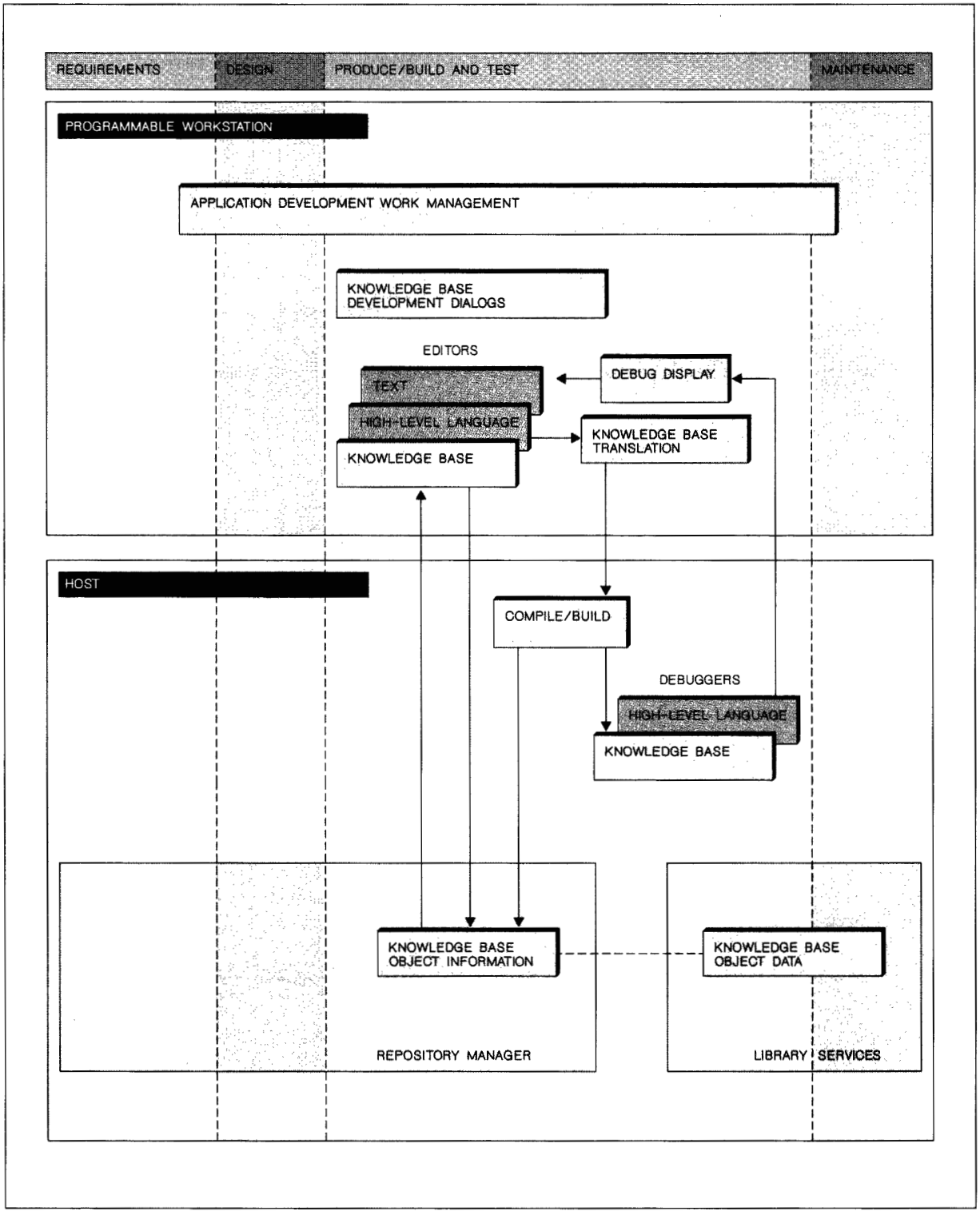
Productivity of the knowledge-based development process

Higher productivity in application development depends on addressing four areas: reuse, reduced complexity, automation of manual tasks, and elimination of redundant work.

Knowledge-based object reuse. In a knowledge-based environment, code reuse implies the reuse of the objects—rules, frames, procedural functions, or groups of these objects. This is enabled through the storage of the knowledge-based objects under the control of the Repository Manager with extensive cross reference information.

Frames for reuse will be located in the Repository Manager by searching for frames containing particular named frame slots (i.e., data items) of interest, by searching for frames containing certain keywords in their text descriptions, or by other methods. Essentially the task is to be able to find a frame that is pertinent to the domain of the application being designed. Once a candidate frame is located, it and

Figure 4 A knowledge-based development environment with the Repository Manager and Library Services



its parent and related frames are displayed graphically by the frame editor, and the user is permitted to browse the frame graph and select portions to be pasted into the frame graph which holds the frame objects of the new application. Using standard knowledge-based system edit facilities, the user can manually create new links between the reused frames and any other frames in the application. The process is shown in Figure 4.

Just as candidate frames can be located and reused, so candidate rules or groups of rules that reference a frame or particular slots within a frame can be found via frame-to-rule cross reference information in the Repository Manager. Such rules encode knowledge about the entities represented by the frames and are also candidates for reuse. The rules may be displayed by the rule editor, evaluated for reuse by the developer, and if suitable copied into the new application.

Automatic generation of knowledge-based objects. If reusable frames do not exist, then facilities will be provided to assist the developer in generating new frame definitions.

The knowledge-based development environment shown in Figure 4 is further extended in Figure 5 to show the automatic building of frame objects from enterprise model and database descriptor information in the Repository Manager.

Typically, frame classes in a knowledge-based application represent classes of entities that exist in the real world and the logic that the applications will use for reasoning. The entity classes, their attributes, and the relationships between them may already be described as an enterprise model, placed in the Repository Manager by enterprise analysis tools. Figure 3 shows the relationships between the knowledge-based objects and some other Repository Manager data that describe enterprise model entities and attributes (and also database descriptors, which are referred to below). Skeleton class frame definitions for a knowledge-based application can be constructed automatically from these enterprise model entities by mapping the entity-relationship model to a frame model with slots and links. Some enterprise analysis tools already organize their identified entities into hierarchic structures, with common properties abstracted into parent classes for inheritance by subclasses. Such structures will map to the inheritance structure of the frame system.

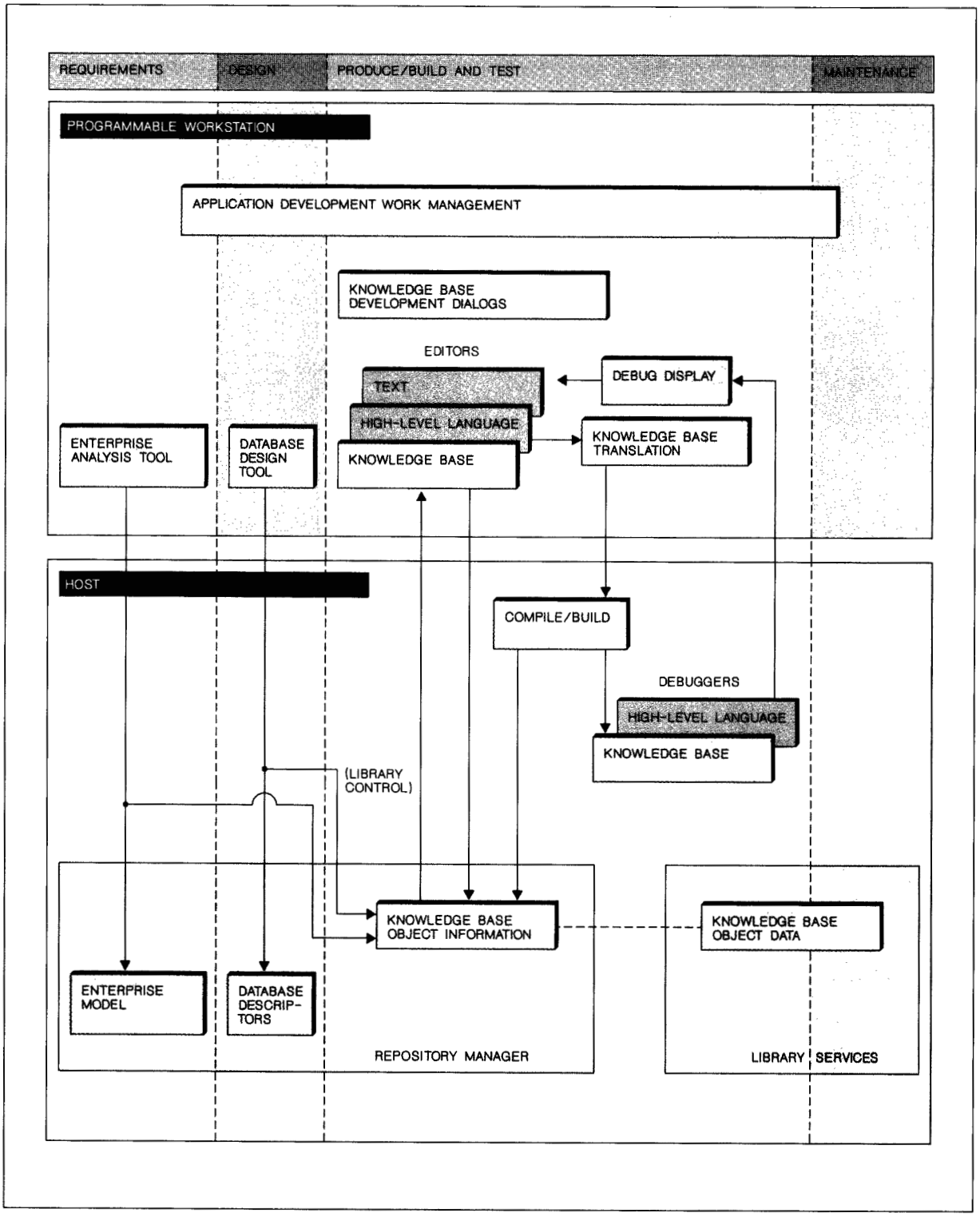
Enterprise model information of interest may be located by searching the Repository Manager for entity or entity attribute names (data items) of interest, or for user-supplied keywords in a text description of the entity. The enterprise model entities and their relationships are displayed graphically, and the user browses the graph and locates a subset of interest. Those entities, with their attributes and relationships, are transformed into a collection of skeleton class frames. On return to the knowledge-based frame editor, the skeleton frames are pasted into the new application's frame graph where they can be further edited and linked to other frames.

Once new frames are constructed in this way they are automatically placed under control of the Repository Manager, and are related to the other knowledge base objects and the enterprise model from which they were derived. Generating frames from the enterprise model in this manner provides the twin benefits of productivity (by elimination of the manual definition task) and accuracy (automatic generation will be more accurate than manual entry).

Another scenario for frame definition occurs when the intent is to build (or source) frame instances at run time from data in a database. In this case, it is necessary to provide a frame class definition to match some view of the database, and to generate a sourcing object containing the SQL statements to perform the database access. The specification of a sourcing object will require that the table (or view) name, column name(s) and their mapping to frame slots, and search arguments are provided. Without Repository Services these specifications would be entered manually. The Repository Manager, however, may contain database descriptions, placed there either by a database design AD tool or manually by the database administrator, which allow a skeleton frame definition and its associated sourcing objects to be constructed. These frame and sourcing objects are stored and linked in the Repository Manager to related knowledge-based objects and also to the database descriptions from which they are derived. Like the generation of frames from enterprise model data, this automatic frame generation from database descriptions provides the twin benefits of productivity and accuracy.

Access to other AD tools. As described in the previous section, knowledge-based application development productivity will benefit from the presence of enterprise modeling tools and database design

Figure 5 Use of other Repository Manager and Library Services objects by knowledge-based application development



tools. In addition, AD/Cycle will provide the developer with access to other AD tools that participate in that environment. Many of these tools address tasks that are potentially as relevant to the development of knowledge-based applications as to conventional applications.

Cross life-cycle tools are used over all life-cycle phases of AD/Cycle. They include documentation tools, project management tools, impact analysis tools, and query/report tools for the Repository Manager. "Screen painters" are used for the creation of text, object, or graphics-oriented dialogs. Other application-enabling facilities, the high-level languages or the application generator, are available for the production of any parts of an application that do not make use of the knowledge-based system. Tools are used to automate regression testing (such as IBM's Workstation Interactive Test Tool) or to manage the testing process.

In some cases, such complementary AD tools will be accessed by the user through the work management facilities of AD/Cycle. In other cases, they will be invoked directly from within the knowledge-based system development facility.

Knowledge acquisition tools. *Knowledge acquisition* is the term given to the collection of knowledge about some problem domain, in a form that allows it to be used within a knowledge base. In effect, knowledge acquisition is the analysis and design front-end phase of knowledge-based technology, and a successful knowledge acquisition tool will improve development productivity.

In its simplest form, knowledge acquisition is simply manual systems analysis with a pad and pencil (and perhaps a tape recorder). The term is more often used, though, to imply special tools or aids, or at least techniques, that are oriented to the construction of knowledge base components such as rules and frames. Well-known examples include repertory grid analysis and induction. More recently, formal, structured techniques that are analogous to the structured analysis techniques of conventional enterprise analysis are beginning to emerge, and it is likely that integrated methods and supporting tools that address the requirements of both conventional and knowledge-based application development will evolve over time.

A review of knowledge acquisition is beyond the scope of this paper, but the important point is that as such tools become available, they will themselves be integrated with AD/Cycle and will communicate their results to the knowledge-based system development environment via appropriate data using Repository Services and Library Services.

Library control of knowledge-based objects. The library control and automated process control described in the last section contribute to enhanced productivity through reduced complexity and automation of manual tasks. The manual alternatives to an automated library control system are generally tedious, time consuming, and error prone.

Knowledge-based assistance of the development process. This paper has focussed on the integration of knowledge-based application development into AD/Cycle, the goal being to use the facilities of AD/Cycle to enhance the development of knowledge-based applications. However, the paper would be incomplete without mention of the "mirror image" topic, namely the use of knowledge-based systems to benefit AD/Cycle by applying an *AI assist* factor to the AD tools and facilities of AD/Cycle. Leading-edge information engineering tools available from some non-IBM developers already use knowledge-based technology extensively. It is expected that this trend will continue in the future and will allow AD tool builders (both IBM and non-IBM) to implement sophisticated development methodologies that will permit higher productivities than could be achieved through conventional designs.

In addition to such truly breakthrough applications of knowledge-based systems, it is expected that this technology will become pervasive throughout the application development environment in a number of unobtrusive ways, such as "smart" (expert system) help facilities that are tightly coupled to the execution of a specific AD tool. Smart defaults may recommend a course of action or a parameter value to a developer based on a knowledge of the user and a history of the development session, and embedded knowledge bases may perform a variety of (normally) invisible background tasks, such as checking for standards conformance. Almost any AD tool or facility within AD/Cycle is a candidate for one or more of these types of assistance.

Summary

This paper has briefly reviewed the evolution of knowledge-based technology, described its current

level of integration into mainstream commercial data processing, and made the point that the logical end result of this evolution is the full integration of knowledge-based systems with IBM's Systems Application Architecture (SAA).

The paper has made the further key point that to realize the potential of this integration with SAA, IBM's knowledge-based application development facilities will also evolve to integrate with AD/Cycle. The expected direction of such AD/Cycle support was outlined, and its rationale was described in terms of the enhanced management control and productivity of the knowledge-based development process that will be essential to ensure its acceptance by the commercial data processing community.

Cited references and note

1. P. Harmon and D. King, *Expert Systems—Artificial Intelligence in Business,* Wiley-Interscience Publishers, New York, NY (1985).
2. D. A. Waterman, *A Guide to Expert Systems*, Addison-Wesley Publishing Co., Reading, MA (1986).
3. E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, Elsevier, New York (1976).
4. R. Duda, P. E. Hart, N. J. Nilsson, P. Barrett, J. G. Gaschnig, K. Konolige, R. Reboh, and J. Slocum, *Development of the PROSPECTOR Consultation System for Mineral Exploration*, SRI Report, Stanford Research Institute, 333 Ravenswood Avenue, Menlo Park, CA (October 1978).
5. M. I. Schor, "Declarative Knowledge Programming: Better than Procedural," *IEEE Expert*, IEEE, 345 E. 47th St., New York (Spring 1986).
6. M. Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P. Winston, Editor, McGraw-Hill Book Co., Inc., New York (1975).
7. *Expert Systems Development Environment User's Guide*, SC38-7006, IBM Corporation; available through IBM branch offices.
8. *Expert Systems Consultation Environment User's Guide*, SC38-7005, IBM Corporation; available through IBM branch offices.
9. *Expert Systems Environment Application Programming Guide*, SC38-7020, IBM Corporation; available through IBM branch offices.
10. *KnowledgeTool, General Information Manual*, GH20-9259, IBM Corporation; available through IBM branch offices.
11. *IBM Prolog for 370, General Information Manual*, GH21-1005, IBM Corporation; available through IBM branch offices.
12. *The Integrated Reasoning Shell, General Information*, GH21-1005, IBM Corporation; available through IBM branch offices.
13. Support of the AIX platforms by The Integrated Reasoning Shell is not addressed in this paper.
14. *Systems Application Architecture—An Overview*, GC26-4341, IBM Corporation; available through IBM branch offices.
15. *IBM Systems Journal* **27,** No. 3 (1988, whole issue).
16. Rules, frames, procedural routines, data sources, and various other components of a knowledge-based application are typically (and specifically, in The Integrated Reasoning Shell) referred to as "objects" to reflect the application developer's perception of these components. The knowledge-based system development environment presents the components graphically as discrete objects that may be individually manipulated for the purposes of definition and interrelation. In addition a knowledge-based system may, depending on its design, implement the knowledge-based application using the object-oriented programming paradigm, with the frames, rules, etc., represented by objects in this sense. However, this latter characteristic is not universal.
17. G. Chroust, H. Goldmann, and O. Gschwandtner, "The Role of Work Management in Application Development," *IBM Systems Journal* **29,** No. 2, 189–208 (1990, this issue).
18. *Application Development Project Support/Application Development Model and Application Development Project Support/Process Mechanism General Information Manual*, GH19-8109, IBM Corporation; available through IBM branch offices.

Douglas M. Hembry *IBM Programming Systems, 2800 Sand Hill Road, Menlo Park, California 94025.* Mr. Hembry's present assignment is in the technical office of the knowledge-based systems development group at IBM's Menlo Park laboratory. He joined IBM United Kingdom in 1965 as a systems engineer, and from 1972 to 1975 worked as a data systems specialist at the UK Systems Center. From 1975 through 1982, he continued the role of data systems specialist at the International Systems Center (then the World Trade Systems Center) at Palo Alto, the Western Region Systems Center of IBM Canada at Vancouver, and the Calgary, Alberta, branch office where he supported the data systems operations of a number of large customers. In 1982 he transferred to the Information Management System (IMS) development group of the IBM General Products Division in Santa Teresa, California, where he had planning responsibility for IMS's high availability characteristics. Mr. Hembry moved to the technical office of the knowledge-based systems group in 1987 and has been responsible for the planning and architecture of IBM's knowledge-based system offerings toward full integration into SAA and AD/Cycle. Mr. Hembry received his B.S. degree from the University of London in 1964, with first-class honors in physics and mathemetics.

Reprint Order No. G321-5399.