Report 85-19
Stanford -- KSL

Scientific DataLink

Evaluating the Existing Tools for Developing Knowledge-Based Systems.
Mark H. Richer,
May 1985

Knowledge Systems Laboratory May 1985

Report No. KSL 85-19

Evaluating the Existing Tools for Developing Knowledge-Based Systems

by

Mark H. Richer

Stanford Knowledge Systems Laboratory
(incorporating the Heuristic Programming Project)
Stanford University
701 Welch Road (Building C)
Palo Alto, CA 94304

Table of Contents

BLANK PAGE

1. Introduction

In recent years there has been a great deal of interest in the commercial applications of knowledge-based (KB) systems (commonly called expert systems). Interest in KB systems was spurred on by the development of programs that can solve complex tasks at an expert level. (See HAYES-ROTH83 and BUCHANAN84 for a comprehensive introduction to expert systems). Other applications include simulations of complex devices or processes. In contrast with conventional software systems, KB systems represent domain and problem-solving knowledge as symbolic structures such as rules or frames.

Many AI applications share common methods for representing knowledge, performing inference, and maintaining large systems. As a result, AI researchers have been able to generalize their methods by developing knowledge representation languages and KB shells [see HAYES-ROTH83 and BUCHANAN84]. Recently, several commercial products have emerged in the marketplace. These products are designed to support the development of KB systems. The more serious development tools are very costly, and require expensive, specialized hardware and scarce high-salaried employees to develop applications. Therefore, current clients are typically Fortune 1000 companies or government agencies that can afford the investment. For these clients, the high price tag is not as much a deterrent because the prospect of building their own tools for less money is unlikely. In the near future, the situation could change dramatically if these tools become available on less expensive hardware, and their utility is well demonstrated. (Currently there exists a class of tools that are available on microcomputers such as the IBM PC. These tools are significantly limited compared to their 'big brothers' and they will not be discussed in this paper).

The purpose of this paper is two-fold: (1) to examine the criteria for evaluating software tools for developing KB systems, and (2) to describe several existing KB system tools. The reader should take note that the information gathered here is necessarily limited to a large extent by what is publicly available. A more complete evaluation must include actual use of each system, and this was not possible at the time this paper was prepared. Therefore, no attempt is made to present a complete evaluation of any one system, and strong comparisons about two or more systems are avoided. Because this paper is not meant to be tutorial in nature, no attempt is made to systematically explain terms and concepts that are commonly used in the artificial intelligence community.

The KB tools described in this section were chosen for review for several reasons. They are all serious development environments that run on high performance work stations or common mini-computers. Current tools that run on microcomputers are not considered here. The tools chosen are all commercially available products. This eliminates research systems even though they may be available as public-domain programs. Also excluded are in-house tools, such as Schlumberger's STROBE, and tools which are not currently being fully supported by vendors, such as Xerox's LOOPS system. The products described herein include ART (Inference Corporation), DUCK (Smart Systems Technology), KEE (Intellicorp), SRL+ (Carnegie Group), and S1 (Teknowledge). The name of each system is a trademark of the respective company that sells the system. Additionally, *ActiveImages* and *TellAndAsk* are trademarks of Intellicorp.

Information on specific system tools was gathered from company brochures, magazine articles, and personal visits to Intellicorp and Teknowledge. The amount of information gathered for each product is not equally distributed so that the sections on specific products vary quite a bit in length. This does not reflect a bias or a judgementfor or against any of the products, but simply reflects the amount of information that was available. To repeat, this paper is not a definitive evaluation of the products described herein; its purpose is to discuss some general criteria for the evaluation of such tools, and to provide a summary of the available information on several (but not all) currently available tools for developing KB systems.

2. Criteria For Evaluation

It is difficult to evaluate any tool without proper consideration of what one intends to use the tool for. A tool that may be excellent for commercial use may be inadequate for research purposes, and vice versa. The amount of support and training required in order to use a tool effectively can vary greatly. Experienced AI programmers may want software support, but probably care less about training. The whole issue of training will take on less importance as the AI industry matures and there is a solid base of experienced AI programmers dispersed throughout industry. Yet in this early stage of technology transfer, many customers may find that the quality and quantity of training they can receive may be one of the more important considerations in choosing an expert system tool. The cost of the software is certainly an important factor, but in the long run calculating the cost effectiveness of one tool versus another is a complicated matter. The total cost of a tool includes the cost of training and support, hardware required, personnel, and so on. This must be weighed against the total gain. If the representational features of one system are such that you can more easily encode the knowledge you are trying to capture in your KB system then the extra cost of the software and required hardware may be offset by the rewards of choosing that system. However, the success of writing a program of any kind can often be determined as much by the programming environment as by the specific representational features provided by the language. The quality of the user-interface, available editors and debuggers are critical components of any software development system. Furthermore, the quality of documentation and on-line help systems are also important factors. It is hard to evaluate any dimension in isolation from another because they can interact with each other. However, in the sections below, one way has been chosen to organize a set of dimensions and each is discussed briefly.

2.1. Representational Features

KB development tools generally provide support for representing symbolic knowledge. This knowledge can include facts, definitions, heuristic judgements, and procedures for doing a task or achieving a goal. The LISP language has proven to be general and flexible enough to represent these kinds of knowledge. However, LISP programming can be painstaking and the resulting code can be opaque. Many researchers have chosen to use or develop representational languages that allow them to encode knowledge at a higher level, which may be closer to how a person may conceptualize it. Various formalisms or techniques have been used including predicate calculus, logic programming, production rules, frames or schemas, and blackboards. Often one or more of these methods have been combined in a single system. Included in these representational languages are reasoning mechanisms including pure deduction, backward and forward chaining, and inheritance. Some systems even allow the user to specify control of inference to some degree.

It can be argued that certain kinds of knowledge are more easily or naturally represented in one formalism than in another. Heuristic knowledge for doing diagnosis can be represented naturally in rules, but some claim that definitions and structural knowledge are more clearly represented in frames or relations. Relations provide a foundation in mathematical logic, but frames can be organized in a hierarchy providing the powerful mechanism of inheritance.

Different researchers with different biases and different problems to solve have often expressed a strong preference for one methodology over another. Recently there has been a trend toward hybrid systems. This is especially true of the commercial KB tools available. In an article in AI magazine [KUNZ84], members of Intellicorp argue for hybrid systems by claiming that the loss of uniformity resulting from combining methodologies is offset by gains in efficiency, flexibility, and the naturalness of the representation. Certainly a system that is more general and more efficient over a range of problems can provide significant advantages to a system builder. However, hybrid systems may be more complicated to learn and use, and for some applications a simpler tool that provides less representational features may be sufficient and more cost effective in practice. On the other hand, Intellicorp argues that the limitations of a single representation system are often overcome by 'tricks' which result in opaque or awkward representations. They argue that the increased complexity of a hybrid system can be justified because "the designer [can] trade the effort to support and teach users to use the tricks with the effort required to support and teach users to use several

methodologies." Hopefully, industrial case histories will be reported that will shed some light on which features are sufficient for doing which problems cost effectively. In addition, continued research studies should help us classify problem types and indicate which representational frameworks are more appropriate for a given problem class.

2.2. Functionality

Functionality in this context refers to all the factors that make a software tool and resulting applications useful to users. This term could be used to encompass other dimensions discussed in this paper including representation and ease of use; however, those topics are treated as separate dimensions instead.

The following issues illustrate what is meant by functionality in this context:

- Appropriateness of a tool for development (or prototyping) versus for end-user applications
    - Is one of these endeavors better supported than the other?
    - Is there a special run-time environment that is more compact, more efficient, or requires less memory during execution?
    - Can run-time applications be executed on cheaper hardware?
- What are the the computational requirements for supporting the system (e.g. minimum memory needed)?
- What are the upper limits on the size of applications? Is there a maximum number of rules or frames that can be used for a given hardware configuration?
- Is the product available on a range of hardware?
- Is the tool only useful for commercial applications or is it appropriate for research projects also?
- Is the tool robust? Is it easy to build robust applications?
- Is the tool extensible? (This certainly interacts with support discussed in section 2.4. Availability of sources, detailed documentation, and technical support can determine how extensible a system actually is.)

In section 3, several commercial tools are discussed, but it is precisely these questions that are difficult to answer without first-hand experience. Therefore, the focus of the discussion in section 3 is limited mostly to describing representational features and specific software tools that can aid a programmer or knowledge engineer build a system.

2.3. The Development Environment, Ease of Learning, and Ease of Use

The following section is admittedly subjective in nature. Precise definitions and methods of measurement are not discussed here and no attempt is made to provide support for several statements that may be open to experimental validation. The comments below appeal to commonsense intuitions and are not meant to represent a formal analysis.

Some of the factors that make a system easier to use for experienced users can also make it easier for novices to learn as well. However, sometimes these goals are at odds, particularly when features which make it convenient for an expert user to do things quickly are opaque to the novice. A well-designed user-interface that provides user settable options and multiple ways of achieving goals (e.g. menus and keyboard commands) can go a long way towards alleviating such conflicts. However there still may be a price to pay in the size, complexity, or efficiency of the system. A slow and cumbersome system is never easy to use.

Ease of learning can be facilitated by emphasizing simplicity and uniformity in the user-interface. Various forms of on-line help are often useful for both novices and experienced users. However, very few software systems can be completely self-explanatory. Therefore, it is inappropriate to evaluate ease of learning without considering the availability and cost (including employee training time) of various kinds of support including training, documentation, and technical assistance provided on the phone.

Ease of use includes how quickly common tasks can be executed. This information can only be acquired through actual use of a system or through published studies. However, the potential customer of an AI tool can find out what kinds of tasks are supported by the tool, and the extent of this support. The availability of various development tools can dramatically influence how easy it is to develop an application. However, the ease by which these tools are learned and used may ultimately determine their actual value. A variety of tools are described below.

Knowledge Acquisition is often a painstaking and time-consuming process. If a given tool provides most of the representational features needed for a given problem, then it is likely that a large amount of the time that is spent on a project will be building the knowledge base and refining it. In applications where the knowledge base will be modified and expanded over time, tools for knowledge acquisition are crucial. Such tools include knowledge base editors which help system builders enter encoded knowledge into the system. An editor for a frame-based system allows you to enter slots and values for individual objects. A grapher package that allows you to interactively add and edit nodes in a taxonomy or causal network can simplify knowledge base construction. If the system builder does not have to know about the underlying represention of objects in the program, then even domain experts may be able to use these tools to assist in knowledge-based construction.

Another class of tools that are useful for knowledge acquisition and validation of the system are tools that check the integrity of the knowledge base to weed out contradictory facts or rules, discover missing or redundant information, and to find syntactical errors. Spelling checkers can also be an invaluable aid because typographical or spelling errors can lead to knowledge base errors that are hard to detect otherwise.

After entering knowledge into a system, the next step is to run the system on some data. Tools that monitor system behavior by providing interactive graphic displays are becoming a necessity in developing complex software systems. These tools are often useful for end-users as well and have opened up a new dimension in explaining system behavior without standard text generation. For system builders monitoring tools should be integrated with other debugging aids, including high level trace and break packages at the 'knowledge level' as opposed to the LISP level. For example, the system builder should be able to set breaks when specified events occur (e.g., a certain rule is tried).

For end-user applications, system builders must provide a good user-interface, graphic displays and explanations of system behavior, requests and decisions. AI tools that support this activity can provide a distinct advantage over tools that do not provide this support. Many commercial tools provide graphic editors to create icons and displays, but these features may vary more among systems than the AI features do. Again the kind of applications a system will be used for determines how important a given feature may be. It is currently difficult to know the exact form an AI system should take in order to be acceptable or useful in a specific application. A system that provides many features may constrain the development of new features which are actually needed. However, it is not necessary that a system that is full of features is less extensible than a simpler system.

In the future, many toolkits may be available with AI tools. Some of these toolkits will be bundled with the system and some will be available at an additional cost. Some may require additional hardware. These could include simulation kits, natural language front-ends, text generation and explanation packages, sound-generating and voice recognition packages, and so on. Eventually, KB systems may be integrated with traditional software tools including word processors, spreadsheets, and communications programs. It seems likely that the future success of KB systems in the business or office market will depend on how well they can be integrated with existing systems.

2.4. Training, Documentation and Support

The companies making KB system tools are in the business of technology transfer. Therefore, many of these companies provide instruction in AI including classes in the LISP language. These general educational endeavors may not be directly related to products these companies sell so they are outside the scope of this paper. However, most KB system tools are sold with an agreement for training and support. Often the customer sends one or more employees to the site of the software company for a one or two week training session. This training can be a vital component in the success of using a particular tool. The training for a first copy is often mandatory and expensive. It includes support of the product which includes technical assistance over a period of time. Most of the companies provide a hotline phone service to answer technical questions.

An important factor in evaluating a software development system is the quality and availability of documentation. The kind and amount of documentation needed will vary depending on the design of the system, its intended purpose, its actual use, the people using it, and so on. Detailed documentation or source code may be needed in order to make major modifications and extensions to a system. Sometimes additional documentation could cost extra money. Source code is often very expensive to obtain, and a non-disclosure agreement will commonly be required. This can be a big problem for groups that are funded under government grants and wish to disseminate their systems for little or no charge.

An alternative to making in-house modifications to a KB tool is to hire the original software company (or someone else) to make custom modifications. In cases where this is a likely need, the product must be evaluated in the context of the availability and cost of custom work. Furthermore, it would be necessary to negotiate rights to any new products developed. Most software companies would like to retain the rights to any custom packages they develop, but the contracting company may feel that they paid a high price for this work and want to retain any licensing rights.

Some companies may prefer the company that supplies the tool to also produce the application system including the encoding of expert knowledge, a process which is referred to as knowledge engineering. Several companies offer this service, presumably at a high cost to the customer.

2.5. Validation of Knowledge-Based Systems

Evaluating a KB system application is obviously different than evaluating a KB system tool for developing applications. For an application, verifying that it performs its task correctly is crucial. It is difficult to verify beforehand that adding domain knowledge to a shell will result in correct problem solving behavior. Earlier, it was suggested that knowledge acquisition aids can help insure the integrity of a knowledge base by checking for completeness, inconsistencies, and synactic, typographical, or spelling errors. However, there is no way of automatically validating that a system's performance is satisfactory. We would like to assume that the inference procedures in commercial products are error-free, but in reality there may be bugs that arise unexpectedly. Part of supporting a product is providing a good vehicle for reporting bugs to the vendor, and issuing patch code and new releases of the system to correct any known bugs.

There are several features of a system that can facillitate system testing. These include the monitoring and debugging tools discussed above. Also a feature that allows a system builder to run a large number of cases in batch facilitates testing and maintenance of a knowledge base. Automatic record keeping can help a KB maintainer keep track of any changes in system performance on a given case. Such tools could help avoid introducing subtle bugs into a system that could make a system fail unexpectedly on a case it previously performed well on. More work surely needs to be done in this area. For further reading see Part 10 of BUCHANAN84.

2.6. Cost Factors

These include the price of the software, support, training, required hardware, and skilled personnel. Multiple copies of the software are usually sold at a reduced cost. However, the cost of developing an application and distributing it in-house or selling it as a product can be difficult to access. For instance, the cost of training includes not only the price that the software company charges, but all other expenses incurred including employee air fare, room and board, and the salary of the employee for the training period. The period of time needed to learn how to effectively use a tool will probably extend beyond the formal training sessions. However, a high training cost can be offset if a given tool can improve productivity more than one with a lower training cost. Unfortunately, it is difficult to determine from company literature alone how easy a tool is to learn or use, let alone how productive someone will become using it at a future time.

When an application is intended to be widely disseminated, then development costs can be distributed among many users. However, such systems are too costly unless they can be placed on low-end hardware configurations. The availability of a compact, robust, and efficient run-time environment may be necessary. An inexpensive licensing agreement for run-time only systems is highly desirable in such cases.

Once a system is developed, staff may be required just to maintain the system and support end-users. These additional costs are not trivial. It would be nice if we could determine that a particular tool will facilitate building application systems that are more robust and easier to maintain and support than another tool; of course, this is also difficult to determine at present.

It is clear that the cost of developing AI applications is very high, and the cost of purchasing a KB system tool is only the beginning. The degree to which a tool can keep other costs to a minimum may determine its true cost effectiveness. Again, experience using existing tools to build AI applications has been too limited (and seldom described in print) for this to be determined in any practical way.

3. Review of Several Existing KB System Tools

3.1. S.1 (Teknowledge)

S.1 is a tool for developing rule-based consultation systems that are designed to solve *structured selection* problems in which there is an enumerated set of solutions. Technical details concerning the implementation of S.1 are not available, but it appears to be strongly based on the EMYCIN system, but with the addition of control blocks which are found in the original ONCOCIN system[see BUCHANAN84]. Users are expected to be computer professionals with some AI training (which could be provided by Teknowledge). S.1 is a less complex tool than KEE, SRL+, or ART since its use is constrained to a certain class of problems which it is tuned for. The idea is that many applications can be created with a minimal amount of low-level programming. However, Teknowledge admits that S.1 is not appropriate for planning, design, constraint-satisfaction, or modeling problems. However, a large class of diagnosis and catalog selection problems are amenable to S.1 techniques.

3.1.1. Representational Features in S.1

Rule-Based Reasoning

Rules in S.1 are not described in much detail in the company literature, but they appear to be similar to EMYCIN rules in that variables are not allowed, certainty factors can be added, procedural attachment is allowed, and backward chaining is the control used for rule interpretation. It is assumed that rule clauses contain object-attribute-value triples, with optional certainty factors.

Control Blocks

High-level, algorithmic procedural knowledge for controlling problem solving is represented

in control blocks. This provides a structured high-level language for representing procedures. A control block statement can create an object, determine the value of an attribute, display text to the user, or invoke another control block. A consultation is started by invoking a top-level control block.

Frame System

Teknowledge says that assertions about objects and relationships between objects are represented in a specialized 'frame system.' Users can define classes and attributes. S.1 'frames' do not seem to provide explicit support for general inheritance, run-time slot value type-checking, or active values (demons). However, S.1 'frames' do support propagation of subsumption relations which is used during reasoning.

Uncertainty

Facts and rules in S.1 can be modified by certainty factors.

Lisp Programming

By using control blocks or procedural attachment in rule clauses, the knowledge programmer can access the underlying lisp system to execute arbitrary function calls including graphics calls.

3.1.2. Using S.1 for Software Development

User-Interface and Graphics Support

The user-interface in S.1 is mostly menu-driven and the user can use a mouse device to make selections. S.1 makes use of the multiple window systems found on current lisp machines to provide a much more user-friendly consultation system than EMYCIN provided on a typical mainframe terminal. The user can select commands and valid responses to questions from menus that are easily used. Keyboard commands and input are also allowed at any time, in which case, abbreviations that uniquely specify the input can be used. S.1 has a help window which describes the current question, commands, and options.

Objects in the knowledge base are associated with text strings which allow S.1 to generate English text for questioning, explanation and other purposes. Rules can be automatically translated into English if-then statements.

The S.1 environment does not provide advanced tools for the user to create graphic displays and interfaces, but these can be programmed using the underlying lisp graphics tools. S.1 makes use of graphics to display various kinds of information regarding the knowledge base and the dynamic state of execution during a consultation. These tools are described below.

KB Tools

(1) S.1 provides a knowledge base editor.

(2) When the editor is exited, the knowledge base can be loaded into S.1 and the system automatically checks for syntax and consistency errors.

(3) Test cases can be saved and used again for system testing. S.1 allows the user to override the answer to a question in a saved case.

(4) S.1 allows the user to trace and break a variety of events.

(5) An event tree can be displayed in a trace window. Nodes in this tree can be selected to get explanations of system behavior, to find out values and certainty factors, and obtain English translations. This dynamic display aids the developer in browsing the knowledge base, and in monitoring and debugging system behavior.

Explanation

S.1 can generate English text to answer HOW and WHY questions in the EMYCIN sense.

End-User Systems

A 'consultation only' system can be created which prevents end-users from modifying the knowledge base.

Hardware Required

S.1 is available on XEROX 1100 series systems running Interlisp-D, SYMBOLICS running Zetalisp, and VAX Franz LISP under VMS.

Range of Applications

Teknowledge is very discrete concerning any applications that S.1 is used for because this is often required of them by their customers. One application that is commonly mentioned is a drilling advisor that diagnoses why oil drill bits are sticking. In general, Teknowledge asserts that S.1 is appropriate for structured selection (or classification) problems which include many diagnosis and catalog problems.

3.1.3. Support and Training for S.1

Teknowledge provides extensive training, documentation and support with S.1. A first copy license fee includes two participants in the two-week S.1 training course. Software maintenance and updates is provided free of charge for the first year, and at a charge thereafter. Telephone support is provided to a single designated individual and one alternate person. Teknowledge's Applications Engineering Service is only available to S.1 clients and assists customers in designing and implementing applications on a contract basis.

3.1.4. Cost of S.1

The first copy of S.1 is $50,000 on Xerox and Symbolics hardware, and $80,000 on the DEC hardware. Second and subsequent copies can be purchased without training for $15,000 on Xerox and Symbolics hardware, and $25,000 on DEC hardware. After the first year maintenance and updates cost 10% of the current license fee. Additional training is offered at a cost.

3.2. DUCK (Smart Systems Technology)

DUCK is primarily a logic programming language implemented in a portable dialect of lisp called NLISP. It supports development of rule-based systems with both forward and backward chaining, and non-monotonic reasoning. NLISP is included with DUCK. DUCK is a general purpose language and does not provide as much explicit support for specific applications as does S.1, and provides less representational features than KEE, SRL+, or ART. However, it is an order of magnitude cheaper than the other products described in this report.

3.2.1. Representational Features in Duck

Logic Programming

The notation in DUCK is based on first order predicate calculus and allows disjunctive (OR) as well as conjunctive (AND) expressions. DUCK uses unification and backward chaining to respond to queries.

Non-Monotonic Reasoning

DUCK provides direct support for non-monotonic reasoning, which allows programs to reason with assumptions and inconsistent information. DUCK does this by employing dependency-directed backtracking which is implemented using a truth maintenance system and 'data pools.' One feature this provides is that variables can be instantiated with default values, but changed later if an assumption is shown to be false. Data pools allow hypothetical reasoning to be specified in a knowledge base. Only the differences from the original database are actually maintained and saved, allowing many hypothetical situations to be considered while using a minumum of storage. This provides much of the capability of ART's viewpoint mechanism, which is described in Section 3.5.1.

Rule-Based Programming

Rules are expressed in DUCK's logic language. Both forward and backward chaining are provided, and they can be mixed allowing the user to specify partial searches to prune a large search tree.

3.2.2. Using DUCK for Software Development

Very little about the programming environment tools are described in the literature obtained from the company. (It is assumed the tools are not as extensive as the other systems that sell for much more money.) It is not clear how much more leverage DUCK gives a knowledge base application developer as compared to another logic programming language (e.g. PROLOG) with the addition of forward chaining and non-monotonic reasoning, though this may be significant in itself.

Hardware Required

DUCK is implemented in NLISP which is implemented in Symbolics Zeta LISP, VAX Franz LISP under UNIX or VMS, Apollo workstations running T Lisp. It will soon be implemented in Common LISP, IBM mainframes running LISP under VM, and Xerox Interlisp-D.

Range of Applications

SST builds KB systems under contract with government or industry. Projects described in company literature include:

- Fault Isolation in Avionics
- Satellite Anomaly Diagnosis
- Communication Network Diagnosis
- Economist Workstation: Representing, manipulating, and displaying time series data for comparative analysis.
- Executive Decision Support System to assist in the development of distribution plans for fielding new pieces of military equipment. (Implemented on a Symbolics using FLAVORS)
- Personnel Expert System to assist army personnel planning and processing
- Intelligent Analyst Workstation for Air Force analysts.

3.2.3. Support and Training for DUCK

SST does not include training with their product, but does offer it at an additional cost. SST also offers a variety of classes in AI, and does a considerable amount of contract work as described above.

3.2.4. Cost of DUCK

At the time of this writing, DUCK with NLISP included costs $6000 for the first computer, substantially less than the other tools available. Additional copies cost $1200 each.

3.3. KEE (Knowledge Engineering Environment, Intellicorp)

KEE is a hybrid tool integrating several AI methodologies into a single system. It provides support for frame-based knowledge representation with taxonomic inheritance, rule-based reasoning, logic representations, data-driven reasoning , object-oriented programming, and LISP functional programming. In addition, KEE provides direct support for various uses of interactive graphics. Important ideas in KEE include "early prototype development, incremental refinement of the problem description, use of multiple integrated solution

methods, and emphasis on visibility of both the problem-solution process and the explicit description of the problem domain." [KUNZ84]

3.3.1. Representational Features in KEE

Application specific knowledge in KEE is called a knowledge base and is loaded into the KEE system. The knowledge base includes frames that represent or describe objects and classes of objects.

Frames

Frames in KEE (internally called UNITS):

(1) are a data structure consisting of a frame name and a set of attribute descriptions, each defined by a slot. Associated with each slot is the "attribute's value(s), constraints that the value must satisfy, procedures that are called whenever the slot is accessed or its values change, and the source of the inherited values." [INTELLICORP84] Therefore, values of attributes can be stored, inherited from other frames, computed when needed by executing LISP code or deduced from a specified set of rules.

(2) describe attributes of objects that are either declarative or procedural. Attribute values can be restricted to meet constraints including class (e.g. must be female), range (e.g. greater than 21), cardinality (e.g. number of natural parents is 2), and data-type (e.g. integer). The value class can be a boolean combination of class descriptions such as (FEMALE-PERSON (UNION DOCTORS LAWYERS) (NOT.ONE.OF SANDRA- DAY-O'CONNOR)), which refers to all female doctors and lawyers except Sandra Day O'Connor. Support for partial descriptions of slot values provides type checking, a way to deduce certain facts (e.g. Joe is not the mother of Bob), and a way to represent some disjunctive and negative facts..

(3) can either describe a class of objects (e.g. pipes) or an individual object itself (e.g. pipe.3). An individual object can be described as an instance or member of one or more classes. As a result of this distinction between classes and individuals there is need for two kinds of attributes, those that describe the class itself and those that describe an individual of a class. "Own attributes" describe a class and include properties such as lightest, longest, smallest, and oldest. "Member attributes" describe individuals of the class and include properties such as length, weight, height, and age.

(4) form taxonomies with the property of inheritance. Objects include classes, subclasses, and individuals. Any frame or unit can have multiple parents. A subclass or an individual inherits the properties of the classes it belongs to. Inherited properties can be restricted by use of an OVERRIDE option. For instance, you could make 3-wheel cars a subclass of cars which have 4 wheels by default. OVERRIDE would be used to change the number-of-wheels attribute to 3; however, other attributes would continue to be inherited.

Intellicorp says frames can be used to describe structure and function better than rules, and can allow for deeper models and reasoning than pure rule-based systems. "Rules describe relations between individual propositions. While propositional representations are powerful, they focus on relations among attributes of objects, rather than on objects themselves, and they do not represent temporal or spatial relations in a natural way."[KUNZ85] "Frames are not well suited for representing all kinds of knowledge, however. For example, they provide no particular advantages for describing behavior, only limited facilities for expressing universal and existential quantification, and no facilities at all for representing most negative and disjunctive facts." [INTELLICORP brochure]

Rules

Rules in KEE:

(1) are to used to encode heuristics and procedural knowledge. Rule clauses can include logical expressions with conjunctions, disjunctions, and negations, as well as arbitrary LISP s-expressions.

(2) are also represented as frames in a class hierarchy. Other objects (besides rules) can refer

to classes of rules which in turn can affect the objects when the rules are applied.

(3) are supported with backward and forward chaining rule interpreters.

An example rule from a KEE brochure that could be "used to determine which feedstock bin can supply alkali to a reactor whose pH level is lower than desired" is reprinted below:

```
NEEDS.ALKALI RULE
    (IF ((?SOME.PLANT IS IN CLASS PLANTS)
          AND
           (A REACTOR OF ?SOME.PLANT IS ?SOME.REACTOR)
          AND
           (THE PH.SATE OF ?SOME.REACTOR IS LOW)
          AND
           (A FEED.STOCK.INPUT OF ?SOME.REACTOR IS ?SOME.BIN)
           AND
            (THE CONSTITUENT OF ?SOME.BIN IS ALKALI))
          THEN
          (THE NEW.CONSTITUENT.BIN OF ?SOME.PLANT IS ?SOME.BIN))
```

(4) are useful for analysis of problems in complicated domains since they are easy to use, and "allow and force a focus on decision making and simulation."

(5) can be used for multiple purposes such as explanation by describing or displaying a chain of reasoning as the chain of rules that were applied.

(6) User-supplied functions can tailor rule interpretation to specify depth-first, breadth-first, or other search strategies for backward chaining, and resolve conflicts in forward chaining when more than one rule is applicable.

Logic

The use of logic in KEE includes a database assertion and query language based on a subset of predicate calculus, called *TellAndAsk*. For example you can assert:

```
(THE FLOW OF (THE FEEDWATER.PUMP OF R1) IS LOW).
```

This will result in storing the value low (if it's a legal value) in the FLOW slot of frame R1. Retrieval requests can specify a pattern with a variable. To determine which objects have a feedwater pump with a low flow rate the user could request:

```
(THE FLOW OF (THE FEEDWATER.PUMP OF ?X) IS LOW)
```

If the earlier assertion above had been made then R1 would have been returned as one object that satisfied the user's request. The assertion and retrieval functions know how to store and access values in frames as well as how to store and retrieve facts that have no frame to be stored in. These facts are placed in an "unstructured facts list." This can be used as a debugging aid since it indicates to the developer those facts that the system has no way of representing in frames. The *TellAndAsk* language provides relations that are commonly needed, but the user can add his or her own relations. Rules can be applied when assertions are made providing antecedent reasoning, and retrieval can be done by deducing a value from rules, providing consequent reasoning. *TellAndAsk* is described in more detail in [NADO85].

Logical expressions can also be used to describe constraints on slot values. Furthermore, logical expressions are used in rules.

Graphical Representation

Icons can represent objects or values of object attributes that are linked to the underlying frame representation of the object in the knowledge base as a slot value. Manipulating these graphical images interactively updates the values in the knowledge base and vice versa. For instance fluid level in a device could be changed simply by using a mouse device to 'drag' the level up or down. This capability is part of KEE's *ActiveImages* package which is included with the system. It is particularly effective for interactive simulations where several parts of a

system are interconnected and changes in one subpart or device affects the whole system. KEE can propagate these changes automatically (on the screen and in the knowledge base) if the necessary relationships and actions have been represented in the knowledge base. (*ActiveImages* is described further in section 3.3.2 under GRAPHICS SUPPORT).

Object-Oriented Programming

Frame systems are inherently object-oriented as the frames each represent an object (a class is also an object). Behavior is invoked by passing messages to objects. The information needed to respond to a message is stored in the attribute slots. Storing behavioral responses directly with an object can facilitate creating more modular programs.

Functional LISP Programming

At its lowest level (and when necessary or desirable) KEE is programmed in LISP and provides full access to the underlying LISP system. Therefore, developers can write LISP code to describe procedural behavior whenever needed.

Data-Driven Reasoning

When the value of a slot is accessed or changed procedural behavior can be invoked through a function call. This can be used to support data-driven reasoning or control of the system. (Active values also provide part of the facility for connecting values to graphical icons as the graphical image can be updated when a value is changed.)

3.3.2. Using KEE for Software Development

Flexibility and Generality

KEE was designed to be a flexible and general environment for the development of knowledge-based systems. It is an evolving product, and can be customized for specific applications.

User-Interface and Graphics Support

KEE was designed to provide a powerful, flexible, and usable user-interface for system developers. In addition, explicit support is provided to the developer for creating user-interfaces for end-user applications. The *ActiveImages* package is an example of this kind of support.

KEE makes extensive use of graphics to provide a powerful and friendly user-interface. The bitmap displays provided on current LISP machines are exploited by utilizing multiple windows, pop up menus, graphs, pictures, gauges, and animation. The knowledge base can be interactively edited by manipulating the nodes in knowledge base graphs that represent underlying data structures in the knowledge base. An interactive knowledge base editor for creating and modifying frames is also provided.

Class and Object hierarchies can be automatically displayed as lattices in a screen window. Solid lines represent subclass links while dashed lines represent object links. This provides a standard way of viewing the objects in a knowledge base and makes the relationships between objects immediately clear, including multiple parent (inheritance) links. The nodes representing classes and objects can each be selected in the graph to get more detailed information. This helps provide a graphical user-interface to the knowledge base that can be used to drive the system.

The *ActiveImages* package provides direct support for dynamic applications that use graphics for display and interaction with the user. The KEE developer can create "graphical displays for both viewing and controlling KEE objects. Images such as histograms, thermometers, switches, push-buttons, plots, meters, pipes, and valves can be attached to the slots of KEE objects. *ActiveImages* also includes facilities for creating and attaching icons to objects, for collecting and saving collections of active images within control panels, and for customizing images and control panels through reshaping, annotation, and modification of descriptive parameters. Because the *ActiveImages* package is implemented as a KEE knowledge base, it can easily be extended by the user to create new kinds of images." [KEE BROCHURE]

Explanation and Debugging

Explanation and debugging facilities in KEE:

(1) can explain how the value of an attribute is determined or why information is requested from the user.

(3) include a Backward Chainer Explanation Window that displays a graph that shows the links between object attributes (the top goal or a rule premise) and the rules that conclude about the attributes during backward chaining. Rules are marked and boxed to distinguish them from the other nodes in the graph. The boxing makes it easy to see quickly which premises (object-attribute-value triples written in English syntax) were concluded from rules and which were derived from facts in the knowledge base.

(4) include rule class graphs which display the static relationships between rules in a class. Similar to the backward chaining window described above these graphs have links between premises and rules that conclude about the premise.

(5) include a "How" graph showing the derivation tree for any conclusion.

(6) include "Why" explanations for any question asked during rule interpretation.

(7) allow tracing of the rule interpreter during forward and backward chaining, and break commands for interrupting the rule interpreter to allow more detailed debugging.

Hardware Requirements

KEE is available on a range of hardware which currently includes LISP machines sold by LMI, Symbolics, and Xerox. Intellicorp intends to implement KEE on new hardware that may enter the marketplace in the coming years.

Range of Applications

KEE is meant to be useful in a wide range of applications. Potential and actual applications of KEE that have been described in publications and company brochures include:

- monitoring a nuclear reactor.
- simulating and diagnosing problems in a biotech processing plant.
- a system to assist satellite operators to diagnose and correct spacecraft malfunctions
- a discrete event simulation of a factory

3.3.3. Training and Support for KEE

Intellicorp is committed to continuing and dependable support of their products. This includes training courses, on-site consulting, a phone hot-line, and electronic mail.

3.3.4. Cost of KEE

The first copy of KEE costs $60,000; half of the cost going towards training and support.

3.4. SRL+ (Schema Representation Language, Carnegie Group Inc.)

SRL+ is also a hybrid tool with characteristics similar to KEE. It uses schemas (called frames or units in KEE) and also provides object-oriented, rule-based, and logic programming language facilities. In addition, it provides an agenda mechanism, a window/canvas interface, and an embedded database for large applications on a DEC VAX. The Carnegie Group says that SRL+ is implemented in Common Lisp, a potential advantage for anyone interested in porting code across machines that support Common Lisp.

3.4.1. Representational Features in SRL+

The literature that was obtained on SRL+ is limited in detail. Therefore, the description of representational features is necessarily terse and limited in scope.

Schemas

Schemas in SRL+ appear to be similar to KEE frames. A schemata contains slots with values that describe attributes of the object represented by the schemata. As in all frame or schema-based systems, objects include physical things, concepts, processes, as well as classes of objects, and inheritance is explicitly supported. SRL+ allows the programmer to specify search paths during inheritance, and specify which slots and values can be inherited. SRL+ also contains a dependency mechanism that identifies the source of inherited values. Procedures or demons can be attached to slots and users can define error handling routines.

Rules

SRL-OPS-5 provides a forward chaining rule-based language that is integrated into the SRL+ environment.

Logic Programming

SRL-PROLOG provides a logic programming capability with backward chaining.

Object-Oriented Programming

SRL+ represents objects as schemas with inheritance, and supports message passing. Logic expressions, rules, or lisp functions may be executed as a response to a message.

Control

A multi-queue event manager is used to schedule events to occur in either a simulated or normal operating mode. Events are represented as schemas, and are placed on an agenda. Event slots may include the event's assigned queue, its ordering within the queue, and the prescribed time the event will take place. This provides direct support for event-based simulations of complex processes.

Contexts

Contexts allow alternative solutions to be tested when solving a problem. This sounds similar to viewpoints in ART, though the feature is not explained in any detail.

3.4.2. Using SRL+ for Software Development

User-Interface and Graphics Support

(1) Windows, displays, and 'canvases' are represented as schemas and can be instantiated to create mouse-driven interfaces. Unfortunately, no information was available explaining what is meant by a 'canvas.'

(2) A CORE-based graphics package for constructing 2D graphics displays is provided. This is a device independent package that provides functions for manipulating multi-window displays.

(3) SRL+ provides spelling correction and a standard command library.

(4) Graphics are used for system browsing and editing of schemas. The Palm Network Editor displays schema hierarchies as graphs which can be interactively edited.

Version Management

Users can create versions of the same model using a context mechanism.

Database Support

On the DEC VAX a multi-user database database system is provided where frequently used schemata are stored. Temporary changes by local users are cached in memory and do not change the shared database.

Hardware Required

SRL+ is currently available on Symbolics, LMI Lambda, PERQ and VAX computers. The Carnegie plans to continue porting SRL+ to other machines including the TI Explorer.

Range of Applications

The prototype of SRL+, developed at Carnegie-Mellon, has been used for several applications according to company literature:

- Callisto: "a product management system which focuses on the semantic representation of activities and product configuration."
- Isis: "a production management system which models, schedules, and monitors activities."
- Rome: "a quantitative reasoning system for long-range planning."
- PDS: "a rule-based architecture for the sensor-based diagnosis of physical processes."
- INET: "a technique for corporate distribution analysis based on the Knowledge Based Simulation approach to modelling and simulation.

3.4.3. Training and Support for SRL+

Carnegie Group provides continuing support of SRL+ and its purchase includes a two-week hands-on tutorial. The Carnegie Group will build custom systems and claims its expertise lies in four major areas [CARNEGIE GROUP company description]:

(1) Automated Engineering Design CAD/CAM/CAE including VLSI and metal products design aids.

(2) Production Management including factory shop floor management

(3) Sensor-based Machine Control and Diagnosis management of flexible manufacturing systems components and robotics.

(4) Project/Product Management engineering and software products

3.4.4. Cost of SRL+

Was not included in promotional material. Call Carnegie Group Inc. in Pittsburgh for further information.

3.5. ART (Automatic Reasoning Tool, Inference Corporation)

ART is also a hybrid tool that provides the system developer with a great deal of representational and programming features. It is described in detail below.

3.5.1. Representational Features in ART

Facts

Propositions in ART express declarative knowledge including general facts (e.g. dogs are mammals) and situation specific facts (e.g. the dog is sick). A fact in ART consists of a proposition (a recursive sequence structure) and an extent (a scope of validity).

ART differentiates between facts that are explicitly stated to be false and facts that are unknown (not known to be true or false). This is supported by allowing ART propositions to have truth values associated with them.

Patterns

In ART, patterns are propositions with variables. Goal and strategy patterns are distinguished in ART. "Goal patterns define certain conditions under which the known data might suggest actions... [and] describe relationships occurring in the propositional database; the set of active goals are stored in the goal base. .... Strategy patterns represent conditions under which the system's current goals might suggest particular processing approaches ... [and] describe relationships among entries in the goal base; they are stored in a strategy base." [DESIGN ENTRY]

Schemas

Schemas in ART are conceptually similar to frames in KEE, but do not seem to provide all the features found in KEE frames. "Schemata may be thought of as collections of facts, allowing the representation of data as conceptual objects. Users may define relations in the system with their own inverses and transitivity properties. Relations may also introduce new relations automatically. Inheritance of information is allowed between schemata. Users may also define their own inheritance relations in the system." [ART product summary]

Below is a sample definition of a schema [WILLIAMS84]:

```
(defschema deployed-satellite)
   (subset-of satellite)
   (prototype
      (orbit (element-of earth-orbit))
      (position (element-of (-> orbit positions)))
      (attitude (element-of 3-space-vector))
      (attitude-maintenance-frame (element-of (set lvlh mean-of-50)))
```

Rules

Rules are used in ART to encode procedural knowledge. ART provides a rich pattern matching language which is used to create and apply rules. "Patterns may include wild cars, literal values, sequences, segments, variables, and logical connectives, as well as arbitrary predicates and procedural restrictions encoded in LISP. The rule language includes logical connectives and quantification as well as procedural constraints such as a case statement and iteration ... Furthermore, mixed chaining is possible by combining goals and facts within a rule. The conflict resolution strategy may be defined by the user. Arbitrary function calls are allowed in the condition and action statements of a rule." [ART product summary]

ART has forward and backward chaining rules. Below are some sample ART rules [ART brochure] :

Forward Chaining Rule

English: Any expression raised to the zero power can be simplified to 1.

```
ART:    (Defrule Exponent-Zero
 (is ?X (expt ? 0))
=>
 (assert (is ?X 1)))
```

Backward Chaining Rule

English: If you want to know whether person A and person B are cousins, verify that some combination of their parents are siblings.

```
ART: (Defrule Cousin
   (cousins ?A ?B)
<=
     (parent ?parent1 ?A)
        (parent ?parent2 ?B)
        (sibling ?parent1 ?parent2))
```

Art differentiates between inference and production rules. Inference rules add facts to the knowledge base while production rules change facts (e.g. the value of an object attribute). An example production rule [ART brochure] is the AND-Gate-Rule:

If the output of an AND-gate is 1, and one of its inputs go to 0, change the output to 0.

ART also has a rule compiler to produce more efficient run-time applications.

Viewpoints

"A viewpoint is a hypothetical 'world' in which ART can pursue a potential course of action to see where it leads. ART can process a very large number of such viewpoints simultaneously." [ART brochure] ART's viewpoint mechanism is "based on a blackboard model that supports logic programming without backtracking." [ART brochure] You can view an "ART database as a set of hypothetical partial solutions. Hypotheticals may be compared and contrasted, merged together or marked as inconsistent, automatically or by rules. The system also supports multiple levels of hypotheticals. The viewpoint structure can be inspected by the user and matched by rules." [ART product summary]

Viewpoints can be used to represent a point in time, a possible interpretation of incoming data, or a potential solution to a problem. Viewpoints are also useful for representing default and non-monotonic reasoning. Each viewpoint contains a subset of propositions. There is a root viewpoint that can have children that inherit facts from the parent viewpoints. Each child can have children and so on. Children can have multiple parents and hence form merge viewpoints. Each viewpoints can add or delete propositions to its parent's set of propositions.

Uncertainty

Associated with each viewpoint is a confidence rating that gives a numeric estimate of a viewpoint's validity. Numerical certainty factors can also be associated with individual facts in the knowledge base.

3.5.2. Using ART for Software Development

Flexibility and Generality

ART is a general purpose programming language with direct support for developing a wide range of KB applications. It is a complex environment and requires extensive knowledge of programming and AI concepts and techniques. However, it is extremely flexible and provides support for most features typically used in KB programs. ART was designed with real-time applications in mind and Inference Corporation claims that ART can be used to develop very efficient run-time applications "where the critical intervals are in the order of tens of milliseconds." [AR product summary]

User-Interface and Graphics Support

(1) ART has a built-in help system which is available to the user at any time, and is designed to explain system commands and language features.

(2) Command entry in ART is facilitated by spelling correction and command abbreviations.

(3) Extensive use of mouse-based menu system for browsing through the system. However, experienced users can use keyboard commands in place of menu selection.

(4) Graphical representations of hierarchical knowledge structures can be automatically displayed as graphs with elliptical nodes.

(5) A structured knowledge base editor is provided.

(6) ART includes a package called ARTIST (ART Image Synthesis Tool) which assists a developer in icon creation and manipulation. This feature also supports animation. Icons are represented as schemata and are saved in the knowledge base

Debugging

A program monitor allows the developer to observe and debug program behavior in execution. There are schema, rule, and viewpoint browsers included in the system. The debugging environment allows tracing execution, setting breakpoints, and examining the state of execution. The use can manually enter patterns and execute actions when execution is stopped.

Range of Applications

ART is a powerful programming environment and seems to be adequate for solving a large number of products. Examples described in available literature include:

- satellite servicing
- navigation: NAVEX, a system used by NASA to make high-speed decisions about satellite velocity and trajectory during re-entry.

Hardware Required

ART is currently implemented on the Symbolics, LMI LISP machines, and DEC Vax with VAX-LISP.

3.5.3. Training and Support for ART

Not specified in promotional material. Call Inference Corp. in LA.

3.5.4. Cost of ART

Not specified in promotional material. Call Inference Corp. in LA.

4. Conclusion

This paper discussed criteria for evaluating tools that are used for developing KB applications. These tools can be very complex; therefore, evaluation is very difficult and requires the availability of a great deal of information. Hands-on experience is very desirable, but it may be difficult to get while shopping for a system. The author even had difficulty borrowing a programming manual from one of the companies.

Most KB system tools have a core of features in common. The products vary in the amount of explicit support given for certain kinds of development. S.1 is designed for a certain class of problems which Teknowledge refers to as structured selection, while DUCK is an enhanced logic programming language which requires more program development. ART, KEE and SRL+ are all similar in that they are a hybrid of various AI techniques. ART has many built-in representational features including multiple viewpoints. KEE is well-integrated around an object-oriented programming environment and provides strong support for the use of interactive graphics. Less information was available on SRL+, so that it is difficult to make any comparisons, but it seems similar to KEE in its frame-based representation while it has some, but not all features in common with both ART and KEE. I have been told that SRL+ is a very flexible and general programming environment. Therefore, the scarcity of information presented here should not lead a reader to conclude that SRL+ is a little like KEE and ART, but not as sophisticated.

All five products appear to be of high quality, and the vendors appear to be committed to supporting and improving their products. However, these tools (with the possible exception of DUCK) are extremely expensive at this time, and therefore, for the present time, are out of the reach of many potential customers. In the next few years, it should be clearer how useful and cost-effective these tools and the KB system methodology, in general, is for real-world applications.

In this paper, an attempt was made to highlight the strengths of each system. It was not possible to adequately criticize the weaknesses without hands-on experience. For those shopping for a system, make sure you analyze your own needs first and then try to match them with the product and company that will best help you meet your own goals. In addition, other products may be available that are well worth considering.

I. Appendix: AI Companies Whose Products Are Described

The Carnegie Group, 650 Commerce Ct. at Station Sq., Pittsburgh, PA 15219, (412) 642-6900

Inference Corp., 5300 W . Century Blvd., 3rd floor, Los Angeles, CA 90045, (213) 417-7997

Intellicorp, 707 Laurel Street, Menlo Park, CA 94025-3445, (415) 323-8300, [moving summer 1985]

Smart Systems Technology, 6970 Elm Street, McLean, VA 22101, (703) 448-8562

Teknowledge, Inc. 525 University Ave., Palo Alto, CA 94301, (415) 327-6640

Bibliography

Barr, A. and Feigenbaum, E., eds., *The Handbook of Artificial Intelligence*, William Kaufman, Los Altos, CA, 1981-82, Vol. 1-3.

Buchanan, B., and Shortliffe, T., *Rule-Based Expert Systems*, Addison-Wesley, Menlo Park, CA, 1984.

Carnegie Group, company description: Carnegie Group, 7 pages, not dated.

Carnegie Group, "PLUME", product description, 5 pages, not dated.

Carnegie Group, "PLUME: A Tool For Developing Natural Language Interfaces", two-sided summary product summary, not dated.

Carnegie Group, "SRL+ Applications And Advantages", 5 pages, not dated.

Carnegie Group, product summary: "The Carnegie Group Knowledge Representation Language/ An Integrated Knowledge Engineering Environment", 4 pages, not dated

Clayton, B., *ART Programming Primer, ART Version 1.1*, Inference Corporation, LA, CA, Sept. 15, 1984.

Fikes, R., and Kehler, T., "Control of Reasoning in Frame-Based Representation Systems", draft, Intellicorp, Feb. 10, 1985.

Gevarter, W., *Intelligent Machines*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985.

Hayes-Roth, F., Waterman, D., and Lenat, D., eds., *Building Expert Systems*, Addison-Wesley, Reading, MA, 1983.

Inference Corporation, "Application of an Expert System to NAS MCC Support", 1 page company handout, LA, CA, not dated.

Inference Corporation, "Need AI? Art Delivers", company brochure, LA, CA, 1984

Inference Corporation, "ART", product summary, 2 pages, not dated.

Intellicorp, "The Knowledge Engineering Environment", company brochure, Knowledge Systems Division, Menlo Park, CA1984.

Johnson, T., *The Commercial Application Of Expert Systems Technology, Ovum Ltd.*, London, England, 1984.

Kinnucan, P., "Software Tools Speed Expert System Development", High Technology, March 1985, pp.16-20.

Kunz, J., Kehler, T., and WIlliams, M., "Applications Development Using A Hybrid AI Development System", AI Magazine, Vol. 5, No. 3, Fall 1984, pp.41-54

Linden, E., "Intellicorp: The Selling of Artificial Intelligence", High Technology, March 1985, pp.22-25.

Nado, R., Bock, C, and Fikes, R., "An Assertion and Retrieval Interface for a Frame-Based Representation System", Intellicorp, Menlo Park, CA , Jan. 23,1985.

Marsh, A., "NASA to Demonstrate Artificial Intelligence in Flight Operations", Aviation Week & Space Technology, McGraw-Hill Inc., Sept. 17, 1984, 1 page reprint.

Shattuck, William, H., a stock report on Intellicorp, Montgomery Securities, SF, CA, Jan. 11, 1985, 11 pages.

Schindler, M., "Artificial Intelligence Begins To Pay Off With Expert Systems For Engineering", Electronic Design, August 9, 1984, pp.106-146.

Smart Systems Technology, "DUCK A Versatile Logic Programming Language For Building Intelligent Systems", 3 pages, not dated.

Smart Systems Technology, "Helping Customers Get Started In AI With Education And Training Courses", 11 pages, not dated.

Smart Systems Technology, "Helping Customers Get Started With Artificial Intelligence", 6 pages, not dated.

Smart Systems Technology, "NISP/DUCK Ordering Information", not dated.

Stefik, M., Bobrow, D., Mittal, S., and Conway, L., "Knowledge Programming In Loops", AI Magazine, Vol. IV, No. 3, Fall 1983, pp.3-13.

Teknowledge, "About Teknowledge", double-sided page, not dated.

Teknowledge, "S.1", double-sided product summary sheet, not dated.

Teknowledge, "S.1 Interactive Knowledge-Base Graphics", double-sided page, not dated.

Teknowledge, "S.1 Purchase Information", 1 sheet, not dated.

Teknowledge, "S.1 Product Description", 8 pages, not dated.

Williams, C., "ART/The Advanced Reasoning Tool/Conceptual Overview", Inference Corporation, LA, CA, not dated.

Williams, C., "Software Tool Packages The Expertise Needed To Build Expert Systems," Electronic Design, August 9, 1984, pp.153-167.

Copyright © 1985 by KSL and
Comtex Scientific Corporation

FILMED FROM BEST AVAILABLE COPY