EVALUATION AND SELECTION OF EXPERT SYSTEMS TOOLS

Randall Davis
COMMERCIAL TOOLS: 4 ISSUES

- Getting past the jargon
- (Why) Do they work?
- Understanding the claims
- Understanding the kinds of tools available.
ISSUE 1: GETING PAST THE JARGON

** What do all of the following mean?

frame-based representation
declarative representation
procedural representation
object-oriented programming
multiple inference engines
knowledge-programming environment

* Now, which of them do you actually need? Avoid the mousetrap mentality.
ISSUE 2: (WHY) DID IT WORK?

R1 was written in OPS.

R1 is the most successful commercial expert system.

OPS must be a very powerful system building tool.
ISSUE 3: SEEING PAST THE CLAIMS

"This product eliminates the need for knowledge engineering and programming."

"You can generate any kind of expert system automatically."

"Practical, cost-effective expert systems can be created in hours."
ISSUE 4: CATEGORIES OF TOOLS
THE TOOL / LANGUAGE SPECTRUM

TOOLS
S1

PROLOG

LISP

LANGUAGES
CATEGORIES OF TOOLS

- General—purposes languages
  LISP

- Special—purpose languages
  OPS, PROLOG

- Prototype Construction Tools
  M1, PC+

- Experiments in induction

- Master Carpenter Tools Kits
  ART, KEE, LOOPS, S1
LISP: THE DIALECT EXPLOSION

LISP(1957)
  /
LISP 1.5
  /  
BBN LISP  
  /  
INTERLISP  
  /  
INTERLISP-D

MACLISP
  
FRANZ ZETA LM TI NIL T SCHEME

COMMON LISP
  
LISP/VM
LISP

- **ADVANTAGES**
  * Facilitates Symbolic Computation
  * Facilitates IE Construction
  * Interactive
  * Compiled or Interpreted
  * Advanced Programming Environments

- **DRAWBACKS**
  * Profusion of Dialects
    - Common Lisp
  * Large Working Set
PROLOG

• ADVANTAGES
  * Facilitates Symbolic Computation
  * Interactive
  * Built-in IE (Backward Chaining)
  * Built-in Database
  * Built-in Pattern Matcher

• DRAWBACKS
  * The Mismatch Potential
  * Advanced Programming Environments
    Uncommon
  * Large Working Set
PC TOOLS

- Definition: Runs on a PC (but beware fluid categories)
- Widely Available
- Inexpensive
- Run on Standard HW
EVALUATING TOOLS: A BASIC CHECKLIST

- PLATFORM
  * What Hardware
    o Clones?
  * What Operating System
  * How Much Memory
    o To run at all
    o To run quickly
      (try it)
EVALUATING TOOLS

- KNOWLEDGE BASE
  * The KB Development Environment
    o KB Editor
      * Specialized
      * Standard Word Processor
      * Menus (structure or confinement?)
EVALUATING TOOLS

• KNOWLEDGE BASE
  * The KB Development Environment
    o KB Checking
      * Syntactic
        If the number of shareholders is yes ...
  
  * Logical consistency
    If the number of shareholders < 35
    then consider an S corporation.
    If the number of shareholders < 35
    then don't consider an S corporation.
  
  * Subsumption
    If the number of shareholders < 35
    then consider an S corporation.
    If the number of shareholders < 25
    then consider an S corporation.
EVALUATING TOOLS

• KNOWLEDGE BASE
  * The KB Development Environment
    o KB Debugger
      * trace and report
        (at "rule level")
      * testing
    o incremental growth
      * integrated editor
    o How big a KB is convenient
      * browser
      * rule subsets
      * text or graphics
EVALUATING TOOLS

- KNOWLEDGE BASE
  * The KB Development Environment
    o Habitability
      * tag rule with name, date
      * augment with source info
      * comments in files
      * listings (language, subsets?)
EVALUATING TOOLS

• INFERENCE ENGINE
  * Inference Style
    o backward, forward
  * Control of Inference
    o rule order
  * Does it Handle Uncertainty?
    o in rules
    o in answers
  * Other Inference Methods
    o frame matching
    o integrate procedures and rules
  * What-if
  * Sensitivity Testing
  * Volunteered Info
  * Explanations
    o implementation
EVALUATING TOOLS

- KNOWLEDGE REPRESENTATION
  * Rule Form
    - and, or
  * Clause Form
    - what operators
    - attribute, object, values
EVALUATING TOOLS

• HABITABILITY
  * Links to External Resources
    o Routines in same language
    o Routines in other languages
    o O/S utilities
    o Database, spreadsheet, graphics
  * Compiler
  * Runtime System
  * User Interface
    o typewriter or menu
    o change answers
    o “?”
  * Knowledge base lockable?
  * Knowledge base private?
EVALUATING TOOLS

• Manual and Supporting Materials
  * examples
  * on-line tutorial, help
  * sample knowledge bases
EVALUATING TOOLS

• SUPPORT
  * Bug Repairs
  * Application Advice
  * Ongoing Development
  * New Releases
  * Training Courses
  * Future of Vendor
EVALUATING TOOLS

• COST
  * Public Domain
  * Single Copy
  * Site License
  * Updates
  * Runtime Package
  * Source Code
EVALUATING TOOLS

• SPEED

* Save Months not Minutes
SOFTWARE PACKAGES

- Is it debugged?

- Is it supported? By whom? For how long?

- What are long term development plans?

- What machines does it run on? How long has it been running on them?
SOFTWARE PACKAGES
(Continued)

- How large is the user community?
- What is the environment like?
- Is there a manual?
- How fast is it?
HARDWARE

- **High End Machines** ($10K - $50K)
  Symbolics, TI, HP 9000, Xerox

- **Medium Scale** ($5K - $10K)
  DEC Microvax, Sun, IBM RT(?)

- **Small Scale** (under $5K)
  PC's
SUMMARY

- Watch out for jargon
- Understand how the tool helps and how it hinders
- There is a spectrum of languages and tools available
Comparative Evaluation
of Three Expert System Development Tools: KEE, Knowledge Craft, ART.

Jean-Pierre Laurent, Jacqueline Ayel, Franck Thome, Danielle Ziebelin

Laboratoire d'Informatique appliquée
Université de Savoie
BP 1104 73011 CHAMBERY (FRANCE)

ABSTRACT

The purpose of this paper is to evaluate and compare three of the most powerful expert system tools available; KEE from Intellicorp, Knowledge Craft from The Carnegie Group Inc., and ART from Inference Corporation.

These three tools are industrial development environments which are fully supported and well beyond research prototypes. They were implemented on Lisp machines initially, but will soon be available on conventional computers. The three systems are very flexible and offer many ways of representing knowledge.

The first part of the paper is a technical overview of each tool. The second part presents their common features. The third part discusses the advantages and drawbacks of each, according to types of application.

This work has been partially sponsored by the Commission of the European communities within the Esprit Project 932 entitled 'Knowledged-based Real-Time supervision in Computer Integrated Manufacturing'. We are grateful to Didier Gaget-Soufflot (Graphael, France) and Bernard Rausi (CGE/TITN) for their participation in the evaluation of KEE and ART respectively.

INTRODUCTION

This work set out to evaluate hybrid expert system development tools, i.e. tools which combine several well-proven artificial intelligence methodologies, including object oriented languages, frame-based representation and rule-based programming; which allow the use of various inference mechanisms, and which have a user-friendly interface. We limited our study to tools having full commercial support (training courses, releases, technical support etc.).

First, we selected seven tools:

— ART (Automatic Reasoning Tool), from Inference Corporation,
— DUCK, from Smart System Technology,
— KEE (Knowledge engineering environment), from Intellicorp,
— KES II, from Software A and E,
— KC (Knowledge Craft), from Carnegie Group
— RULE MASTER, from Radian Corporation,
— S1, from Teknowledge.

Eventually we only kept ART, KEE and KC and discarded the other ones:

— DUCK, because its knowledge representation is not powerful enough,
— KES II, because it is strongly diagnosis oriented,
— RULE MASTER because it has no user-oriented interfaces, and, in our opinion, because it is more a research tool than an industrial tool.
— S1, because its use is limited to well structured selection domains (it is a system of the emycin family).

The first step in the evaluation was based on documentation and the second step on practical programming exercises run on a Lambda machine (LMI) and on a TI Explorer. These exercises were conceived to evaluate the capacity of each of the tools in solving important problems related to:

— non-monotonic data
— representation of time
— representation of relationships (others than those defining classes and sub-classes)
— sophisticated control strategies
SHORT PRESENTATION OF THE TOOLS

We now give a short technical overview of each tool, keeping in mind that it is impossible to describe all the capabilities of each one. It is assumed in the following that basic notions like objects, attributes, demons, procedural attachments, message passing, forward chaining rules, backward chaining rules, backtracking, conflict set, non-monotonic reasoning...etc...., don't need to be presented in detail.

KEE

KEE came on the market in 1983, and is presently available on most of the Lisp-machines, on DEC-Vaxworstations, HP9000, and soon on IBM PC-RTs. It provides a well-designed representation language based on units, a data-structure similar to frames.

Each unit is defined by its unit name, and a set of attribute descriptions, represented as slots. “Own” slots are used to describe classes or objects own properties. “Member” slots are used to describe class members' properties.

Several kinds of information can be attached to a slot in addition to its value, which can be any LISP expression: procedural attachments, inheritance specification and meta-knowledge can be defined for each slot — allowing complex object-based worlds to be built.

Reasoning in KEE is based on rules, which are represented as units, like any KEE object, and can be used either in forward or backward chaining. In KEE, a reasoning process is usually initialized by a call to the knowledge base Assert and Query Language supplied in the package. This language, called TellAndAsk, provides essentially three basic functions: ASSERT to create a fact, RETRACT to remove it, and QUERY to extract knowledge from the knowledge base. Two syntaxes are provided within TellAndAsk for writing propositions: an English-like form, useful for its readability, and a prefixed form, more compact and faster because it is closer to the syntax of Lisp. Some examples of TellAndAsk follow:

**ASSERT:**

Natural-language form:

(ASSERT ‘(THE power OF my.car IS 700)) = assertion

(ASSERT ‘(THE power OF my.car IS 700) ‘car-rules) = ass. + forward chain.

Prefixed form:

(ASSERT ‘(OWN.VALUE my.car power 700))

**QUERY:**

Natural-language form:

(QUERY ‘(THE power OF my.car IS ?what))

(QUERY ‘(my.car IS IN powerful.cars) ‘car-rules)

**RETRACT:**

Natural-language form:

(RETRACT ‘(THE power OF my.car IS 700))

...etc...

KEE rules have the same syntax as TellAndAsk, for example:

(CAR-TAXES.RULES
  IF (THE power OF my.car IS> 200))
  THEN (THE tax-to-pay IS steep)
  DO (put.some.money.aside))

Such a rule can be used in two ways: either to deduce a conclusion from existing facts, or to find out if a given fact is true by trying to prove premisses of rules, the right-hand side of which contains the fact. This feature allows homogeneous knowledge bases to be built but note that this facility is paid for by a relative weakness in control structures and some users may have difficulties implementing more complex reasoning strategies.

Another feature of KEE is the object-oriented programming facility, use of which is highly recommended by the promoters of the software. Access-oriented programming is possible by means of ActiveValues, or demons, which are values to which are attached functions that are called whenever the value is accessed. Based on ActiveValues, ActiveImages are graphics images, attached to unit slots, and updated each time the corresponding value is modified. Object-oriented programming is based on “methods” attached to slots; these are run when the slot receives a message of the form:
The Knowledge Engineering Review

(UNITMSG 'unit-name 'slot-name)

One of the best points of KEE is its user interface, which is very sophisticated and user-friendly. It provides a graphical unit editor, a graphical backward chainer explanation facility, several permanent windows, which show what KEE is doing and what the user wants to see. It gives access to all features of KEE through pop-up menus.

Knowledge Craft

Knowledge Craft was developed at Carnegie-Mellon University. Formally called SRL+, it is a superset of the SRL language developed by Mark S. Fox.

The implementation of frames, here called schemata, is very complete in Knowledge Craft. Schemata can represent objects, actions, situations, facts, events; the slots attached to a schema can represent attributes, structures or relations associated with the schema. A schema can be considered by the system either as an individual object or as a class of objects, without any particular declaration. When a schema is taken as a class, it is called a prototype. For example:

A prototype schema
((discrete-machine
  resource-of:
  total-capacity:
  queue:
  status:
  current-order:
  no-of-order-processed:
  total-queue-length:
  maximum-queue-length:
  total-idle-time:
  maximum-idle-time:))

An individual schema
((milling-machine-1
  instance: discrete-machine
  resource-of: cc-rooting-1
  total-capacity: 1
  queue: order 14
  status: busy
  current-order: order 13
  no-of-order-processed: 6
  total-queue-length: 3
  maximum-queue-length: 1
  total-idle-time: 3415
  maximum-idle-time: 1736))

In addition to the basic relations “is-a” and “instance” provided by the system which make it possible to define taxonomies, the user can define specialised relationships as schemata which specify:

- which types of objects it relates,
- its inheritance characteristics,
- selective passing on of slots and values,
- introduction of new slots,
- modification of slots or values which can be passed on,
- the transitivity path.

Complex inheritance paths can be defined by means of user-defined relations and grammars of inheritance paths.

Meta-schemata, meta-slots and meta-values (which are themselves user-defined schemata) can be used to give to schemata, slots and values any kind of characteristics, i.e. meta-knowledge about schemata, slots or values. In particular, meta-slots attached to a particular slot make it possible to specify restrictions on the possible values of the slot, inheritance specification and demons. The meta-knowledge about a slot, can be specified globally in a control schema or locally in a meta-slot of a schema. For example Meta-values can be used to specify the units in use: weight might be in pounds, kilos, tons, stones... etc.

Demons allow access-oriented programming and are also represented as schemata; the demon prototype is provided by the system.

Object-oriented programming is based on message passing between objects. A slot can contain the name of a Lisp function, and message passing consists of remote evaluation of this “method” triggered by the function “call-method”.

Reasoning in Knowledge Craft is provided by CRL-OPS rules or with PROLOG predicates:

CRL-OPS. These rules bring the reasoning power of OPS5 to the package. They combine the representational power of CRL’s schemata, and the reasoning power of OPS5’s rules. A CRL-OPS program works by matching the left-hand side clauses of rules to existing “memory elements” now including schemata. CRL-OPS is particularly suitable for data-driven reasoning, especially when facts may change during reasoning (usually called non-monotonic reasoning).
To select the rule which will be fired in a given cycle, CRL-OPS uses a conflict resolution strategy called MEA (for Means End Analysis) the mechanism is based on three underlying principles: refraction, recency, and specificity.

— refraction: rules which have been applied to a given set of facts cannot be applied again to the same set. (Once a rule has been carried out, the instantiation which corresponds to the triggering of the rule is inhibited.)
— recency: the most recent facts are considered first (focus of attention principle)
— specificity: rules containing a larger number of clauses are considered as more specific, and have priority (a choice of the most constrained rules).

CRL-OPS programming provides a more powerful pattern-matching mechanism than OPS5, since it combines the features offered by OPS5 with the possibilities provided by CRL, in particular the inheritance facilities.

CRL-PROLOG provides a mechanism for intelligent deductive queries and goal-directed reasoning. It is well suited for applications where assertions have to be proved or disproved.

A CRL-PROLOG program is stored in a schema where the current Prolog predicates and the propositions are grouped together in a single slot called “axioms”.

CRL-PROLOG combines the deductive power of Prolog and the representational flexibility of object-oriented programming. Since CRL-PROLOG is accessed through Lisp functions (like (find-all '( query )) or (find-some 'query)), a Prolog program can be called from the right part of a CRL-OPS rule, provided that the Prolog question be preceded by the execution of the function (load-cr-program <program-name>) which “loads”, i.e. initializes the Prolog program.

Hypothetical reasoning is made possible by the context mechanism. Contexts allow the system to save, at any time, as many copies of the Knowledge Base as wanted. In fact, the system actually only saves in memory data which are modified in the new context, thus saving both time and memory. With this feature, the user can test the consequences of a given action without affecting the initial knowledge base.

The context mechanism provides the following facilities:

— managing as many virtual copies of the knowledge base as one wishes.
— reasoning about hypothetical data
— keeping several versions of a knowledge base
— modelling and testing different situations.

By using only these means, very complex control structures can be built, but Knowledge Craft provides yet another feature, which is a multi-queue event manager which enables the user to program by events. Events are actions (represented as schemata) which are programmed to occur at a given time. Four types of queue are provided: schedule-queue, imperative-queue, simulation-queue, manual-queue. This is a very useful feature for implementing simulations or real-time processes.

The user interface in Knowledge Craft is very complete. A graphical schema editor is provided for editing knowledge trees, the PALM-EDITOR. For editing schemata in a more detailed way, a schema editor, P-EDIT, is supplied, which can be graphically called from PALM (i.e. by pointing with the mouse to the name of the schema to edit in the tree edited by PALM) The software also allows the user to build custom-made schema editors, which can be called by simply calling the appropriate method.

For program testing and debugging, two “workbenches” are provided in Knowledge Craft: one for OPS5 programs and one for PROLOG programs. Using these tools, OPS and Prolog programs can be traced, run step by step, or modified. The working memory can be modified between two steps. The conflict set can be displayed. In the OPS-Workbench, the programmer can return to an earlier step in the execution, since each step is stored in a context. Steps can be recalled by clicking on the corresponding icon of the graphical conflict set displayed by Knowledge Craft.

Lastly, many facilities are supplied for developing professional user interfaces: a significant part of Knowledge Craft's internal command interface is directly accessible to the user for its own applications, (implementing a graphical editor like PALM for example is a matter of a couple of hours for a beginner) and it results in a very compact program (one or two pages of Lisp code). For inserting graphics in the interface, the CORE standard is fully supported, allowing the user to build and manipulate complex graphical objects, represented as schemata.
All in all, Knowledge Craft is a very complete tool, but the great diversity of the offered functionalities is paid for by relative complexity and in practice a long training period (at least two months) before being able to use the tool efficiently.

ART

ART was put on the market in 1984 by Inference Corp. It is a powerful tool which combines sophisticated artificial intelligence methodologies and speed.

The representation by schema is less complete than in Knowledge Craft or KEE (neither meta-knowledge nor demons can be attached to slots). But ART can cope with propositions to which are attached truth values (specifying whether a proposition is true, false or unknown) and patterns, which are propositions with variables. These can include wildcards, and Lisp conditions. Goals and strategies in ART are represented as patterns.

Rules are either forward or backward chaining, though the rule compiler gives only one rule representation, which gives ART its speed.

Backward chaining rules are simulated by using goals, patterns and viewpoints.

Viewpoints are the most powerful feature of ART. They are similar in functionality to deKleer's assumption-based truth maintenance system, and can support complex hypothetical reasoning.

Uncertainty in ART can be represented by confidence values attached to viewpoints, or to individual facts.

ART's user interface can show graphical displays of the knowledge base, but does not provide a graphical schema editor similar to KEE's or KC's. A program editor allows a program to be traced. Moreover, the knowledge base can be modified while the program is running.

ART is a powerful and well-integrated expert system tool. Its stronger points are the ease of use of complex AI paradigms, and the speed provided by its knowledge compiler.

COMMON TECHNICAL CHARACTERISTICS

Knowledge representation

The three tools provide similar frame-based knowledge representation with slot inheritance customization, and for two of them, procedural attachment (called active values in KEE, and demons in Knowledge Craft). These data structures help to organise knowledge efficiently.

In all three tools, slot inheritance through different paths of the data tree can be specified by the user, so that a first level of inference can be obtained in this manner: a physical object will usually be defined as an instantiation of predefined classes, represented as "prototype" frames. For instance the object "Renault-billancourt-plant" would be defined as an instantiation of the class "Car-plants" which would be a sub-class of "Plants", itself a sub-class of "Sites". But besides this, the "Renault-Billancourt-plant" object could also be an instantiation of the class "Employers" or of the class "Expensive-necessities", and would thus inherit the properties of both these classes.

In all three tools, the frames allow the user to define his objects rapidly and efficiently in a quite natural way. Moreover the ability to define relations between objects, or at least to define the way in which slots are inherited, allows specific problems such as multiple inheritance and exceptions to be solved simply. The famous example of the ostrich, which is a bird who cannot fly, can be implemented very easily as follows: the class "Birds" has a slot "can-fly" set to True by default. Thus all birds will inherit the value True for the slot "can-fly", except the frame "ostrich" where the slot "can-fly" will be explicitly set to False.

Using the frame-based representation technique, the three tools also provide the means for attaching restrictions on slot values. For example the type (integer, string, list, class...etc...), the range (from 0 to 100) values within a set (blue, red, white) ...etc, or the cardinality, i.e. the maximum number of values that the slot can contain.

Finally, KEE and knowledge Craft provide facilities for attaching procedural behaviour to slots. Demons, as they are usually called, enable the user to build slots, whose value is calculated or modified at the time of access. But this feature not only permits self-adapting slots to be built, it also allows slots to have a role in the inference mechanism by modifying the working memory (the fact base) each time the slot is accessed.

For example access to specific slots can trigger a process such as incrementing a variable containing the number of times the slot has been accessed, or creating a fact which matches the left hand side of a meta-rule. The latter could be used to implement data-driven reasoning.
Another use of demons is the connection of slots to graphical icons in such a way that the graphical image is automatically updated each time the value is changed. This is the way ActiveImages are implemented in KEE and it is a very flexible way of having varying gauges and level meters on the screen.

Knowledge operation

Rules. Each of the tools provides rules as the main inference mechanism but the form and use are quite different. All three tools provide the means for forward-chaining, backward-chaining and some kinds of meta-rules, i.e. rules to control the way other rules are fired. Rules, in the three tools, fully exploit the possibilities of the frame-based knowledge base including property inheritance, active values, constraints attached to slots ... etc... More generally, all the possibilities of the tool, either representational or inferential, are accessible from the rules.

Logic programming and backward chaining. All three tools provide ways of implementing logical assertions and queries though the aspect may differ from one tool to another. Each of KEE's TellAndAsk, ART's language and Knowledge Craft's CRL-Prolog allow logical programming and backward chaining.

In KEE, backward chaining is started by a call to the function QUERY, the argument of which is a fact where variables may replace unknown objects or a class of rules. For example:


In Knowledge Craft, this is a schema-based Prolog, CRL-PROLOG, which gives access to backward chaining. CRL-PROLOG is a DEC10-like Prolog, implemented in Lisp, and accessed through Lisp functions.

In ART another paradigm gives access to backward chaining. Rules which are entered as backward chaining rules are translated into forward chaining rules at compilation by creating a goal corresponding to the left hand side of the rule, and opening a viewpoint to begin the search.

Object-oriented programming. Compared with other well-known object-oriented languages such as Smalltalk-80, the hybrid approach in KEE and in Knowledge Craft gives advantages of both an object representation (which deals with meta-classes by using prototype schemata) and a powerful inference engine.

The integration of object-oriented programming within KEE and Knowledge Craft makes these tools very powerful if they are used in conjunction with their other features. Yet, it seems useful to point out that hybrid tools such as KEE and KC don't provide a real message passing facility as in recent object-oriented languages, like LRO2, where new concepts such as messages sent to unknown receivers, message buses, etc... are introduced. In fact the "message-passing" which is done in these tools is just a remote function evaluation.

Event programming. Events are only available in ART and in Knowledge Craft, where a very complete agenda mechanism is provided. Events, as well as the different mechanisms which allow them to be used, are represented as schemata. They allow the user to create as many tasks as he wishes, which will automatically start at the time specified in the corresponding slots. Events are placed on queues where they wait to be performed. The time taken for reference can be either real-time or a simulated one. This is a valuable feature for implementing simulation programs.

The user-interface

Knowledge editing. KEE, Knowledge Craft and ART provide graphical displays of the frame hierarchy, showing inheritance paths and relations, and give a good overview of the knowledge base. The three tools fully exploit the possibilities of Lisp-machines: windows of any shape and any kind can be added on the screen, including ZMACS windows; all schema names or icons which appear on KEE, KC or ART's windows are mouse-sensitive. The schema editors make it possible to modify schema contents, include meta-knowledge attached to the schema or the slots, and to move along inheritance paths (IS-A and INSTANCE relations). Facilities are provided to make life easier for the programmer: keyword completion, choice in a pop-up menu with the mouse, or icon selection.

Not all interfaces provide the same comfort in use (Knowledge Craft in this respect is perhaps less pleasant than KEE) but they all provide roughly the same features. Facilities for creating new interfaces adapted to user needs are provided, such as Coconut in Knowledge Craft. It is possible to build interfaces for collecting data from the user, interfaces for providing help, possible answers, graphical facilities... etc...
The Knowledge Engineering Review

Program development and supervision aids. When developing a rule-based program it is important to be able to trace and control rule execution, and to intervene while the program is in progress. KEE, KC and ART all provide sophisticated workbenches which enable a program to be followed during its execution, with graphical display of the fired rules, or the viewpoints created in ART's case. Several watch levels are provided to choose the information that one wants to be displayed during a run.

COMPARISON OF THE THREE TOOLS

The strength of KEE: user-friendly and easy to use

KEE appeared to us superior to KC and ART in that KEE is easier to learn and to exploit, and the user interface is very pleasant.

KEE is easy to learn and to exploit. The access to KEE’s knowledge base is made through the TellAndAsk language which is an English-like language and thus easy to understand and to learn. A beginner will remember and use the syntax.

(ASSERT '(THE AGE OF MY.BROTHER IS 28))
more easily than:

(NEW VALUE 'MY.BROTHER 'AGE 28) which is the CRL syntax.

Rules can be expressed either in English-like form, (using words like THE, ALL, OF, IS, IS IN, ARE), attractive for its readability, or in Lisp-like form wherever speed is required. For testing prototypes, the English-like form is very attractive.

KEE rules have two other advantages:

— First, they are represented as schemata and thus can be handled like any other object, which is not quite true in KC and ART where they are compiled. KEE rules are classified in a hierarchical way, like any other unit. This feature makes it possible to select sets of rules for solving specific problems, and tune the conflict set according to user’s needs. This is in fact the only way to control which rules should be fired. It has the drawback of being quite static, since it is not very easy to change the rule hierarchy during the reasoning process.

— Second, the rules can be used either in forward or in backward chaining, which spares the user rewriting rules which are to be used in both ways.

Because of its simplicity and its naturalness, simple applications can be programmed faster with KEE than with KC or ART, where complexity is a handicap. But if problems become more complex, functionalities must be added to KEE by means of Lisp functions and thus KC and ART may be more suitable.

The user interface of KEE is very pleasant. KEE has a better user interface than the others in that it makes full use of graphics and the large screen. All the functionalities of the software are immediate by available in pop-up menus. The screen is well organized and understandable when it is used for development; it can be easily set up for closely following the progress of the program during a run.

However this powerful interface is paid for by relative slowness, which makes KEE less well suited for building big systems.

The strength of KC and ART: problem solving capacities

The strength of KC and ART for problem solving, are:

1. The user can define several levels of meta-knowledge and attach any kind of meta-knowledge in schemas (KC)
2. User-defined relations are supported
3. Temporal reasoning is possible
4. Sophisticated reasoning mechanisms are provided
5. A task can be divided into several interactive subtasks.
Meta-knowledge representation. This section concerns only KC because ART doesn’t provide meta-knowledge except a single-line of documentation at the top of the schemas. In Knowledge Craft, meta-schemas, meta-slots, or meta-values (which are themselves user-defined schemas) can be used to give schemas, slots and values special characteristics. This is not the case in KEE where meta-knowledge is encoded in a fixed form as facets attached to slots. Meta-knowledge in KEE is simpler to use and requires fewer operations, but it is less powerful. In Knowledge Craft, multiple levels of meta-knowledge can be encoded.

User-defined relations. In KC and ART, explicit relationships can be defined between objects or classes of objects, in addition to the basic ones provided like IS-A and INSTANCE.

Among other possible implications, this makes it possible to have, at one time, several different semantic networks built with the same objects but with different relations, and to specify which properties can be passed from one object to another. In the real world, concepts are related by numerous and various relationships, for example: Subtask-of, part-of works-for, works-with...etc... This possibility of defining explicit relationships allows better modeling of the world and is fundamental for applications where complex models must be built, such as factory models for computer integrated manufacturing.

Another obvious application of user-defined relations is in spatial relations (e.g. in-front-of, is-behind, is-at-left-of, is-at-right-of) and temporal relations (e.g. is-at-the-same-time-as, is-before, is-after, is-at-the-beginning-of, is-at-the-end-of, etc...)

Temporal reasoning. The implementation of events in KC and ART provides several advantages:

— operations can be placed on queues and programmed to be fired at a given time
— meta-rules can be fired periodically to control the reasoning process in time
— any incoming data can be treated as events (keyboard inputs, mouse inputs, sensor values).
— a life cycle can be attached to an event, for simulating the time of validity of data items.

Contexts (or viewpoints) can be used as temporal contexts, i.e. contexts which fix the knowledge base at a given time and allow the system to reason about intervals. With this feature conclusions can be dated.

Inference modes. Knowledge Craft and ART make extensive use of forward chaining with backward chaining as an additional mode, while KEE relies heavily on backward chaining.

Another strong point of KC and ART is that rules are precompiled and are much faster than KEE, where rules are interpreted.

The strength of the contexts. Contexts, or viewpoints, are available in KC and ART and are essential for programming a complex knowledge-based system. The reasons for this include:

— Contexts make it possible to represent many possible scenarios associated with hypotheses. It is a basic feature for building a powerful prediction expert system.
— Several versions of a model can be kept, so that comparing different models of a specific object is easy.
— Contexts are essential for hypothetical reasoning as they allow immediate backtracks to be made. For a planning expert system this is of first importance.
— The access to information can be faster since contexts divide the knowledge base into points of view, and allow the user to create sections in the knowledge base.

Meta-rules. For controlling the reasoning process, a tool must provide support for implementing meta-rules. In KC and ART several levels of meta-rules can be implemented by means of goals.

Task management. The task managers in KC and in ART allow the user to create hierarchies of tasks using the same knowledge base; the tasks interact by starting, interrupting, killing other tasks. This feature provides more direct access to secondary tasks, like graphical tasks, inferential tasks, interface tasks..., which must collaborate to be efficient.

A task manager is attractive for handling processes in simulation systems, but also for managing the communications between different cells of "cooperating" expert systems.
CONCLUSION

Our studies and experiments with the three tools have led us to the following overall conclusions:

KEE is a nice tool to use, especially because of its user interface, but the user is quickly limited by the weakness of KEE's inferential features and by its slowness, which is unacceptable for a $50000 tool. The scope of KEE seems to us limited to prototyping. It seems to be difficult to develop professional and large applications with the current version of the software. Let us hope that KEE 3.0, announced for summer 1986, will bring some additional speed, and power with the "KEE-worlds", which should be close to ART's viewpoints.

ART seems to be the most suitable tool for complex applications, and especially for real-time applications (such as intelligent interpretation of sensor data from a factory floor), for which it provides very specific features (non-monotonic-reasoning based on temporal viewpoints, treatment of uncertainty, possibility to believe or to poison a viewpoint...etc...). ART combines the power of Knowledge Craft and the clean integration of KEE, in an elegant homogeneous system. However applications which really need object-oriented programming should not be programmed in ART 2.0, which does not provide it. Another drawback of ART, and not the least, is the price, about $75000, which is 50% more than its competitors, usually priced around $50000 (these are the public prices; large discounts are accorded to universities and educational institutions).

Knowledge Craft has various interesting features, which make it significant for people looking for the maximum at the best price. Knowledge Craft is not so well integrated as its two competitors. The contexts are not as well integrated and efficient as ART's viewpoints and do not provide a Truth Maintenance System, but they are present, and very useful for sophisticated reasoning. It gives an impression of having been built from pieces gathered together, however Knowledge Craft provides the most powerful knowledge representation. Knowledge Craft is currently the only expert system environment commercially available on VAX, on which up to four users can work simultaneously (VAX 11/780). For the VAX, a multi-user database facility is available. On a Lisp-machine, Knowledge Craft provides a good window-based interface. Unlike ART, Knowledge Craft is very open and gives access to many of its internal features.

Finally, our personal feeling would be that looking at the releases available in mid-86 Knowledge Craft is probably the best compromise between representation of power and problem solving power. But each reader may have different requirements so we include their main features under the following headings:

- frame-based knowledge representation,
- rule-based programming,
- logic programming,
- control of reasoning,
- dealing with uncertainty,
- graphics capabilities,
- environments for developers and users

The following tables are intended to help each reader make his own judgement.
<table>
<thead>
<tr>
<th>FRAMES</th>
<th>RULES</th>
<th>OBJECT-ORIENTED PROGRAMMING</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FRAMES</strong></td>
<td><strong>RULES</strong></td>
<td><strong>OBJECT-ORIENTED PROGRAMMING</strong></td>
<td><strong>CONTROL</strong></td>
</tr>
<tr>
<td><strong>KEE</strong></td>
<td><strong>KEE</strong></td>
<td><strong>KEE</strong></td>
<td><strong>KEE</strong></td>
</tr>
<tr>
<td>&quot;List&quot;</td>
<td>&quot;List&quot;</td>
<td>&quot;List&quot;</td>
<td>&quot;List&quot;</td>
</tr>
<tr>
<td>• member slots</td>
<td>• units, members of the</td>
<td>• Active values</td>
<td>• Tell and Ask is an assertion and</td>
</tr>
<tr>
<td>• own slots</td>
<td>rule class</td>
<td>procedural behaviour</td>
<td>query language which allows:</td>
</tr>
<tr>
<td>• multiple inheritance:</td>
<td>• forward or backward</td>
<td>• in particular Active</td>
<td>— creation or removal of facts</td>
</tr>
<tr>
<td>some user-defined possibilities</td>
<td>chaining</td>
<td>images: the graphic</td>
<td>(ASSERT, RETRACT)</td>
</tr>
<tr>
<td>through member links</td>
<td>6 possible conflict</td>
<td>images are controlled</td>
<td>— to ask questions (QUERY)</td>
</tr>
<tr>
<td>or subclass links</td>
<td>resolution</td>
<td>by slot values</td>
<td>— to activate forward or</td>
</tr>
<tr>
<td>• demons attached to slots</td>
<td>strategies:</td>
<td>strategies:</td>
<td>backward chaining for one class</td>
</tr>
<tr>
<td>type checking</td>
<td>— Least Promise Complexity</td>
<td>— Least Promise</td>
<td>of rules and with one of the possible</td>
</tr>
<tr>
<td>integrity constraints</td>
<td>— Greater</td>
<td>Complexity</td>
<td>fixed strategies</td>
</tr>
<tr>
<td></td>
<td>— Weighted Least Promise</td>
<td>— Weighted Least</td>
<td>(ASSERT, QUERY)</td>
</tr>
<tr>
<td></td>
<td>Complexity</td>
<td>Weight</td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Least Weight</td>
<td>Greatest Weight</td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Greatest Weight</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• rules are interpreted</td>
<td>slow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• KC</td>
<td>• CRL-OPS combines the</td>
<td>• KC demons are</td>
<td>• Possibilities of CRL-OPS in</td>
</tr>
<tr>
<td>&quot;schema&quot;</td>
<td>representational power of</td>
<td>procedural attachments</td>
<td>forward chaining</td>
</tr>
<tr>
<td></td>
<td>the schemata with the reasoning</td>
<td>similar to the Active</td>
<td>and of CRL-Prolog in backward</td>
</tr>
<tr>
<td></td>
<td>power of the well-known OPS-5</td>
<td>values of KEE. They</td>
<td>chaining, mixing is possible.</td>
</tr>
<tr>
<td></td>
<td>engine (good integration of</td>
<td>allow calls to LISP</td>
<td>— Multisource Event Manager</td>
</tr>
<tr>
<td></td>
<td>chaining and of inheritance</td>
<td>functions.</td>
<td>Events are schemas that may be</td>
</tr>
<tr>
<td></td>
<td>properties)</td>
<td></td>
<td>placed on an agenda. Their slots</td>
</tr>
<tr>
<td></td>
<td>• rules are compiled in an</td>
<td>may include the event's assigned</td>
<td>can include the event's assigned</td>
</tr>
<tr>
<td></td>
<td>internal network</td>
<td>queue, the prescribed time the</td>
<td>target, the prescribed time the</td>
</tr>
<tr>
<td></td>
<td>faster</td>
<td>event will take place... Time can</td>
<td>event will take place... Time can</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be simulated or real time.</td>
<td>be simulated or real time.</td>
</tr>
<tr>
<td>• ART</td>
<td>• ART</td>
<td>• ART</td>
<td>• ART</td>
</tr>
<tr>
<td>&quot;schema&quot;</td>
<td>&quot;schema&quot;</td>
<td>&quot;schema&quot;</td>
<td>&quot;schema&quot;</td>
</tr>
<tr>
<td>• multiple inheritance</td>
<td>• multiple inheritance</td>
<td>• multiple inheritance</td>
<td>• multiple inheritance</td>
</tr>
<tr>
<td>• user-defined relations</td>
<td>• user-defined relations</td>
<td>• user-defined relations</td>
<td>• user-defined relations</td>
</tr>
<tr>
<td>with their inverses and</td>
<td>with their inverses and</td>
<td>with their inverses and</td>
<td>with their inverses and</td>
</tr>
<tr>
<td>their inheritance properties</td>
<td>their inheritance properties</td>
<td>their inheritance properties</td>
<td>their inheritance properties</td>
</tr>
<tr>
<td>• no demons, no meta-knowledge</td>
<td>• no demons, no meta-knowledge</td>
<td>• no demons, no meta-knowledge</td>
<td>• no demons, no meta-knowledge</td>
</tr>
<tr>
<td>to ART: the assign is more</td>
<td>to ART: the assign is more</td>
<td>to ART: the assign is more</td>
<td>to ART: the assign is more</td>
</tr>
<tr>
<td>on facts and rules than on</td>
<td>on facts and rules than on</td>
<td>on facts and rules than on</td>
<td>on facts and rules than on</td>
</tr>
<tr>
<td>classes and objects</td>
<td>classes and objects</td>
<td>classes and objects</td>
<td>classes and objects</td>
</tr>
<tr>
<td>OBJECT-ORIENTED PROGRAMMING</td>
<td>CONTROL</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ART</strong></td>
<td><strong>KEE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Nothing in 2.0</td>
<td>• Active values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(but announced in</td>
<td>procedural behaviour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>release 3.6)</td>
<td>attached to slots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• no possibility to</td>
<td>through a call to a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>state constraints about</td>
<td>LISP function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>slot values or to attach</td>
<td>• in particular Active</td>
<td></td>
<td></td>
</tr>
<tr>
<td>procedural behaviour</td>
<td>images: the graphic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• In ART no object-oriented</td>
<td>images are controlled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>programming.</td>
<td>by slot values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming is rule-oriented</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LOGIC PROGRAMMING**

<table>
<thead>
<tr>
<th>KEE</th>
<th>ART</th>
</tr>
</thead>
<tbody>
<tr>
<td>• nothing</td>
<td>• nothing</td>
</tr>
</tbody>
</table>

**CONTROL**

<table>
<thead>
<tr>
<th>KEE</th>
<th>ART</th>
</tr>
</thead>
<tbody>
<tr>
<td>• CRL-Prolog may be</td>
<td>• The possibilities for sophisticated</td>
</tr>
<tr>
<td>used to exploit the</td>
<td>control and strategies are very</td>
</tr>
<tr>
<td>schemata in backward</td>
<td>large as implemented by means</td>
</tr>
<tr>
<td>chaining, to define</td>
<td>of goals and strategy patterns</td>
</tr>
<tr>
<td>Prolog predicates and</td>
<td>within the rules.</td>
</tr>
<tr>
<td>to build real Prolog</td>
<td>• Events - the viewpoint mechanism</td>
</tr>
<tr>
<td>programs</td>
<td>is comparable with the context</td>
</tr>
<tr>
<td></td>
<td>mechanism of KC, but easier</td>
</tr>
<tr>
<td></td>
<td>and faster. The belief rules allow</td>
</tr>
<tr>
<td></td>
<td>hypothetical reasoning.</td>
</tr>
<tr>
<td>UNCERTAINTY</td>
<td>GRAPHICS</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>KEE</strong></td>
<td><strong>KEE</strong></td>
</tr>
<tr>
<td>• Not provided must be programmed</td>
<td>• objects icons modifications = modifications of objects • very interesting feature for interactive simulation</td>
</tr>
<tr>
<td>• as KEE</td>
<td></td>
</tr>
<tr>
<td><strong>KC</strong></td>
<td><strong>KC</strong></td>
</tr>
<tr>
<td></td>
<td>• certainty factors are used in the KB and a mechanism is embedded to deal with them</td>
</tr>
<tr>
<td>• ART</td>
<td>• as KEE</td>
</tr>
<tr>
<td>• ART</td>
<td>• ART</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• certainty factors are used in the KB and a mechanism is embedded to deal with them</td>
</tr>
<tr>
<td></td>
<td>• confidence factors for the viewpoints that may be used for control</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY


CLAYTON B.: ART, Automated Reasoning Tool, Programming primer, 1985, Inference Corporation, Los Angeles, California, U.S.A.

CLAYTON B.: ART, Automated Reasoning Tool, Programming Tutorial, volume 1, 2, 3, 1985, Inference Corporation, Los Angeles, California, U.S.A.


INTELLICORP: KEE, Software development system user's manual KEE version 2.1, 1985, Intelllicorp, U.S.A.


ROCHE C. LAURENT J.P.: LRO2: Artificial intelligence and object oriented languages: 10th World Computer Congress, IFIP '86, Dublin, IRELAND

VIGNARD P.: Representation des connaissances, mécanisme d'exploitation et d'apprentissage, INRIA, Roquencourt, FRANCE
H. REICHLGELT, F. VAN HARMELIN
Criteria for Choosing Representation Languages
and Control Regimes for Expert Systems
.................................................. 2

J.P. LAURENT, J. AYEL, F. THOME, D. ZIEBELIN
Comparative Evaluation of Three Expert System Development Tools:
Kee, Knowledge Craft, Art
.................................................. 19

B.P. BLOOMFIELD
Capturing Expertise by Rule Induction
.................................................. 31