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)

©*J.P. Laurent, J. Ayel, F. Thome, D. Ziebelin*

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 Raust (CGE/TITN) for their participation in the evaluation of KEE and ART respectivily.

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 suport (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-Vax-worstations, 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 exampls 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:

(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. Formely 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 on a given set of facts cannot be applied again o 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 numer 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-crp-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 with out 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 canbe 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 progam 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 instantiationof 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 possibliities 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:

(QUERY '(THE state-of-health OF my-patient IS ?what) 'medical-rules)

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... .

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 inteface 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 additon 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 possibliity of defining explicit relationships allows better modelling 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, possiblility 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: 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.

FRAMES

KEE

"Unit"

- member slots
- own slots
- multiple inheritance: some user-defined possibilities through member links or subclass links
- demons attached to slots type checking integrity constraints

KC

"schema"

- multiple inheritance with complex, user-defined possiblities (transitivity, restrictions,...)
- user-defined relations (ex: part-of, worker-of, above, before,...) Their attributes define inheritance properties
- demons
- metaknowledge about schemas and slots (metaschemas and metaslots) to describe inheritance of procedural behaviours

ART

"schema"

- multiple inheritance
- use-defined relations with their inverse and their inheritance properties
- no demons, no metaknowledge in ART the accent is more on facts and rules than on classes and objects

RULES

KEE

- = units, members of the rule class
- forward or backward chaining
- 6 possible conflict resolution strategies:
  - — Least Premise Complexity
  - — Greatest
  - — Weighted Least Premise Complexity
  - — Least Weight
  - — Greatest Weight
- rules are interpreted slow

KC

- CRL-OPS combines the representational power of the schemata with the reasoning power of the well-known OPS-5 engine (good integration of chaining and of inheritance properties)
- rules are compiled in an internal network faster

ART

- may mix forward and backward chaining by combining goals and facts within a rule
- — inference rules
  - — production rules
  - — hypothesis rules
  - — constraints rules
  - — belief rules
- rules are compiled fast

OBJECT-ORIENTED PROGRAMMING

KEE

- Active values procedural behaviour attached to slots through a call to a LISP function
- in particular Active Images: the graphic images are controlled by slot values

KC

- KC demons are procedural attachments similar to the Active values of KEE. They allow calls to LISP functions
- Metaschemas and metaslots increase the possibilities of Object-Oriented Programming

ART

- Nothing in 2.0 (but announced in release 3.0.) no possibility to state constraints about slot values or to attach procedural behaviour
- In ART no object-oriented programming. Programming is rule-oriented

LOGIC PROGRAMMING

KEE

- nothing

KC

- CRL-Prolog may be used to exploit the schemas in backward chaining, to define Prolog predicates and to build real Prolog programs

ART

- nothing

CONTROL

KEE

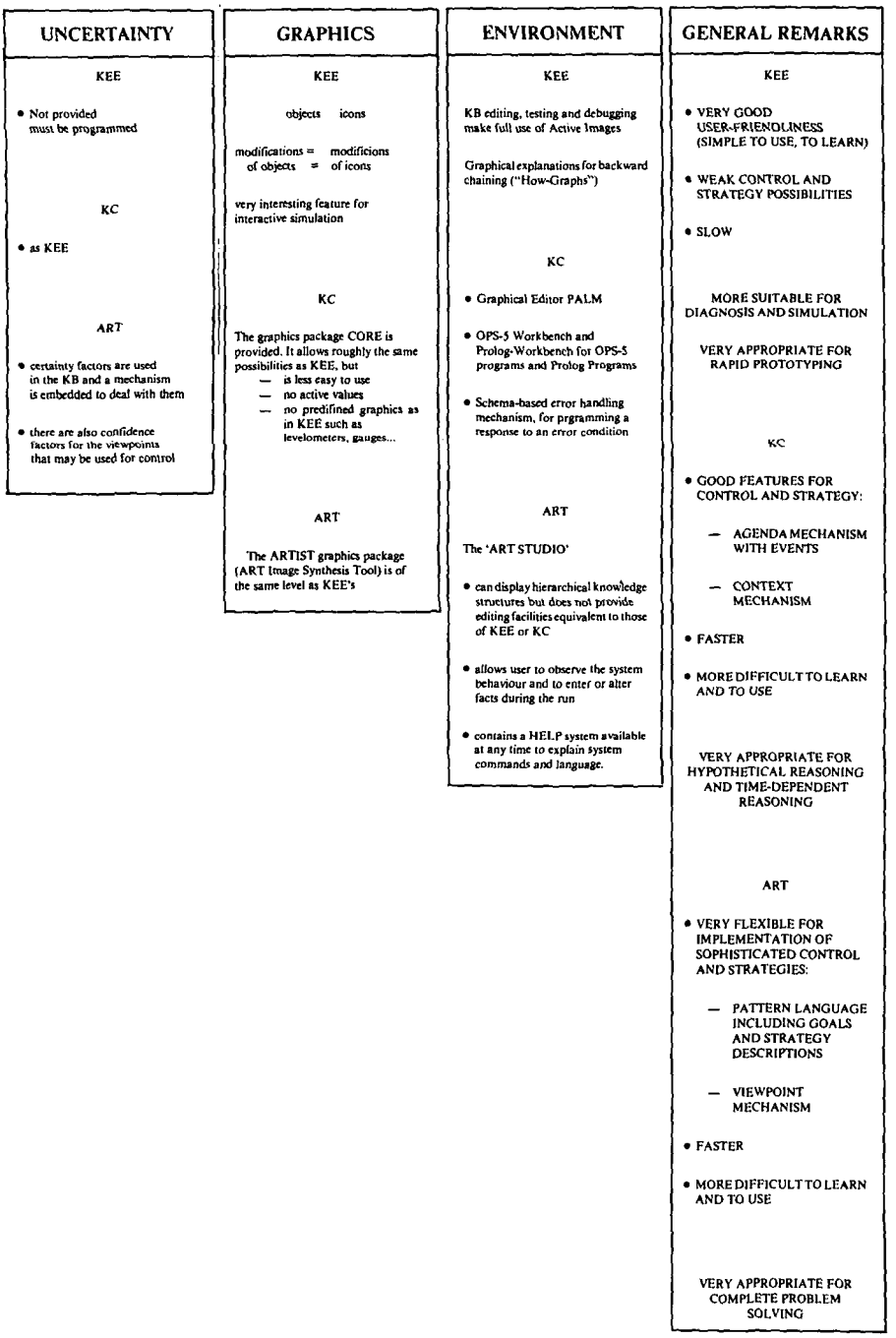
- Tell and Ask is an assertion and query language which allows:
  - — creation or removal of facts (ASSERT, RETRACT)
  - — to ask questions (QUERY)
  - — to activate forward or backward chaining for one class of rules and with one of the possible fixed strategies (ASSERT, QUERY)

KC

- Possibilities of CRL-OPS in forward chaining and of CRL-Prolog in backward chaining. Mixing is possible.
- Multiqueue Event Manager Events are schemas that may be placed on an agenda. Their slots may include the event's assigned queue, the prescribed time the event will take place,...Time can be a simulated or real clock. this provides direct support for event-based simulations of complex processes.
- Contexts. The creation of one context allows preservation of the current state of the KB. The modifications on the current KB state are only valid for this context and its successors. The context mechanism allows restoring of previous states if reasoning seems no more promising (intelligent backtracking) This feature is very important for NON-monotonic reasoning and for hypothetical reasoning.

ART

- The possibilities for sophisticated control and strategies are very large as implemented by means of goals and strategy patterns within the rules.
- Events = the viewpoint mechanism is comparable with the context mechanism of KC, but easier and faster. The belief rules allow hypothetical worlds.


UNCERTAINTY
KEE
• Not provided must be programmed
KC
• as KEE
ART
• certainty factors are used in the KB and a mechanism is embedded to deal with them
• there are also confidence factors for the viewpoints that may be used for control
GRAPHICS
KEE
objects icons
modifications = modifications of objects = of icons
very interesting feature for interactive simulation
KC
The graphics package CORE is provided. It allows roughly the same possibilities as KEE, but
— is less easy to use
— no active values
— no predifined graphics as in KEE such as levelometers, gauges...
ART
The ARTIST graphics package (ART Image Synthesis Tool) is of the same level as KEE's
ENVIRONMENT
KEE
KB editing, testing and debugging make full use of Active Images
Graphical explanations for backward chaining ("How-Graphs")
KC
• Graphical Editor PALM
• OPS-5 Workbench and Prolog-Workbench for OPS-5 programs and Prolog Programs
• Schema-based error handling mechanism, for prgramming a response to an error condition
ART
The 'ART STUDIO'
• can display hierarchical knowledge structures but does not provide editing facilities equivalent to those of KEE or KC
• allows user to observe the system behaviour and to enter or alter facts during the run
• contains a HELP system available at any time to explain system commands and language.
GENERAL REMARKS
KEE
• VERY GOOD USER-FRIENDLINESS (SIMPLE TO USE, TO LEARN)
• WEAK CONTROL AND STRATEGY POSSIBILITIES
• SLOW
MORE SUITABLE FOR DIAGNOSIS AND SIMULATION
VERY APPROPRIATE FOR RAPID PROTOTYPING
KC
• GOOD FEATURES FOR CONTROL AND STRATEGY:
— AGENDA MECHANISM WITH EVENTS
— CONTEXT MECHANISM
• FASTER
• MORE DIFFICULT TO LEARN AND TO USE
VERY APPROPRIATE FOR HYPOTHETICAL REASONING AND TIME-DEPENDENT REASONING
ART
• VERY FLEXIBLE FOR IMPLEMENTATION OF SOPHISTICATED CONTROL AND STRATEGIES:
— PATTERN LANGUAGE INCLUDING GOALS AND STRATEGY DESCRIPTIONS
— VIEWPOINT MECHANISM
• FASTER
• MORE DIFFICULT TO LEARN AND TO USE
VERY APPROPRIATE FOR COMPLETE PROBLEM SOLVING

BIBLIOGRAPHY

L. BROWSTON, R. FARELL, E. KANT, N. MARTIN: Programming expert systems in OPS5, an introduction to rule-based programming. Addison-Wesley Publishing Company, 1985, U.S.A.

CARNEGIE GROUP: Knowledge Craft 3.0, reference manual, volume 1, 2, 1985, Carnegie Group Inc, Pittsburgh, Pennsylvania, U.S.A

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.

DE KLEER J.: An assumption-based TMS, Draft of report, 1985. Intelligent systems Laboratory, Xerox Palo Alto Research Center, Palo Alto, California, U.S.A.

FORGY C.: On the efficient implementation of production systems, Ph. D. Thesis, 1979, Carnegie-Mellon University, Pittsburgh, Pennsylvania,

FORGY C.: OPS5 USER'S MANUAL, Technical Report CMU-CS-18-135, 1981, Carnegie-Mellon University, Pittsburgh, Pennsylvania, U.S.A.

FOX M.: On inheritance in Knowledge representation, IJCAI 1979, p282-284.

FOX M.: Reasoning with incomplete knowledge in resource limited environments; integrating reasoning and knowledge acquisition, *IJCAI 1981, p 313-318.*

FOX M.: Constraint-directed search: A case study of job-shop scheduling, *Ph. D. Thesis, 1983, Carnegie-Mellon University, Pittsburgh, Pennsylvania, U.S.A.*

F. HAYES-ROTH, D. WATERMAN, D. LENAT: Building expert Systems *Addison-Wesley 1983, U.S.A.*

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

LAURENT J.P.: Control structures in expert systems. *Technology and Sciences of informatics, vol.3. P 147-162, 1983, Afcte-bordas, Paris, FRANCE*

RICHER M.: An evaluation of expert system development tools, *Report °KSL 85-19 1986, Knowledge Systems Laboratory, Stanford University, Stanford California, U.S.A*

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

VIGNARD P.: Representation des connaissances, mécanisme dèxploitation et đapprentissage, INRIA, Roquencourt, FRANCE