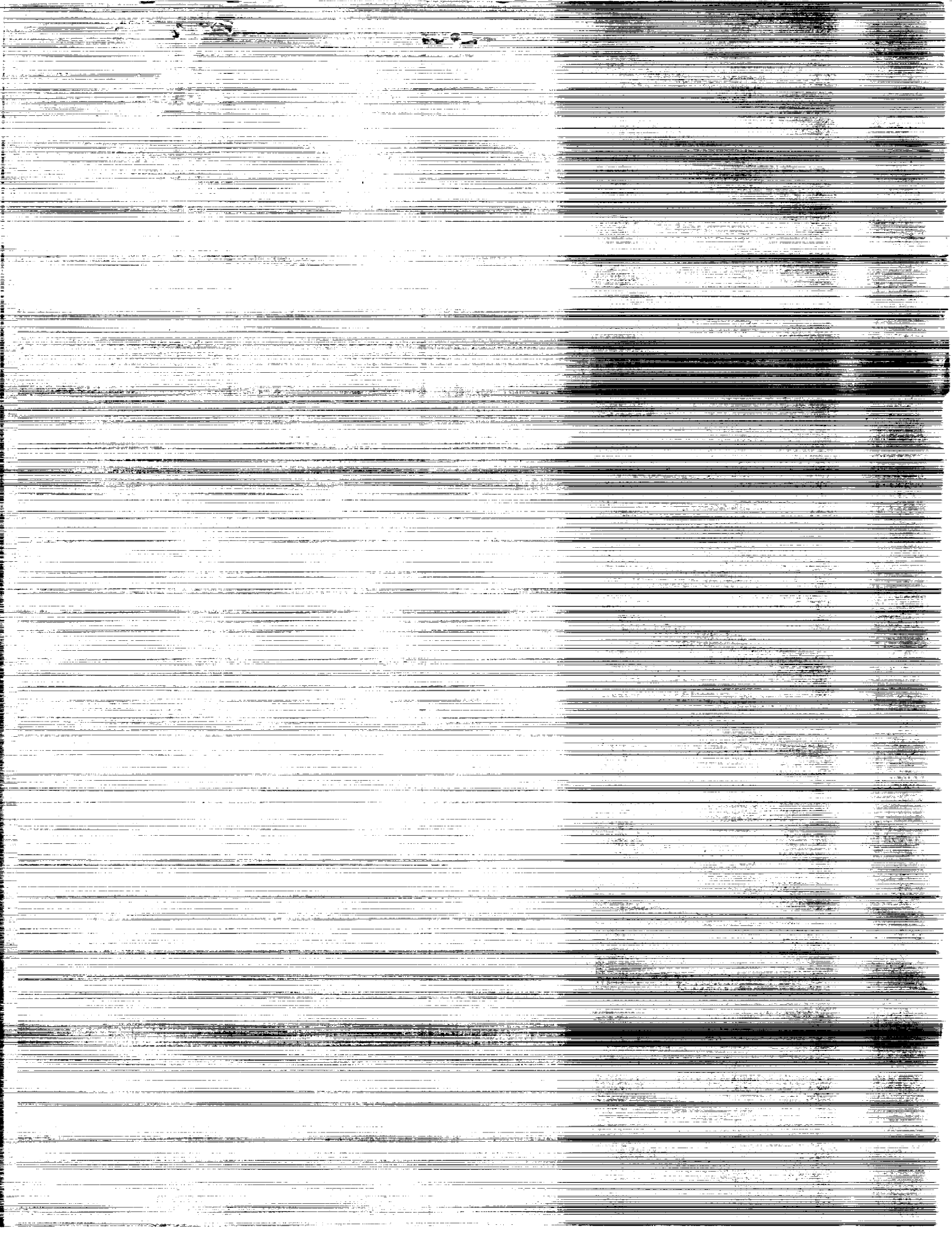
NASA Contractor Report 4273

Knowledge Management: An Abstraction of Knowledge Base and Database Management Systems

Joel D. Riedesel

CONTRACT NAS8-36433
JANUARY 1990



NASA Contractor Report 4273

Knowledge Management: An Abstraction of Knowledge Base and Database Management Systems

Joel D. Riedesel
Martin Marietta Astronautics Group
Denver, Colorado

Prepared for
George C. Marshall Space Flight Center
under Contract NAS8-36433

National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Division

1990

TABLE OF CONTENTS

	Page
INTRODUCTION	1
ARTIFICIAL INTELLIGENCE APPLICATION REQUIREMENTS	2
THE KNOMAD ARCHITECTURE	3
EXAMPLE	8
CONCLUSION	12
ACKNOWLEDGEMENTS	12
REFERENCES	13
KNOMAD BNF SPECIFICATION	14

1 Introduction

Early Artificial Intelligence (AI) work centered around domain *independent* representation and reasoning tools. Examples include General Problem Solver [22] and general purpose theorem provers such as resolution theorem proving [26]. These early approaches included powerful knowledge representation languages and sound and complete search methods. However, as large problems were applied these early programs proved to be intractable.

To manage the complexity and tractability of large domains there was a shift to domain *dependent* systems, primarily expert systems. These domain dependent systems could then apply domain heuristics instead of general search and restricted knowledge representation languages with inherent bias to reduce the search space [4, 28]. Although these domain dependent expert systems worked quite well, expert system technology and knowledge representation of various expert system shells did not transfer well from one domain to the next. Witness the diversity of expert system shells currently existing in the commercial marketplace. The number of shells residing in the university research labs are probably an order of magnitude larger. This diversity is a result of two main restrictions put on the knowledge representation language. One, the restriction on knowledge that can be described using the language of the shell and two, the restriction on mechanisms of inference over the knowledge. Because of these restrictions, it became very important to analyze a domain very carefully to be certain that it could be represented in an existing shell. If a good match was not found two approaches could be taken; one, the shell could be used anyway and the application *bent* to fit it, or two, a home-grown shell that better matched the application could be built. These domain dependent solutions turned out to be very tractable while trading generality. This does not only apply to expert system shells, but to software and AI tools in general [17].

Recently there has been a better understanding of the issues of search and heuristics and how they can be used together to solve complex problems. An example system is SOAR which does large amounts of search initially to solve a problem but learns from the solutions found. The learned solutions may then be applied as heuristics for future problems [15]. The point of all this being that heuristics can eliminate the need for search while search compensates for the lack of heuristics [23]. To make use of this concept in AI tools, the tools must be capable of supporting the representation of domain dependent heuristics in a general fashion. The tools need to have heuristic adequacy [33] as well as general search mechanisms. They need to be very general with the intent that the knowledge engineer specializes or instantiates the tool to the specific application, thus making the tool efficient. The current generation of AI tools do not support this sort of engineering environment[1].

Real world problems need more than just heuristic adequacy in a tool. They need a collection of tools for representing and reasoning about the various aspects that make up the problem. No one tool will be able to manage a large scale, real world problem [17, 16]. Furthermore, a collection of tools ought to be expandable and modular. They ought to be able to reason about the same data and share data easily.

The Knowledge Management Design System (KNOMAD), described here, more properly belongs to the next generation of AI tools in the applications world. It may be considered as a knowledge engineering *environment*. It provides a set of tools including rule management, database, constraint system, frame system, model support, and so forth. These tools are organized in a modular fashion so that more tools may be added and other tools may be removed or substituted as the application requires. The rule management system tool, for example, provides a powerful rule language while also providing mechanisms to keep the tool efficient.

[1] Although [32] is a step in the right direction.

2 Artificial Intelligence Application Requirements

The wide variety of available applications impose a large and diverse set of requirements on necessary tools for encoding knowledge about them as well as performing inference over them. Furthermore, most applications require more than one tool to represent the necessary knowledge. The following is a list of tools, some set of which will be required by any application:

- A model building tool
- A database for both distributed and local data
- Procedural representation and execution
- Abstraction for both data and procedures
- Scripts and frames
- Object oriented programming
- Constraint posting and propagation
- Analytic, qualitative, and quantitative reasoning
- Semantic nets
- Rule management

In addition, representation of both temporal and non-monotonic knowledge may need to be mixed in with some of the above tools as an application requires.

Combinations of these tools may be put together to build applications in planning and prediction domains, diagnosis domains, and control domains, for example.

2.1 Expressibility and Efficiency

Expressivity has at least two meanings that may be distinguished. One meaning of expressiveness asks whether a language allows something to be said. In this meaning various languages can be compared in terms of formal language theory and classified as Type 0, 1, 2, or 3 [19]. We find that most computer languages are Turing equivalent and therefore it becomes moot to compare languages on the simple basis of whether or not something can be said in them. The other meaning of expressiveness asks how easy can it be said, how clear or understandable is it using the language? This meaning has a tendency to be more subjective but may also be seen as a variation of the first. For example, anything can be expressed using propositional logic, the problem being that an exponential number of statements may be needed to say it. First order predicate calculus can immediately get rid of the exponential number of statements (perhaps at the loss of clarity). It is the second meaning of expressivity, that is, how naturally can something be stated, that is important here.

As a language becomes more expressive, the problem of tractability becomes more important. There are two aspects to managing this problem. The first aspect is to manage tractability by providing heuristic adequacy in the language [33]. This method manages tractability by using heuristics in place of search; providing direct paths to a solution. The second method is by providing efficiency in the language itself, for example, the RETE network in OPS5.

There are at least three options available for managing the expressivity versus tractability problem when using general search methods [18]. First Order Logic (FOL) will be used to represent complete expressivity. Most will agree that practically anything can be represented in FOL. Furthermore, resolution theorem provers have been built which are sound and complete and will derive a proof in FOL if the proof exists in the theory. The only problem is that the theorem prover may take an exponentially long period

of time deriving the proof. It may not even return at all if the proof does not exist. The alternatives are to: One, restrict what the language can express (e.g. allow only conjunction) or two, restrict the definition of implication (or derivability). One way of doing this is to define an implication operator that is sound but incomplete. Finally, a third alternative is to use defaults and assumptions. Defaults and assumptions are probably the least understood option. In fact, it is even possible that the use of defaults and assumptions (introducing non-monotonicity) may be more intractable than monotonic reasoning.

Common mechanisms for restricting the expressivity of the representation language are by the use of databases, logic programs, versions of FOL, expert system languages, etc. The power of machine learning programs are also usually a function of what they can represent. The representation language in combination with the domain knowledge defines the size of the search space [12]. Limits on knowledge expression determine bounds on the size of the search space. Thus, this restriction on knowledge representation is a mechanism for reducing the size of the search space.

The most common way of restricting implication is to define methods of inference that are sound but not complete. One of the easiest ways to do this is to not allow backtracking. For example, in the area of expert systems, there may be more than one rule in the conflict set. During forward chaining only one rule may be allowed to fire from the conflict set. Once this rule is fired no backtracking is allowed to try the other rules. Therefore, this strategy can be seen to be sound but not complete. All possible solutions may not be found.

Finally, defaults and assumptions may be used to infer knowledge without having to explicitly derive it using an inference strategy. Knowledge is available due to inheritance and other mechanisms. This approach helps tractability by eliminating the need to infer defaulted and assumed knowledge. On the other hand, managing defaults and assumptions (as well as inheritance with exceptions) may cause some intractability problems of its own.

3 The KNOMAD Architecture

To solve the problems of flexibility and expressivity the KNOMAD architecture was developed. While trying to be both an expressive and powerful language, efficiency was also a primary issue. The KNOMAD architecture is a layered architecture as shown in Figure 1. The central component is the database, a place for storing working memory data, for transferring and sharing data, and for storing long term data. The database is designed as a module and may be implemented as a distributed database. A distributed database may then be used to support multiple cooperating knowledge agents, each residing in different physical locations. The next layer is an interface to the database that provides a frame system for abstracting both data and procedure as well as a mechanism for storing simple facts. The top layer is where various tools are defined and implemented. All the tools make use of the same data representation and may thus easily share data across domains and functions.

The architecture will be discussed in some detail in the following sections including aspects of implementation and efficiency. KNOMAD's current syntax is given in the appendix.

3.1 The Database

The first level of support for KNOMAD is the database. The KNOMAD architecture views the database as a plug compatible module. This provides the ability to take advantage of existing databases and database management mechanisms such as ORACLE or INGRES, for example.

KNOMAD takes a tuple space view of the world and requires that the database represent them. A tuple is an ordered sequence of (possibly) typed fields. Conceptually, a tuple may be thought of as an object that exists independent of the process that created it. This implies that a place to store it is needed if it is to live a life independent of processes used to create and destroy it. The tuple space is sufficient for representing FOL and also sufficient for representing at least relational databases [24].

The tuple space has also been proposed as a mechanism for supporting distributed processing [5]. There are three basic mechanisms that a process may perform on tuples in the tuple space: **IN**, **OUT**,

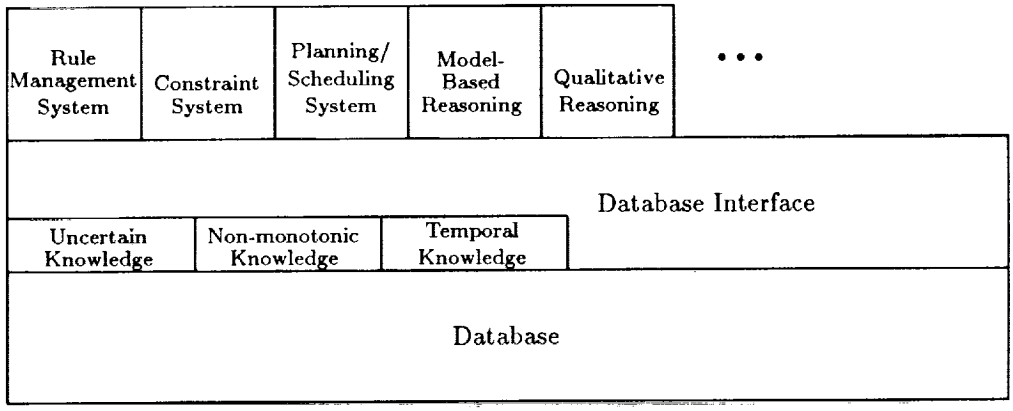


Figure 1: KNOMAD Layered Architecture

and **READ**. The tuple fields in the **IN** and **READ** operations may optionally contain variable arguments for matching. **IN** and **READ** block until the tuple is present in the database. **IN** subsequently removes the tuple from the database while **READ** does not. **OUT** installs a tuple in the database. The benefit to distributed processing that the tuple space provides is the ability to add processes to the environment and have them communicate with existing processes without first having to encode knowledge about existing processes directly into their formalism. It has also been proposed that these basic tuple space operations are sufficient for supporting other communication mechanisms such as those defined in contract nets and Actors. On the other hand, the use of **IN** must be judicious in order to support the addition of other processes that may also need to use the tuple. See [5] and [20] for more details.

The database provides three operations similar to the tuple space operations; these are: **STORE**, **RETRIEVE/MATCH**, and **DB-REMOVE**. The major difference between these operators and the tuple space operators **IN**, **READ**, and **OUT**, is that these do not block. Another difference is that while **IN** and **READ** will non-deterministically select one of multiple matching tuples, **MATCH** will return all of them (**RETRIEVE** will behave non-deterministically). Although the database operations defined here are not identical to the tuple space operations in the LINDA model, it is probably sufficient for handling distributed processing protocols as the LINDA model [5].

This view of the database allows for many different implementations of the database to be used, including distributed databases. A distributed database would provide the ability to support distributed knowledge agents defined and managed using KNOMAD. Their communication is then supported by the tuple space mechanism as implemented in the distributed database.

The implementation of the database for KNOMAD also supports a form of integrity constraints and a restricted version of views. An integrity constraint is defined by a tuple. Fields of the tuple may contain one of four items: An actual field descriptor (formal argument), a variable (to match an argument), a type specifier—meaning that the field may match tuples with the corresponding field of the same type, and finally, a type declarator—meaning that for all tuples matched on the other three options, this field must be of the declared type. Views provide different databases for storage and retrieval. This is quite different to the traditional use of views in most databases. What this provides is a way to organize data into more logical groupings. Currently, the use of integrity constraints is global across all views, in the future this should use the view mechanism just as any other storage or retrieval operation does.

3.2 The Database Interface

The database interface is the module that links the database to KNOMAD and KNOMAD to the database. The interface provides the necessary hooks for proper accessing and notification of data in the database and of interest to KNOMAD. The database interface provides the operations **STORE!**, **REMOVE!**,

MATCH! and RETRIEVE!. It also provides a data and procedure abstraction mechanism—a frame system [13, 21, 29]. Frames are used to extend the knowledge representation language of the various tools for supporting complex domains requiring abstraction mechanisms.

Frames are an abstract organization of data into conceptual units. In this definition data may also be procedure if it is subsequently executed. A frame may have any number of slots. Frames may be defined as children of multiple parents (making inheritance potentially more complex) and may also have code attached to them that is executed whenever a new instance of the frame or one of its children is created. Slots have six optional aspects. The most used aspect is the `:value` aspect. This aspect is where a value for the slot is located. The `:if-needed` aspect is used to store code that is executed if a slot value is asked for. The `:if-added` aspect is used to store code that is executed whenever the slot gets a new value. There are two aspects that are used to constrain the value of the slot. The `:constraint` aspect is used to store code that checks if an added value passes the constraint. Since this is user-defined code that is used here there is no restriction on what it may do, only that it return a true or false status indicating the result of the constraint. The `:mustbe` aspect constrains the value to be one of a list of formal values or frames. Finally, the `:distribution` aspect is used to determine if the value of the slot is for global distribution, accessible to all knowledge agents, or only for local usage.

Frame data is stored in the database as 5-tuples of the form: `(frame <name> <slot> <aspect> <value>)`. Obviously, there is no restriction on the value aspects, they can be any normal data type as well as executable code. Facts are stored in the database as 4-tuples: `(fact <name> :value <value>)`.

Frame inheritance information must also be stored in the database so that inheritance over frames may take place in response to different knowledge agents requests for data values. The current implementation does not yet store this data in the database.

It is the responsibility of the database interface to both store and retrieve information and to notify the various tools of changes to data made by other knowledge agents. Thus if knowledge agent 1 makes a change to the value of a fact and knowledge agent 2 uses that fact on the left hand side (LHS) of a rule, it is the responsibility of the database interface (and distributed database) to notify knowledge agent 2 of the changed fact.

3.3 The Tool Layer

The tool layer is where the various reasoning tools are defined. This is where the rule management system, the constraint system, the planning system, the model-based reasoning system, the qualitative reasoning system, etc. are defined and implemented. Each of these various tools may define different language representations and certainly define different reasoning mechanisms. However, all the tools must use the underlying database and interface for storage and retrieval of data [14, 27, 2, 6, 31, 1, 30, 8, 7].

The power and flexibility of KNOMAD is in its layered architecture and in the power and flexibility of the various tools. The rule management system is the only tool that has been implemented so far. The following section describes this tool in detail and makes apparent the power of the rule language and flexibility of inference that the rule system has.

3.3.1 The Rule Management System

The rule management system consists of two layers. The base layer is the rule language. This language defines what rules are and how they are evaluated and interpreted. The second layer defines a knowledge base management system (KBMS) that is built on top of the rule language. While the KBMS is a layer on top of the rule language, it does not add anything to it; it is completely defined in terms of the rule language[2]. The correct way to think of this is that the rule language is the basic level of expression for the knowledge engineer. The knowledge engineer then writes a number of rules that define the KBMS.

[2]However, to make this work efficiently there has been some optimization. The rule language makes certain assumptions about the existence of certain KBMS domain constructs. An example of this is the assumed existence of a rule-group frame with a "rules" slot and a "*lhs-tickled-queue*" slot.

This KBMS is like another shell that may be instantiated with a domain application, thereby defining an expert system.

In KNOMAD, the rule language and the KBMS work hand in hand. The KBMS is a predefined object available to the knowledge engineer. However, the KBMS definition may be easily modified and extended for special applications. Furthermore, control strategies and conflict resolution strategies may be defined using the rule language itself or by writing procedures (a kind of compiled form of the strategy). These user definable strategies in combination with the expressibility of the rule language provide the power to the rule management system for application to a wide variety of domains.

The KBMS defines a knowledge base to consist of a number of rule groups as well as domain knowledge. Each rule group provides mechanisms for defining aspects of the inference strategy using either further rule groups or user-defined functions. A rule group also has a mechanism for specifying the level of determinism desired over rule execution [3], [11]. A mechanism for explicitly stating a level of deterministic control over a set of rules is one of the things missing from many expert system shells. Without this, the only way to get the necessary control is by inserting control knowledge into the rules themselves—where it does not belong. By explicitly stating the difference between control over knowledge and the knowledge itself, the maintenance problem becomes easier. The way control is added is by examining the inference cycle of an expert system. Basically, the rule group inference strategy consists of a match, evaluate, and fire cycle. The match phase is supported by the database interface which automatically queues up potential rules on the tickled-queues of the rule group. The evaluate phase then checks the rules on the tickled-queues to determine which ones belong in the conflict set. Finally, one rule is selected and fired. This approach is completely non-deterministic in selecting which rules get fired from the conflict set. In between the match and evaluate phases a control phase is added. This control phase consists of the definition of a regular expression that defines which rules may be examined and possibly fired next. In this implementation the transition table derived from the regular expression is used instead of the regular expression itself. This is much simpler from the user's perspective. For the user defining a number of rules and wanting to insert some control over them, it is easier to specify a transition table over the rules than it is to define the regular expression that the transition table can be derived from. Furthermore, it is easier to maintain a transition table during rule group modification.

The language of the rule management system is very complex. It supports more traditional data access such as facts and frames. It supports procedural execution directly from rules in the form of messages. This is how the KBMS is initially linked to the rule language. A default execution strategy, including control strategy and conflict resolution strategy exists which may be started from a rule using the message form. The rule language also supports a form of typed quantification.

3.3.2 Tractability and the Rule Management System

Considering the generality and expressivity of the rule language and the KBMS, efficiency has the potential of being forgotten, both in development and implementation. There are three ways of maintaining tractability in the rule language.

The first way of maintaining tractability is to manage the match phase of the inference cycle efficiently. The conceptual definition of the match phase is to check each rule and evaluate its LHS and if it is `:ok` then to queue it up for later interpretation. Obviously, this is also the slowest approach. The step this implementation takes is to compile rules into a rule constraint network that maintains rules in a form more suited for recognizing when data becomes available that has the potential of satisfying the LHS of a rule. To be specific, all the reference variables of the LHS of a rule must have values in order to determine if the LHS is satisfied. The rule constraint network represents variables as nodes in the network and rules as rule-nodes (see Figure 2). As variables get values, the nodes representing the variables are triggered. All the rule-nodes connected to the node are then triggered. These rule-nodes are then checked to see if all the nodes representing the LHS variables have values. If so, the rule represented by the rule-node is then put on the appropriate queue. As can be seen, this method allows rules to be queued that may not be satisfied. The rule constraint network only checks to see if variables have values, not if they have

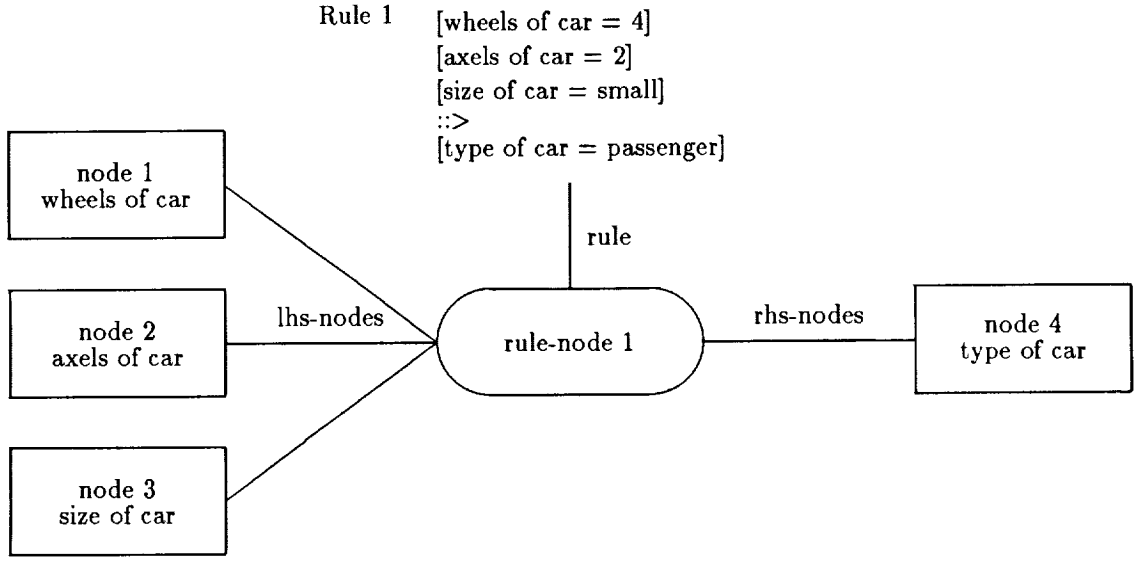


Figure 2: Rule Constraint Network Example

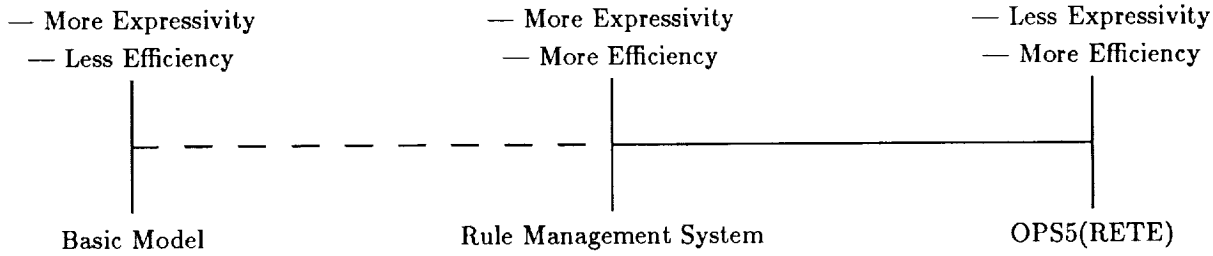


Figure 3: Expressivity versus Efficiency

the correct value. Thus the rules on the tickled queues must still be evaluated for satisfaction.

Alternatively, rules could be compiled more completely so that variables are checked to see if they have the right value. The complexity of the rule language implemented here makes this a difficult task and it has not been determined that it would be cost effective. Languages with less expressivity can be completely compiled much more easily (such as OPS5 using the RETE network [9] [10]). Expressivity and tractability is always lurking behind the scenes; see Figure 3 for an idea of where the rule management system fits.

The second way of maintaining tractability is by providing code for inference strategies instead of providing declarative rules defining the inference strategy. This is making use of the compiled versus interpreted option. This means that a programmer fluent in the rule language and its implementation needs to be available for the knowledge engineering task. However, considering the expressivity of the rule language, this may be a cost effective option. It is very easy for the programmer to express what he wants without having to go into contortions over representational limits.

The third way to maintain tractability is by the effective use of knowledge. This is to make use of Raj Reddy's fifth principle: *Knowledge eliminates the need for search.* In other words, the domain to be represented needs to be analyzed for maximum efficiency in terms of knowledge organization. For example, I can have two rule groups; one rule group forward chains on various data and computes a value for the variable `diagnosis`. The other rule group then uses the value of `diagnosis` to output the results to the user. If there are 50 rules in each rule group that use the variable, then 100 rules are triggered

whenever the value of the variable changes. If both of these rule groups are made to be sub-rule groups of a control rule group, the first rule group can compute a value for `diagnosis`. The control rule group can then set the value for the variable `the-diagnosis` as the value of `diagnosis`. `the-diagnosis` is then used by the second rule group instead of `diagnosis`. Now only 50 rules get triggered at any time.

The representation of knowledge should make maximum use of divide and conquer principles of knowledge organization. Another approach along the same lines is to provide strong heuristic knowledge to eliminate the need for search. This can come in many forms including domain guided inference strategies over rule groups as well as judicious use of control over the execution of rules in a rule group (i.e. the transition table method of control over rules).

3.4 Mixins

Diverse knowledge representation requirements may be needed by various domains. These may include temporal knowledge and non-monotonic knowledge. Neither of these have been implemented in KNOMAD and may involve major modification to the database interface and structure. Conceptually, the use of these aspects of knowledge are optional depending on the needs of the application. This is why they are considered as a mixable item to those items that are necessary for a collection of tools.

Temporal knowledge, non-monotonic knowledge, and uncertain knowledge should be a set of extensions to the database and database interface. They have not been implemented and therefore will not be discussed in any more detail.

4 Example

KNOMAD has been tested on two distinct applications so far. The application presented here is the Freight Agency Knowledge Base. It is a toy application requiring only ten rules. Its purpose is to compute how to send a package given its length, width, height, urgency, and destination. Considering that the application is very simple, we have implemented a control strategy for it in the rule language to demonstrate the rule language's power and flexibility.

The second application is a knowledge base for fault diagnosis in a power management and distribution system for a space station like platform [25]. This application requires almost 200 rules to analyze a fault and produce a diagnosis. In addition to the rules, the size of the domain is over 1500 facts for representing the model. This application is not presented here due to its size.

4.1 The Freight Agency Knowledge Base

```
@@
@@ The Freight Agency Knowledge Base
@@
@@ This knowledge base defines two rule groups, the main one for
@@ determining how to send a package, and a secondary one for implementing
@@ the control strategy of the first.

KB : freight

@@
@@ This rule group determines how to send a package.
@@ It is assumed that the package
@@ is being sent from somewhere in europe.  Initially, we need to compute the
@@ volume of the package based on its length, width, and height.  This is done
@@ in frule10.  The zone of were the package is sent is computed in frules6-9
@@ based on the country being shipped to.  The only other piece of information
@@ that might be needed is whether or not it is urgent.
@@
rule-Group : RG1
@@
```

```
@@ The control specifies the equivalent of a finite state machine
@@ representation of what rules are allowable after a rule is executed.
@@ The initial state for a rule group is always the start state.
@@ Here the only rule allowable after the start state is frule10.
@@ When frule10 fires we may allow any of frules1-9.
@@ After any of these the same set of frule1-9 is always
@@ available.  Thus we can have any level of non-determinism in our rule group
@@ as we desire.
@@
  CONTROL : ((start (frule10))
             (frule10 (frule1 frule2 frule3 frule4 frule5 frule6 frule7
                       frule8 frule9))
             (frule1 (frule1 frule2 frule3 frule4 frule5 frule6 frule7
                      frule8 frule9))
             (frule2 (frule1 frule2 frule3 frule4 frule5 frule6 frule7
                      frule8 frule9))
             (frule3 (frule1 frule2 frule3 frule4 frule5 frule6 frule7
                      frule8 frule9))
             (frule4 (frule1 frule2 frule3 frule4 frule5 frule6 frule7
                      frule8 frule9))
             (frule5 (frule1 frule2 frule3 frule4 frule5 frule6 frule7
                      frule8 frule9))
             (frule6 (frule1 frule2 frule3 frule4 frule5 frule6 frule7
                      frule8 frule9))
             (frule7 (frule1 frule2 frule3 frule4 frule5 frule6 frule7
                      frule8 frule9))
             (frule8 (frule1 frule2 frule3 frule4 frule5 frule6 frule7
                      frule8 frule9))
             (frule9 (frule1 frule2 frule3 frule4 frule5 frule6 frule7
                      frule8 frule9)))
@@
@@ The control strategy for this rule group is specified to be declared as the
@@ rule group freight-cs.
@@
  CONTROL-STRATEGY : freight-cs
@@
@@ Frule1:
@@      IF the zone is europe AND the distance is less than 100
@@        AND the volume is less than 100
@@      THEN the mode to send the package is road.
@@
  Frule1
  [ zone = europe ]
  [ distance < 100 ]
  [ volume < 100 ]
  ::>
  [ mode = road ]
 ;
@@
@@ Frule2:
@@      IF the zone is europe AND the volume is less than 200
@@        OR
@@         the distance is greater than or equal to 100
@@         AND the volume is less than 100
@@        THEN the mode is rail
@@
  Frule2
  [ zone = europe ]
  [ volume < 200 ]
  OR
  [ distance >= 100 ]
  [ volume < 100 ]
  ::>
  [ mode = rail ]
```

```
;
 Frule3
 [ zone = europe ]
 [ distance >= 100 ]
 ::>
 [ mode = special ]
;
 Frule4
 [ zone <> europe ]
 OR
 [ urgent = no ]
 [ volume >= 50 ]
 ::>
 [ mode = sea ]
;
 Frule5
 [ zone <> europe ]
 [ urgent = yes ]
 [ volume < 50 ]
 ::>
 [ mode = air ]
;
 Frule6
 [ country memberin european-countries ]
 ::>
 [ zone = europe ]
;
 Frule7
 [ country memberin american-countries ]
 ::>
 [ zone = america ]
;
 Frule8
 [ country memberin asian-countries ]
 ::>
 [ zone = asia ]
;
 Frule9
 [ country memberin australasian-countries ]
 ::>
 [ zone = australasia ]
;
 Frule10
 ::>
 [ volume = length TIMES width TIMES height ]
:
 [ mode memberin modes ]

@@
@@ This rule group implements (declares) the control strategy for the
@@ main rule group for the freight agency knowledge base.
@@ It is actually somewhat complex (although, not terribly so).
@@ FYI, this rule group performs a forward chaining implementation
@@ of a control strategy.
@@
Rule-Group : freight-cs
@@
@@ The first rule to be run is CS-Rule1.
@@ Following this we only allow CS-Rule2.
@@ After CS-Rule2, we non-deterministically only allow CS-Rule3-5.
@@
  CONTROL : ((start (CS-Rule1))
             (CS-Rule1 (CS-Rule2))
             (CS-Rule2 (CS-Rule3 CS-Rule4 CS-Rule5))
```

```
            (CS-Rule3 (CS-Rule3 CS-Rule4 CS-Rule5))
            (CS-Rule4 (CS-Rule3 CS-Rule4 CS-Rule5)))
@@
@@ The variable that we quantify over is rule-group.
@@ This variable will be bound to
@@ the rule group that this control strategy is applied to.
@@
  RG-VAR : rule-group
@@
@@ CS-Rule1 is an initialization rule.
@@ It simply makes sure that we start with an
@@ empty list.
@@
  CS-Rule1
  ::>
  [ rule-result-list = empty ]
 ;
@@
@@ CS-Rule2 evaluates all the rules in the viable-set of the rule-group and
@@ records the result in the rule-result-list.  Each result takes the form of
@@ (rule result), the result can be one of: :ok, :ng, or :missing-patterns
@@
  CS-Rule2
  FOR ALL rule in viable-set of rule-group
    < ::>
      [ rule-result-list =
         rule-result-list PLUS lisp :: my-evaluate ( rule ) ] >
 ;
@@
@@ CS-Rule3 adds all the rules which evaluated to :ok to the conflict set of
@@ the rule group.  It also removes them from the viable set.
@@
  CS-Rule3
  FOR ALL rule-result in rule-result-list
    WHERE [ lisp :: second ( rule-result ) = :ok ]
    < ::>
      [ conflict-set of rule-group = conflict-set of rule-group
                                        PLUS lisp :: first ( rule-result ) ]
      [ viable-set of rule-group = viable-set of rule-group
                                        MINUS lisp :: first ( rule-result ) ] >
 ;
@@
@@ Similarly, CS-Rule4 watches for rules with a :ng result.  These that have an
@@ else are fired (via my-interpret-else) and also removed from the viable set.
@@
  CS-Rule4
  FOR ALL rule-result in rule-result-list
    WHERE [ lisp :: second ( rule-result ) = :ng ]
    < ::>
      [ lisp :: my-interpret-else ( lisp :: first ( rule-result ) ) ]
      [ viable-set of rule-group = viable-set of rule-group
                                        MINUS lisp :: first ( rule-result ) ] >
 ;
@@
@@ CS-Rule5 watches for the rules that had :missing-patterns and simply removes
@@ them from the viable set of the rule group.
@@
  CS-Rule5
  FOR ALL rule-result in rule-result-list
    WHERE [ lisp :: second ( rule-result ) = :missing-patterns ]
    < ::>
      [ viable-set of rule-group = viable-set of rule-group
                                        MINUS lisp :: first ( rule-result ) ] >
 ;
```

```
@@
@@ Finally, this rule group is done when the viable set is empty.
@@
  [ viable-set of rule-group = empty ]

@@
@@ The domain knowledge.  A number of constants and facts particular
@@ to this knowledge base.
@@
Domain-Knowledge :  @ these are constants just to this kb
  constants :
    europe ;
    america ;
    asia ;
    australasia ;
    road ;
    rail ;
    special ;
    sea ;
    air ;
    no ;
    yes ;
    :ok ; :ng ; :missing-patterns .
  facts :  @ some initialized facts
    european-countries =
      ( france belgium spain germany uk portugal italy austria
 poland ) ;
    american-countries = ( usa canada mexico brazil ) ;
    asian-countries = ( china japan india ussr ) ;
    australasian-countries = ( australia new-zealand ) ;
    modes = ( road rail special sea air ) ;
    empty = ( ) .

@@
@@ Initially we want to begin execution of the main rule group: rg1.
@@
Begin : RG1

@@
@@ That's All Folks
@@
End-KB
```

5 Conclusion

The architecture of KNOMAD has been shown to be both flexible and powerful. The example shows both aspects while only making use of the rule management system and the frame system embodied in the database interface. This architecture may be easily extended by adding additional tools such as those described earlier.

Our goals are to add the constraint system and the planning/scheduling systems next. At the same time we intend to use a distributed database and look into partitioning knowledge for solving complex problems among physically distinct knowledge agents. We feel that KNOMAD is both a necessary and sufficient part of our goals of intelligent automation in complex, real world problem solving.

6 Acknowledgements

Many thanks go to Barry Ashworth who has continually supported and encouraged me while providing useful comments and insights on KNOMAD.

This work was performed by Martin Marietta Astronautics Group under contract number NAS8-36433 to NASA, George C. Marshall Space Flight Center, Huntsville, Alabama.

References

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] Barry R. Ashworth. Reactive autonomous planning in spacecraft. In *Proceedings of the Conference on Aerospace Applications of Artificial Intelligence*, 1989.

[3] A. B. Baskin. Combining deterministic and non-deterministic rule scheduling in an expert system. In *AAMSI*, 1986.

[4] Bruce G. Buchanan and Richard O. Duda. *Principles of Rule-Based Expert Systems.* Technical Report HPP-82-14, Stanford Heuristic Programming Project, 1982.

[5] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, 32(4), 1989.

[6] David Chapman. *Planning for Conjunctive Goals.* MIT Industrial Liaison Program Report 10-20-86, Massachusetts Institute of Technology, 1986.

[7] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24(3):347–410, 1984.

[8] K. Forbus. Qualitative process theory. *Artificial Intelligence*, (24), 1984.

[9] Charles L. Forgy. Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1), 1982.

[10] Charles L. Forgy and Susan J. Shepard. Rete: a fast match algorithm. *AI Expert*, January 1987.

[11] M.P. Georgeff. Procedural control in production systems. *Artificial Intelligence*, 18:175–201, 1982.

[12] D. Haussler. Bias, version spaces, and valiant's learning framework. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 324–336, 1987.

[13] P.J. Hayes. The logic of frames. In B.L. Webber and Nils J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 451–458, Morgan Kaufmann, 1981.

[14] Guy Lewis Steele Jr. and Gerald Jay Sussman. *Constraints.* Technical Report AI Memo No. 502, Massachusetts Institute of Technology Artificial Intelligence Laboratory, November 1978.

[15] J.E. Laird, A. Newell, and P.S. Rosenbloom. Soar: an architecture for general intelligence. *Artificial Intelligence*, 33, 1987.

[16] Doug Lenat and R.V. Guha. *The World According to CYC.* Technical Report ACA-AI-300-88, Microelectronics and Computer Technology Corporation, September 1988.

[17] Douglas B. Lenat. Ontological versus knowledge engineering. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):84–88, March 1989.

[18] Hector J. Levesque. Knowledge representation and reasoning. *Annual Reviews of Computer Science*, 1986.

[19] C.L. Liu. *Elements of Discrete Mathematics.* McGraw Hill Book Co., 1985.

[20] Satoshi Matsuoka and Satoru Kawai. Using tuple space communication in distributed object-oriented languages. In *OOPSLA*, 1988.

[21] M. Minsky. A framework for representing knowledge. In P. Winston, editor, *The Psychology of Computer Vision*, pages 211–277, McGraw-Hill, 1975.

[22] A. Newell, J.C. Shaw, and H.A. Simon. A general problem-solving program for a computer. In *Information Processing: Proceedings of the International Conference on Information Processing*, pages 256–264, UNESCO, Paris, 1960.

[23] Raj Reddy. Presidential Address at the American Association for Artificial Intelligence Conference, 1988.

[24] R. Reiter. Towards a logical reconstruction of relational database theory. In M. Brodie, J. Mylopoulos, and J.W. Schmidt, editors, *On Conceptual Modelling*, Springer-Verlag, 1984.

[25] Joel D. Riedesel, Chris Myers, and Barry Ashworth. Intelligent space power automation. In *Proceedings of the Fourth IEEE International Symposium on Intelligent Control*, 1989.

[26] J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, January 1965.

[27] Y. Shoham. *Reasoning about Change*. MIT Press, 1987.

[28] Mark Stefik, Jan Aikins, Robert Balzer, John Benoit, Lawrence Birnbaum, Frederick Hayes-Roth, and Earl Sacerdoti. The organization of expert systems. *Articial Intelligence*, 18, 1982.

[29] Mark Stefik and Daniel G. Bobrow. Object-oriented programming: themes and variations. *The AI Magazine*, 40–62, 1986.

[30] P. Thyagarajan and Arthur M. Farley. *Design and Implementation of a Qualitative Constraint Satisfication System.* Technical Report CIS-TR-87-03, Department of Computer and Information Science University of Oregon, March 1987.

[31] Steven A. Vere. Planning in time: windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(3):246–267, May 1983.

[32] Masanobu Watanabe, Toru Yamanouchi, Masahiko Iwamoto, and Yuriko Ushioda. Cl: a flexible and efficient tool for constructing knowledge-based expert systems. *IEEE Expert*, 41–50, Fall 1989.

[33] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, 1988.

A KNOMAD BNF Specification

A.1 Definitions

This appendix describes the syntax of KNOMAD using extended BNF notation. The meanings of the meta-characters are given in the table.

Symbol	Meaning
{	begin optional grouping
}	end optional grouping
|	alternative
+	one or more
*	zero or more
ID	analogous to any LISP atom
STRING	a string
NUMBER	a number

A.2 Rule Management System

```
knowledge-base ::= KB : ID { DOMAIN : ID } rule-group+ { domain-knowledge }
                   BEGIN : ID+ END-KB

rule-group ::= RULE-GROUP : ID
               { CONTROL : transition-table }
               { CONTROL-STRATEGY : control-strategy }
               { CONFLICT-RESOLUTION-STRATEGY : conflict-resolution-strategy }
               { RG-VAR : ID }
               { QUANTIFIED-VARS : q-vars }
               rules
               { : termination-condition }

domain-knowledge ::= { CONSTANTS : constants . }
                     { FACTS : facts . }
                     { FRAMES : frames . }

control-strategy ::= FUNCTION | ID

conflict-resolution-strategy ::= FUNCTION | ID

termination-condition ::= condition

transition-table ::= ( ( ID ( ID+ ) )+ )

q-vars ::= ID | ID , q-vars

constants ::= ID | ID ; constants

facts ::= fact | fact ; facts

fact ::= ID = ID | ID = ( ID* )

frames ::= frame+

frame ::= ( FRAME :NAME ID
                { :PARENTS parents }
                { :ISNEW FUNCTION }
                { :SLOTS ( ( ID aspect VALUE { aspect VALUE }* )* ) } )
```

```
parents ::= ID | ( ID+ )

aspect ::= :IF-NEEDED | :IF-ADDED | :CONSTRAINT | :MUSTBE | :VALUE |
           :DISTRIBUTION

rules ::= rule | rule ; rules

rule ::= { condition } ::> rhs |
         { condition } quantifier ID IN expr { WHERE condition } < rule >
         { ELSE condition }

rhs ::= condition { ELSE condition } | { ELSE condition }

quantifier ::= FOR ALL | THERE EXISTS

condition ::= selector+ { choice selector+ }

choice ::= EXCPT | OR

selector ::= [ quantifier ID IN expr { WHERE condition } < condition > } ] |
             [ expr { relation reference } ]

relation ::= NOT MEMBERIN | NOT UNIQUEIN | MEMBERIN | UNIQUEIN |
             = | <> | >= | <= | > | <

reference ::= expr

expr ::= message | constant | path | expr op expr

constant ::= STRING | NUMBER

op ::= PLUS | MINUS | UNION | TIMES | IDIV | RDIV | MOD

path ::= ID | ID OF path

message ::= path :: ID { ( expr+ ) }
```

A.3 Frames

```
frame ::= ( FRAME :NAME ID
                  { :PARENTS parents }
                  { :ISNEW FUNCTION }
                  { :SLOTS ( ( ID aspect VALUE { aspect VALUE }* )* ) } )

parents ::= ID | ( ID+ )

aspect ::= :IF-NEEDED | :IF-ADDED | :CONSTRAINT | :MUSTBE | :VALUE |
           :DISTRIBUTION
```

A.4 Database Assertions

In the following, VALUE represents any object.

```
fact ::= ( FACT ID :value VALUE )

frame ::= ( FRAME ID slot aspect VALUE )

slot ::= ID

aspect ::= :IF-NEEDED | :IF-ADDED | :CONSTRAINT | :MUSTBE | :VALUE |
           :DISTRIBUTION
```

A.5 Integrity Constraints

```
i-constraint ::= ( item+ ) | ID

item ::= i-constraint | type-decl | type-spec

type-decl ::= ( :type-decl ID )

type-spec ::= ( :type-spec ID )
```

NASA

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-4273		

4. Title and Subtitle	5. Report Date
Knowledge Management: An Abstraction of Knowledge Base and Database Management Systems	January 1990
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Joel D. Riedesel	
	10. Work Unit No. M-625

9. Performing Organization Name and Address	11. Contract or Grant No.
Martin Marietta Astronautics Group P. O. Box 179, MS: S-0550 Denver, CO 80201	NAS8-36433
	13. Type of Report and Period Covered

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered / 14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, DC 20546	Contractor Report
	14. Sponsoring Agency Code

15. Supplementary Notes

NASA Technical Monitor: Bryan Walls, Information and Electronic Systems Laboratory, Marshall Space Flight Center, Alabama 35812

16. Abstract

Artificial Intelligence application requirements demand powerful representation capabilities as well as efficiency for real-time domains. Many tools exist, the most prevalent being expert systems tools such as ART, KEE, OPS5, and CLIPS. Other tools just emerging from the research environment are truth maintenance systems for representing non-monotonic knowledge, constraint systems, object oriented programming, and qualitative reasoning. Unfortunately, as many knowledge engineers have experienced, simply applying a tool to an application requires a large amount of effort to bend the application to fit. Much work goes into supporting work to make the tool integrate effectively.

KNOMAD, a Knowledge Management Design System, described here, is a collection of tools built in layers. The layered architecture provides two major benefits; the ability to flexibly apply only those tools that are necessary for an application, and the ability to keep overhead, and thus inefficiency, to a minimum. KNOMAD is designed to manage many knowledge bases in a distributed environment providing maximum flexibility and expressivity to the knowledge engineer while also providing support for efficiency.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
KNOMAD Artificial Intelligence Knowledge-Base Systems Expert Systems AI Tools	Unclassified-Unlimited Subject Category: 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	19	A03

NASA FORM 1626 OCT 86

For sale by the National Technical Information Service, Springfield, VA 22161-2171

NASA-Langley, 1990