NAS8 - 36955

Cooperating Systems: Layered MAS

JRC Research Report No. 90-21

May, 1990

Final Report D.O. 50

Prepared for:

Mr. Tim Crumbley

NASA Marshall Space Flight Center

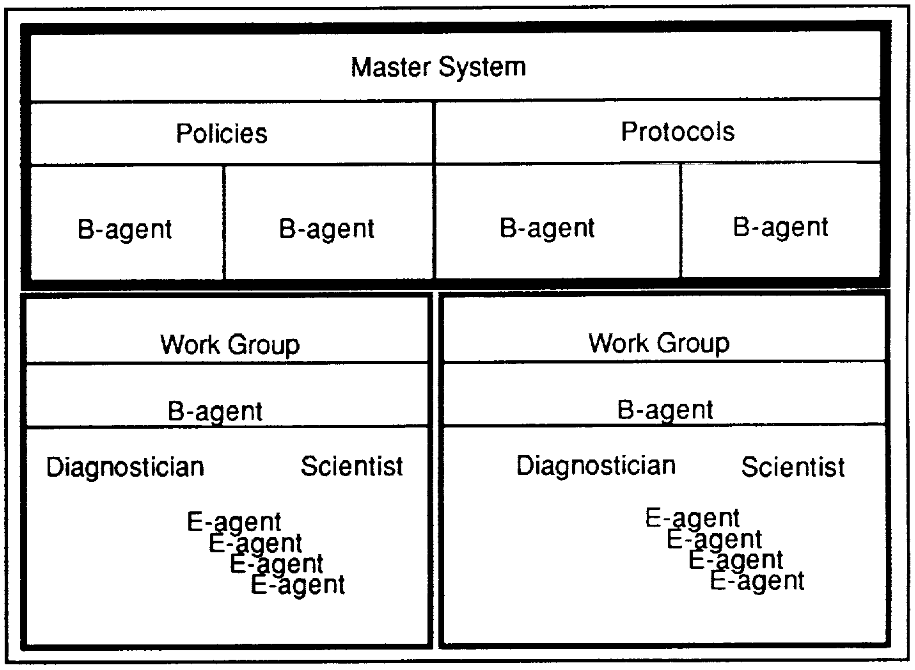
By

Daniel Rochowiak
Research Scientist
Johnson Research Center
University of Alabama in Huntsville
Hutsville, AL 35899
(205) 895-6583
(205) 895-6672

A BUREAUCRATIC MODEL FOR MUTIAGENT SYSTEMS

INTRODUCTION

Distributed intelligent systems can be distinguished by the models that they use. The model developed in this report focuses on layered multiagent systems conceived of as a bureaucracy in which a distributed database serves as a central means of communication. In this section the various generic bureaus of such a system will be described and a basic vocabulary for such systems will be presented. In presenting the bureaus and vocabularies special attention will be given to the sorts of reasonings that are appropriate.



The bureaucratic system (B-system) is composed of a collection of E-agents and B-agents that operate in a cooperative way through a collection of protocols and policies. The E-agents like diagnosticians and scientists perform specialized services. These specialized services are monitored and facilitated by B-agents. Within the B-system that task decomposition is more-or-less fixed in terms of the bureaus or work groups of the bureaucracy.

In brief a bureaucratic model has a hierarchy of master system and work group that organizes the E-agents and B-agents.

The master system provides the administrative services and support facilities for the work groups. The goal of the master system is to stay in a stable state in which the communications between work groups can continue and the results of the work groups can be shared. The workgroups are collections of agents. The minimal workgroup would be composed of one B-agent and one E-agent. The administrative oversight and communications of the E-agent would be provided by the B-agent. Additionally, if there were several E-agents, the B-agent would provide the facilities for cooperation and especially result sharing among the E-agents.

In order to accomplish these goals the use of a DDB is essential. The DDB acts a central repository for information communicated by both the builder/designers of the system and the current states of the agents in the bureaucracy. The distributed database also serves to account for the actions of the agents and provide each agent with some account of the general nature of other agents. In order to generate cooperation among the agents of the system there must be specified protocols and policies. The protocols provide for the exchange of information and knowledge with out which the bureaucracy would collapse into chaos. Without the protocols no agent would know how to communicate and no agent would be able to share its knowledge. The policies of the master system establish the distribution of the tasks and the general modes of interaction among work groups. With out these no work group would know either what it was supposed to do nor the manner in which a task was to be done.

This general model was developed during prior work for Marshall Space Flight Center and reported in UAH Report No. 804. In this report, the framework is extended and applied to considerations of the role simulations, the ECLSS software for Space Station Freedom, a tool for knowledge management, and explanation.

KBS AND SIMULATION

One of the obvious benefits of a KBS is that it is capable of constructing an analysis on the basis of the state of knowledge at a given point in time. A KBS can be understood as using an engine to apply knowledge to a packet of atoms and issue messages that constitute the analysis. The engine is the code that applies the knowledge to the atoms of the packet. The atoms can be though of as the statements or facts the knowledge is designed to reason. The knowledge may be represented in many forms, but in what follows a rule like representation will be assumed. The messages are the results of the engine applying the knowledge to the atoms of the packet, and the messages may be either conjunctions of clauses or symbols that stand for operations. The messages are various and can range from messages to display something to messages to reassign values for the parameters or atoms.

The operation of a KBS is composed of cycles. Descriptively the cycle includes tasks to load a packet, load knowledge, run engine, issue messages, and to reset the packet, knowledge, and engine. Although this descriptive notion is linear, it is possible to have some of the tasks occur in parallel or to have some of the task units operate in a parallel fashion. However, it should be noted that the idea of a KBS cycle is not well defined under a range of operations. In particular it should be noted that the descriptive account does not clearly indicate the condition of termination. One way in which termination might be defined and the cycle established is to allow that the cycle is completed when the messages are sent or formulated. Alternatively, the cycle may be said to be complete when the engine is unable to generate any new messages. The second sense is preferable since it accords with the idea that that a cycle should be complete in the sense all of the information in the packet, relative to the available knowledge, is used before the process terminates. Adopting this second way allows the knowable content of the packet to be defined as the total collection of messages issued by a specific engine with specific knowledge.

A further restriction on the notion of a KBS cycle and knowable content must be imposed, if the knowable content of a packet is to be produced in a cycle. The restriction is that the only changes that a message may make are either to send a message to some other agent or to change the state of a variable from an unknown state to a value. The point of the restriction is to prohibit the change of a value in an atom of the packet. A KBS that accords with this restriction will be called a closed KBS, and one that does not will be said to be open.

CLOSED KBS

A closed KBS might be designed for several reasons. First, the closed KBS directly satisfies conditions for individuating packets. Each packet consists of a specified number of atoms with a specified set of values. If a KBS resets the values of the atoms, then either a new standard of individuation is needed or the resulting packet is deemed a new packet. If the packet is deemed a new packet, then the processing of the first packet would not terminate. Thus, as will be discussed below, allowing a new value to be set for an atom of a packet will at least require a new criterion of individuation if the notion of a cycle is to be kept in a meaningful way. Second, if the KBS can alter the value of an atom in a packet, then it would be difficult to define the notion of consistency in a packet. This would be so since the setting of a new value would be equivalent to saying that "Atom A has value X, and it does not have the value X." Finally, a closed KBS might be desirable since it most clearly specifies the knowable content of a packet given a certain knowledge structure and engine.

A closed KBS is such that for a given packet, engine, and knowledge structure a specific message will be produced, remembering the message might be a conjunction of clauses or a symbol for further operations. To simplify the issue any apparent messages that appear to change a parameter from an unknown state to a known state can be conceived of as operating as part of a chain argument. This process of chaining can be eliminated by allowing that if the inference is a chain, then the complex chain can be replaced by a complex antecedent. (See example 1)

Example 1

Atoms in packet: P, Q, R
Rule 1: $(P\&Q) \supset S$
Rule 2: $(R\&S) \supset T$
Message 1: S by Rule 1
Message 2: T by Rule 2 and Message 1

Chain to T		Eliminate Chain	
1 P&Q&R	Packet	1 (P&Q) ⊃ S	Rule 1
2 (P&Q) ⊃ S	Rule 1	2 (R&S) ⊃ T	Rule 2
3 (R&S) ⊃ T	Rule 2	3 P&Q&R	Assume
4 P&Q	Simp 1	4 P&Q	Simp 3
5 S	MP 4,2 Message 1	5 S	MP 4,2
6 R	Simp 1	6 R	Simp 3
7. R&S	Conj 6,7	7 R&S	Conj 6,5
8 T	MP 7,3 Message 2	8 T	MP 6,2
		9 (P&Q&R) ⊃ T	Cond. Proof 3-7

In the example the formula created by the conditional proof allows the chaining to be eliminated. That is, the packet, 'P&Q&R' and the new conditional, '(P&Q&R) ⊃ T' can allow for the issuing of Message 2 without the additional issuing of Message 1. This process can be generalized so that all intermediary assignments to atoms whose values are unknown and do not appear in either the packet or the message can be eliminated.

The general strategy works in a similar way in the case of disjunctive elements of rules. In this case, however, it is easier to consider any rule in which a disjunctive element occurs to be represented in terms of two rules.

Example 2

Packet: P, Q, R
Rule 1: (P&(QvS)) ⊃ T
Rule 2: (R&T) ⊃ U
Message 1: T by Rule 1
Message 2: U by Rule 2 and Message 1

Chain to U		Separation of Rules		Eliminate Chain	
1 P&Q&R	Packet	1 P&Q&R	Packet	1 (P&(QvS)) ⊃ T	Rule 1
2 (P&(QvS)) ⊃ T	Rule 1	2 (P&Q) ⊃ T	Rule 1*	2 (R&T) ⊃ U	Rule 2
3 (R&T) ⊃ U	Rule 2	3 (R&T) ⊃ U	Rule 2	3 P&(QvS)&R	Assume
4 Q	Simp 1	4 T	Message 1	4 P	Simp 3
5 QvS	Add 4	5 U	Message 2	5 Q	Simp 3
6 P	Simp 1			6 QvS	Add 5
7 P&(QvS)	Conj 6,5	1 P&S&R	Packet	7 P&(QvS)	Conj 4,6
	Message 1	2 (P&S) ⊃ T	Rule 1**	8 T	MP 7,1
8 T	MP 7,2	3 (R&T) ⊃ U	Rule 2	9 R	Simp 3
9 R	Simp 1	4 T	Message 1	10 R&T	Conj 9,8
10 R&T	Conj 9,8	5 U	Message 2	11 U	MP 10,2
11 U	MP 10,3			12 (P&(QvS)&R) ⊃U	CP 3-11
	Message 2				

The 'Chain to U' column illustrates the way in which the intermediary messages would be sent and used. The 'Separation of Rules' column illustrates in a brief form how the separation would produce the same result with different packets. The 'Eliminate Chain' column illustrates how the intermediate messages could be removed through a single rule.

In turn the rule that results at step 12 of the column could be separated into two rules, '$(P\&Q\&R) \supset U$' and '$(P\&S\&R) \supset U$.' The two new rules would then clearly illustrate the nature of a closed KBS.

Allowing that all of the rules in a knowledge structure of a KBS can be formulated using only the packets to which they respond on the left hand side of the rule and messages to be issued on the right hand side, then given any packet the engine would apply one and only one rule for a closed system. In effect the rules would represent no more than a large two dimensional array of packets and messages.

Open KBS

An open KBS does not obey the restriction that the only changes that a message may make are either to send a message to some other agent or to change the state of a variable from an unknown state to a value. Thus an open KBS can change the value of an atom in a packet, or change the value of some atom for which a value has already been inferred. There are several reasons for violating the restriction.

The violation of the restriction on closed systems is reasonable when there is a need to set a seed value or to make an assumption. In both cases this occurs as a first step in determining a value for an atom. A typical example is the use of iteration to find a value within given limits by first "guessing" a value. However, it should be immediately noted that there are two ways in which such guesses can be made. Given a sufficiently powerful algorithm for determining the desired value, any guess will do. Guesses for such algorithms are often informed by background knowledge and produce quicker results. These cases need not be serious violations of the restriction, however. The restriction can be amended such that the algorithm is locked into a procedure such that from the point of view of the flow of knowledge only the final value of the procedure is assigned to the atom of the KBS. In this way the spirit of the restriction is kept, but the operation of the engine is altered such that temporary computational assignments of values to the atoms are kept distinct from the permanent values assigned to the atoms for the purposes of reasoning.

Non-Monotonic Reasoning and Open KBS

Even the amended restriction may be violated in cases that, from a knowledge perspective, require the assignment of a new value based on further reasoning. This is the case in which reasoning behaves in a non-monotonic way. Reasoning behaves in a monotonic way just in case the collection of true inferred claims always grows, or the validity of the inference is unaffected by additional information. Note that in example 1 and 2 (above) it was not necessary to violate the amended restriction when using the logical rule of conditional proof (CP) even though assumptions are made. In processes like CP the assumption used in the deduction is discharged from the list of true statements in the proof, but is hidden in the antecedent of the conditional that is formed in the conclusion. However, in the case of non-monotonic reasoning the truth of a claim is retracted and the list of proven true claims is contracted. The contraction clearly indicates the non-monotonic character of the reasoning.

The way in which non-monotonic reasoning operates can be illustrated by a slight variation on a case generated by Dov Gabby (1982). Gabby's example focuses on a case in which a package tour company must make decisions about what to do with the passengers on one of its tour flights. The flight deviates from standard conditions because the Paris airport is one of the stops on the tour flight and the airport has been closed because of terrorist action. Abstracting from the specifics of the case the essential elements are the following:

1 there is set of procedures that should be followed
2. there is limited knowledge of relevant events
3. at time T a conclusion C is deduced
4. at time T' a conclusion not-C is deduced.

Turner (1984) looks at various ways in which situations of this kind can be handled. Given the abstract statement of the problem it should be clear that cases of this sort must be handled, if they can be handled at all, by open KBS. The reason should be clear if 1 represents the knowledge structure and the engine remains unchanged, then the only source for the change in the conclusion must rest with 2 which can be considered to be the packet. In short, the analysis of this sort of problem requires that the known value of some atom in the packet must change. This is a direct violation of the restriction that generates a closed KBS. Further it should be clear that without any new restriction the termination of the cycle has as its upper limit the changes of all values in all atoms in the packet such that the atom appears in the left hand side of any rule.

Consider the following example (Example 3). The example is similar to those found in the previous cases. The only difference is that a contradiction in the set up of the problem is made explicit. It should be clear that if the packet contained P, Q, and ~R or P, ~Q, and R then contradiction would not appear. In the former case S would be sent as the message, while in the latter no message would be sent. Further note that if backward chaining were to be used and the target was either T or ~T a solution could be found. In the former the solution would be P, Q and R and in the latter Q and R. This would seem to be at least an unhappy state of affairs since the added information P would generate the contradiction. It should be notic hat the reasoning to T, ~T, and T&~T is formally valid. If an additional non-formal metarule were applied, then some decision about the ultimate conclusion could be reached. If, however, there is no such rule, then the reasoning that leads to the inferred rule, '(P&Q&R) ⊃ T&~T' would be valid but with the addition of the packet could not be sound. Thus, it would seem that some bit of information of knowledge should be rejected. However, the formal side of the bivalued logic cannot be a help in this case.

Example 3

Atoms in packet: P, Q, R
Rule 1: (P&Q) ⊃ S
Rule 2: (R&S) ⊃ T
Rule 3: (R&Q) ⊃ ~T
Message 1: S by Rule 1
Message 2: ~T by Rule 3
Message 3: T by Rule 2 and Message 1

Chain to T&~T		Eliminate Chain	
1 P&Q&R	Packet	1 (P&Q) ⊃ S	Rule 1
2 (P&Q) ⊃ S	Rule 1	2 (R&S) ⊃ T	Rule 2
3 (R&S) ⊃ T	Rule 2	3 (R&Q) ⊃ ~T	Rule 3
4 (R&Q) ⊃ ~T	Rule 3	4 P&Q&R	Assume
5 P&Q	Simp 1	5 P&Q	Simp 4
6 S	MP 5,2 Message 1	6 S	MP 5,1
7 Q&R	Simp 1	7 Q&R	Simp 1
8 ~T	MP 7,4 Message 2	8 ~T	MP 7,3
9 R	Simp 1	9 R	Simp 4
10 R&S	Conj 6,9	10 R&S	Conj 6,9
11 T	MP 10,3 Message 3	11 T	MP 10,2
12 T&~T	Conj 9,11	12 T&~T	Conj 11,8
		12 (P&Q&R) ⊃ T&~T	Cond. Proof 4-12

The case of non-monotonic reasoning is both difficult to solve and illustrative of a wide range of problems to which KBS might be applied. In order to see why this is so, a closer examination of the issues is required.

Consider the sequence of events in a KBS. A packet is delivered to the engine with a particular knowledge structure in place. For example, a packet of data is delivered to an engine whose knowledge structure is tuned to fault diagnosis. Now there are several possibilities.

The first possibility can be established by considering the data in the packet and the knowledge elements in the knowledge structure to be fixed and exhaustive. If such elements are fixed their values do not change and if the collection of elements is exhaustive, then all combinations of values of atoms in the packet are such that some rule can be applied that will issue an appropriate message. This is the case for a closed KBS and the problem reduces to the case of finding the correct row in the decision table. This possibility leads to the conclusion that 3 and 4 differ because they are acting on different packets. At T and at T' the cycle is completed and the appropriate message is issued. Part of the message issued in each case is to flush the KBS. This means that all values of atoms are returned to an unknown state and the engine is reset to accept a new packet. It is as if the KBS has amnesia; it neither remembers what it did in a previous cycle nor that there was a previous cycle. In this case non-monotonic reasoning cannot occur since the engine does not have the memory that would allow it to know that the reasoning is non-monotonic.

The second possibility can be established by considering the data in the packet and the knowledge elements in the knowledge structure to be fixed but not exhaustive. Two subcases should be considered. The first subcase focuses on the rules in the knowledge structure. In this subcase some combinations of values for the atoms will not match the arrangement of atoms on the left hand side of any rule in the KBS even allowing for backward chaining. That is there some situation in which the engine applying the knowledge in the structure to the atoms in the packet will be unable to issue a message. To force the completion of the cycle a rule is generally added to the effect that if no message can be issued by the rules in the structure, then a message that this is so should be issued. This solution avoids the problem of non-monotonic reasoning by forcing the KBS to be closed. In general, the solution to the lack of exhaustive knowledge for the domain is to force the KBS to be closed. The second subcase creates a different situation. In the second subcase it is the atoms in the packet that are not exhaustive. This subcase will collapse into the previous case only when the elements of the knowledge structure, especially if they are rules, are in complete declarative form. The knowledge element is in complete declarative form just in case the condition of the element (the left hand side of the rule) references every atom in the KBS. In such a case the lack of a specification of an atom in the packet will prohibit the KBS from issuing any message accept a message that no other message can be issued. However, if the elements are not in full declarative form, then even though the entire collection of element conditions may reference every atom of the system, it is possible either that the system will generate no message or that it will generate multiple messages. The former option leads to the previous line of solution; the KBS is forced to be closed. The latter option provides a place for non-monotonic reasoning.

The third possibility can be established by considering the data in the packet and the knowledge elements in the knowledge structure to be variable and exhaustive. This case establishes a possibility for non-monotonic reasoning when it is applied to either the packet or the knowledge structure. If the values of the atoms in the packet can be reset to a new known value or if the applicability of the rules can be altered, then the KBS will be open. Further, in each case it is clearly possible to generate claims such as 2 and 3 on a cycle of the engine. This can happen either because an intermediate atom forces the consideration of a group of knowledge elements which in turn resets the value of an atom in a packet, or because some intermediary creates a condition in which the value of an atom in a packet is directly reset. In either case there is an opportunity for non-monotonic reasoning.

The fourth possibility is established by considering the packet and the knowledge structure to be both variable and non-exhaustive. In this case it is clear that non-monotonic reasoning will occur and that the KBS will be open.

Consideration of the four possibilities strongly suggests that non-monotonic reasoning should be a prevalent phenomenon, and since this is so should be considered an important class of problem. In practice this can be clearly seen. Most of the work that goes into the knowledge engineering of a KBS is an attempt to convert non-monotonic reasoning into monotonic reasoning, and thereby convert an open KBS into a close KBS. During the life cycle of the KBS there is a tacit assumption that as the KBS develops it will become a closed monotonic KBS.

Consider a case of fault detection and diagnosis. In this case it initially might be assumed that all of the sensors of the physical system are truthful, that all of the sensor readings can be delivered as a packet to the engine, and that the knowledge elements are fixed and exhaustive. Under these assumptions the process of knowledge engineering the KBS is a process of cataloguing all of the atoms (sensor readings) and associating with every combination of atoms some rule. It may be assumed that some of the rules might generate intermediates that are used by other rules, but given the considerations advanced above for the eliminability of chaining and disjunction, these can be considered 'logical sugar.' This is so because the use of these structures makes it easier for the human to get at or organize the knowledge, but is not required by the logic of the system. Thus, for every configuration of the sensors there exists some rule such that that rule will issue a message that represents the detection and diagnosis of some fault if it exists or issues a message that the system is 'healthy.'

So far the presentation has stressed ways in which non-monotonic reasoning can be eliminated to create a closed KBS and has emphasized the formulation of non-monotonicity in terms of the growth of truths. At this point it is useful to introduce an alternative sense of non-monotonicity in which a system is said to be non-monotonic if the introduction of new data, information, or knowledge invalidates previous results. If the values of the atoms in the packet are also considered to be a result, then changing the value of an atom in a packet will create non-monotonic reasoning. When it is not the case that the sensors are truthful, when it is not the case that the sensors can deliver their readings as a packet, or when the knowledge elements are not fixed and exhaustive the possibility of ineliminable non-monotonic reasoning arises.

If the sensors of the system are not truthful, then the messages issued by the closed KBS may be fallacious. This is clearly undesirable. Thus it may seem appropriate to add a new KBS to the general system such that it is a closed KBS that either indicates the fault in a sensor or reports that the sensors are healthy. But what would such a KBS be like? Consider a two sensor system in which the only message is to be a message about the health of the sensors. Suppose that the sensor checking apparatus terminates its cycle when it detects a bad sensor. In this case only two checking operations are needed; one for the first sensor and one for the second. Now suppose that the sensor readings are linked in some way. For example suppose the second sensor acts as a check on the first sensor. If the values differ, then the fallacious sensor cannot be directly isolated from the packet of values delivered to the KBS. If the sensors agree, the same result will occur since it will be possible that both sensors are in error. Adding additional sensors will not help either since the same type of problem will occur if another sensor is added. Depending on the way the sensors are arranged and the processes that intervene greater complexity could result. The point here is that from a logical point of view the closed KBS could do its work only if it already knew the truthful sensors (or the fallacious ones). Of course in this case, the KBS would not be needed. Further it should be clear that even if there could be a correct judgment about the sensors, it would still require that the fault detection KBS be an open system, since the result of the sensor KBS would alter the values in the packet, and pass that packet to the fault detection KBS. In any case removing the assumption of the truthfulness of the sensors will create a condition under which non-monotonic reasoning is ineliminable.

If all of the sensor readings cannot be delivered to the engine as a packet, then either the engine must wait until the information is so delivered, or engage in non-monotonic reasoning. If the engine waits for the whole packet to be delivered, then critical actions may not be taken. Consider the case of fire detection and suppression (FDS) on board the Space Station. If the KBS charged with the monitoring of the FDS must wait for the packet of all sensor information which is relevant to the FDS KBS knowledge structure, then the action taken on a single sensor item that indicates a fire would be postponed until the packet is assembled. This wait might easily be deemed unacceptable; the KBS might be charged with acting on the fire as soon as it is noticed. Note that the previous considerations of multiple sensors would also apply with a vengeance. If the sensor that indicated fire was subject to the action of a sensor KBS prior to the fault KBS, then the delay would increase. Faster processing may help, of course, but the fact of the delays would remain. It should also be remembered that the reason for waiting is to ensure that a closed KBS can be applied. To avoid the waiting the KBS must be open and engage in non-monotonic reasoning. In this case item 2 of the specification of non-monotonic reasoning is at issue. At one point in time conclusion C is generated, and at another time conclusion not-C is generated. In the case of FDS it would be unreasonable to think that the system would be purged at the termination of a cycle since the fact that the message of fire was generated would seem to require a focus of attention and historical analysis related to that fire event. In brief, the KBS would ideally use the information that a fire message had been sent as part of a continuous reasoning process. This continuous reasoning process might even provide a facility for the plausible inference that the fire detector that reported the condition was in error. This would be so if other related sensors (temperature, atmospheric contaminants) reported findings that were inconsistent with the fire detector's report. In any case non-monotonic reasoning is needed.

If the knowledge elements are not fixed or are not exhaustive, then the analysis given above concerning open systems will apply. In practice this situation can obtain either when there is the opportunity for metareasoning about possible conclusions (messages) or when there is simply a lack of knowledge. The former creates a situation that requires non-monotonic reasoning in an open KBS because the metareasoning may require the system to backtrack if an unacceptable or incoherent conclusion is reached. If, for example, a KBS could generate a solution that indicates a device or process is in a physically unrealizable situation, then the engine should backtrack. In this case the new knowledge would cause a retraction of an inference previously deemed valid. The latter creates a situation that is much more typical of the kind problem that is actually encountered. Either it is not possible to get all of the information into a packet, or the elements of the knowledge structure do not cover all of the possible cases. In any case the reasoning that is appropriate would be non-monotonic.

Thus, it seems that non-monotonic reasoning is an important and in many cases ineliminable part of reasoning. Since a closed KBS prohibits such reasoning, systems that require non-monotonic reasoning must be open systems.

OPEN KBS AND SIMULATION

A simulation is a computer program that projects what events will take place given certain initial conditions and structures. A KBS is a computer program that generates messages (conclusions) given certain packets and knowledge. At a very high level both simulation and KBS have a similar structure: If certain conditions are satisfied and certain structures exist, then certain results are produced. However, the simulation and the KBS differ in that the simulation is forward looking, while the KBS is backward looking.

The difference in whether the engine is forward or backward looking points to the incapacities of both the simulation engine and the KBS engine. The incapacities of each are due to the fact that certain information the would make either more knowledgeable is lacking.

Consider the sort of information that is typically lacking in a KBS.

First, there is in general little that is done to validate the atoms of the packet. In part this is due to the emphasis on closed KBS. However, there is another deeper reason why a typical KBS does little to validate the packet. The deeper issue is that the packet constitutes that which is to be analyzed by the engine and knowledge structure; it is akin to the 'givens' in a proof. To alter or even to question the veracity of the packet is to fundamentally alter the character of the KBS. Rather than issuing messages that are diagnoses or conclusions about the intended domain, the messages would constitute plausible hypotheses about the domain. That is, each message of the KBS would function as though it were a hypothetical conditional of the form: "If the conditions in the packet are correct, then the message is a plausible conclusion." This would be tantamount to creating a new rule or knowledge element that should be used in coming to a conclusion. The only way to come to the conclusion at this point would be to show that the conditions in the packet are correct. In general, this is not an ordinary line of attack since it is at least unclear, if not impossible, to determine whether the atoms of the packet are veridical. The reason for this is that if one is to determine the veracity of the atoms in the packet this can only be done in the KBS framework by having another KBS agent determine the veracity of the atoms. Of course, that KBS agent must itself have a packet delivered to it. Since this is the case, this second KBS must have the atoms of its packet checked. And so on. Thus, the deeper problem is that if at some point the atoms of the packet are not taken as veridical givens, an infinite regress ensues. If the principle to be used is that no concluding message is to be established until the veracity of the atoms of the packet is established, then it would appear that no concluding message can be established.

Second, the information in the knowledge structure in general represents knowledge about how to extract additional knowledge content from the packet. The assumption used in building the knowledge structure for a KBS is that the knowledge in the knowledge structure makes explicit the information about the domain as captured in the packet. The use of a deductive model of how knowledge applies to information, generates this sort of situation since the knowable content of the packet is the set of messages that can be generated from it. It is the set of consequences of the packet, given the knowledge structure. Thus it is reasonable on such a deductive account to interpret the function of the engine on the packet and knowledge structure as making explicit the knowable content of the packet. What is lacking in this case is the sort of knowledge that comes about through the projection of consequences, the sort of knowledge that constitutes background knowledge, and the sort of knowledge that fuses disjunctive reasoning into reasoning that more strongly supports the conclusion. The focus of this section will be on the consequences of a lack of projective knowledge and a partial remedy.

In each of the foregoing cases, increasing the strength of the reasoning on the packet can be understood as at least partially a function of the projective strength of the reasoning structure and its ability to handle non-monotonic sorts of reasoning. In particular, it is the ability of the system to project the next state of affairs and backtrack, if necessary, that is important. If the KBS only tries to make explicit that which is already contained in the packet, then the projective capacity of the system is weak. At best it can generate conclusions of the form: "This condition (error, problem, difficulty) has been detected, and the following action should be taken." However, there is typically little done about projecting the consequences of taking such an action. The projection should act as a prediction that if satisfied demonstrates the soundness of the reasoning, and if not calls for some modification.

The soundness of the reasoning is what is at least in part at issue in the consideration of non-monotonic reasoning. Using a closed conception and accepting the atoms of the packet as veridical, the issue of the soundness of the reasoning does not even appear. Each cycle of a closed KBS will generate messages that are both valid and sound; that is simply the nature of the closed KBS. As soon as questions of soundness are raised issues of openness arise. In the present context the issue of openness can be examined from the combined perspective of the simulation plus KBS.

The open KBS in its initial work operates much as closed KBS would operate. The knowledge structure of the system is applied to the packet by the engine. The system then issues a message based on that application. However, the cycle of an open KBS with a simulation agent does not end at this point. Rather the action in the message is applied to the packet by the simulation agent in order to generate predictions of the activity of the physical system. That is, the simulation agent makes a predication of what will happen if the proposed action is taken given the state of the physical system represented in the atoms of the packet. The KBS continues its cycle with the arrival of a new packet. At this point rather than the KBS simply applying the knowledge structure to the packet a comparison phase is entered. If the data in the packet matches the predicted values, then the KBS accepts the reasoning about the previous packet as sound and the first open KBS cycle terminates. The new packet is loaded and the system starts again. However, if the new values do not match the predicted values, then two possibilities must be examined. The first possibility is that the reasoning on the previous packet was incorrect. The second is that the reasoning on the first packet was correct and that a new problem has arisen. These two possibilities present many of the same difficulties as the case of the veridicality of sensors does.

How should the simulation plus KBS system determine whether its previous reasoning was incorrect or a new error has arisen? One answer would be to check whether the values that do not accord with the prediction would have occurred if some atom in the original packet had been different. This case could easily go beyond the bounds of available computational power when there are either many atoms in the packet or when the values of the atoms cooperate in generating a new condition. Thus this approach is not acceptable. There is something right about this approach, however. What is right about it is that the prima facie case is strong that a relevant possibility is that the original results of the KBS were in error. What should prove more tractable is a subtractive approach.

The subtractive approach allows the engine to operate on the packet and attempts to determine what the condition is and what its resolution might be. Once the condition is determined the simulator sets to work again. This time it takes as its inputs the projected state of the first simulation with the currently detected error and compares these to the values in the second packet. If there is a match at this point, then the first reasoning of the combined system is accepted as sound, the cycle is terminated and the combined system begins again. If, however, the two do not match, there are again two cases: either the current diagnosis is incorrect or the previous diagnosis was incorrect. At this point the situation is somewhat different than the previous situation. If the current diagnosis is incorrect, then it can be hoped that the next cycle will capture it. If the previous diagnosis were incorrect, then the simulator can be put to work in determining what the state of the system would have been if the first action had not been taken. The difference between the states of the physical system and the two simulations will provide a rule of thumb for distinguishing were the blame lies. If the new packet more closely resembles the state of affairs generated by the simulation without the action, then there is reason to believe that the first diagnosis was at fault. If the values in the new packet more closely resemble the simulation values with the action taken, then there is good reason to think that the first diagnosis was correct. If the latter is the case, then the first cycle terminates. If the former is the case, then the first diagnosis is retracted and the values of the simulation without the action are used as the comparison values for a new cycle. The new cycle keeps these values for the testing of the soundness of the second diagnosis and action.

Although the combined systems appears complex, and may even seem needlessly so, there is reason for it. What the combined system is trying to avoid is the deadlock of example 3. The combined system is attempting to escape from the condition of contradiction by using both historical and simulation knowledge to determine what the best conclusion is and retract any conclusions that do not match that best conclusion. Of course determining the best conclusion is the problem. Th following table may help. (See Table 1) The first column indicates the round of the reasoning. A round of reasoning is the operation of the system up to the point of action. In round 0 no judgement is made. The second column indicates the actions of the system. These actions include making inferences, issuing messages, and running the simulation. It should be noted that the simulation is used to predict values. The final column indicates the judgements that are made by the system about the activities of the KBS and simulation agents. The judgements are based on comparisons of the values in the packets and the values generated by the simulation with and without actions being taken. If The action based on the first packet is incorrect, then the action is withdrawn and a corrective action is taken. If it is only suggestive that the first reasoning is unsound, then the cycle is terminated, but corrective action is not taken.

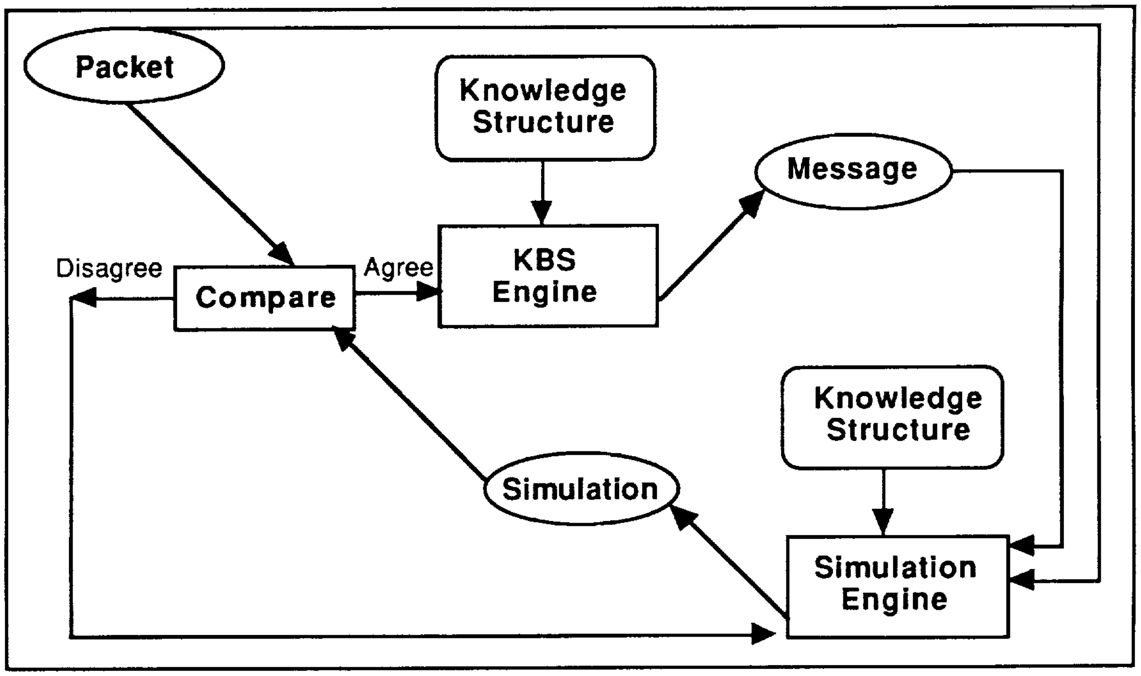
Table 1

Round	Action	Judgement
Round 0	Packet 0, knowledge structure and engine generate message 0 Message 0 and packet are used to generate simulation 0 that predicts the values for round 1	
Round 1	Packet 1 is compared to simulation 0 values	If the values of packet 1 and simulation 0 match, then the reasoning of round 0 is accepted as sound and the cycle 1 terminates. If the values of packet 1 and simulation 0 do not match then enter round 2.
Round 2	The values of packet 0 are used to generate simulation 1	If the values of simulation 1 and packet 1 match, then the reasoning of round 0 is rejected as unsound and corrective (backtracking) action is taken. Cycle 1 terminates. If the values of simulation 1 and packet 1 do not match, then if the values in the packet more closely resemble the values in simulation 0 than simulation 1, then the reasoning of round 0 is accepted as correct and the cycle is terminated. If the values more closely resemble simulation 1, then the reasoning of round 0 is rejected as unsound, and the knowledge structure is used to generate a new message. Cycle 1 is terminated

The judgments rendered by the combined system do not guarantee the truth of the conclusion or the soundness of the reasoning. That is simply not a possible or reasonable demand in any case that requires an open KBS. It should be noted, however, that this lack of a guarantee is something that afflicts most of human activity. It is rarely the acse thast such guarantees can be issued and it is certainly not the case that the can be issued in any strong case for the empirical world. The best that can be said is that on the basis of available knowledge the fforgoing procedure can issue a principled judgment and that the principled judgement is based on more and better information than would be available for a KBS alone.

Next consider the sort of information that is typically lacking in a simulation. In a simulation the packet and knowledge structure that is given to the siulation agent, generate a group of values. By itself the simulator makes no judgment about wheteher the value are good, veridical, indicative of problems, or anything else of interest. The purpose of the simulator is simply to take initial values and apply the knowledge structure to project new values. In the best of worlds were the construction of the simulator is such that if veridical values are input, then veridical values are output, then the simulator simply predicts the sate of affairs at some later time. This predictive function is crucial for the making of judgements about the soundness of reasoning. This should be so since if the reasoning is sound, then the actions taken on the basis of the conclusions of that reasoning should have predictable consequences if no new fault eneter the physical system being examined. What the simulation does not include is rules of thumb for making judgements about the data that it produces.

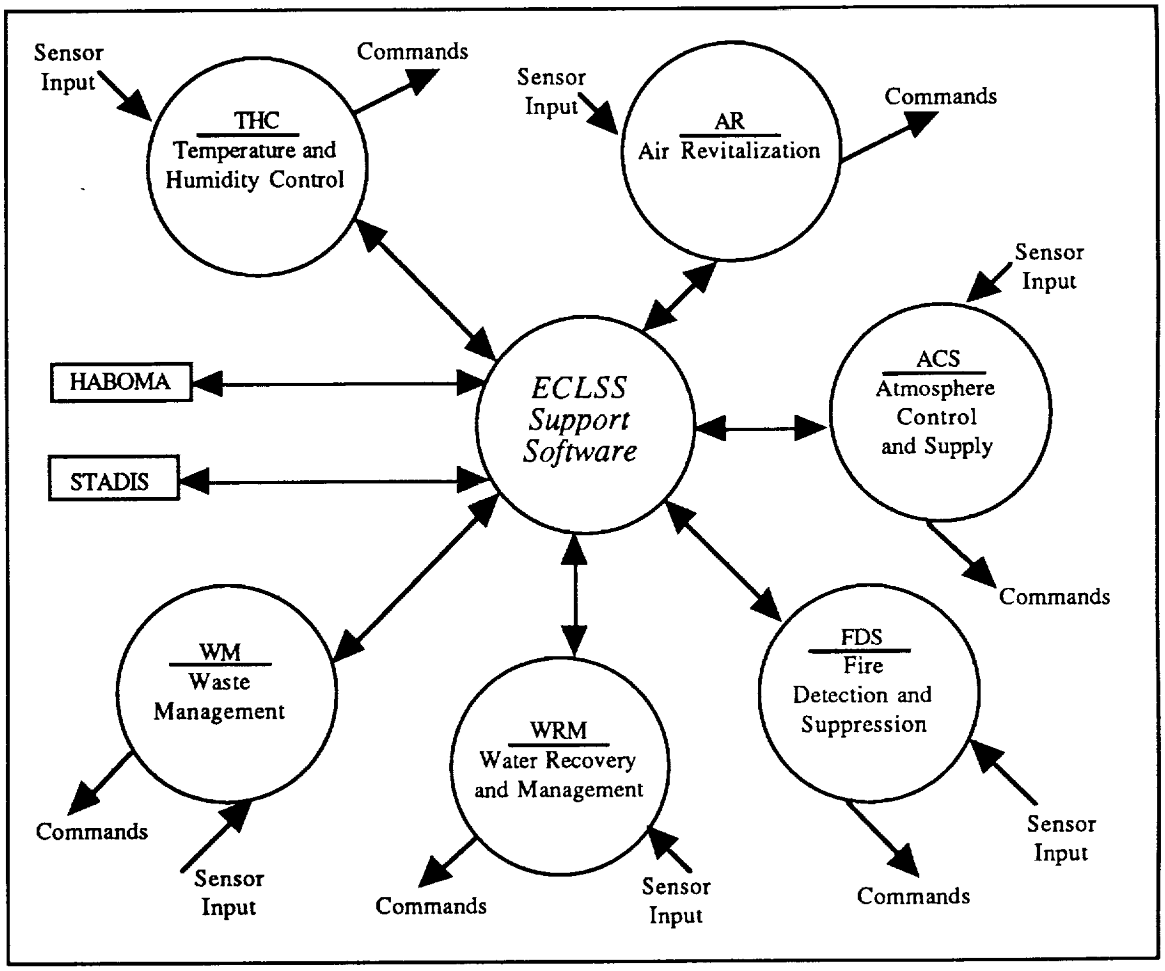
This separation of the rules of KBS and a simulation ought not to be surprising. Diagram 1 indicates the general flow of the simulation plus KBS structure.



As the diagram illustrates there are seperate knowledge sources for both the KBS and simulation engines. This is not unreasonable. The knowledge needed to make a diagnosis or evaluation may be and often is distinct from the knowledge needed to simulate a physical system. Indeed it may be the case that simulation will accept any initial values and attempt to set output values. This is a desirable feature of the simulation since the simulation should be used to simulate 'bad' as well as 'good' events. Further, it should be noticed that the separation into results that agree and results that disagree can be altered to advantage of different comparison strategies. The subtractive strategy is just one strategy. Other strategies may rely on goodness of fit, trend analysis, etc. The result of the comparison once agreement is reached is either to discharge or accept the previous reasoning. In either situation the reasoning is in principle non-monotonic since the information present at a later time is used to accept or reject the reasoning at a prior time. Simply put what at one time appeared to be the correct message, no longer does.

ECLSS SOFTWARE

In a previous report UAH Report No. 823 the following overview of the ECLSS software suppport was presented in the context of an appraisal of the ECLSS for Space Station Freedom. In this report that material will be extended with a specific focus on the ways in which a layered MAS might be added to the system and the places where simulations could act in cooperation. For full references see:



Context Diagram for an Overview of ECLSS Software

The software for the ECLSS incorporates several different levels of function and processing. This section of the report will focus on the way in which the software leads from the ECLSS Software Support module (now called ECLSSMGR) to the potable water software (POTH2O). No attempt is made in this section to give an exhaustive account of the software for each ECLSS subsystem. However, the discussion of the ECLSS Support Software will apply to all of the subsystems.

ECLSS Support Software (ECLSSMGR)

The ECLSS Support Software is intended to aid in the administration of the subsystems of the ECLSS. It is through the ECLSS Support Software that the Space Station's HABOMA and STADIS software are informed of the conditions and needs of the ECLSS system. It should be noted that the organization of the software is a logical organization and not a physical organization. It is not a question of where the software is, but of what the software does. Thus, the software for the subsystems and the ECLSS support software may reside in physically distinct computers, or the same computer.

Level	Short Name	Long Name	New Name
Station			
	STADIS	Station Distributed System	
Habitat Module			
	HABOMA	Habitat Operations Management Application	
ECLSS			
	ECLSS Software Support		ECLSSMGR — ECLSS Manager
	THC	Temperature and Humidity Control	
	AR	Air Revitalization	
	ACS	Atmosphere Control and Supply	
	FDS	Fire Detection and Suppression	
	WRM	Water Recovery and Management	
	WM	Waste Management	

The context diagram indicates the ways in which the ECLSS Support Software (now called ECLSSMGR) coordinates the activities of its subsystems (THC, AR, ACS, FDS, WRM, WM) and passes information to the habitat module level software (HABOMA) and the station level software (STADIS). (Larger context diagrams appear at the end of this report.) Each subsystem of ECLSS is represented in the diagram. At this context level, each subsystem can be thought of as taking in sensor data and, on the basis of its code and communication with the ECLSS Support Software, issuing commands to the hardware. Taken in this way, the software packages for each of the physical ECLSS subsystems represents the actions that ought to take place given certain sensor readings and communication with the ECLSS Support Software. In terms of the context diagram at this level, there is no direct communication of the software for the ECLSS subsystems to the habitat or station level software. Rather it is the ECLSS Support Software that communicates with those software packages.

The ECLSS Support Software receives commands from both HABOMA and STADIS and sends requests and status information to HABOMA while operational data is sent to STADIS. Each subsystem sends requests and status information to the ECLSS Support Software, while the ECLSS Support Software sends commands to the subsystems.

At the level of the overall system there are several issues that a layered MAS will need to address. The first is the specific cites at which it would be appropriate to have reasoning rather than purely computational agents. Although NASA appears to be committed to incorporating advanced technology into Space Station Freedom the cites should be carefully chosen and there should be a clear reason for using the newer knowledge based technologies. The second issue to be addressed concerns the way in which the various knowledge based agents should communicate. In part this will be an issue about where (in terms of processor units) the software is located and what the communications protocols are. This can be illustrated readily. If two logically distinct agents are to be used on the same processor, then the memory of that processor can be shared by the agents. This ideally would be the case for bureaucratic and scientific agents in the same domain. However, it need not be the case that the agent's can share physical memory. In this case the information and knowledge in memory must be packaged and communicated to another agent. This may lead to complications that make a multiple agent system oppressive. The third issue concerns the decomposition of the tasks about which the agents are to cooperate. Here there are several different ways in which complicating design factors may play a role. As an example, it might be the case all FDS operations in all parts of the Space Station are to be centralized. In this situation there is only one FDS agent, all subcomponents concerned with FDS are parts of the FDS agent, and the physically different parts of the Space Station are different end nodes of the FDS. On the other hand, one could design the FDS in such a way that there are multiple FDS agents and these agents correspond to the physical parts of the Space Station. In this situation there is no agent that is the FDS agent. Actions occur because of the actions of various FDS agents in the various physical components of the Space Station. In the first alternative the ECLSS agent is to be taken as a centralized bureau with the Space Station; in the latter the ECLSS agent stands for a collection of individual agents distributed through out the physical Space Station.

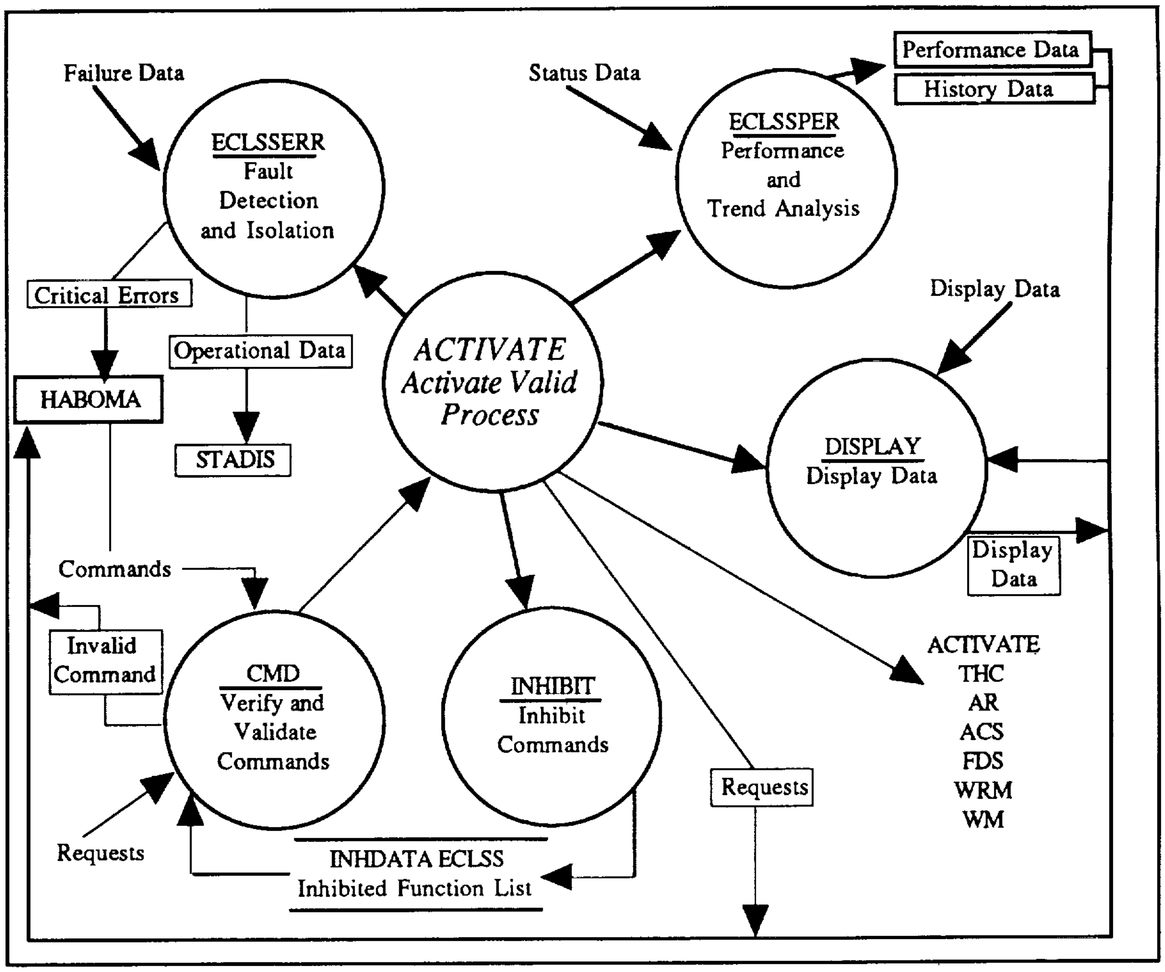
The issues of cites, communications, and distribution are issues that should be addressed early and often in the design of the Space Station. In terms of a layered MAS approach this is a necessity since these factors amount to the operationalization of the problem decomposition. Since the layered MAS approach assumes such a decomposition, the actual work of deigning this sort of cooperative system cannot begin until the decomposition is specified. It should be clear, however, that these sorts of issues gain prominence when the layered MAS perspective is under consideration. Hence at this level the adoption of the layered MAS perspective leads to a focusing on particular design issues that will be useful even if a system of cooperating agents is not adopted.

Inside of the ECLSS Support Software are six subcomponents that handle the incoming information from the ECLSS subsystems, HABOMA, and STADIS.

Short Name	Long Name
ACTIVATE	Activate Valid ECLSS Process
INHIBIT	Process ECLSS Inhibit Commands
INHDATA ECLSS	Inhibited Function List
CMD	Verify and Validate ECLSS Commands
ECLSSERR	ECLSS Fault Detection and Isolation
ECLSSPER	ECLSS Performance and Trend Analysis
DISPLAY	ECLSS Display

As indicated in the context diagram for the ECLSS Support Software, ACTIVATE is the central subcomponent, since it issues commands to ECLSS subsystems and requests to HABOMA. The commands, however, are checked before an activation. INHIBIT, INHDATA ECLSS, and CMD are the modules that check for processes that are inhibited. ACTIVATE sends a process name to INHIBIT which in turn sends a message to the inhibited function list in INHDATA ECLSS. The resulting process status is sent to CMD. CMD receives requests from all ECLSS subcomponents, as well as commands from HABOMA and STADIS. CMD indicates invalid commands to HABOMA and valid commands to ACTIVATE.

ACTIVATE also sends commands to ECLSSERR, ECLSSPER, and DISPLAY. ECLSSERR receives failure data from all ECLSS subsystems and sends critical errors to HABOMA and operational data to STADIS. ECLSSPER receives status data from all ECLSS subsystems and sends history, performance, and status data to HABOMA and DISPLAY. DISPLAY receives display data from all ECLSS subsystems as well as ECLSSPER, and sends display data to HABOMA.

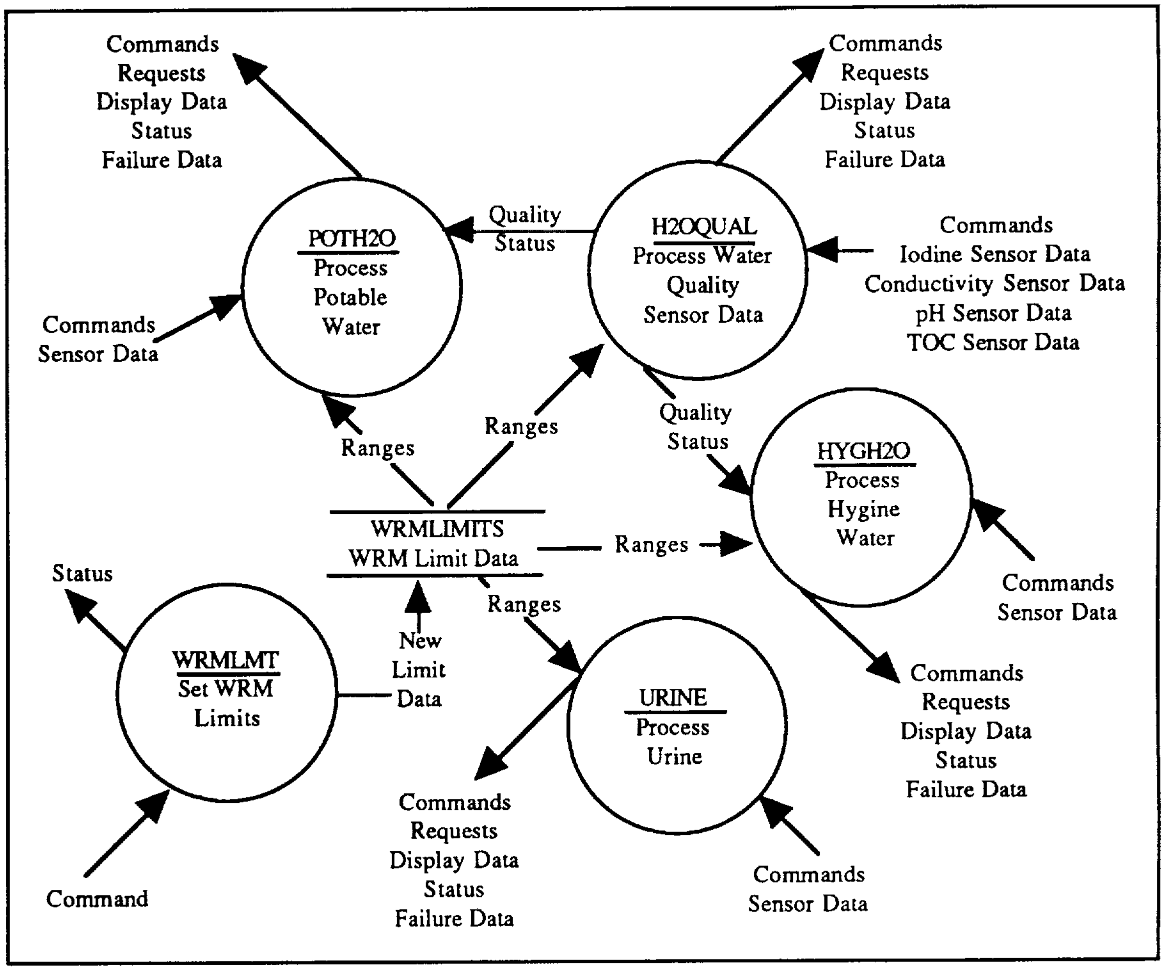


Context Diagram for ECLSS Support Software — ECLSSMGR

In terms of the layered MAS approach this organization of ECLSS components exhibits the central characteristic of the approach. There is a heavy reliance on result sharing. This is especially so in considering the dynamics of the process and the operations of ECLSSERR and ECLSSPER. In this case the results of the two agents would be shared. In this sense the results of performance analysis are relevant to fault detection and the results of fault detection are relevant to performance analysis. The case of DISPLAY is somewhat different. If DISPLAY is the agent that receives all ECLSS data and displays it to the operator, then the information that is received from the other agents will be more complexly structured and include information on the priority of the event, the time of the event, and perhaps on the way in which the information should be displayed. The results of display are not shared with other computational agents, but with a human. However, even here there may be a form of result sharing if DISPLAY is also charged with accepting user inputs that initiate actions. In this case DISPLAY must reason about which other agents should be notified.

As in the top level of ECLSS there are again problems connected with whether the collection of agents is taken to be a centralized or distributed bureaucracy. Are the agents in this collections agents for all ECLSS events, or are there multiple agents in charge of only ECLSS events in a particular part of the Space Station? This sort of question is persistent in the design as it currently stands.

WATER RECOVERY AND MANAGEMENT SUBSYSTEM (WRM)



Context Diagram for ECLSS WRM Software

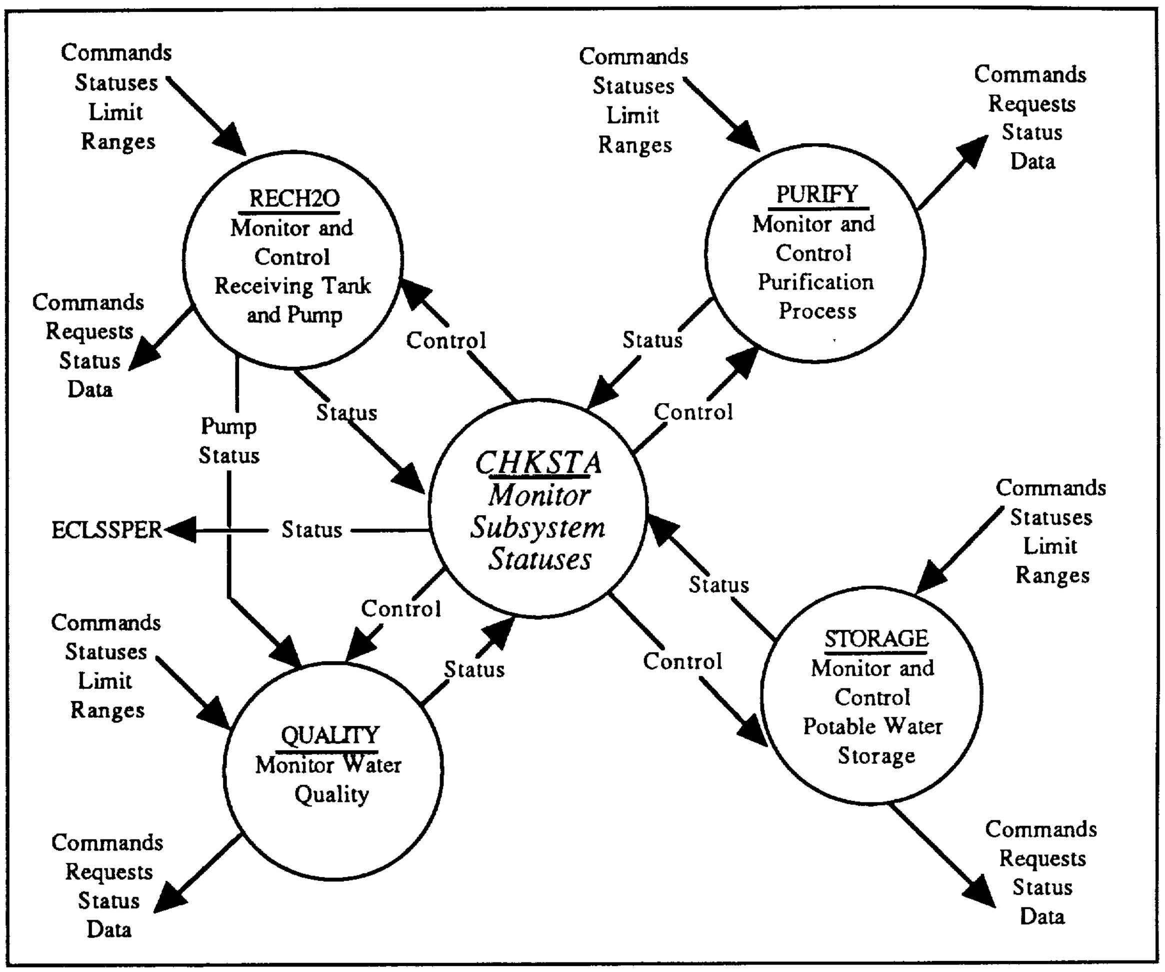
As the context diagram for the WRM indicates there are six modules that include both the potable and hygiene water systems. These modules analyze and control the water recovery functions in terms of the limits set for the subsystem. In general, each unit receives information sensors and various software units, and on the basis its processing generates new data and issues commands and requests.

Short Name	Long Name
WRMLMT	Set WRM Limits
WRMLIMITS	WRM Limit Data
POTH2O	Process Potable Water
H2OQUAL	Process Water Quality Sensor Data
HYGH2O	Process Hygiene Water
URINE	Process Urine

On the basis of commands from ACTIVATE, WRMLMT constructs new limit data for WRMLIMITS and reports its status to ECLSSPER. WRMLIMITS establishes the ranges for the main subcomponents POTH2O, H2OQUAL, HYGH2O, and URINE. Each of these subcomponents receives commands from ACTIVATE and sensor inputs from hardware. The case of H2OQUAL is somewhat special since it receives special sensor inputs for iodine, conductivity, pH, and TOC (Total Organic Carbon). Additionally, each subcomponent sends commands to hardware, requests to CMD, display data to DISPLAY, status data to ECLSSPER, and failure data to ECLSSERR. Additionally, H2OQUAL sends quality status to POTH2O and HYGH2O.

Within WRM, POTH2O will be taken as an example since the potable water system is the focus of the demonstration software produced for the overall effort of this research and presented with the report on the ECLSS project. The details of the demonstration system are presented in UAH research report No. 824. It should be noticed, however, that the demonstration software overlaps the functionality of POTH2O and H2OQUAL. In part this is because the physical potable water system overlaps both software components, and in part because the potable water design effort is geared to producing water of a specified quality. This raises the important fact that the breakdown of software components need not match precisely the breakdown of the function of the physical system. In the case at hand there seems to be an obvious reason why the software and physical breakdowns do not correspond. Although the potable and hygiene water systems are distinct physically, the process of monitoring water quality is sufficiently similar that one module (H2OQUAL) can satisfy the demands of both the potable water system (POTH2O) and the hygiene water system (HYGH2O). For reasons of economy, the monitoring process is placed in one module rather than in two.

The use of H2OQUAL by two distinct agents illustrates one of the advantages of result sharing in a layered MAS system. The advantage is that the centralization of function can result in an economy of computational units provided that the agent in question knows how to deal with various inputs and the agents sending messages know how to encode them. Further, it should be clear that H2OQUAL acts as an initial scientific agent that assembles the relevant data and builds a hypothesis on that data. The actual control of the processes is in the control of the other agents. The requests for diagnosis in these cases is forwarded to ECLSSERR.



Context Diagram for ECLSS WRM POTH2O Software

As indicated in the context diagram, POTH2O consists of five modules. The central module is CHKSTA. CHKSTA receives status information from each of the other modules and issues controls to each. Further, CHKSTA provides status information to ECLSSPER.

Short Name	Long Name
CHKSTA	Monitor Subsystem Statuses
RECH2O	Monitor and Control Receiving Tank and Pump
PURIFY	Monitor and Control the Purification Process
STORAGE	Monitor and Control Potable Water Storage
QUALITY	Monitor the Water Quality

Each of the modules that report to CHKSTA act on the basis of commands from ACTIVATE, limit ranges from WRMLIMITS, and controls from CHKSTA, and each module sends its status to CHKSTA and ECLSSPER, requests to CMD, failure data to ECLSSERR, and display data to DISPLAY. The modules differ in the statuses that are input and the hardware commands that are output. RECH2O receives sensor statuses from the liquid, speed, pressure, pressure transducer, and flow meter, as well as statuses from pump and valve, and sends commands to pump and valve as well as passing the pump status to QUALITY. PURIFY receives sensor statuses from pressure, pressure transducer, and temperature, as well as the status from heater, and sends commands to heater. STORAGE receives sensor data from liquid sensor, as well as status from valve, and sends commands to valve. QUALITY receives sensor status from temperature, as well as statuses from pump, valve, and H2OQUAL, and sends commands to valve.

At the lowest level of this part of ECLSS are the agents that are the most mechanical in nature. They function to monitor and control various aspects of the physical process and their results are sent to other knowledge containing agents which may in turn cooperate with other knowledge containing agents. This sort of structure allows that the system as a whole contains knowledge, but at the lowest level there is no need for such knowledge. Rather at the lowest level there is a need for specific functions that either produce the effects of a knowledge based decision or forward information to the knowledgeable agent. Look at as sort of organism this makes sense. At this level one is dealing with the computational analogues of sensors and effectors. Knowledge and knowledge containing agents are to be found elsewhere in the system

THE ROLE OF SIMULATIONS

Simulations in the ECLSS can provide a vital facility for checking the results of the operation of some knowledge based agent. The specific ways in which this might occur have been examined in the previous section of this report. At this point and with some of the detail in mind it should be clearer that such simulations are needed not only to check the results of some inferential process, but also to predict some of the actions that other agents may take. For example a general simulator in ECLSSERR or ECLSSPER might be able to predict not only what state the system should go to after some change, but also what agents should be activated and what they should do. Such a simulation may help to diagnose the health of the computational knowledge containing agents in way that is similar to the way in which a simulation may help with the faulty sensor problem. Again as in that case there can be no guaranty that the results of such an operation are accurate. The possibilities for error are vast and interactive. However, such a scheme can supply information that may aid in decisions about the general health of the system

SUMMARY

The ECLSS software is a layered collection of software modules that may be thought of as a logical hierarchy that can be located in physically distinct computers. Although only the path from the ECLSS support software to POTH2O has been traced analogous tracings could be produced for the other ECLSS subsystems. The use of a layered MAS approach in the analysis and design of this system may prove useful since there are multiple agents which may contain knowledge, there is a tendency to share results, and the decomposition of the system is contained in its design. Further, there appear to interesting reasons for joining the diagnostic and simulation approaches in the ECLSS system.

KNOWLEDGE ACQUISITION

INTRODUCTION

The familiarization aspect of knowledge acquisition (KA) continues from the beginning of a knowledge based programming project until the final content for the project is determined. Familiarization, in this sense, is not a distinct episode of knowledge engineering, but consists in all those activities which the knowledge engineer (KE) engages to prepare for the project and to prepare for particular knowledge acquisition sessions. It is important to note that these preparatory activities are both important and time consuming. They are important since they lay the ground for shared common content about the domain, and are time consuming since the knowledge engineer is required to become acquainted with terms, concepts, methods, and theories that may be far different from those with which he or she has already become familiar. In the familiarization process at the beginning of the project the knowledge engineer attempts to find the sources of important information, organize that information, read the documents, charts and other materials that have been assembled, and gain an elementary mastery of the vocabulary of the domain. As the project progresses the KE will continually need to become familiar with new material. However, the familiarization process for knowledge acquisition sessions based on this new material ideally should be less time consuming since a base has been established by previous efforts.

Familiarization is document-driven. Documents play a primary role even when there is a mentor to guide the KE through the material. The documents become a base on which KA can proceed. There are two reasons for this. (9) The first is practical. In order to interact effectively with an expert, the KE and the expert must have some shared conception of the domain. The shared conception is not to be understood as a detailed, precise, accurate or comprehensive account of the domain. Rather the shared account is the base that will continue to develop in the KA process. If the interaction with the expert requires that there be some common understanding, then it should be clear that in the beginning this must be provided in a way other than the interview process. In general, the information needed to establish this shared level of understanding is contained in documents associated with the expert's domain. The second reason is structural. Organizations collect knowledge in documents. These documents represent the stored knowledge of the organization. As such, the knowledge in these documents is social and intersubjective, and constitutes the background against which both individual knowledge and expertise are defined. Thus, for practical and organizational reasons the familiarization aspect of KA is document-driven.

The focus on documentation generates advantages.

- Documents are often "approved" knowledge sources.
- The writers of the documents have "decompiled" to some degree the domain knowledge.
- Documents tie down references in the knowledge dictionary.
- Documents make multiple lines of reasoning available.
- The documents provide a context needed to gain access to specific expertise.
- The documents provide a source of material for both explanation and help facilities.
- Attention to the documentation leads to the identification of weaknesses in the documents.
- Attention to the documents provides a point of reference for the expert and the user.
- Attention to documents leads to tighter coupling and resource-sharing between KE and technical writer.

Methods and aids must be devised to gain these advantages, however.(4) There are at least two ways in which such methods and aids might be built. The first focuses on the direct analysis of existing documents. The objective of this way is to create tools that would directly analyze documents and abstract knowledge. The second way focuses on the management of the familiarization process associated with the documents. We have adopted the latter way. The methods and aids that we are developing focus on the idea of a knowledge dictionary that is similar to the idea of a data dictionary in traditional database operations, and the expanded model of reasoning articulated by Toulmin, Rieke, and Janik (10).

A Documentation Approach to Knowledge Acquisition

Traditional approaches to knowledge engineering emphasize the interview process. Interview driven methods assume that interviewing an expert is the best way to acquire knowledge that is "chunked" and "compiled". Knowledge is "chunked" when items of knowledge are organized into meaningful units. Such chunking is believed to increase the performance of human experts. Unfortunately, such chunking makes knowledge acquisition more difficult especially when such chunks are "compiled"."Compiled" knowledge is knowledge that has been distilled and abstracted of all unnecessary elements; elements which may have originally been needed to gain the knowledge are removed. Further, the organizational schemas may be altered to increase the efficiency of recalling the items in the chunks. Compiled chunks may account for the fact that experts recall all of the content of one chunk before processing a subsequent chunk.

Knowledge engineers are familiar with the problems of chunked and compiled knowledge, and have developed various techniques for acquiring various kinds of knowledge. The documentation approach is consistent with the assumption that knowledge is chunked and compiled, and adds to the available techniques, especially those available during the familiarization activities of knowledge acquisition. Such familiarization activities are part of the episodic units in knowledge acquisition. The episodic units of knowledge acquisition include preparing, conducting, and reviewing interviews. The preparation activities which include familiarization are most intensive during the initial phase of a project. Although it is difficult to obtain data on this topic, informed, but informal, estimates suggest that during the initial phase of a project the ratio of preparation time to session is as high as 8 to 1, while over the life of the project the ratio might be closer to 3.5 to 1. (3) In either case it is clear that a significant portion of a knowledge engineer's time is spent in preparation activities and that such activities are more time consuming during the beginning of the project.

Preparation during the initial phase of a project is a complex undertaking. The knowledge engineer's activities are geared to becoming familiar with the domain. But how does one become familiar with a domain and in what does that familiarity consist? Our suggestion is that the KE becomes familiar with the domain through documents and that this familiarity leads to the production of a knowledge dictionary.

During preparation, the KE attempts to amass documents about the domain. Such documents include text books, reports, instructional materials, design plans, and, in general, any written (hard copy or electronic) materials about the domain. In using the documentation approach, it is assumed that documents have a degree of authority for the experts in the domain, that the experts in the domain would recognize the authority of the documents, and that documents are written and revised in order to establish common understandings and frameworks. We do not assume that there is any direct correspondence between the chunks and terms identified in the documents and those used by domain experts. We do assume, however, that the documents act as constraints on the domain expert. In this sense, the documents constitute an official and authoritative framework within which the expert brings his or her skill to bear. These documents act as a backdrop for two important KEing tasks: building a knowledge dictionary and analyzing it.

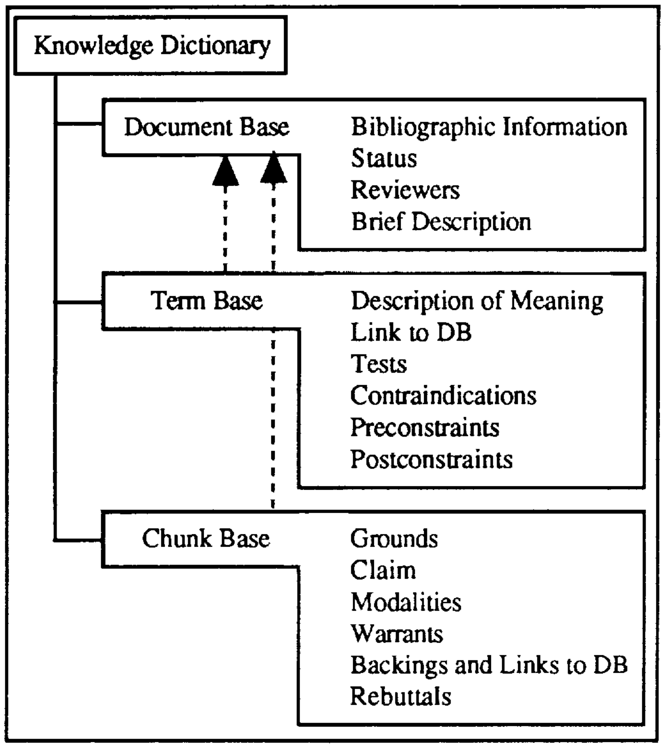


Figure 1

A knowledge dictionary, in its initial formulation, consists of a document base, a term base, and a chunk base. (See Figure 1) These three bases provide a map of the domain and a "first pass" collection of materials for automation.

The document base consists of bibliographic material, the status of the document, indications of whether and by whom a document has been reviewed, and a brief general description of the content and utility of the document. The materials in the term and chunk bases are keyed to these document base. Since in many cases the documents undergo revision as the project evolves, keying the terms and chunks to the document base provides a way of systematically reviewing the materials in the knowledge dictionary in light of revised documentation.

The term base provides information about the meaning and application of the term. An entry for a term provides a brief ordinary language description of the term and any appropriate abbreviation or symbol for it. Additionally, an entry contains typical information about the values the term may take, tests associate with the term, contraindications for the application of the term, and pre and post constraints on the application of the term. The specification of the source for the information provides a link to the documents base.

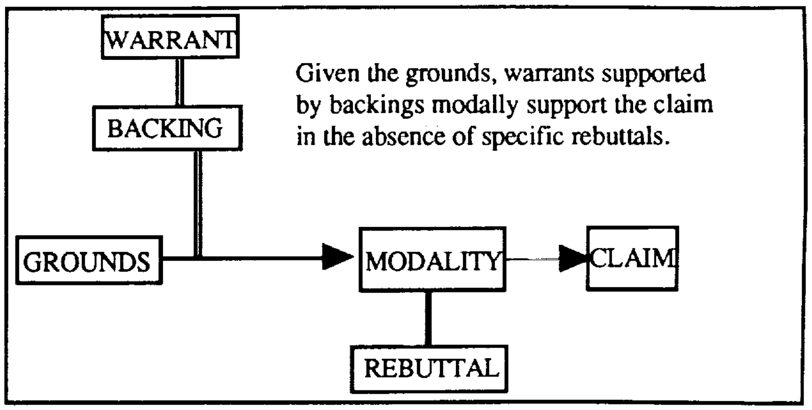

WARRANT
BACKING
Given the grounds, warrants supported by backings modally support the claim in the absence of specific rebuttals.
GROUNDS
MODALITY
CLAIM
REBUTTAL

Figure 2

Knowledge in the chunk base is represented using the Toulmin, Rieke, and Janik (TRJ) model of reasoning. (See Figure 2). Using this model a knowledge chunk is treated as an argumentive or inferential structure, However the model allows for greater depth and flexibility than more strictly logical models. When working with documents, one notices the flexibility of arguments. Even in highly technical areas, assumptions and premises are often not made explicit. (6) Further, the use of language makes it possible to link various knowledge elements in subtle, but important ways. The TRJ model more closely represents this sort reasoning.

In the TRJ model a claim is analogous to a conclusion in a deductive model or the the facts that are added to the knowledge base after a rule is fired in a rule based system. The grounds are analogous to premises or the facts in the knowledge base before a rule is fired. The modality can be thought of as a confidence value or some other measure of the strength of the claim based on the grounds. The warrant is most often the conditional statement that allows the grounds to lead to the claim. The warrants may be expressed as a rule, but other representations are possible. The backing indicates the support or basis for the warrant. The rebuttal indicates the considerations that would inhibit or prevent the assertion of the claim.

The TRJ model of reasoning is much more flexible than traditional models that emphasize the logical (propositional and predicate) style of representation. First, it should be noted that a warrant might have multiple backings. If this is so, then the removal of any one of the backings is not sufficient for retracting a warrant. This suggests that a modal logic might be applied to reasonings about warrants. This has been explored in Rochowiak (5). There a very minimal modal system, **T**, showed promise. Second, the use of the rebuttal notion may prove valuable in nonmonotonic reasonings. For example, if the reasoning unit were implemented in a frame like structure, then the rebuttal slot could be used as a trigger for retracting the rule's application and the retraction of the facts asserted in the claim. Or, it could be used to prohibit the application of a rule that would otherwise match a pattern in the facts. Third, it should be noticed that the notion of a backing provides a very natural way to include references to documents and can be easily extended to include the statements of experts in interview situations. Finally, the availability of backings for warrants (rules) allows for a clearer separation of the system and domain oriented notions of explanation . (7,8).

Given a knowledge dictionary composed of the bases specified above how is it to be analyzed? This question can begin to be given an answer by specifying the sorts of operations that a KE would want to have performed on the bases.

Beginning with the simplest case the KE should be advised of possible alteration sites when a document is updated, revised, deleted, or in some way altered. In an effort to build an essentially bureaucratic system this would be of great importance. A bureaucratic system is one that attempts to automate some process in a bureaucracy. The administration of loans and the purchasing of parts are typical examples. In these cases new rules or forms may require an alteration in either the term or the chunk bases. Another important arena is that in which the KE activity is occurring while the domain is being constructed. In this arena changes to the domain in terms of designs or specifications may force changes in the dictionary. An operation for alerting the KE to potential changes is needed for the analysis of the dictionary. A more complex case involves the grouping and reporting of the materials in the dictionary. Operations that would provide reports of how terms, chunks, and documents are linked, as well as the frequency of such linking, would help the KE to better understand how the knowledge is clustered. At another level operations that would identify some gaps or sites for decomposition are desirable. Such operations might begin with the identification of terms used in the chunks that are not defined or the identification of missing elements in the definition of the term. The identification of empty elements in the chunks would be equally important. Additional operations would be desirable for allowing multiple views of the knowledge dictionary, tracing of particular elements (say particular backings or typical tests) through the knowledge dictionary, and identifying links between chunks (grounds to claims, claims to grounds). Finally in keeping with the spirit of the documentation approach, there should be operations that would generate relevant sections of the knowledge dictionary as a document. Such documents would be useful in creating reports, setting agendas for interviews, and constructing materials for interviews. This is not, of course, an exhaustive account of the sorts of operations that a KE might need in analyzing the knowledge dictionary, but it is indicative of the kinds of concerns that are relevant in the construction of a tool for generating and analyzing a knowledge dictionary.

The process of analyzing and updating the knowledge dictionary is one that will continue over the life of the project. Ideally, the final dictionary would contain all of the information required to document the knowledge aspect of the finished system. For example, the coding that implements a chunk should be tied to the dictionary. This sort of tie would facilitate the identification of the locations in the code that need to be update as a result of some change in the domain knowledge.

A TOOL FOR THE DOCUMENTATION APPROACH

A tool that implements the documentation approach to knowledge acquisition is being developed. "Knowledge Management Tools" (KMT) is constructed primarily in HyperTalk ™ and CLIPS for the Macintosh™ family of computers. The use of a hypertext system is desirable since the hypertext facilities are of themselves useful in KA activities. (1,11). CLIPS is being used since it provides a readily available inference system.

The associational character of the construction and analysis of the knowledge dictionary strongly suggests that a hypertext system is appropriate. During familiarization the entry of data is not strongly structured. The KE may obtain information on one topic and then another without there being a clear connection between the units of information. However, as more information is entered into the dictionary it is reasonable to think that patterns will emerge. These patterns can be quickly and easily captured in associational links. Such links can provide a map of the material in the dictionary and represents the KE's view of the structure of the domain. Further, in familiarization the KE may become aware of new elements that should be added to the dictionary only in the process of reviewing information already entered. This again suggests that an associational link should be created that will allow the KE to easily add the needed information. Finally when there is more than one KE it will often be necessary to review what another KE has done. This review is again an associational link. Each of these reasons suggests the desirability of using a hypertext approach to the management of the familiarization process.

From a management point of view the knowledge dictionary can be treated as a (nonlinear) text. The production of the text should be such that a KE or a member of a KE team can add additional text to an entry during review. This factor means that the text in the knowledge dictionary not only can serve as the background against which a KE formulates interview sessions, but also is a means of communication for members of a KA team. The knowledge dictionary, in this sense, serves as a shared, common background for further knowledge acquisition. These management features again suggest that a hypertext approach is appropriate.

The inclusion of CLIPS in KMT is both an illustrative and cautionary tale. The inclusion of CLIPS was motivated by practical considerations. CLIPS is readily available, and some projects needed to use CLIPS. Further, since reading CLIPS code can be difficult, the direct association of the CLIPS code and the text in the knowledge dictionary would seem desirable. That is, the material in the knowledge dictionary would indicate what a segment of CLIPS code was intended to represent. On the other hand, this approach leads to an effort to coerce the information and knowledge into CLIPS form. This coercion while having some practical advantages leads to difficulties. Most importantly, rather that trying to capture knowledge and information as would seem to be natural, an effort is made to capture knowledge and information in a way that is amenable to CLIPS. It is almost as if an assumption is made that CLIPS is the appropriate tool for the domain. This difficulty is a general one. The problem can be put clearly in the following way: Should the selection of the tool be a determinate of the KA process, or should the KA process be a determinate of the tool? This essay will not attempt to resolve this difficulty, but it should be noted as a serious one.

KMT-CLIPS includes a K-edit stack, a K-dictionary stack, and a K-document stack. Separation of these three stacks adds to the efficiency of the system. The links between the stacks are in the beginning directed toward the K-document. The system as a whole attempts to keep a list of the associations. As the dictionary develops other links are established. Lists of these associations are also kept by the system. Internally, KMT is a collection of associated lists of association links. Access to the text information stored in the system is provided in various ways be accessing these lists.

The key stack is the K-edit stack. Currently this stack contains four screens. Future screens for analysis are planned. The idea of the K-edit stack is to allow the user to enter knowledge based elements and later provide the CLIPS code. However, this is not required and all materials can be entered at one time. Additionally, CLIPS is interactively available. The K-edit stack implements the idea of a chunk base only in a partial way in its rule card. Backings for the warrants are limited to References. Further since CLIPS is a rule based inference system the grounds and claims components of the TRJ model are identified as Conditions and Actions. The idea of a term base is also only partially implemented in the parameter card. The parameter card does not contain fields for all of the features identified above.

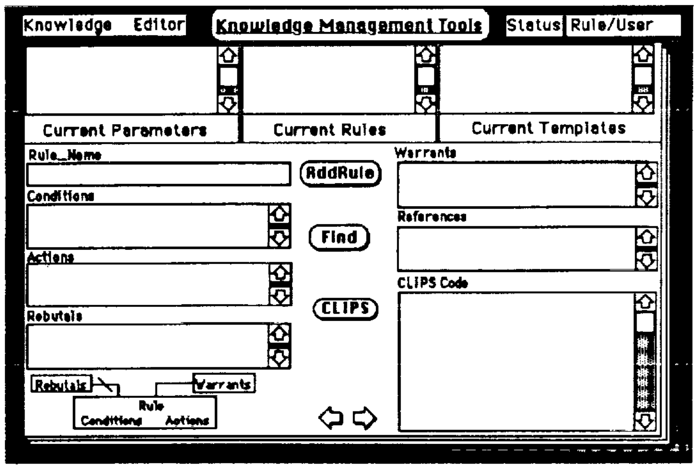


Figure 3

The first card of K-edit is a rule card that contains a chunk template. (See Figure 3) The user provides the name of the rule, its conditions, and actions as ordinary text. Rebuttals are specific circumstances that would prevent the application of the rule. Warrants are the reasons why the rule is being asserted. Both of these are ordinary text, as is the field for references. The diagram in the lower left illustrates the structure. At the top of this card and every card is a list of the currently known parameters, rules, and templates. This provides an interactive access to other parts of the knowledge dictionary. The entries for the term base are found in the Current Parameter and Current Template list. These identifications were selected to provide a more CLIPS-like interface.

The field at the lower right is used for CLIPS code. This may be added or not at the time the chunk is entered. Additionally, it can be tested by clicking the CLIPS button. This button will add the CLIPS code to a user specified text file, and additionally, if desired, load CLIPS and place that file in a buffer. The buffer can then be compiled and run. On exiting CLIPS, the user will be returned to the stack with the clipboard in tact. Thus, if modifications are made to the CLIPS code in the CLIPS environment, those changes can be copied and pasted into the card. It might be handy to have the new rule load into a buffer and then load the previous rules or templates into another buffer. The KE can then paste the new rule into the old CLIPS code and test it. When it is the way the KE wants it, it can be saved as a CLIPS code file. If this approach is taken the KE will need to clean up the dictionary at a latter date. By treating the CLIPS code as a document and building tools that understand CLIPS code, cleaning up the dictionary will be much easier. We are currently developing such tools.

The AddRule button adds the rule elements to a database indexed by the name. The Find button allows the user to find previous elements in the different bases. By selecting an element from those currently known and clicking Find, the user is taken to the database element in K-dictionary. Clicking the Return button in K-dictionary will return to the current card.

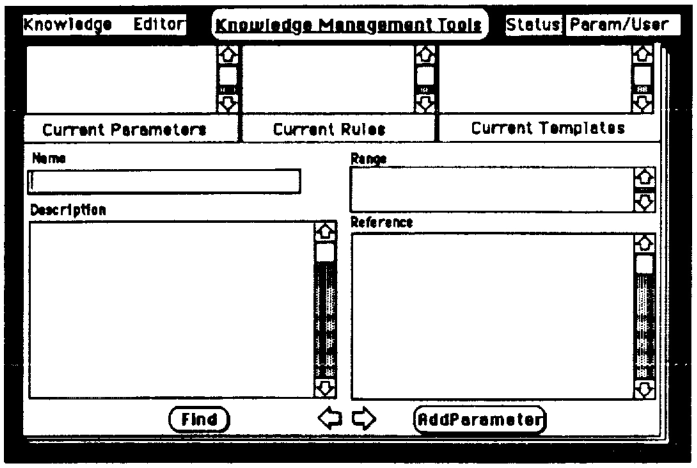


Figure 4

The parameter card partially implements the idea of a term base and functions in a way similar to the rule card. (See Figure 4) The parameters are used to identify the terms in the dictionary. Currently, only the description of the meaning of the term, the range of values, and the reference for the term are included. Fields for additional elements can be added. The AddParameter button adds the data in the fields to the database indexed by name in K-dictionary. Find will find the selected item as in the case of the chunk cards.

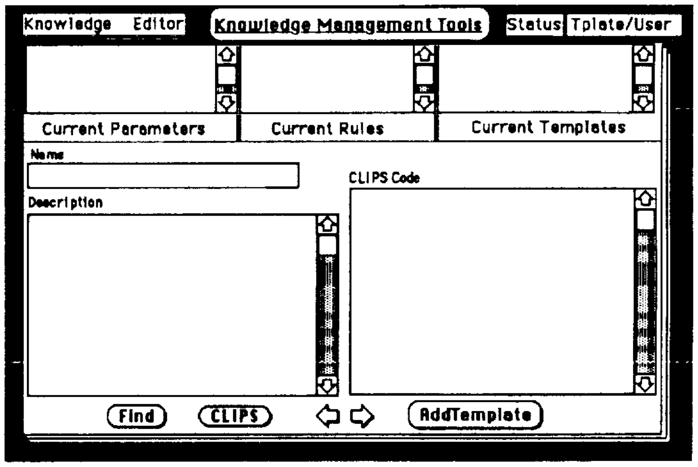


Figure 5

The template card is again similar to the rule card. (See Figure 5) The idea of a template is needed in order to make the general ideas of the documentation approach amenable to the latest version of CLIPS (4.3). A template is a structure that is somewhat similar to a frame and allows for more flexible access to and modification of facts. The template card also provides access to CLIPS since CLIPS code is used to define templates. It should be noted that the file to which the template is saved will load into the CLIPS buffer. If there is any additional editing to be done, it can be done there. The AddTemplate button adds the field to the database in K-dictionary indexed by name. Find will find the selected dictionary item

Currently work is under way to provide more of the features of the documentation approach and to generalize KMT. While there are practical reasons for orienting the system toward a specific inferencing mechanism, that selection also brings problems. The focus on rules and the need for a specific template card are examples. From the beginning, the KE is thinking in terms of the concepts and structures that will ultimately be used in the encoding of the knowledge rather than the knowledge itself. In an ideal case, the software should conform to the knowledge, rather than the knowledge conforming to the software. In improving KMT a more general approach will be taken.

The generalization of KMT will allow for alternative ways of entering information and provide a greater integration with tools that can be used in the interview process. Treating KMT as a "poor man's" knowledge acquisition tool, provides a way of adding different strategies. (4) Of particular interest are the additional representational strategies found in BDM-Kat and MacKat. (2)

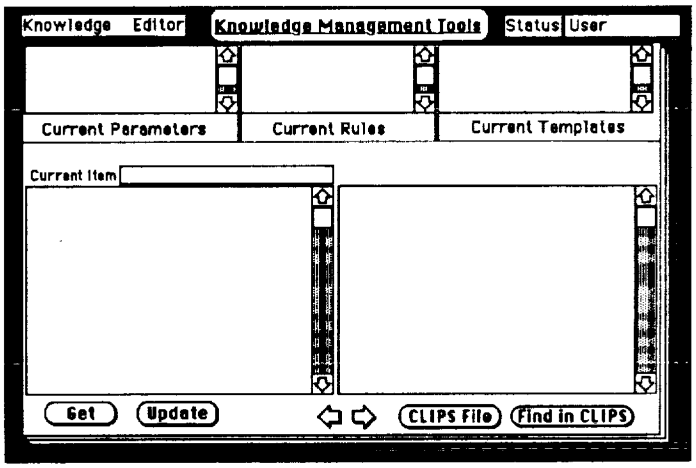


Figure 6

The compare card is used to generate comparisons between the database and the current CLIPS file. The CLIPS File button allows the KE to select a CLIPS knowledge base. The knowledge base is loaded into the field on the right of the card. Selecting a current parameter, rule, or template and clicking the Get button will load the database item from K-dictionary into the field at the left, place its name into the current item field, and move the CLIPS code to the first occurrence of the item. The items text can then be edited. Copy and paste can be used on the two fields. The Update button updates the database in K-dictionary. Similar procedures allow the KE to update the CLIPS code.

While the compare card performs several useful functions, there is much more that it should be able to do. Currently several features are being added that improve on the analysis capabilities of KMT and make greater use of inferencing about the material in the cards. For example, scanning the materials in either of the two fields should be able to produce a list of common items, and a list of items contained in the CLIPS code but not in the database. This would alert the user to check the common elements and to determine if new entries are needed for the elements in the CLIPS code that do not match terms in the dictionary.

K-dictionary and K-document currently share the same structure. The fields on the cards and the operations available are, however, easily tailored to specific needs.

The main card for the two stacks controls the operations of the stack.(See Figure 7) The buttons along the bottom of the card allow the KE to enter or alter the material in the stack rather freely. The New Term Button allows the user to enter a new term or document into the appropriate stack and indexes the entry. The Remove Term button removes the term and updates the index. In each case the scrolling list in the Dictionary Entries field is updated and alphabetized. The Write Dict. button allows the KE to build a text file of the materials in the stack. This file can be imported into a word processor, or saved as a separate file that can be loaded by Build Dict. This allows the user to operate with multiple files. The Clear Dict. button is used in conjunction with the two previous buttons to initialize the stack and prepare for a new file. The Browse by Letter area allows the user to select a letter and browse the entries for that letter. The scrolling list in the Referenced Topics field operates in two ways. In the first way the KE simply create a list of terms that he or she needs to add to the stack. Selecting one of these terms and clicking the Find Selection button will notify the user if the term already exists. If it does not the KE can then enter the information. The Find Selection button also works in connection with the items in the Dictionary Entries field. The Remove Ref Topics button clears the Referenced Topics field. Items in this field can also be cleared manually.

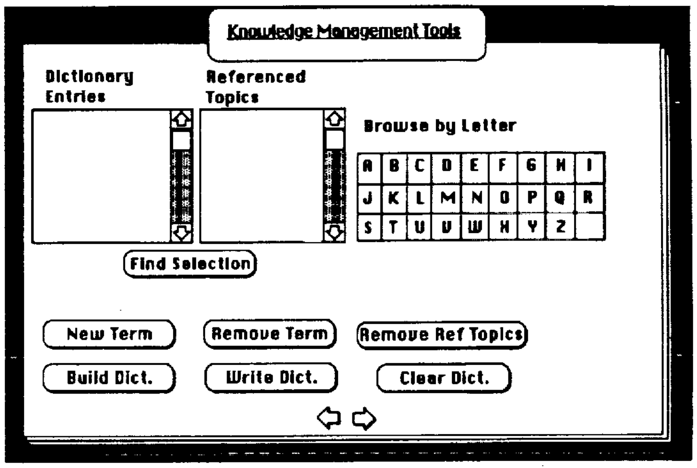


Figure 7

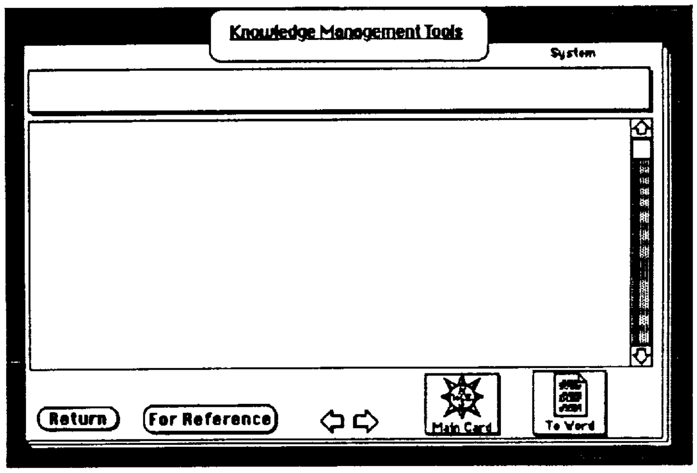


Figure 8

The cards that store the information currently have minimal structure. (See Figure 8) The information can, however, be structured in multiple fields, and resources in the stack allow the information to be gathered in multiple ways. The buttons on the bottom of the card provide a number of functions. The To Word button links to the KE's word processor. The word processor is loaded with the clipboard in tact so that the KE can paste the information into the document. The Main Card button takes the user to the main card of the stack. By highlighting material in the cards entry field and clicking the For Reference button, the Referenced Topic field of the main card is updated. This allows the KE to scan through a stack and quickly note terms that need definitions. The Return button takes the KE back to the K-edit stack, if this stack was entered through it.

The K-dictionary and K-document stacks currently share the same structure and are only distinguished by their content. Links can be established between the two stacks in several ways. We are currently working on ways to make the linking of the information in the three stacks easier. It should also be noted that the K-dictionary and K-document stack contain resources for formulating a frequency-recency model of the user interaction. This may prove to be helpful when an expert is allowed to view the stack or when tracing the flow of knowledge through a stack.

CONCLUSION

The documentation approach to the management of knowledge acquisition provides a way in which the familiarization aspect of knowledge acquisition can be made more productive. The emphasis on existing documentation, especially in bureaucratic systems or systems in the design phase, can be significant. The KMT tools partially implement the documentation approach. The KMT tools have been used on several projects and have been very useful. It should be remembered that the documents in a bureaucracy have been the traditional repository for its knowledge. The documentation approach is directed toward making use of this repository and augments the classical interview approach to knowledge acquisition.

ACKNOWLEDGEMENTS

Results presented at the 1990 NASA—UAH AI for Space Applications Conference.

REFERENCES

(1) Barrett , Edward(ed). *Text, ConText, and HyperText* (Cambridge, Mass.: MIT Press, 1988).

(2) McGraw, Karen L. "Developing a cognitively-based toolkit for knowledge acquisition," *Workshop on Knowledge Acquisition* (1989) 7-9.

(3) McGraw, Karen L. and Karan Harbison-Briggs. *Knowledge Acquisition* (Englewood Cliffs: Prentice Hall, 1989).

(4) Moseley, Warren. Final Report for FAST-1, "Automated Software Tools," 1989.

(5) Rochowiak, Daniel. "Expertise and reasoning with possibility," *Proceedings of the Second Conference on Artificial Intelligence for Space Applications*, 1987.

(6) _________ "Extensibility and completeness: an essay on scientific reasoning," *The Journal of Speculative Philosophy* 2 (1988): 241-266.

(7) _________ "Simple explanation and reasoning," *Proceedings of the AAAI'88 Workshop on Explanation,* 1988, 95-98.

(8) Rochowiak, Daniel, B. Ragsdale, and L. Wurzelbacher. "An integrated hypertext and rule based system for explanation," *Proceedings of Expert Systems '89*, 1989, 345-352.

(9) Rochowiak, Daniel, D. Hays, and D. Ford. "Document driven knowledge acquisition in the construction of expert systems for aquaculture," *Proceedings of Expert Systems '89*, 1989, 109-120.

(10) Toulmin, S., R. Rieke, and A. Janik, *An Introduction to Reasoning* (New York: Macmillan, 1984).

(11) Wells, Tracy. "Hypertext as a means for knowledge acquisition," *SIGART Newsletter* 108 (April 1989): 136-138.

Types of Explanation and Looseness of Knowledge

A central issue of the first AAAI Workshop on Explanation concerned whether explanations could or should be based on knowledge that is only loosely related to the knowledge used to solve a problem (Wick, 1989). In constructing a response to this issue it is important not to over state the response. It is highly unlikely that all explanations have the same form and that one type of explanation will be suitably universal to characterize all instances of explanations. I will argue that there are several types of explanation and that these types of explanation show why there are cases in which additional knowledge should not be used and others in which it should. Finally, I will indicate why knowledge that appears to be only loosely related has this appearance because of the way in which knowledge is acquired and not because the knowledge itself is loosely related.

Explanations come in a variety of types. (Rochowiak, 1988) In ordinary life explanations are generated in great abundance. (Schank, 1986) As a working principle one might hold that for every action an individual can generate some explanation for the act. In saying that the individual can generate some explanation this does not mean that the explanation is a good, valid, legitimate, or acceptable explanation. To add the qualifier is to appeal to some criterion by which the virtues of explanation are established and the character of particular explanations appraised.

A Pass at a Generic account of Explanation

One classic way to attempt to generate a generic account of explanation, is to follow out the views of the logical positivist school of philosophy. Simply put this conception holds that explanation is a deductive relation between initial conditions, scientific laws, and a the proposition (which describes an event or state) that is to be explained. This account holds the promise of giving what appears to be a general account of the way in which science operates. If the initial conditions and laws are known, then a proposition about a predicted event can be generated. If a proposition about an event is known, then the specification of laws and initial conditions will explain it. If the initial conditions and a proposition about an event are known, then induction might be used to discover a new law. Clearly this is an oversimplification, but it does capture the ideal of generic explanation.

There have been numerous criticisms of this generic account within the literature on the philosophy of science. These criticisms assert that the idea is too broad and, therefore, will allow arguments that are not explanations to be counted as such, and that the idea is too narrow and, therefore, will not allow for explanations that are otherwise legitimate. The specific nature of these criticisms are various. Some emphasize the notion of causality, and others the role of theoretical models. Some emphasize the role of history, and others the independence of explanation and prediction. Still others directly attack the notion of a generic account and the lack of a pragmatic dimension. It is the last pair of notions that is of interest in the present context.

One way in which an attack can be made on the generic account of explanation is to note the different ways in which reasons and causes may be used. For example, reason explanations often make references to intentions. For example, given a puzzling piece of behavior by Sally, one might offer the explanation that Sally intended to do this thing. This is in some ways a curious explanation, but it is often used. "Why did you do that?" "Because I intended to." Such explanation explain by attempting to remove the surprising character of some behavior. Of course such explanations can be expanded through further questions, or through the specifications of a series of things related to a cognate term. For example one might reasonably ask for explanation of the intention, "Why did you intend to do that?" On the other hand, one might have offered a related, but distinct, initial explanation, "I wanted to do X, and it appeared to me that doing A would get me (closer to) X."

What is interesting about these sorts of reason explanations is that they do not fit the generic model since they involve intensional elements and seem to violate the conditions on causal explanation. The former is so, since notions of intention, desire, and belief do not satisfy the extensional demands of substitution. The latter is so since it is a principle of causal explanation that the event that is the cause should be describable independent of the event that is the effect, and this does not seem to be so for intentions, beliefs, and desires. In short the deductive seductions of the generic model are undermined by the attempt to explain human actions through reasons. This generates a situation in which either this particular generic account must be given up and another produced, or it must be admitted that there are varieties of explanations.

That there are varieties of explanations is supported by the consideration of the pragmatics of explanation. The pragmatics of explanation involve the construction and presenting of explanations. Note that this is not an issue in the generic account. In the generic account an argument that satisfies specific formal and empirical considerations is an explanation even if it fails to explain anything to anyone. Thus, on the generic formalist view it is quite possible to assert that some structure S is the explanation of A, even though you don't understand why. In this situation the issue is simply that presenting the right sort of structure counts as an explanation even if one does not know why the elements of the structure are relevant, or what the elements mean. This is not an uncommon experience in the learning of science where the student can respond to a request for an explanation on an examination, yet still claim to not understand the material.

The pragmatics of explanation in which there is a demand that information in the explanation generates some understanding of the event to be explained raises problems of determining exactly what is to be explained, how it is to be explained, and to whom it is to be explained assuming that resources for constructing explanations are available. Thus, when considering the pragmatics of an explanation it is not sufficient to indicate a generic pattern of explanation unless it can be shown that it is reasonable to claim that a specified group of resources will be sufficient for determining what is to be explained and how it is to be explained to a potential diverse group of individuals. Ordinary experience seems to argue against this, however. Thus, in terms of the pragmatics of explanation it seems that there will be different types of explanations.

Assembling the different parts of the objections to the generic account of explanation, it appears that explanations must :

use available resources to construct accounts
respond to some surprising event
make the event less surprising
generate some increased measure of understanding.

Since there is no reason to suppose that these requirements can be satisfied with one generic mode of explanation, it seems reasonable to suppose that there will be multiple explanation types.

In the construct of automated explanation systems, it will be useful to consider the varieties of explanation from the metaphorical intentional stance. This stance assumes that the explanation system is metaphorically an agent in the same way that people are. This does not entail a claim of identity and only requires that that explanation systems and persons be treated similarly in ways that are relevant to explanation. Given the requirements noted above this entails that both the machine and human systems must satisfy the foregoing conditions if either is to be understood as presenting an explanation.

TYPES OF EXPLANATION

Ordinary explanations of actions tend to identify the reasons for the action and in general identify states of the agent that are prior to the agent's act. For example, leaving a wake-up call for 7:00 AM, might be explained by the agent's desire to be at a meeting by 9:00 AM. Or, if an agent has the belief that a due date for an abstract is April 6, this might be explained by the agent's prior belief that all abstracts are due on the same day and the belief that an abstract for another section is due on April 6. In both examples, the explanation of the action — leaving a wake-up call or believing that the due date is April 6 — is based on the presumed prior state of the agent and some pattern of explanation. These sorts of explanations might be called agent explanations since they are designed to explain the agent's actions in terms of the agent's states.

Such ordinary explanations have corollaries in automated explanation systems. One kind of explanation that an automated system might give is an agent explanation. Automated explanation systems give agent explanations when the system is designed to explain the system's action in terms of the system's prior states. As in the ordinary case, the explanation itself is structured in terms of some previously established pattern. That pattern might, for example, refer to the subgoals of a system's operation, the identification of a candidate schema that would solve the problem at hand, or the sequence of inferences that have been made by the system. In any case there is a significant disanalogy between the ordinary case and the automated system case. In the ordinary case it is at least difficult, and perhaps impossible, to determine what the prior states of the agent were. In the automated case this is not a problem.

A second kind of explanation does not focus on the agent's actions or prior states. In the ordinary case these explanations are explanations of external events, conditions, or things. The request for an explanation of the status of a piece of equipment or the color of someone's clothing, is not ordinarily treated as a request for an agent explanation. Rather, it is treated as a request for an ontic explanation. An ontic explanation is designed to explain why some state of affairs is the way it is, or fits into a pattern of the way things can be. In giving such an ontic explanation the account of how an agent came to a conclusion is not to the point. This so even if one holds that explaining is something that agents do. (Thagard, 1988)

In the realm of automated explanation, ontic explanations present a problem. They are a problem if automated explanation is intended only to give explanations of what the computational agent is doing. However, if the computational agent also attempts to explain why something is the case, then agent explanation is no longer satisfactory and knowledge outside of that by which an agent comes to a a particular conclusion may be needed. Hence to engage in the generation of ontic explanations is admit the possibility of using knowledge that was not used in coming to a conclusion.

In the case of both agent and ontic explanations knowledge not directly coded into the inferencing system is needed.

In the case of agent explanations this additional knowledge will concern the way in which the system is attempting to reason. Such knowledge might be called introspective knowledge and be represented in terms of meta-rules. (Rolston, 1988). This way of specifying what sort of additional knowledge is needed for the explanation system is attractive while the focus of the system is agent explanation. In agent explanation it should be clear that the system is attempting to explain to another what it is doing. Assuming that the inferencing system is a rule system, such knowledge about what the system is doing is compiled out of the system. That is, for the purpose of doing the inferencing, such knowledge is not needed. (It should be noted that this sense of a meta-rule is distinct from that in which meta-rules are used to guide or control inferencing.) Agent explanation at this level is akin to debugging in that the user is more or less presented with a report of what has happened. However, agent explanation can be taken to deeper levels by adding knowledge about the strategies that are being used in making the inference. Such strategic knowledge would be akin to the sort of knowledge ordinary human agents use in elaborating upon their intentions.

In the case of ontic explanations the additional knowledge concerns the way in which the system that is the object of the inferencing operates. This additional information does not add to the knowledge store concerning the way in which the inferences are made. Rather, the additional knowledge attempts to explain a conclusion of an inferencing process in terms of the event or state described in the conclusion. The knowledge required in ontic explanation is knowledge that is closely related to the sort of knowledge that would be required in building a simulation. Ontic explanations explain an event described in a proposition in terms of a model or theory of the world and not in terms of a model or theory of an inference maker. It is this fact that seems to require ontic explanation systems to be at least partially independent of the inferencing system. The distinction is between explaining how I came to a conclusion, and explaining how the event or state described in the conclusion follows from other known events or states. This distinction is all the more important when it is remembered that expert's knowledge is chunked, compiled, and formed in accord with certain heuristics. These allow the expert to quickly access a broad base of knowledge and come to an answer without it being the case that the way in which the expert would explain a conclusion could be easily mapped onto the way in which the expert produced it.

Thus in both cases it appears that additional knowledge is required for explanation. This additional knowledge may, however, be closely related to the knowledge used in the inferencing system. This is most clearly the case in agent explanations. In this case the additional knowledge required for the explanation system is in some sense knowledge already contained in the construction of the inferencing system. What is needed is a way of making that knowledge accessible to the end user. On the other hand ontic explanations do not seem to be directly or indirectly an extension of the inferencing system. The focus of the explanation system has changed. Rather than attempting to explain an agent's actions the system is attempting to explain why some event or state described in a proposition should be the case. Ontic explanation systems will require knowledge of a different sort since the system will not be a model of an expert's inferencing, but the content of the expert's knowledge.

LOOSELY RELATED KNOWLEDGE?

The determination of whether or not knowledge is 'loosely related' can be made in terms of whether the knowledge was used in coming to a conclusion. This is satisfactory in the case of agent explanation and loosely related knowledge should be avoided. In ontic explanation this is not so. Knowledge not used in coming to the conclusion may be exactly the sort of knowledge needed to explain why something is the case. In ontic explanation, however, it is not quite accurate to say that the knowledge used in generating such an explanation is loosely related. Rather such knowledge is tightly related to this type of explanation, even if loosely related in terms of agent explanation.

The relation of one item of knowledge to another is a function of the target to which the knowledge is to be applied. This is especially so if one adopts a declarative understanding of knowledge units. In the declarative interpretation a knowledge unit can be applied at any time. This is most clear in a rule based system in which the inference is made whenever the conditions of the rules are satisfied. In such a system knowledge is tightly related to the system when it is used to make an inference about some goal state. Thus, if there were a rule that was never fire in coming to satisfy any goal for which the system was designed, then the knowledge could be eliminated because it was loosely related. Further, if there were two more rules such that the terms on the right side of some of the rules where only intermediaries that were immediately used on the left side of other rules, then condensing the rules into single rule would be permissible if there were no loss of functionality. In this case the knowledge that was represented in the uncondensed rules represented knowledge that was only loosely related related to the system. Both of these cases occur in the construction of rule systems either at the coding phase or at the knowledge acquisition phase. In either case, the are a compilation of knowledge that removes knowledge from the system. In such cases the elimination of such knowledge is legitimated on the grounds that it is only loosely related to the inferencing process and that the removal of the knowledge may improve the performance of the system.

The attempt to provide systems that can build either agent or ontic explanations requires that decisions about such explanation capacities be made when the system is being designed (McKeown and Swartout, 1987). Once such decisions are made determinations of what knowledge is and is not loosely related can be made. This will have a most direct affect in the knowledge acquisition phase of system construction.

The apparent looseness of the knowledge is also a function of the knowledge acquisition process employed in developing a system. (Rochowiak and Mosley, 1990) If an effort is made to remove the explanatory component of the knowledge obtained during the knowledge acquisition process, then the knowledge used in ontic explanation will appear to be only loosely tied. However, if it is recognized that the knowledge acquisition process must also focus on the acquisition of knowledge for explanation as well as knowledge for reasoning, then much of the appearance of the looseness of the relation disappears.

CONCLUSION

It is dubious the there will be a satisfactory generic account of explanation. This is especially so when one considers the pragmatics of explanation. Given the desire for an explanation that satisfies pragmatic goals, more knowledge than that required for inferencing will be required.

Agent explanations that focus on explaining how the agent makes inferences to arrive at some conclusion, and ontic explanations that attempt to show why some event or state described in a conclusion are distinct. Each type of explanation will require knowledge not contained in the inferencing part of the system. Further the type of knowledge required for each of these types of explanation will be distinct.

The degree to which knowledge is loosely tied to a system is a function of the kind of explanation facilities that are to be included in the system. Knowledge not required for the inferencing part of the system will be required for the explanation part of the system. While the knowledge needed for agent explanation can reasonably considered to be an extension of the knowledge needed for the inferencing part of the system, this is less so for ontic explanations.

REFERENCES

McKeown and Swartout, 1987. "Language generation and explanation," *Annual Review of Computer Science* 2(1987): 401-449.

Rochowiak, 1988. D. Rochowiak, "Simple explanations and reasoning: from philosophy of science to expert systems." In *Proceedings of the AAAI'88 Workshop on Explanation*, p. 95-98.

Rochowiak and Mosley, 1990. D. Rochowiak and M. Mosley, "Documentation and knowledge acquisition," To be included in the *Proceedings of the Fifth Conference on Artificial Intelligence for Space Applications.*

Rolston, 1988. David Rolston, *Principles of Artificial Intelligence and Expert Systems Development* (McGraw-Hill, 1988)

Schank, 1986. Roger Schank, *Explanation Patterns* (Lawrence Erlbaum, 1986).

Thagard, 1988. P. Thagard, *Computational Philosophy of Science* (Bradford Book / MIT Press, 1988).

Wick, 1989. M. Wick "The 1988 AAAI Workshop on Explanation," *AI Magazine*, Fall 1989 p. 22-23, 25-26.