MULTIS: A Knowledge Acquisition System Based on Problem-Solving Primitives

Yuri Adrian Tijerino
Tadahiro Kitahashi
Riichiro Mizoguchi

Institute of Scientific and Industrial Research,
Osaka University

Abstract

Building knowledge-based systems is not an easy task, especially when it comes to acquiring and coding expertise from human experts into machine understandable forms. Attempts, some successful, have been made to automate this very tedious task. Nevertheless, those efforts have been focused on specific task areas of knowledge-rich human problem solving, e.g. diagnosis, classification, etc. So far, few automated knowledge acquisition systems exist that are useful in more that one such area. The aim of this project is to develop a method for automating knowledge acquisition that can be employed to automatically generate knowledge-based systems making few initial constraints on the type of problem solving mechanism. This method is based on the following concepts: (1) human problem solving can be broken down into conceptual primitives that can be reused and rearranged to model problem solving processes from vast domains and task types, and (2) there exist enough formal problem solving mechanisms in algorithmic form, or at least those mechanism can be easily developed by computer experts, to implement those models in a machine understandable code. To prove the tractability of these concepts we are currently developing MULTIS (MULti-Task Interview System).

Keywords: Interview system, cognitive primitive, generic vocabulary, generic process, building block, case-based reasoning.

1. Introduction

In the past two decades intensive research has taken place in the area of knowledge-based system in numerous countries. As a result, so-called "expert systems" have been constructed to store human problem solving knowledge in the computer in a dynamic form that can be used as an aid to the knowledge provider and other parties interested. Originally, those systems, which will be referred to as knowledge-based systems (KBS) hereof, were computer programs designed by "knowledge engineers," a term coined by Prof. Faigenbaum in the early years of artificial intelligence. Knowledge engineers are usually persons that would spend numerous hours talking to human experts, who in some cases would think of them as intruders poking in their own business. Through this painful but often rewarding period, knowledge engineers have accumulated readily useful knowledge of their own that has been written in the form of papers, books, etc. This knowledge of how to acquire knowledge from experts and encode it into computer programs, can now serve another very important purpose, that is, to create systems based on that knowledge to automate the process of knowledge acquisition.

Interview systems are computer programs that are constructed with that purpose in mind. Early interview systems were generated from existing knowledge-based systems and were useful in aiding in the construction of other knowledge-based systems that presented problem solving mechanisms similar to that of the original one. Based on this fact, it was proposed that there exist cognitive primitives (Chandrasekran, 1986; Clancey, 1985; McDermott, 1988, Tijerino et al., 1990) in which terms most human problem solving could be modeled. But how could that be possible for all facets of problems? Problem solving knowledge was subdivided in two major areas: task knowledge and domain knowledge. From this perspective, primitives could then be identified for task knowledge. Since it was also proposed by the above researchers that task knowledge presented strong guidance in the process of acquisition of domain knowledge, then those primitives would be of great help in that process too.

The aim of this project is to develop an interview system, MULTIS, based on the concept of cognitive primitives, also known as mediating representations. The major difference between this project and other related ongoing projects is that we make a distinction between domain expert's cognitive primitives and knowledge engineering primitives. Other projects consider the only the latter leaving it up to the knowledge engineer to analyze the domain expert's problem solving process and synthesize KBSs with reusable mechanisms. We propose that the former can be extremely useful in the process of modeling problem solving interactively with the domain expert and that the former can be mapped to the latter, thus automatically generating the target KBS.

2. Two-Level Domain Expert's Cognitive Primitves with different Grain Size

We make no claim that we have identified "the" primitives of human thought. On the contrary, in this paper an experimental framework is presented in which conceptual primitives do not necessarily have to be precise in modeling human thought, but only accurate in creating systems that model thought processes in the form of computer programs.

In this section, we will present two types cognitive primitives used to model expert problem solving processes: generic vocabulary and generic processes. Notice that these are only equivalent to domain expert's cognitive primitives and no attempt is made to implement them directly as computer programs. As figure 2-1 illustrates, their implementation is accomplished by mapping them to mechanisms that represent knowledge engineer's primitives which we refer to as building blocks and will introduce in another section. A comparison of our research with other ongoing research efforts will also be given. Last, we will provide an example of how an already existent system can be modeled with those primitives.

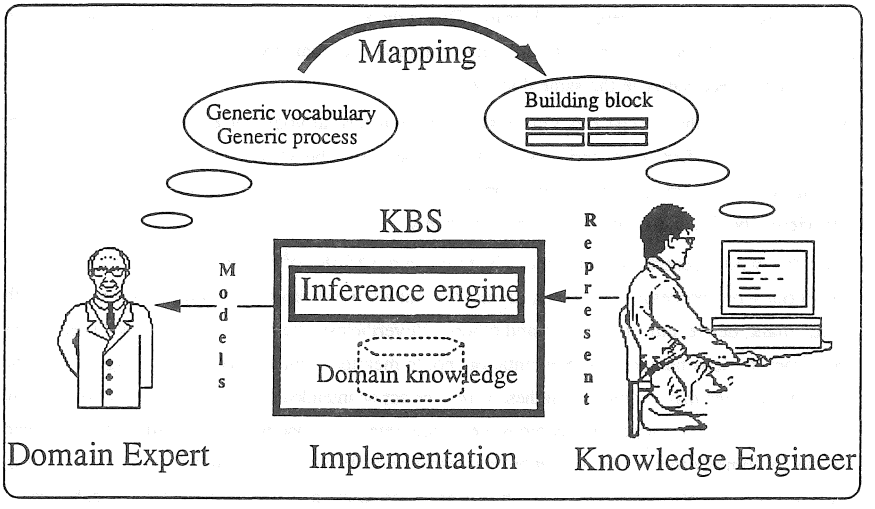

Figure 2-1. Implementation of a KBS is accomplished by mapping domain expert's primitves to knowledge engineer's primitives.

2.1 Generic Vocabulary

One kind of cognitive primitive used by all fellow human beings can be identified as the vocabulary. Not a particular language itself, e.g. English, Japanese or Spanish, but the concept represented by the vocabulary of the language, e.g. the concept given by the word "book." It is important to make this distinction because it is what makes communication possible or impossible between two parties. For instance, let's suppose a person A does not know the meaning of the word "book." Let's suppose further that another person B utters the word "book" to A in A's language. Tough A will not understand the word, it does not imply A's ignorance of the word itself, e.g., by hearing the word A has just learned of its existence. On the other hand, A will not learn the meaning of the word, i.e. the concept implied by the word, until either B shows a book or explains its meaning to A. Furthermore, once the assimilation process has been completed, A will add the new concept given by the word to his/her vocabulary, not his/her language, and will be able to use it or understand it in future occasions.

Vocabulary can either be very generalized or very specialized. Generalized vocabulary is learned at the period of adaptation during one's youth and as one becomes older or experienced at doing some particular task or job one learns more specialized vocabulary. The latter is usually explained in terms

of the former, which is not a reciprocal relation. For instance, the concept of book can be explained in terms of words such as paper, writing, tales, drawing, etc, but those words cannot be explained in terms of book without major effort. As such vocabulary becomes more and more specialized, it becomes dependent on some particular task and common only to individuals with experience in that task. Furthermore, as one gains a vocabulary related to an specific task in which one is experienced, e.g. medical or equipment failure diagnosis, that vocabulary becomes the basis for explaining concepts more specialized and for communicating with colleagues, in common task-dependent vocabulary, without resorting to more generalized terms. This is the kind of vocabulary to which we will refer to as "generic vocabulary." Generic because it is common to domain experts performing the same task type, but not necessarily in the same domain.

We assume that there exist this kind task-dependent terminology for each PS task that can be represented in the form of generic vocabulary. Generic vocabulary can serve two purposes, that is, to analyze the tasks for hidden cognitive primitives, which we call generic processes* and to help the interviewer be fluent and coherent during interview.

In the real world, generic vocabulary can be found in most fields of human cognitive processes. Let's revisit our word-processing example. In most, if not all, word-processing programs one can use the words mark, delete, cut, copy, paste and others. Nevertheless, the name of this actions (verbs) may vary from implementation to implementation, not to mention from language to language. No matter how different one may choose to call these verbs, the action intended is still very similar if not the same. Due to this fact, when a person becomes well acquainted with one word processor it becomes very simple for him/her to use any other word processor once the necessary mapping or translation of the previous word-processor's vocabulary has been made to the new word-processor's vocabulary.

In this early stages of our project, we have identified a generic vocabulary for the task of scheduling. In table 2-1, the vocabulary we have identified for scheduling is depicted. The words in *italics* represent verbs or actions which we also call active vocabulary. The rest are called passive vocabulary because usually actions are performed on them. The passive vocabulary is divided into two types: those with strong task dependance, represented in outline font, and those with weak task dependance, represented in normal font.

2.2 Generic Processes

Generic processes are another kind of mediating representation between the domain expert's cognitive primitives and knowledge engineering primitives. Their main difference with generic vocabulary is that generic vocabulary are mostly partial concepts with no significant meaning unless combined with others and generic processes are primitive concepts of activity described in terms of generic vocabulary. Generic vocabulary are passive in the sense that though some of them represent an action, they must be combined with other vocabulary to completely describe the action, e.g. what is the object of the action. As the terms mediating representation implies, generic processes are not "the" cognitive primitives of human thought and no attempt is made to categorize them as such. The generic processes presented here for scheduling are experimental processes that have been changing since first introduced in order to better and more generally represent the processes undertaken in scheduling. They stand some where in the spectrum between real domain expert's cognitive primitives and knowledge engineer's building blocks.

Table 2-1. Experimental generic vocabulary for scheduling.

analyze	*delete*	*relax*
apply	*generate*	requirement
assign	*identify*	resource
batch	job	*satisfy*
capacity	*merge*	*select*
categorize	*modify*	sequence
category	preference	schedule
conflict	priority	subschedule
constraint	process	tolerance
		update

Italics : action Normal: weak task-dependence

Outline: strong task dependencce

Generic processes are primitives defined in terms of generic vocabulary. In our word-processing example, a generic process could be paragraph-deletion for which a paragraph has to be marked before being deleted. The generic process to generic vocabulary relation can be written as: paragraph-deletion = delete marked paragraph. The most important relation between generic vocabulary and generic processes is that generic processes usually combine two or more generic words to describe a general action, not another word.

We have identified some generic processes for planning and scheduling, but intend to expand to other PS tasks such as monitoring, diagnosis and so forth. Although, perfect generic vocabulary and generic process libraries cannot be constructed, we believe that they provide enough guidance for analysis and synthesis of the tasks under consideration. What matters the most, is that the mechanisms, to which the generic processes were mapped, are able to successfully accomplish their final goal, that is, that they will help construct (synthesize) the desired KBS. In MULTIS, a mapping between generic processes and knowledge engineer's mechanisms is made in such a way that one or more generic processes may correspond to a single mechanism.

Figure 2-2 presents some generic processes identified for scheduling and simple planning problems. We have divided the generic processes in two levels: generic processes and generic sub-processes. The former are more specific to the task and the later more general, that is, the later are more general and can be used to explain the process of the former. In this figure, the left column depicts the generic processes and the right columns the generic sub-processes. The lines between columns indicate modularity or specialization. Appendix A presents a more detailed explanation for each of the processes shown in figure 2-2.

Generic processes can be interconnected in what we refer to as generic process networks. The connectability between generic processes is restricted by the kind of conceptual input or output, represented in the form of generic vocabulary, that they can receive or send respectively. Thanks to this connectability relation, inference strategies can be created to aid in the process of identifying what generic processes a particular domain expert employs when performing a specific task.

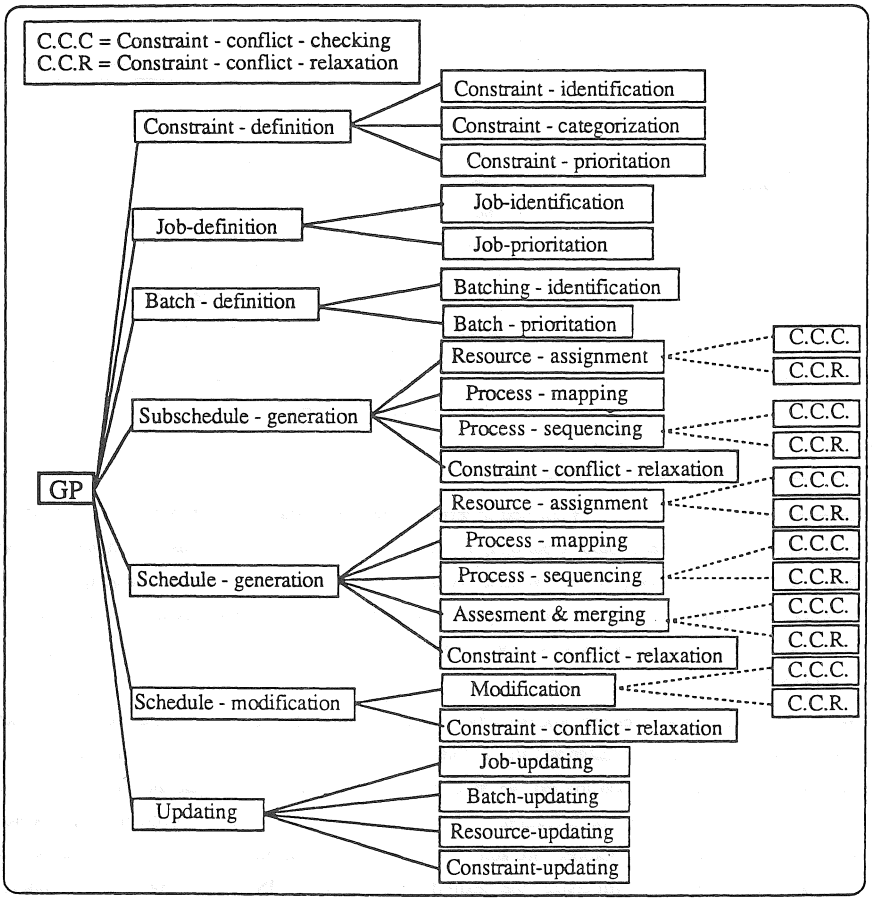

Figure 2-2. Experimental generic processes for scheduling.

2.3 Related Work

What follows is an enumeration of some related research efforts being undertaken in the area of cognitive primitives.

2.3.1 Chandrasekaran's Generic Tasks

Chandrasekaran (1986, 1987) proposes units of search control mechanisms that are very useful to construct inference engines for KBSs and to guide in the elicitation of domain knowledge. However, those units called "generic tasks" or "building blocks," though generalized for a specific type of PS tasks, are too specialized in the context of knowledge engineering. Therefore, they are very difficult to be understood by the domain expert himself.

Comparison: Our generic processes differ from Chandrasekaran's generic tasks in that our generic processes have to be combined in order to obtain knowledge-based system building blocks. In other

words, our generic processes are used to model human PS at the cognitive level and, if properly translated, can be easily understood by the domain expert. Chandrasekaran's generic tasks can be used as building blocks as they are.

2.3.2 Clancey's heuristic classification

Clancey (1985) proposes heuristic classification as a method that is useful to explain the mechanism of most KBS systems constructed in the area of analysis tasks. Again, this type of method is easily understood by knowledge engineers, but not by domain experts.

Comparison: In MULTIS we attack the problem of generalization of problem solving methods from quite a different perspective from that of heuristic classification. Heuristic classification, though very successful a method, is too general and doesn't provide enough insights for knowledge acquisition. It is difficult to believe that, because problems usually involve more than one type of task, all analysis problems can be best modeled with only heuristic classification. In MULTIS, the general assumption is made that more than just one problem solving method can be used to represent human problem solving processes in computer programs. However, if Heuristic classification is broken up in smaller component pieces, it would still be possible to provide generic processes that could be mapped to those component pieces, thus behaving as mediating representations.

2.3.3 McDermott's Half Weak Methods

Introduction of half-weak methods by professor McDermott (1988) contributed greatly to knowledge engineers in building KBSs. However, as Chandrasekaran's generic tasks, it is difficult for an average domain expert to understand his/her own PS process in terms of such half-weak methods.

Comparison: Again, these so-called half-weak methods are similar to MULTIS's building blocks, because they are helpful in modeling of inference engines for knowledge based system construction. Conceptually, half-weak methods are better employed to explain the way computers solve problems than the way humans do. However, as it is the case with heuristic classification, we believe that generic processes and generic vocabulary can be found to stand as mediating representations between the domain expert's cognitive primitives and McDermott's half-weak methods.

2.3.4 Weilinga's Knowledge Sources

Breuker and Weilinga (1989) propose to make models of human PS and call the units used in those models "knowledge sources." This is an interesting idea because those knowledge sources seem to be somewhere in between what we have called generic vocabulary and generic processes. Therefore, though KADS, the system that implements their proposal, is intended for knowledge engineers, its potential for usability by domain experts seems to be high.

Comparison: KADS's knowledge modeling framework uses generic processes and generic vocabularies (though not called that way) to model human problem solving. Although this is extremely similar to the MULTIS's modeling, KADS doesn't remember models nor it tries to adapt those models to other domain experts and was intended as a knowledge engineer's tool.

2.4 Discussion

As noticed in previous subsections, our idea of cognitive primitives depart somewhat from the context of knowledge engineering. However, we believe that cognitive primitives proposed by other

researchers are very useful as implementation tools or units for KBSs. The cognitive primitives proposed in this paper are intermediaries between human cognitive primitives and computer search control mechanism. Therefore, they provide a way to overcome the representation mismatch between domain expert and knowledge engineer, not human and machine.

2.5 An example

Let's suppose that expert A is an expert at performing a scheduling task in domain X. Let's further suppose that there is another expert B who also is an expert at performing a scheduling task, but in domain Y. Let's also assume that X is different from Y significantly. Lastly, let C be a knowledge engineer with the task of constructing two KBSs, both for modeling A's and B's problem solving knowledge in scheduling for domains X and Y respectively. Since A and B perform a common task in different domains, their task dependent primitives might also be common. Therefore, if those primitives are translated accordingly, A and B can understantd each other's problem solving processes. C, on the other hand, will have to study much deeper about each of A's and B's domains and their relation to scheduling before endeavoring to model KBSs for them respectively. In other words, C has to map his/her primitives (building blocks) to A and B and learn about X and Y in order to construct useful KBSs. Once KBSs for doing scheduling in X and Y have been modeled, their task-dependent mechanisms could be very similar, because they were modeled using C's primitives (building blocks). In this example, A and B are able to understand each other after some comparatively short discussion of X and Y. This is true because their task is the same, thus, they possess common primitives for performing it. Most of the time spent at discussing would probably be used in translating vocabulary related to X or Y. On the contrary, C has to do more effort because he has to learn new primitives (generic vocabulary and generic processes) and/or map them to his own primitives (building blocks) that are not from scheduling, but from computer programming. However, as time goes by and C constructs more scheduling KBSs, he will become more and more acquainted with such mappings and consequently more efficient at building the KBSs.

3. Building Blocks

Building blocks are transparent to the interviewee, but not to knowledge engineers who usually use them to design and construct a KBS. Though MULTIS builds a model of PS interactively with the interviewee using generic vocabulary and generic processes, a mapping of such model is made transparently to the building blocks needed to construct the inference engine of the knowledge-based system intended. This approach is consistent with the knowledge engineer's approach because they code the inference engine without showing it to the interviewee after building his/her PS model interactively.

At the end of the previous section, we compared our mediating representations with other research projects. It was concluded that the representations we introduced could still be used as mediating representations between domain expert's primitives and the primitives (building blocks) proposed by those projects. However, in this section we will introduce some experimental KBS building blocks used to prototype KBSs that perform scheduling tasks.

3.1 Summary of Mechanisms

As the name building block implies, they are mechanisms used to implement KBSs. In figure 3-1, we present some experimental building blocks. These building blocks are intended to be as general as possible in representing problem solving mechanisms identified by knowledge engineers through the years. Their combination will result in an implementable prototype for the inference engine of a KBS. Though mechanisms proposed by others (Chandrasekaran, 1986 and 1987; Clancey, 1985; McDermott, 1988) could be used as building blocks also, we have adopted this approach for simplicity reasons. Thus, we don't make any claim that the building blocks presented here cover all facets of knowledge engineering problem solving primitives. On the contrary, we propose that building blocks presented by other researchers be also used.

In figure 3-1, building blocks are divided into two types: primitive building blocks and composite building blocks. The former are more general than the later which are derived as combinations of the former. Again, the same argument introduced previously, that there exist general primitives and more specific primitives, is still valid because as can be seen, the primitive building blocks are general knowledge engineering concepts, while the combinatorial building blocks are more specific to given situations. We could also argue that there is almost no difference between the level of generic processes and this building blocks, since both endeavor to represent human cognitive primitives. However, there exist some relevant features that make distinction between them possible, that is, generic processes can be differentiated from building blocks on the following two accounts:

1) There exist no consensus in any domain or task on what generic processes could be used to successfully model problem solving for that domain or task. On the contrary, building blocks have been proposed so far that can be used to generally model knowledge engineer's problem solving.

2) Generic processes mainly model domain expert problem solving in a representation that is understandable to them, but cannot be used to explicitly implement a executable computer program. On the other hand, building blocks are representations understandable by the knowledge engineer and can be used to construct computer programs, i.e. KBSs, that implement their problem solving models.

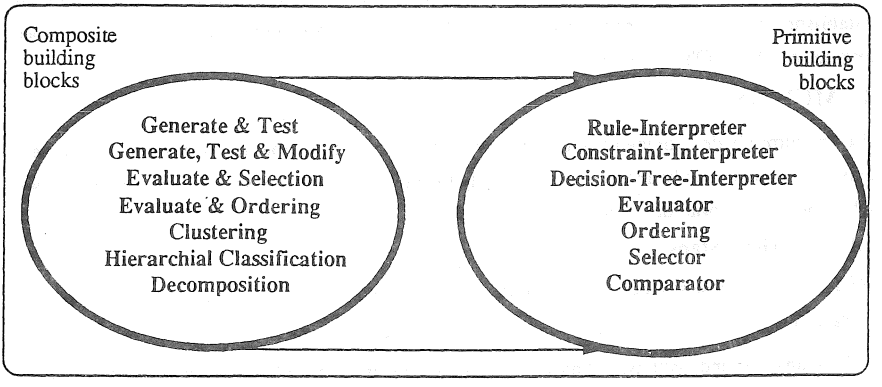

Figure 3-1. Experimental building blocks divided in composite and primitive ones.

Appendix B, gives a non-exhaustive enumeration of experimental, combinatorial building blocks being proposed in this paper.

3.2 Mapping of Generic Processes to Building Blocks

The mapping of generic processes to building blocks requires expertise of knowledge engineers in doing such mapping. Though not formally recognized previously, knowledge engineers have engaged in the task of doing such mappings when confronted with construction of KBSs. The knowledge engineer interviews domain experts, who can generally only express their problem solving in their own way, and then identifies what could be used as implementing blocks also in a rather personal manner. Consequently, a mapping is implicitly established between the domain expert's cognitive primitives and the knowledge engineers library of implementable mechanisms. The expertise used in creating this mappings can be extracted from knowledge engineers and used to perform mappings between networks of generic processes and building blocks proposed in this paper.

In appendix B, there is some guidance as to what kind of knowledge can be used for mapping generic processes to building blocks. For each generic process presented, there is an entry by the name of "mechanism identification inference," from which interview strategies could be generated to identify building blocks useful to implement the domain expert's problem solving model.

3.3 An Example

CABPRO (Schaefer et al., 1988) is a simple KBS developed to schedule processes when assembling cables. A cable could be very complex or very simple to make and very few cables could be considered standard, thus, the processes for making them vary in great degree. A schedule for making a cable is a very detailed set of instructions called a "traveler," because the person making the cable must travel from working station to station to complete the cable according to the instructions given in the traveler. Although a traveler is considered as a schedule, it must be taken into consideration that there are many travelers at any time in the factory, hence, a traveler is only a sub-schedule to the whole scheduling problem. For this reason, a traveler will be referred to as a sub-schedule.

Figure 3-2a shows what generic processes can be used to model CABPRO's task of scheduling. In figure3-2b, a translation to the CABPRO's vocabulary is given. Figure 3-2c illustrates the mapping established between the generic processes and the building blocks used to construct the inference engine of CABPRO's prototype.

4. MULTIS

We are currently developing an interview system based on the previous considerations that domain expert's primitves can be mapped automatically to knowledge engineer's primitives. What follows, what follows is a brief description of the system.

4.1 Interview Steps

MULTIS's interview strategy consists of the following four phases which are described below:

1. Task type identification. Identify the PS task the expert performs.
2. Task analysis and synthesis. Analyze the task and identify needed search control knowledge.
3. Knowledge acquisition. Elicit domain knowledge and task implementation knowledge.

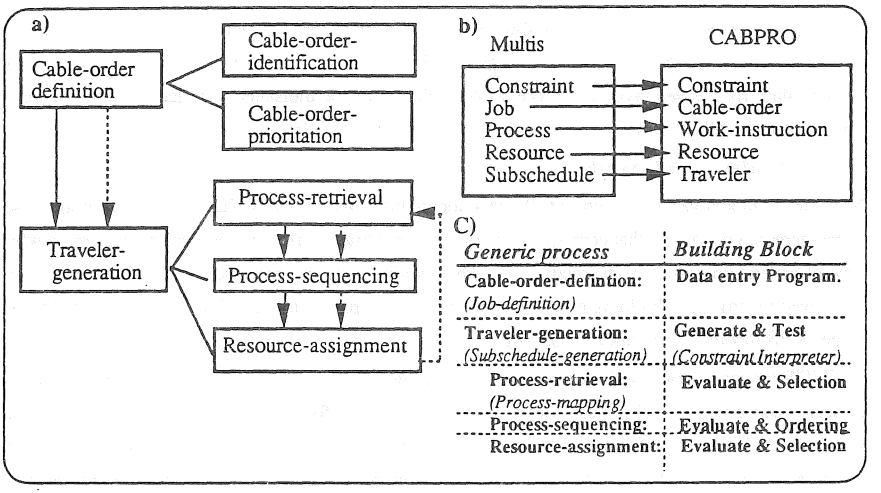

Figure 3-2. CABPRO's a) generic process network, b) generic vocabulary translation, and c) generic process to building block mapping.

4. System evaluation. Evaluate the system by imputing an already solved problem.

4.1.1 Task Type Identification

MULTIS first identifies the task the domain expert, from whom the knowledge is to be elicited, performs during PS. During this phase of the interview, the tasks for which MULTIS can elicit knowledge are currently simple planning and scheduling, but can be expanded later to cover other tasks such as diagnosis, monitoring and so on. Since a taxonomy of tasks already exists for the type of tasks being performed by humans , this phase can be implemented as a classification problem. Therefore, the interview is based on heuristics about the taxonomy of tasks. Roget (Bennett, 1985) is an earlier interview system that attempts to identify the type of task the interviewee performs before engaging in knowledge acquisition. However, it limits the type of task from which it can elicit knowledge to diagnosis by using sentence-parsing techniques that could prove somewhat inefficient if other tasks were to be added. MULTIS performs interactive interview with a user friendly interface to avoid falling into a similar problem.

Since an expert sometimes performs more than a single task in his/her field of expertise, it is necessary to identify what is the main task being performed. Once that task is identified, the rest of the tasks being performed can be ignored, because later during the interview their corresponding generic processes will be incrementally added to the model of the main task. This is a problem that hasn't been treated properly in the past because most interview systems have assumed that human experts perform only a single task. However, human PS is usually more complex, that is, it consists of a mixture of analysis and synthesis at different levels. For instance, a medical doctor mainly performs diagnosis (analysis), but he also designs treatments (synthesis) to fit special cases of patients. With this in mind, MULTIS identifies the main task the expert performs and then incrementally identifies what other tasks are being performed.

MULTIS then extracts the generic vocabulary corresponding to the task identified and translates it to the interviewee's vocabulary. As the term generic implies, a generic vocabulary is generalization of a vocabulary for a given task. Therefore, a vocabulary must be translated to reflect the interviewee's vocabulary. In MULTIS the vocabulary is translated interactively by the user himself.

4.1.2 Analysis and Synthesis

Next step is to analyze and synthesize the task identified in the previous phase. MULTIS makes use of the generic vocabulary that corresponds to the task identified in the previous step to determine what type of generic processes can be related to it. Then, with the generic processes identified heuristics are employed to single out building blocks that can be used as part of the search-control structure for the target KBS through case-based reasoning.

MULTIS makes use of case-based reasoning during its interview to reuse previous combinations of generic tasks encountered in past interviews to make search for generic tasks more efficient. The vocabulary corresponding to the expert's task and its translation are employed as indexes to search in the case base for a similar case or cases. Later, the generic task combination from the extracted case is modified with use of a graphical interface (under development) to accurately represent the task being performed by the domain expert. Finally, the generated generic task combination is analyzed heuristically to identify the right mix of building blocks that should be used as part of the search-control structure of the target KBS. The details of case-based reasoning are given in another paper (Tijerino, 1990).

PS modeling and inference engine mapping take place in this phase. Interactive modeling is implemented through the graphical interface mentioned after a prestored modifiable model has been found. The inference engine mapping is performed transparently after employing heuristics to identify building blocks useful to reflect the patterns given by the model.

4.1.3. Acquisition of Domain Knowledge

In this step, the required additional task-implementation knowledge and domain knowledge is elicited from the domain expert. For this purpose, the assumption is made that each generic process and each building block, related to the task involved, have structures that guide the processes of elicitation of domain knowledge, and task-implementation knowledge respectively. Chandrasekaran (1987) and McDermott (1988) have made similar assumptions. Put in other terms, a strategy for knowledge acquisition can be directly mapped to each generic task or each building block.

As explained before, the main characteristic of MULTIS is its ability to help acquire knowledge for various tasks. Therefore, we have concentrated mostly in the design of inference engines for those tasks through building block combination. However, acquisition of domain knowledge is also a very important issue that cannot be ignored. Consequently, we are carrying out intensive research on the mapping of existing knowledge acquisition methods to the search-control structures resulting from MULTIS's interview.

4.1.4 System Evaluation

The last phase in the interview strategy, is system evaluation. At this point, MULTIS tries to solve a problem already solved by the expert being interviewed. If the expert is satisfied with the solution then

no more questions are asked and the target KBS is constructed. Otherwise, MULTIS checks for missing or erroneous domain knowledge, or erroneous search control knowledge.

4.2 Architecture

MULTIS architecture is depicted in figure 4-1. It consists of the following modules and submodules:

1) Task Type Identification Module, which interactively identifies what is the main task type that the domain expert perform during problem solving. The purpose of this module is to help MULTIS focus on a task type to avoid unnecessary search for PS primitives from other task types

2) Generic Vocabulary Translator, which also interactively helps the domain expert translate his/her vocabulary to make the rest of the interview flow coherent with the interviewee.

3) Case-based Reasoning Module. This module uses the translation of generic vocabulary and patterns found in it to search for previous case which present some similarity with the current. Next, it interactively modifies the generic process network of the similar case with help of the Generic Process Simulator to reflect the PS structure of the interviewee. Then, it maps the generic process network obtained to implementable building blocks to yield the inference engine of the target KBS.

3.1) Generic Process Simulator. This is an user-friendly module that takes the interviewee through tutoring sessions on generic processes, shows examples of similar cases' generic process networks and simulates the flow of those networks during PS. It also provides the interviewee with powerful means to modify those networks in such a way that he/she can represent his/her own PS structure in a new generic process network.

3.2) Generic-Process-to-Building-Block Mapping Module. This module uses heuristics obtained from knowledge engineers to map generic process networks to building blocks yielding an executable inference engine.

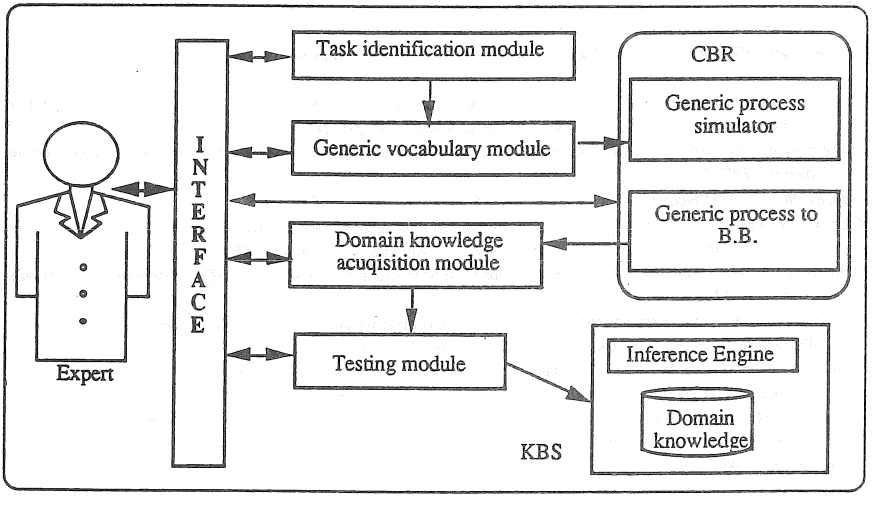

Figure 4-1. Multis Architecture.

4) Domain Knowledge Acquisition Module. This module employs interview strategies attached to building blocks and generic processes present in the current case.

5) Testing Module. This module attempts to solve an already solved case as explained in 4.1.4.

Details of each sub-module will be presented in future papers.

5. Conclusion

MULTIS is still under implementation. Once fully constructed it should be able to:

1. Engage in coherent and smooth interview with domain experts with help of generic vocabulary.
2. Model the PS processes of domain experts in various fields in the form of generic process networks.
3. Identify which prefabricated search control structures can be implemented to construct a KBS that performs the desired problem-solving.
4. Make use of strategies related to the generic processes and building blocks to elicit domain and task-implementation knowledge.
5. Help construct or quickly prototype KBSs for various tasks.
6. Make use of CBR to improve efficiency during the interview.

Currently, MULTIS possesses a case base for process scheduling with seven cases. We plan to add other cases that cover other tasks such as planning, monitoring and diagnosis. Also, mapping between generic tasks and building blocks, a very important issue, is being investigated along with the corresponding mapping to domain knowledge acquisition methods.

References

Bareiss, R. (1989). *Exemplar-Based Knowledge Acquisition.* Chandrasekaran, B. (ed) Academic Press.

Bennett J. S. (1985) ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System, *J. of Automated Reasoning* I, pp. 49-74.

Breuker, J and Wielinga, B. (1989) Models of expertise in knowledge acquisition. *In topics in Expert System Design*, G Guida and C Tasso, eds. 265-295.

Bylander, T and Chandrasekaran B. (1987) Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition. *Int. J. Man-Machine Studies.* 26, 231-243.

Chandrasekaran, B. (1986) Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert* 1, 23-30.

Clancey, W. J. (1985) Heuristic Classification. *Artificial Intelligence* 27, 289-350.

Kahn, G., Nowlan, S. and McDermott, J. (1985) MORE; an intelligent knowledge acquisition tool. *Proceedings of the 9th Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, 581-584.

McDermott J. (1988) Using problem-solving methods to impose structure on knowledge. *IEEE International Workshop on Artificial Intelligence for Industrial Applications.* 7-11.

McDermott, J. et al. (1990) Explorations in how to make application programming easier. *In Porceeding of 1st Japanese Knowledge Acquisition fo Knowledge-Based Systems Workshop.* 134-147.

Numao, M. and Morishita S. (1988) Scheplan—A scheduling expert for steel-making process. *In Proceedings of IEEE International Workshp on Artificial Intelligence for Industrial Applications*, 457-472.

Schaefer, R. M., Colmer, J. S. and Miley, M. (1988) CABPRO: A rule-based expert system for process planning of assembled multi-wire cables. *IEEE* CH2552-8/88 (181-186).

Rappaport, A. (1988a) Cognitive Primitives. *International Journal of Man-Machine Studies*, 29, 733-747.

Rappaport, A. (1988b) Integration of acquisition and performance systems. *Proceedings of the third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.* Banff Canada.

Tijerino, Y. A., Kitahashi, T. and Mizoguchi, R. (1990) A task-analysis interview system that uses a problem solving model. *In Porceeding of 1st Japanese Knowledge Acquisition fo Knowledge-Based Systems Workshop.* 331-344.

Appendix A: Generic Processes for Scheduling

What follows is an enumeration of the generic processes and sub-generic processes presented in figure 2-2. The order is given in the same manner as they appear in figure 2-2. Each generic process will be described according to the following description criteria:

1) a definition in terms of generic vocabulary,

2) input and output taken,

3) a general explanation of the concept they describe,

(0) *Constraint-definition*

Definition: Identify and prioritize constraints neede for scheduling.

Input: Constraints from the world.

Output: Constraints.

Explanation: With this generic task the individual performing scheduling defines what are the constraints needed in performing problem solving for scheduling.

(0.1) *Constraint-identification*

Definition: Identify constraints needed for scheduling.

Input: Constraints from the world.

Output: Constraints.

Explanation: Identify the relevant constraints that will be considered during scheduling.

(0.2) *Constraint-prioritation*

Definition: Prioritize constraint.

Input: Constraints.

Output: Contraints and priorities.

Explanation: Give an order of importance to each constraint.

(1) *Job-definition*

Definition: identify and/or assign priorities to jobs to be scheduled.

Input: Varies according to what is considered a job, but usually it is the orders taken from the customers.

Output: Jobs, Job priorities

Explanation: Job definition is a process that is performed mostly before the actual problem solving task of scheduling takes place. Although this is a beginning process, it doesn't have to be performed necessarily by the expert himself. This process deals with defining what is to be considered a job, what are the jobs to be scheduled and in some cases what are the priorities to be assigned to those jobs.

(1.1) *Job-identification*

Definition: Identify jobs to be scheduled.

Input: same as (1).

Output: Jobs.

Explanation: Job definition is a process that is performed mostly before the actual problem solving task of scheduling takes place. Although this a beginning process, it doesn't have to be performed

necessarily by the expert himself. This process deals with defining what is to be considered a job and what are the jobs to be scheduled.

(1.2) *Job-prioritation*

Definition: Assign priorities to jobs.

Input: jobs

Output: jobs, job priorities

Explanation: Prioritation of jobs can be as simple as determining the priority by a single attribute of the job or as complex as considering multiple attributes as well as other parameters external to the jobs. In essence, this process deals with assigning priorities to jobs before trying to do any scheduling.

(2) *Batch-definition*

Definition: Generate batches, with or without priorities, from jobs.

Input: Jobs, job priorities.

Output: Batches, batch priorities.

Explanation: Jobs may sometimes have similar processes that, if performed simultaneously, permit more efficient and inexpensive factory operation. In this cases, the grouping of jobs into batches is indispensable for scheduling. This process deals with generating groups or sets of jobs that can be treated as units in scheduling. A job priority is one of the many attributes that a batch might posses. However, it is a very important one when it comes to scheduling because jobs with higher priority give a starting point for scheduling. If all jobs are prioritized then there is an implicit partial order from where a schedule can be generated.

(2.1) *Batch-identification*

Definition: Generate batches from jobs.

Input: Jobs, job priorities.

Output: Batches.

Explanation: A batch is a group of jobs with one or more attributes in common. This process uses some grouping criteria to decide what jobs should be grouped. Such criteria might be the combination of various job attributes or a single one.

(2.2) *Batch-prioritation*

Definition: assign priorities to each batch.

Input: Batches.

Output: Batches, batch priorities.

Explanation: A batch priority is one of the many attributes that a batch might posses. However, it is a very important one when it comes to scheduling because batches with higher priority give a starting point for scheduling. If all batches are prioritized then there is an implicit partial order from where a schedule can be generated.

(3) *Subschedule-generation*

Definition: Generate subschedules that satisfy batch or job requirements and/or constraints.

Input: Jobs, Job priorities, Batches, Batch priorities.

Output: Subschedules.

Explanation: This process deals with generating partial, modular subschedules that satisfy local constraints to jobs or batches. Each subschedule should give a series of steps to perform the subschedule locally and in some cases identify what resources have to be used. For some scheduling problems scheduling finishes with this process, that is, by implementing a number of modular schedules problem solving is accomplished.

(3.1) *Process-mapping*

Definition: Select or generate and assign a process to a job or batch.

Input: Job, Batch, processes

Output: Processes assigned to jobs or batches.

Explanation: Process-mapping can be a very simple process, such as assigning a single process to a job or batch, or a very complex one, such as assigning numerous processes in an intricate manner. Processes can also be very simple, being composed of a small set of procedures, or very complex, a large set of procedures. For this reason process-mapping might depend on data-bases where the information about processes is prestored reflecting previous experiences gained during problem solving.

(3.2) *Process-sequencing*

Definition: Generate a sequence of processes for a job or batch.

Input: Processes, Jobs, Batches.

Output: Sequence (partial subschedule).

Explanation: Sequencing is where most of the actual scheduling takes place, because a set of ordered instructions is given in the form of a sequence. Process-sequencing deals with determining an order in which processes must be performed, where processes themselves are predetermined sets of instructions or procedures.

(3.3) *Resource-assignment*

Definition: Select and assign resources to processes, jobs, batches in sequences.

Input: Processes, Jobs, Batches, Sequences.

Output: Processes, jobs, batches in sequences with resources assigned (tentative subschedule or schedule).

Explanation: The main difference between scheduling and planning lies in that in planning the only resource considered is that of time. In scheduling the successful assignment of resources determines the overall success of the task. Resource assignment deals with choosing what resources and in what amount can be assign to a particular job, batch, processes in a sequence.

(3.4) *Constraint-conflict-checking*

Definition: Identify conflict between constraints

Input: constraints

Output: constraint conflict

Explanation: This process deals with identifying what can be treated as a conflict between the various constraints taken into consideration in problem solving, e.g. job, batch, process, resources or other global constraints. This is a process that involves some effort on the part of the problem solver to find which constraints are in conflict with others.

(3.5) *Constraint-conflict-relaxation*

Definition: Relax conflicts between constraints in within bounds assigned.

Input: Constraints in conflict, priorities.

Output: Constraints.

Explanation: When constraints are found to be in conflict with each other the problem solver must decide whether those constraints are flexible, what priorities they have and so on. Then, he/she must decide how to resolve the conflict, e.g. by satisfying only the ones with higher priority, changing the requirements in within the bounds of the constraints, etc. Constraint-conflict-relaxation is the process that is involved in making those changes so as to resolve those conflicts.

(4) *Schedule-generation*

Definition: Generate an overall schedule from jobs or batches, or merge subschedules into a single schedule.

Input: Jobs, Batches, Subschedules.

Output: Schedule.

Explanation: Schedule generation might seem too general a name for this process, however, it is intended to mean the process undertaken when the final schedule is generated from either merging subschedules or constructing one from jobs or batches. Hence this can be a process that yields the final or tentative schedule from subschedules or from either scheduling jobs or batches.

(4.1) *Assessment-and-merging*

Definition: Generate a schedule by merging subschedules.

Input: Subschedules.

Output: Schedule.

Explanation: This process deals with merging subschedules into an overall schedule of the problem being solved (the scheduled being generated). Assessment of how subschedules can be merged forms great part of this process. For most cases, modification of subschedules is performed so that they can be merged without conflicting with each other.

(4.2) *Resource-assignment*

Same as (3.3).

4.3) *Constraint-conflict-checking*

Same as (3.4)

(4.4) *Constraint-conflict-relaxation*

Same as (3.5)

(5) *Schedule-modification*

Definition: Modify a schedule into one or more schedules.

Input: Schedule, constraints, others.

Output: Schedule

Explanation: This process can be as complex as generating parallel schedules during problem solving or as simple as modifying the schedules in a small way at the end of problem solving.

(5.1) *Modification*

Definition: Modify schedule

Input: Schedule.

Output: Schedules.

Explanation: This is a more general process of modification because it serves as the modification engine itself of (5).

(6) *Updating*

Definition: Delete, modify or add jobs, batches, resources, constraints.

Input: Jobs, batches, resources, constraints.

Output: Jobs, batches, resources, constraints.

Explanation: This process deals with deleting , modifying or adding jobs, batches, resources or constraints, during or after problem solving.

(6.1) *Job-updating*

Definition: Delete, modify or add jobs.

Input: Jobs.

Output: Jobs.

Explanation: Same as (6), but only for jobs.

(6.2) *Batch-updating*

Definition: Delete, modify or add batches.

Input: Batches.

Output: Batches.

Explanation: Same as (6), but only for jobs.

(6.3) *Resource-updating*

Definition: Delete, modify or add resources.

Input: Resources.

Output: Resources.

Explanation: Same as (6), but only for resources.

(6.4) *Constraint-updating*

Definition: Delete, modify or add constraints.

Input: Constraints.

Output: Constraints.

Explanation: Same as (6), but only for constraints.

Appendix B: Experimental Building Blocks

The following is an explanation of the building blocks presented in figure 3-1.

Generate & Test.

Mechanism: Generate a schedule according to some given criteria and test the schedule for appropriateness. If the schedule is appropriate, then give it as an output to the problem. If the schedule

is not appropriate, then start from the generation process all over again. There are at least three ways of implementation of generators: Selection from a finite set, computation of formulas and production rule interpretation.

Possible primitive building blocks: Rule-Interpreter or Constraint-Interpreter.

Generate, Test & Modify.

Mechanism: Generate a schedule according to some given criteria and test the schedule for appropriateness. If the schedule is appropriate, then give it as an output to the problem. This time if the schedule is not appropriate, then modify it until it satisfies some given conditions or until it can not be modified more, in which case a new schedule is generated and the process is started all over again.

Possible primitive building blocks: Rule-Interpreter or Constraint Interpreter.

Evaluate & selection.

Mechanism: Evaluate a set of known elements according to some given criteria and select the appropriate schedule that covers prestablished conditions.

Possible primitive building blocks: Evaluator, which can be designed with other primitive building blocks such as Rule-Interpreter or Constraint-Interpreter, and Selector which can also be constructed from others.

Evaluate & Ordering.

Mechanism: Evaluate a set of elements according to some given criteria and return a set with the same elements in an order that satisfies a set of prestablished conditions.

Possible primitive building blocks: Evaluator, which can be designed with other primitive building blocks such as Rule-Interpreter or Constraint-Interpreter, and Orderer which can also be constructed from others.

Hierarchical Classification.

Mechanism: Identify to what leaf of a decision-tree an element with some given features can be classified as belonging to.

Possible primitive building blocks: Decision-Tree-Interpreter.

Decomposition.

Mechanism: Decompose a problem into smaller problems until the subproblems become such that they can be used as subschedules. Then, select the most appropriate subschedules that can be employed as the overall solution to the starting problem.

Possible primitive building blocks: Rule-Interpreter

Clustering.

Mechanism: Make groupings of elements that belong to the same category by comparing their features according to some given criteria.

Possible primitive building blocks: Comparator which can be constructed from such building blocks as Rule-Interpreter and Constraint-Interpreter.