MULTIS II: Enabling End-Users to Design Problem-Solving Engines Via Two-Level Task Ontologies

Yuri Adrian TIJERINO* and Riichiro MIZOGUCHI**

*Advanced Telecommunications Research Institute International (ATR)
**The Institute of Scientific and Industrial Research, Osaka University

Abstract: The aim of this research effort is to develop a method for automating knowledge acquisition, that can be employed to generate knowledge-based systems, making few initial constraints on the type of problem solving mechanism. This method is based on the following principles: (1) problem solving can be described with primitives that can be reused and rearranged to model problem solving processes in various domains and task types, and (2) there exist enough formal problem solving mechanisms in algorithmic form to implement those models in a machine understandable code, or at least those mechanisms can be easily developed by computer experts.

1 Introduction

The aim of this research effort is to develop a method for automating knowledge acquisition, that can be employed to generate knowledge-based systems, making few initial constraints on the type of problem solving mechanism. This method is based on the following principles: (1) problem solving can be described with primitives that can be reused and rearranged to model problem solving processes in various domains and task types, and (2) there exist enough formal problem solving mechanisms in algorithmic form to implement those models in a machine understandable code, or at least those mechanisms can be easily developed by computer experts. With these two ideas in mind the authors embarked in the process of identifying, first the problem solving primitives for scheduling tasks, and second in developing mappings between those primitives and mechanisms implemented as computer programs. To prove the tractability of those ideas we have developed MULTIS II[1], a system that interacts with domain experts of various fields, makes a model of their problem solving task, and generates a customized knowledge-based system (KBS) that performs the given task. This paper presents this system and demonstrates a typical interview session.

Traditionally, KBS's are composed of, at least, an inference engine, which corresponds to a general search-control mechanism and a domain knowledge base, which includes factual and/or expertise knowledge. This paper claims that knowledge acquisition bottleneck is caused by too much generality of such inference engines. Therefore, this paper proposes that inference engines should be constructed with task specificity in mind. In order to achieve task specificity, this paper introduces the idea of two-level task ontologies[2]; the knowledge level and the symbol level (Newell, 1982). The task specificity and the layered format of the ontologies will not only ease the process of domain knowledge acquisition, but could also make it possible for non-knowledge engineers to design KBS's with relatively little training. This can be achieved if the knowledge level ontology is mapped dynamically to the symbol level one, based on user interpretation of the primitives in the ontology. Therefore, it becomes unnecessary for the user to understand the symbol level ontology unless the user specifically wants to do so. Recently, this notion of ontology introduced by (Neches et al., 1991) has become very popular, but the contribution of this

1 MULTIS II is a system that evolved from MULTIS, which will be called MULTIS I henceforth. The main difference between the systems lays in the design and untested ontologies of the earlier system. MULTIS I was presented in earlier papers (Tijerino et al., 1990; Tijerino et al., 1991 and Tijerino et al. 1992).

2 The word ontology means the study of existence in philosophy, but in artificial intelligence it usually means the set of most primitive terms or concepts that describe something, e.g., a task. Though the term is widely accepted in the AI community, lexicon is probably a more appropriate word to describe the idea.

paper is based on task dependency and layered ontologies. This paper presents a two-level ontology for scheduling tasks.

1.1 Problem Definition

Experts solve problems at a high conceptual level while the computer needs much detail, thus making it necessary for the expert to adjust the way they think when using computer aids. In this work, the authors describe research that addresses these issues in the following manner: 1) general concepts and functional code are defined for tasks in the form of task-dependent ontologies, 2) methods for task analysis based on those ontologies are studied, and 3) knowledge acquisition is redefined as the process of mapping models constructed with knowledge-level ontologies to functional code at symbol-level ontologies.

The major objectives of this work are 1) to design a general methodology for knowledge acquisition for KBS construction, 2) to identify general and reusable task components for problem solving in the form of task ontologies, 3) to outline general guidelines for task analysis interview based on two-level task ontologies, 4) to synthesize KBS's from components, and 5) to identify what problems arise from the methodology and how to solve them.

2 Task Ontologies

A KBS is a kind of high-level problem-solving system that employs problem-solving methods that uniquely solve the problem being addressed. So, its so-called *inference engine* should effectively reflect the problem's class structure for which it is intended and the tools provided should embed corresponding functionality. Yet, most KBS's now widely depend on production systems that, while being highly general at representing knowledge, leave much to desire at the conceptual level when it comes to portray the peculiarities of diagnosis, design and other types of problems. This representation paradox introduced by production systems happens to be another primary factor that makes knowledge acquisition a difficult task. Recently, researchers have focused efforts on developing representations that render the essence of human expert thought processes without explicit dependence on the problem's domain but with careful analysis of the task side of the problem by describing the problem solving process at an appropriate abstract level. This new task representation has been called *generic tasks* by Chandrasekaran (1986). McDermott introduced a similar idea called *half-weak methods* (1988) or more recently called *problem-solving mechanisms* (Klinker et al., 1991), which is alternatively called components of expertise by Steels (1990). The ESPRIT project, on the other hand, has also introduced KADS (1989) and more recently KADS II (1992) as a framework for modeling problem solving based on so-called *knowledge sources*.

Based on the above discussion, it becomes obvious that the idea of inference engines is no longer adequate. Problem solving form differs for every generic task, therefore, a monolithic production system cannot possibly depict all their characteristics. For this reason, we call a *problem solving engine* any engine composed of components that capture specific aspects of control knowledge for a specific task. A knowledge-level task ontology is defined as the set of generalized user interpretations of such components. And finally, a symbol-level ontology is defined as the set of reusable components useful for synthesizing problem solving engines for a given task.

The fundamental principle behind the author's KBS construction methodology is synthesis of KBS components from symbol-level task ontologies. More precisely, problem solving engine components are combined by selecting the adequate ones from the symbol-level ontology defined for a task to satisfy problem solving method specifications and later obtaining needed domain knowledge by engaging in interview. Steels (1990) has also been working on a similar assumption.

This paper describes the knowledge-level task ontology in terms of generic vocabulary, generic processes and generic-process networks. On the other hand, the symbol-level task ontology consists of building blocks[3] and problem-solving engines. The generic vocabulary represents the concepts described above, which are called generic nouns, and the actions performed on them, which are called generic verbs. Generic processes are a combination of generic nouns with generic verbs and can be thought as primitive activities recognized by the domain experts. Finally, the building blocks correspond to the programming code mentioned above and combine to form problem-solving engines

3 A Scheduling Ontology

3.1 Generic Vocabulary for the Scheduling Tasks

To make the discussion concrete, let us take scheduling KBS's as an object of description using a task ontology. Though, most practical scheduling problems involve multiple resources being assigned to each other, we discovered that scheduling experts usually think of the assignments being done on a two-dimensional basis. That is, one resource is assigned at a time to another, given that a set of constraints are satisfied by the assignment, then other resources are taken into consideration. Following this line of reasoning, there is always an active resource, which we call the schedule resource for simplicity, and a passive one, which we call the resource recipient (see figure 1.) These concepts of time, assignment, schedule resource and schedule recipient play a crucial role in our schedule task ontology. Tables 1 and 2 show generic vocabulary identified thus far that consists of generic verbs, generic nouns, generic adjectives, constraint-vocabulary, and goals. Generic vocabulary acts as a mediating representation that fills the understanding gap between the domain experts and the computer. Domain experts easily understand those terms shown in the tables. For scheduling, the experts have the concept of what they are scheduling (i.e., schedule-recipient), that of what to assign to them (i.e., schedule-resource), that of conditions under which scheduling takes place (i.e., constraints), that of how much the constraints can be relaxed (i.e., tolerance) and so on. These generic nouns thus indicate important concepts in scheduling tasks. In a similar manner, generic verbs represent fundamental and general activities in scheduling tasks. Mizoguchi, Tijerino and Ikeda (1992) described these ontologies in more detail.

3.2 Generic Process and Generic Process Network

Generic processes are primitives defined in terms of generic vocabulary. 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. Generic processes are represented in terms of generic vocabulary as follows:

Generic process = Generic verb + Generic noun.

Typical examples includes *pickup-schedule-recipient*, *schedule-resource*, *assign-schedule-resource-time-to-schedule-recipient*, *update-priority*, *relax-constraint*, and so on. A domain expert's problem solving process is described in terms of generic processes and the result is configured as a kind of inference network composed of generic processes which is called GPN: Generic Process Network. A GPN can be thought of as task flow defined in terms of general, reusable components that describe meaningful stages of the problem solving process.

A GPN does not represent data flow explicitly, though it manages it implicitly and presents it to the user during the interviewing process to verify the correctness of the GPN built. A GPN is implemented in a two-dimensional language that is programmed by connecting icons of generic processes to each other. The internal representation of GPN is used by the GPNC (Generic-Process-Network Compiler) to generate code of the problem solving engine corresponding to the GPN. A detailed example will be presented in a later section.

3.3 Building Blocks and Problem-Solving Engines

Building blocks are symbol-level primitives readily used as components of problem solving engines of KBS's and are realized as abstract algorithms obtained by analyzing generic processes with respect to the input/output relations, data structure, and knowledge sources required. By knowledge source, we mean constraints or conditions necessary in a building block for its execution. Knowledge source is the domain-dependent knowledge detached from the task knowledge. Some typical examples of building blocks are presented in table 3. Generally speaking, every generic verb is indexed to more than one building block. As table 3 shows, *select* has at least eight building blocks associated with it accord-

3 We borrowed the term *building block* from Chandrasekaran (1988) and use it to refer to the algorithms that implement our generic processes. Though building blocks are at the symbol-level translation much be established to a generic process in the users terminology.

Schedule-Resource \ Time	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Truck 1	Driver3	Driver2	Driver5	Driver7	Driver5	Driver4
Truck 2	Driver2	Driver8	Driver9	Driver6	Driver3	Driver10
Truck 3	Driver10	Driver4	Driver 1	Driver8	Driver9	Driver2
Truck 4	Driver7	Driver6	Driver8	Driver4	Driver7	Driver9
Truck 5	Driver5	Driver10	Driver3	Driver10	Driver 1	Driver8
Truck 6	Driver9	Driver 1	Driver6	Driver5	Driver4	Driver3
...			**Schedule-Recipient**			
Truck n-1	Driver6	Driver3	Driver10	Driver2	Driver8	Driver 1
Truck n	Driver 1	Driver7	Driver4	Driver9	Driver2	Driver5

Figure 1. A Gantt chart for scheduling of Trucks to drivers representing the concepts of time axis, schedule resource and schedule recipient.

Table 1. Generic Vocabulary for Scheduling.

- **Generic verb: (See table 2)** (26)
- **Generic nouns** (40)
  - *Schedule Recipient: RCP*
    - RCP-GRP
  - *Schedule Resource: RSC*
    - RSC-GRP
  - *Schedule*
    - Schedule, Subschedule, Intermediate solution, Final solution, etc.
  - *Schedule representation*
    - Gannett chart, Time table, etc.
  - *Constraint, Goal, Priority, Data/Information*
- **Generic adjective** (11)
  - Unassigned, Previous, Last, Next, Satisfying, Violating, etc.
- **Constraint-vocabulary** (67)
  - *Constraint/Condition*
    - Strong constraint, Weak constraint, etc.
  - *Constraint adjective*
    - Maximal, Minimal, Earliest, Latest, Longest, Shortest, Largest, Smallest, etc.
  - *Constraint-predicate*
    - Equal to, Larger than, Smaller than, Include, Exclude, Overlap, etc.
  - *Attribute* (Component of constraint)
    - Time interval
      - Time available, Assigned time, etc.
    - Time point
      - Due date, Starting time, Ending time, etc.
    - Frequency, Efficiency, Priority, Load, Cost, Tolerance, Amount, etc.
- **Goal** (21)
  - *Status*
    - Maximum, Minimum, Uniform, Continuous
  - *Object*
    - Load balance, Rate of operation, Efficiency, Idle time, Operation time, etc.

(N) is the number of vocabulary identified so far for category

Table 2. Generic Verbs for Scheduling

RSC/RCP verb			
	Generate: Generates objects to process		
		Assign	: assign RSC and time to RCP
		Classify	: classify objects into groups
		Combine	: make tuples of objects
		Compute	: obtain value of object
		Divide	: divide objects into group
		Insert	: insert an object into a list
		Merge	: merge some objects
		Permute	: generate a permutation
		Pickup	: take an objects from list
		Remove	: remove objects from list
		Select	: take objects satisfying a condition from list
		Sequence	: arrange objects in order
	Test: Test if an object satisfies a condition		
		Check	:check object if it satisfies condition
		Evaluate:	:evaluate object to obtain value
	Modify: Modify an object		
		Reassign	
		Exchange	: exchange assignments
		Shift left/right	: shift assignment to left /right
		Update	: update data
Constraint_verb			
	Add	: add constraint	
	Change	: change constraint	
		Increase	: increase the threshold
		Decrease	: decrease the threshold
	Neglect	: neglect constraint	
	Relax	: relax constraint	
	Satisfy	: satisfy the constraint	
	Strengthen	: strengthen constraint	
	Violate	: violate the constraint	

ing to the characteristics of knowledge source. *SelectMaxAll*, for example, makes a list of all the objects of maximum value obtained by the execution of a criterion function and *Sequence* has two variants depending on whether the given ordering information is total ordering or partial ordering. Alternatively, a rule interpreter can be a building block for every generic verb, since it may require complex heuristic search for performing the task.

3.4 Task Ontology and Knowledge Acquisition

One of the major difficulties in knowledge acquisition comes from the mixture of domain knowledge and control knowledge in the expertise being acquired. It is not easy for domain experts to articulate separately these two kinds of knowledge. In this work on task ontology, we are aiming at realization of clear discrimination between the two in order to make it easier to acquire knowledge from domain experts.

Table 3. Some examples of building blocks. (Total 35)

Assign	: make a list of RSC, time and RCP; (RSC, time, RCP)	
	input:	three objects, Obj1, Obj2 and Obj3
	output:	(Obj1 Obj2 Obj3)
Select	:select object satisfying constraint	
	input:	list
	output:	atom or list
	knowledge source:	constraint, criterion function, set of rules for evaluation
	SelectMaxAll	make a list of all the objects of maximum value obtained by the criterion function
	SelectMinAll	make a list of all the objects of minimum value obtained by the criterion function
	SelectMaxOne	select an object of maximum value obtained by the criterion function
	SelectMinOne	select an object of minimum value obtained by the criterion function
	SelectMaxpAll	make a list of all the objects of maximum partial order values obtained by the rules
	SelectMaxpOne	select an object of maximum partial order value obtained by the rules
	SelectSatisfyAll	make a list of all the objects satisfying the condition
	SelectSatisfyOne	select an object satisfying the condition
Pickup	: take the first object in a list	
	input:	list
	output:	the first object in the list
	control:	Form a loop terminating when the list is empty
Classify	:classify objects into some groups	
	input:	objects
	output:	list of list of objects
	knowledge source:	Decision tree, set of rules, distance or similarity
	ClassifyDT	classify objects using decision tree
	ClassifyAttr	classify objects of the same attribute values
	ClassifyDist	classify objects based on distances among them
	ClassifyRule	classify objects by interpreting rules
Evaluate	: obtain value of object by evaluateing it	
	input:	list of objects
	output:	list of pairs of object and its value
	knowledge source:	criterion function, condition, set of rules
Sequence	: arrange objects in order	
	input:	list of objects with ordering information
	output:	list of objects
	SequenceT	: arrange objects according to values of total ordering
	SequenceP	: arrange objects according to values of partial ordering

Another principle in performing task ontology research is "to homogenize the knowledge sources required by every building block." Determination of the granularity is one of the serious problems in ontology research. Although it is not easy to cope with this problem, the above principle helps us identify the right size of building blocks. Furthermore, it becomes easier to acquire domain knowledge, since interview could be well-focussed and well-situated. Let us examine how the above building blocks satisfy the principle.
(1) *Assign*, *Pickup* and *Sequence* do not require any domain knowledge, since necessary information is given as input information. They are primitive building blocks in this sense.
(2) *Select* and *Evaluate* require criterion functions, conditions, or set of rules for evaluation. *Classify* requires decision trees, distance or similarity functions, or set of rules. These knowledge sources partly depend on what object is processed by the building block.

In evaluating MULTIS II's knowledge-level ontology, the authors have coordinated efforts with a working group sponsored by ASTEM/R (Advanced Software Technology & Mechatornics Research Institute of Kyoto), which consists of members from NEC Corp., Toshiba Corp., Mitsubishi Electric Corp., CSK Corp., Nippon Steel Corp., Hitachi Ltd., Ritsumeikan University and chaired by Riichiro Mizoguchi of Osaka University. Members of the consortium evaluated MULTIS II's vocabulary using it to describe several real tasks. Results of the evaluation, which were very promising, are reported in detail on two tecnical reports (Mizoguchi et al., 1992 & ASTEM 92). The group's main goal is to use task ontology for defining and accessing very large knowledge bases. Another evaluation was made by Honda et al. with description of nine scheduling KBS's who reported that the ontology was sufficiently expressive (1991). Future plans for the ontology include the augmentation of the case base, formalization of more generic vocabulary and generic processes, and evaluation of interview behavior of the system's ontologies.

4 MULTIS II: Its Structure and Mechanism

The basic idea employed in MULTIS II is that a problem-solving engine of a KBS under consideration can be synthesized from pre-fabricated components. For this purpose, it has a library of building blocks that correspond to the components of the problem-solving engine. One of the major issues in this research is how to enable domain experts to synthesize problem solving engines for their tasks. Generic vocabulary and generic processes are designed to this end. They are easy for domain experts to understand than the building blocks, because they don't need to know the details of their implementation.

Even if the vocabulary the interview system uses is understandable to domain experts, it is not easy for them to write a control structure from scratch. MULTIS II employs Case-Based Reasoning (CBR) that presents the domain expert appropriate cases of several KBS's control structures described as GPN's. Domain experts can build their problem solving engines by modifying the case data. Thus, description of case data and retrieval of them are crucial issues in MULTIS II. GPN's are stored in the case base with several kinds of indexes such as 1) domain, 2) solution representation, 3) goal, 4) group or dependencies between schedule-recipients, and 5) time axis.

MULTIS II shows cases to the user in several ways —one in terms of generic vocabulary, another one in terms of the domain concepts of the particular domain and yet another one in terms of the domain concepts under consideration— since cases have correspondence between generic vocabulary and domain concepts. The translation between generic vocabulary and domain concepts is made very smoothly. This helps domain experts understand what the GPN is and how to modify cases to generate one that describes the underlying task. For a detailed description of the search mechanism consult (Tijerino et al., 1990).

4.1 MULTIS II Structure

Figure 2 shows the structure of MULTIS II. It is composed of a Task Identification Module (TIM), a CBR Module, a Generic-Process-Network Editor (GPNE), a Generic-Vocabulary Browser (GVB), a Generic-Process-Network Compiler (GPNC), a Domain-Knowledge Acquisition Module, a Testing Module, and an Interface that combines all the modules to make them work in concert with each other. The following sub-sections describe all of these modules.

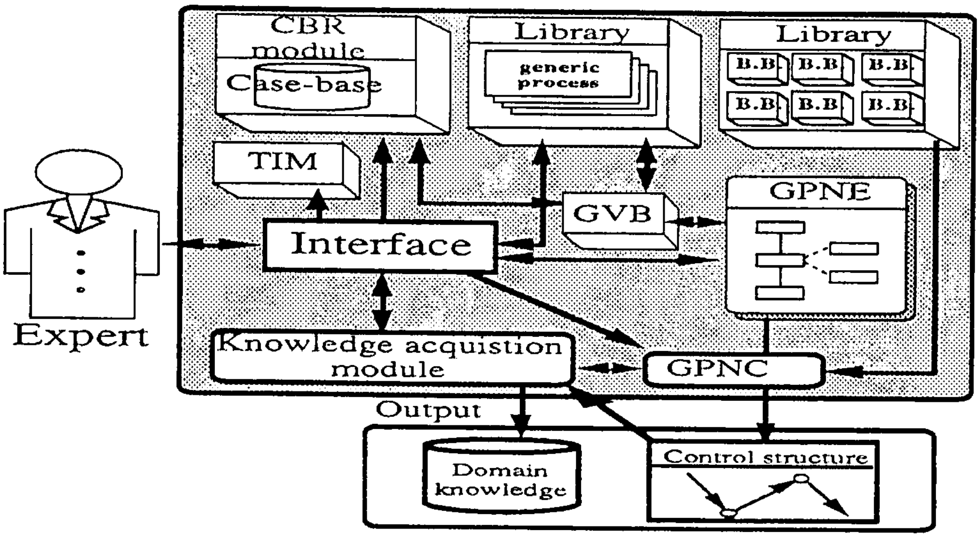


Figure 2. MULTIS's architecture. CBR = Case-Based Reasoning, TIM = Task Identification Module, GVB = Generic Vocabulary Browser, GPNE = Generic Process Network Editor and GPNC = Generic Process Network Compiler

4.2 Building a Scheduling KBS with MULTIS II

In order to illustrate better MULTIS II's Mechanism, this section presents an example of how MULTIS II empowers domain experts to model their problem-solving processes. This illustrative example shows MULTIS II's functional components and how they interact with a domain expert that does not necessarily knows much about building KBS's.

4.2.1 Scheduling Belt Production

Throughout the years, John has been in charge of scheduling production of leather belts for his company, Acme Co., at Marlboro, Massachusetts. The main goal is to have continuity in production of orders as the first constraint expresses. Figure 3 shows what a typical order list looks like.

Belt production consists of many sequential processes. Each process requires a processing workstation. This example, will only describe the five processes that produce the finished leather part of the belt. They are: 1) dying the belt with 5 possible colors, 2) cutting it to 4 different possible sizes, 3) punching varying number of holes of appropriate diameter, 4) engraving the belt with 3 possible patterns, and 5) sawing the belt borders. These processes are only part of a more complex web of processes, but only these will be considered for the sake of simplicity. Figure 4, shows what a typical schedule looks like for the orders presented in figure 3.

This is a scheduling problem because John assigns workstations to orders at specific times. Thus, when John starts a new session with MULTIS, he can select scheduling as the "Type of task." Figure 5, shows the New Session window.

Order Num.	Color	Sizes	Hole Size/Num	Engraving Pattern	Thread Color/Size	Amount	Due Date
1023	Brown1	S-M-L	1/4		Brown/1	100/ea	1/29
1101	Black	S-M-L	1/5	Patt1	Black/1	150/ea	1/27
1129	Blue1	S-M-L	1/5		Black/1	50/75/100	1/25
1165	Brown1	S-M	1/4	Patt3	Black/1	150/200	1/28
1178	Brown2	S-M-L	1/5		Brown/1	200/ea	1/29
1179	Blue2	S-M-L-X	1/4	Patt2	Black/1	50/ea	1/27
1191	White	S-M-L	1/4		White/1	150/ea	1/27
1211	Black	S-M	1/4		Black/1	150/300	1/29
1235	White	M-L	1/5	Patt1	White/1	200/250	1/26
1237	Brown1	S	1/4		Brown/1	45	1/25
1253	Brown2	M	1/5	Patt3	Black/1	70	1/27
1266	Blue1	L	1/5		Black/1	85	1/29

Figure 3. A typical order form for John's belt scheduling problem.

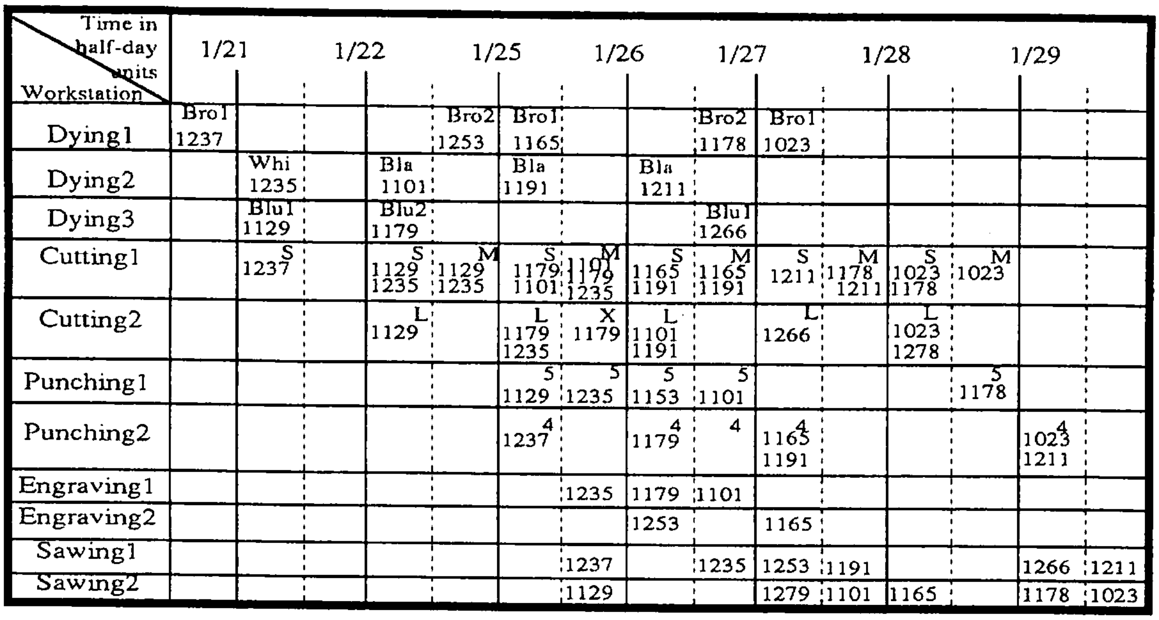


Figure 4. A typical scheduling solution for belt production at Acme co.

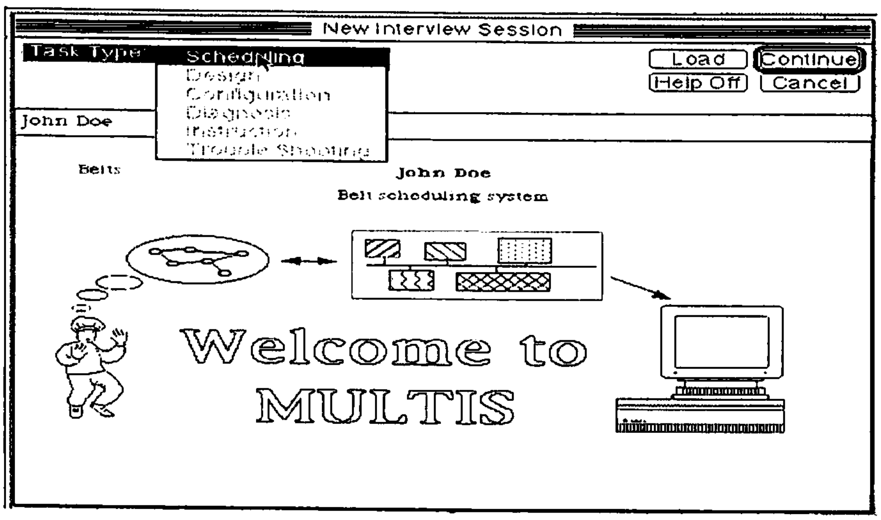


Figure 5. The New Interview Session Window

4.2.2 Task Analysis and Modeling

Task analysis begins after reducing the problem space in the previous step to scheduling tasks. Because the system knows that Gantt charts and tables usually represent solutions to scheduling problems, it shows some typical generalizations of these solutions to John, as figure 6 illustrates.

After looking through the choices of Gantt charts and reading the explanations of the titles by clicking on the Explain button, John notices that the one with the title "Continuous Recipient; Resource=1+" is the most similar to the Gantt chart given in figure 4. Continuous recipient means that the schedule recipient, in this case, orders go through different processing stages. Resource=1+ means that there are more than one schedule resource at the abscissa of the chart.

Figure 7, shows the system requesting more information about the solution representation for John's problem. It asks for vital information for task concepts in the form of generic vocabulary, such as *schedule-resource*, *schedule-recipient* and *time*. John chooses *RECIPIENT-TIME-CONTINUITY* from a list of four main goals. *Recipient-time-continuity* means that the recipient has to go through the schedule resources continuously after it has been started.

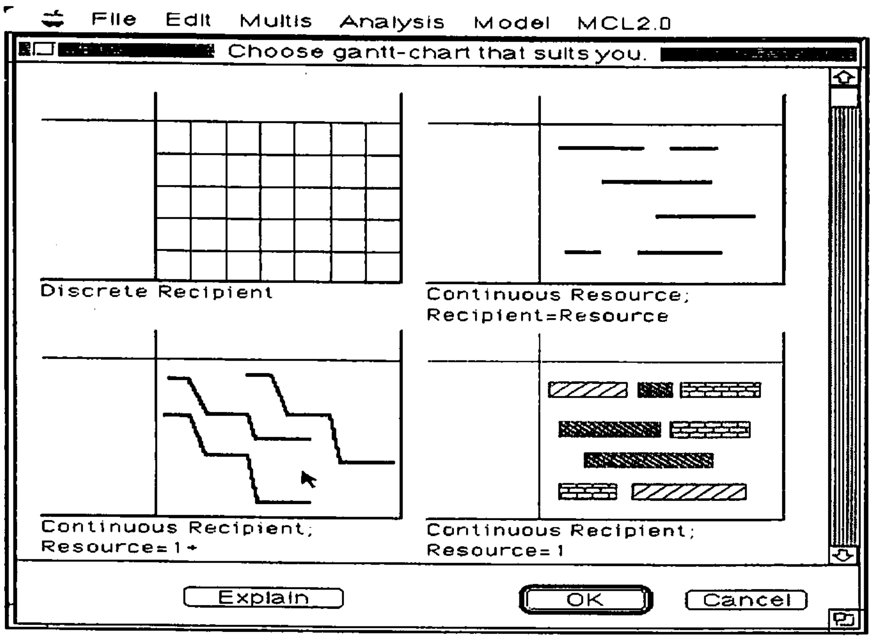


Figure 6. Different choices for scheduling solution representations in scheduling problems.

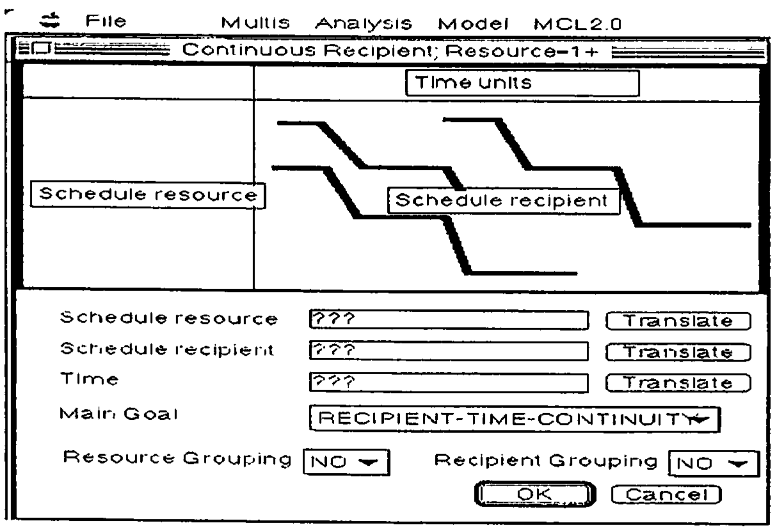


Figure 7. More data being requested about the solution representation.

Next John chooses to translate the vocabulary that he inputs for the concepts of *schedule-resource*, *schedule-recipient* and *time*. He chooses the translate buttons and the window shown in figure 8 appears. This window is the interface of the *vocabulary browser*, an idea borrowed from Klinker et al. (1992). The vocabulary hierarchy shows the hierarchy of the terms used to translate the term, in this case *workstation*. John double-clicks on *processor* from the "Most Specific Terms," which seems to be the most similar term to *workstation* and the term *processor* is put at the lowest place in the vocabulary hierarchy as illustrated in figure 9. Notice that there is also a new list of terms shown as examples of *processor*. John can double chooses any of these terms and the system will show him where those terms are actually used. After looking at an example of where *furnace* is used (see figure 10a), John decides that *processor* is the appropriate term. He then adds the term *workstation* to the example list by choosing the "Add Term as Example" from the "Edit" menu of the window (see figure 10b). Later, he goes through the same process for *orders* and *time*, which are translated into *order* and *time* respectively.

So far, John has input information that will be useful for a preliminary search of the scheduling case base for systems that solved similar problems to that of John's. He can now start analyzing how he solves scheduling problems with help of the case-base reasoning

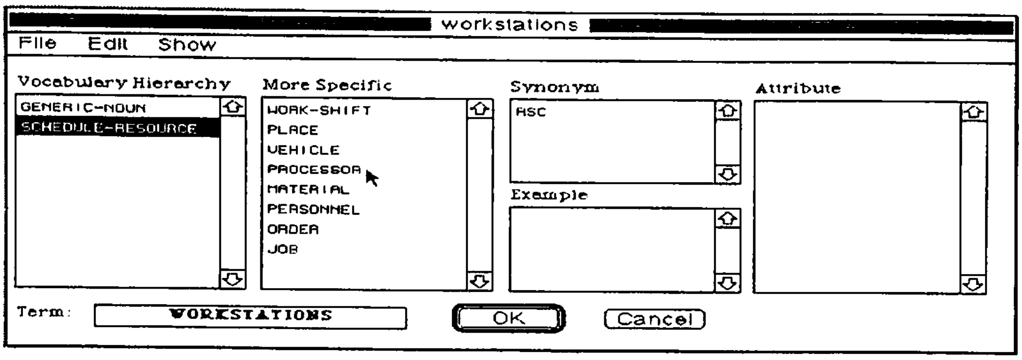


Figure 8. The Generic Vocabulary translator helping John translate the term workstation.

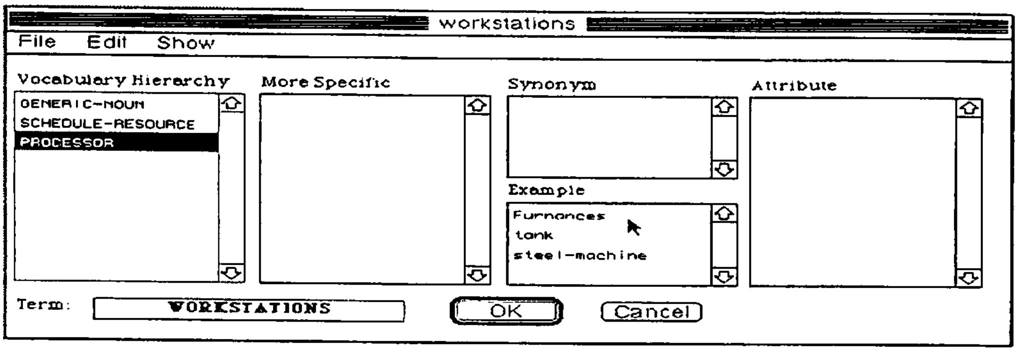


Figure 9. The final translation of workstation to process.

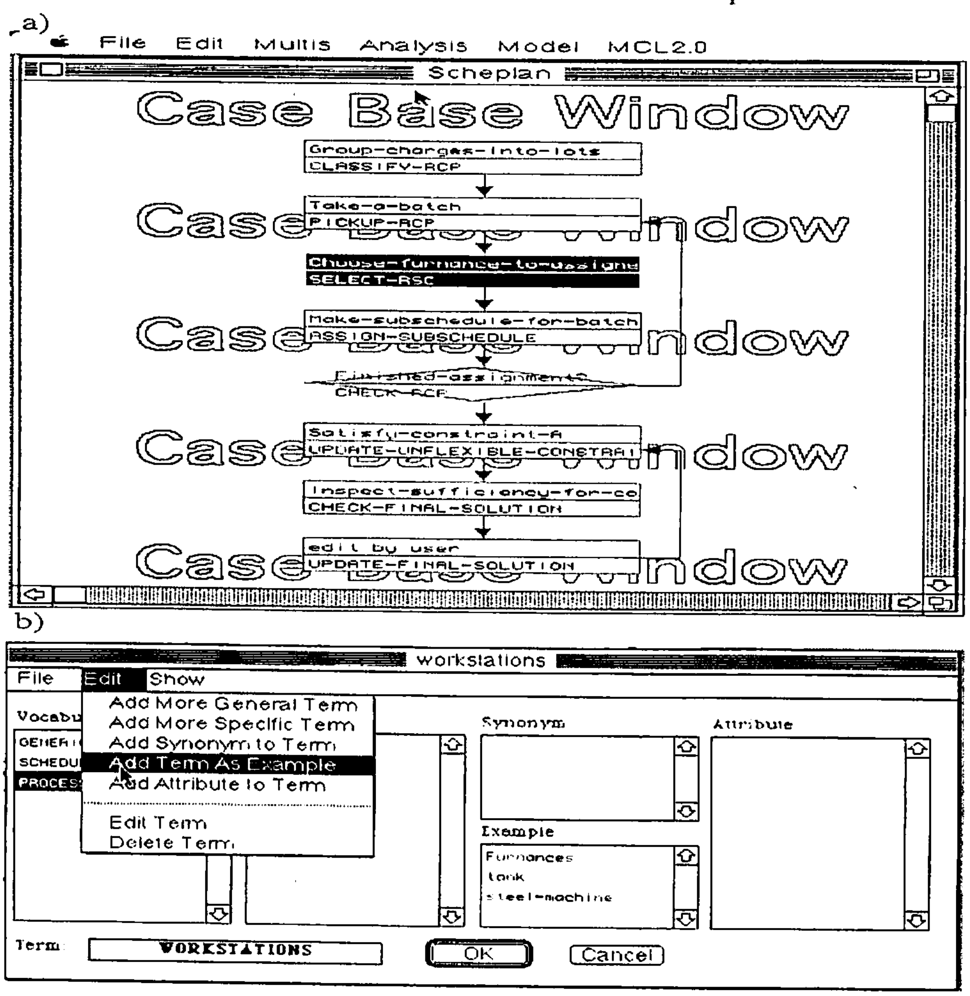


Figure 10. a) Example of where furnace is used. b) Workstation is added to example list.

engine in MULTIS II. Figure 11 shows a GPNE window in which the model of the task can be constructed. There are four buttons at the upper left of the window, with which he can select various tools to generate a task description in terms of a GPN. This tool provides enough functionality so that John can build his problem solving model from scratch by performing actions on objects such as naming them, translating them to MULTIS II's ontology or cutting, copying, pasting and dragging them around.

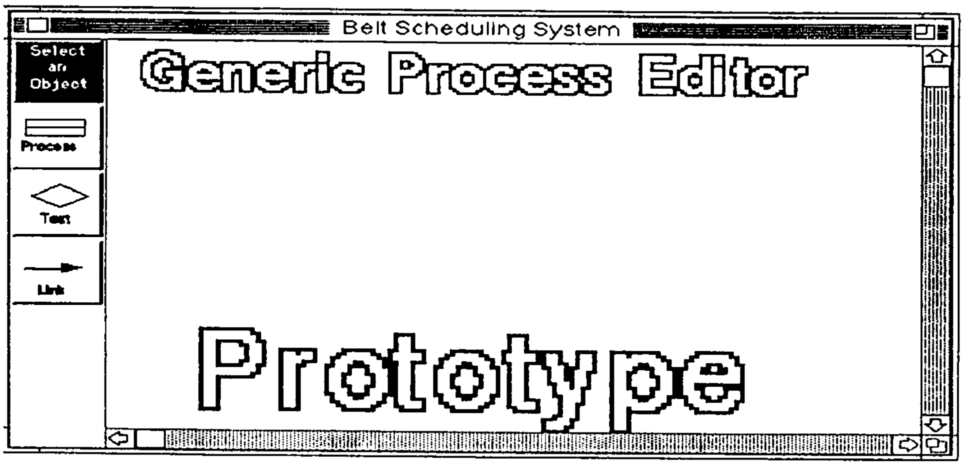


Figure 11. A typical window for the Generic Process Network Editor.

Next John decides to see whether the case base has a case with similarity to his problem. He selects the "Search Case Base..." menu item from the "Model" menu. The system finds some similar cases and asks John how many he wants displayed (see figure 12) and John answers that he wants to see 3. The system then shows the generic process network for each of the 3 cases in a different window, as shown in figure 13.

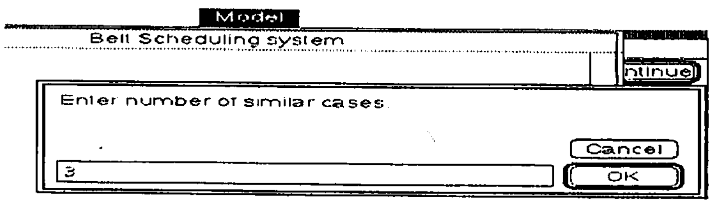


Figure 12. A window requesting the number of similar cases to display.

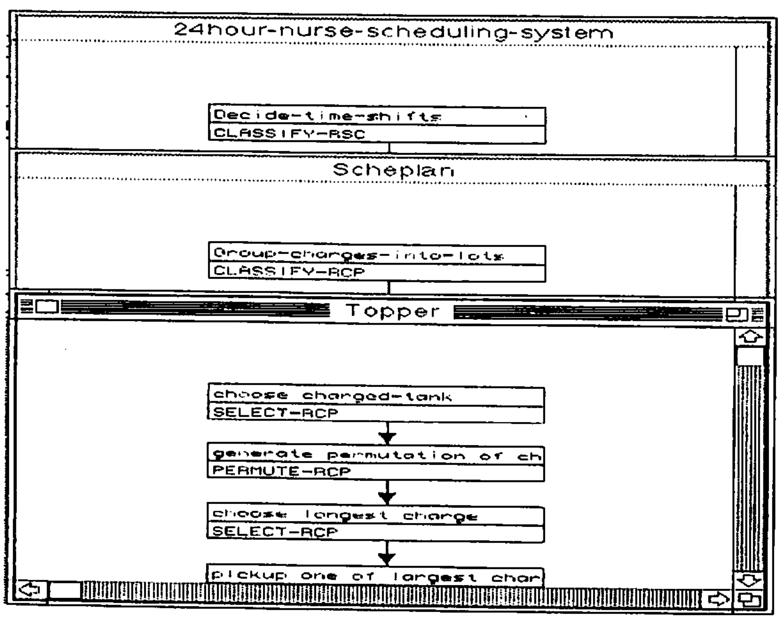


Figure 13. 3 similar case to John's problem.

John can look at each of the cases extracted with case-based reasoning and copy parts of it if he thinks they are appropriate or can fill in parts that he feels confident about. These cases were extracted mainly according to an indexing method that is similar to that of Protos (Bareiss, 1989; see also Tijerino et al., 1990) In this case, the following indexes were used:
Reminders:

1) Scheplan has same type of solution representation
2) Both Scheplan and Topper have processor as the schedule resource.

Prototypicality:

Topper is chosen as the most similar case because it is the prototypical case for cases with processor as the schedule resource.

4.2.3 Model Refinement

John copies the Topper case and starts to modify it as shown in figure 14. Notice that the system exchanged the labels of the generic processes to reflect what it already knows about John's vocabulary. Those terms in capital letters on the upper-half of the processes are the ones substituted by MULTIS II. The others are left the same as in the Topper case.

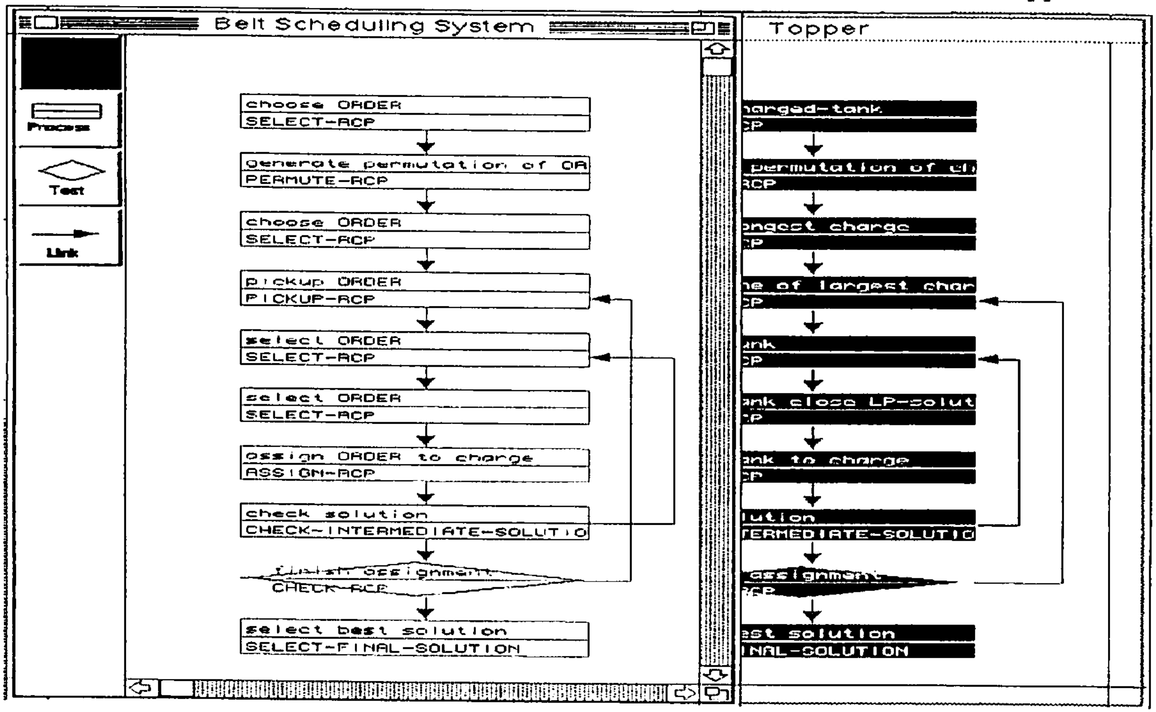


Figure 14. Case modification. John modifies a case named Topper to reflect his problem solving process.

John now modifies the tentative representation to reflect his. He first starts by changing the names of the processes. Figure 15 shows how he changed the name of the first process from "choose ORDER" to "sort orders by due date." This figure also shows how he is about to translate the process differently from how it is translated in Topper. First, he inputs the corresponding verb and nouns of the phrase. He, then, clicks on the translate button next to the verb that he just input. Figure 16 and 17 depict how John browses through the vocabulary hierarchy for generic verbs and chooses the verb SEQUENCE as the translation for sort. Figure 18 displays the final translation of the verb and noun.

John goes through a similar process for the rest of the processes producing a representation of his task that is satisfactory. Figure 19 shows this representation. The processes consist of:

4.2.4 Building Blocks and Resulting Problem-Solving Engine

MULTIS II still doesn't have enough information to start building a system for scheduling belts based on the network which John described. MULTIS II needs to map the processes in the network to appropriate building blocks in its symbol-level ontology. How-

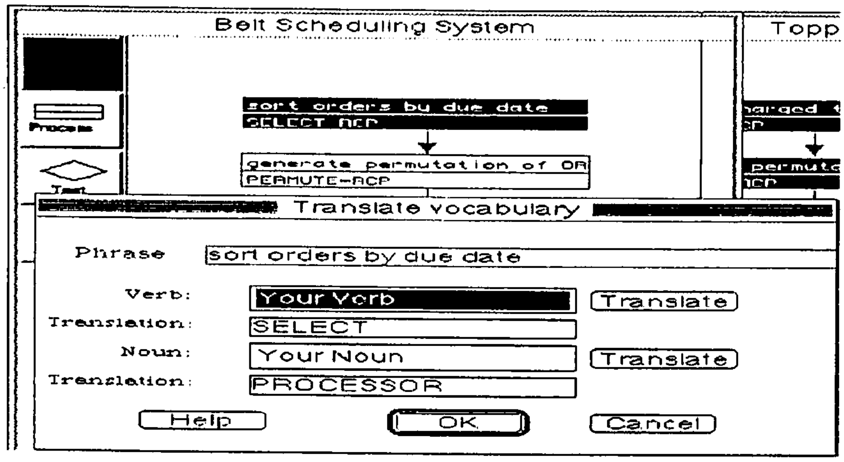


Figure 15. Vocabulary translation from Topper's GPN.

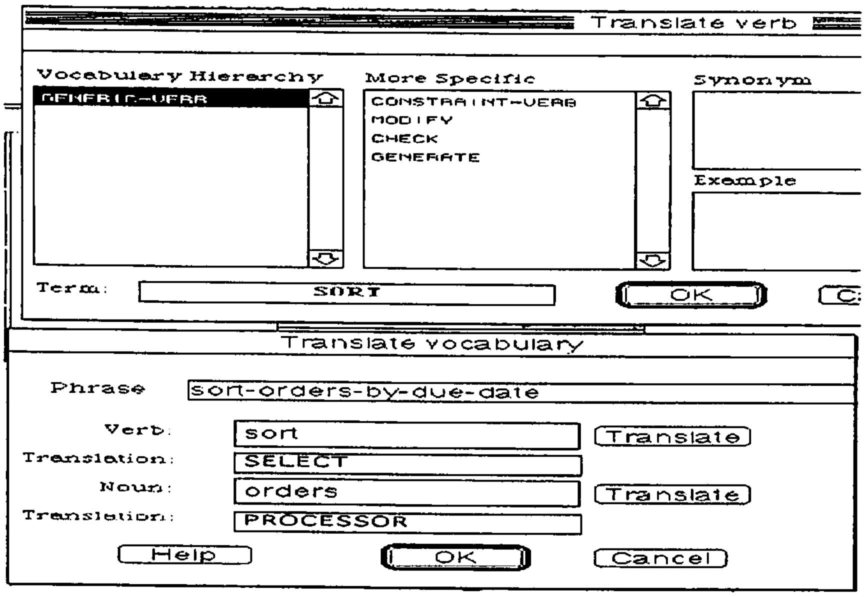


Figure 16. MULTIS II's vocabulary browser.

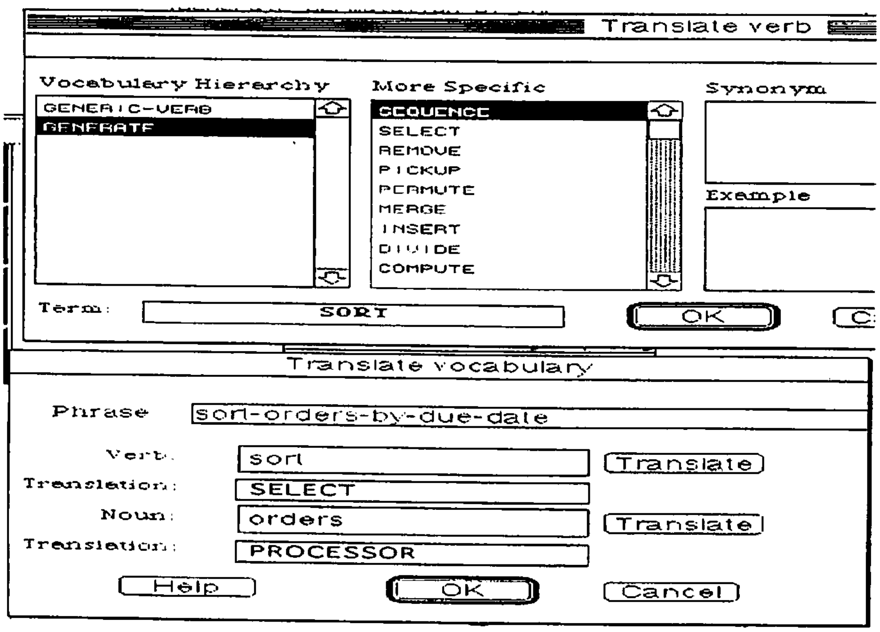


Figure 17. SEQUENCE being translated to SELECT with the GVB.

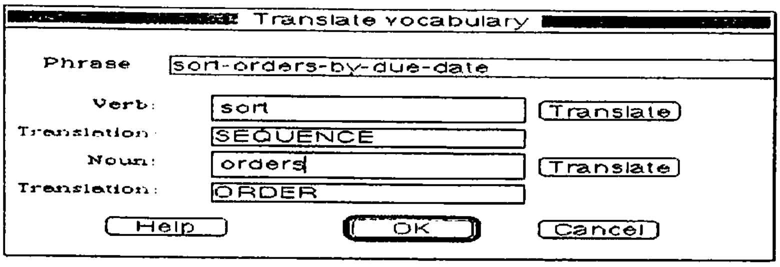
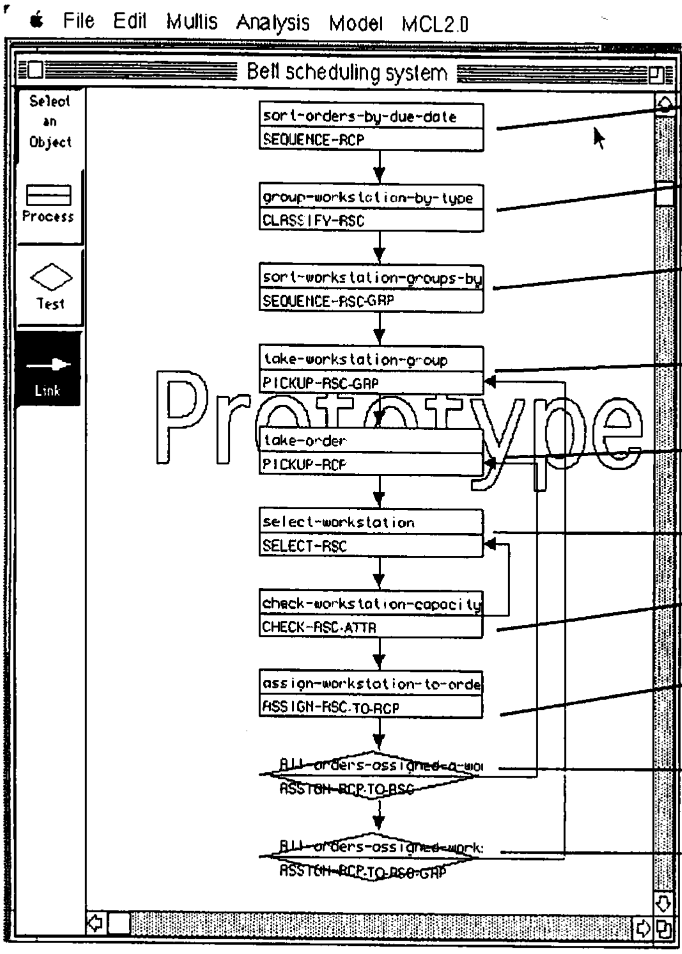


Figure 18. The final translation for sort-orders-by-due-date.



1) sort orders by due date, which is translated to SEQUENCE-RECIPIENT,
2) group workstations by type, which is translated to CLASSIFY-RESOURCE,
3) sort workstation groups by process sequence, which is translated to SEQUENCE-RESOURCE-GROUP,
4) take workstation group, which is translated to PICKUP-RESOURCE-GROUP,
5) select-workstation, which is translated to SELECT-RESOURCE-GROUP,
6) select-order, which is translated to SELECT-RECIPIENT
7) check-workstation-capacity, which is translated to CHECK-RESOURCE-ATTRIBUTE,
8) assign-workstation-to-order, which is translated to ASSIGN-RESOURCE-TO-RECIPIENT,
9) All-orders-assigned-a-workstation, which is translated to ASSIGNED-RECIPIENT-TO-RESOURCE? and
10) All-orders-assigned-workstation-type, which is translated to ASSIGNED-RECIPIENT-TO-RESOURCE-GROUP?

Figure 19. The Belt Scheduling System's final GPN representation.

ever, MULTIS II already has constrained the search space with John's help because he translated the processes to terms MULTIS II understand. Since these terms are associated with building blocks MULTIS II has selected the building blocks depicted in figure 20.

After some interaction with John, MULTIS II infers that the building blocks in figure 21 are the most appropriate to construct the underlying problem-solving engine for John's system. The structure of the building blocks is represented in listing 1 and the resulting engine is depicted in listing 2. MULTIS II's GPN compiler constructed this engine taking into consideration input/output relations between the building blocks in listing 1 and by asking questions to John when it fell in deadlocks or needed more information. Listings 1 and 2 are in LISP format and should give an idea of the simplicity and modularity of the kind of code that MULTIS generates.

4.2.5 Domain Knowledge Acquisition

The domain knowledge elicitation process is quite straight forward because the building blocks and the problem-solving engine help to guide it implicitly because of their data and knowledge requirements. The system generates queries such as: "Enumerate worksta-

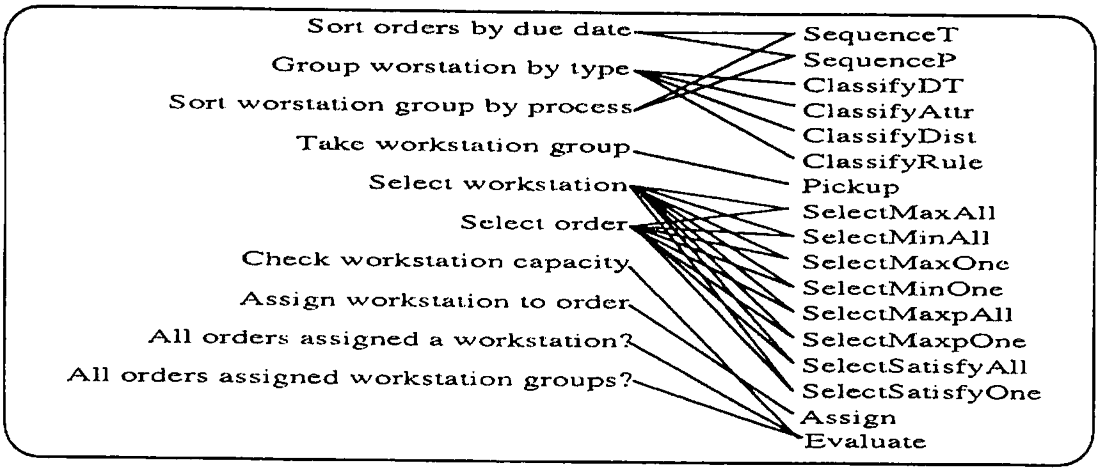


Figure 20. The choices of building blocks (right) that can implement John's processes (left).

Sort orders by due date	SequenceT
Group worstation by type	ClassifyAttr
Sort worstation group by process	SequenceT
Take workstation group	Pickup
Select workstation	SelectMaxOne
Select order	SelectMinAll
Check workstation capacity	Evaluate
Assign workstation to order	Assign
All orders assigned a workstation?	Evaluate
All orders assigned workstation groups?	Evaluate

Figure 21. Building blocks selected by MULTIS II to implement John's processes.

```
(defmacro sequenceT
        (list &optional (condition nil))
  `(if (null ,condition)
     (setf ,list (sort ,list #'>))
     (setf ,list (sort ,list ,condition))))

(defun Classify-Attr (list get-attr)
  (prog ((result '())
         attr class
         (ans '()))
    (dolist (obj list)
      (setq attr (funcall get-attr obj))
      (setf class (assoc attr result))
      (if (null class)
        (push (list attr obj) result)
        (rplacd (assoc attr result)
                (push obj (cdr class)))))
    (setf result (mapcar #'cdr result))
    (dolist (tmp result)
      (push (nreverse tmp) ans))
    (return ans)))

(defmacro pickup (list-top list)
  `(setf ,list-top (pop ,list)))

(defun check (target &optional
                     (condition #'null))
  (funcall condition target))
```

```
(defmacro select-min-all (object list condition)
  (let ((ans (gensym)) (tmp (gensym)) (top-value (gensym)))
    `(prog ((,ans '()) ,tmp ,top-value)
       (setf ,tmp (mapcar #'(lambda (obj)
                              (list (funcall ,condition obj) obj))
                          ,list))
       (setf ,tmp (delete-if #'(lambda (x) (eq (car x) nil)) ,tmp))
       (setf ,tmp (sort ,tmp #'< :key #'car))
       (setf ,top-value (car (car ,tmp)))
       (dolist (element ,tmp)
         (if (equal ,top-value (car element))
           (push element ,ans)))
       (setf ,ans (nreverse ,ans))
       (setf ,object (mapcar #'second ,ans)))))

(defmacro assign (sol time obj1 obj2)
  `(if (and (not (null time))
            (not (null obj1))
            (not (null obj2)))
     (setf ,sol (list ,time ,obj1 ,obj2))))

(defmacro select-max-one (object list condition)
  (let ((ans (gensym)))
    `(prog (,ans)
       (select-max-all ,ans ,list ,condition)
       (setf ,object (car ,ans)))))
```

Listing 1. Building blocks in MULTIS II used to represent the symbol-level model of the Belt Scheduling System. Notice the simplicity which is due to the fact that most of the knowledge is in the form of constraint conditions, that is, domain knowledge.

```
(defvar *rsc* nil)
(defvar rsc nil)
(defvar *rcp* nil)
(defvar rcp nil)
(defvar *solution* nil)
(defvar *stack* nil)

(defvar sequence-cond1 nil)
(defvar classify-cond1 nil)
(defvar sequence-cond2 nil)
(defvar select-cond1 nil)
(defvar check-cond1 nil)
(defvar check-cond2 nil)
(defvar check-cond3 nil)

(setq rcp (copy-list *rcp*))
(setq rsc (copy-list *rsc*))
```

```
(defun belt-scheduling-problem-solving-engine ()
 (prog (rsc1 rsc2 rsc3 rsc4 rcp1 rcp2 time sol)
   (setq rcp1 (sequencet rcp sequence-cond1))
   (setq rsc1 (classify-attr rsc classify-cond1))
   (setq rsc2 (sequencet rsc1 sequence-cond2))
1
   (pickup rsc3 rsc2)
2
   (pickup rcp2 rcp1)
3
   (select-max-one rsc4 rsc3 select-cond1)
   (if (check rsc4 check-cond1)
     (go 3))
   (assign-all sol rsc3 rcp2 time)
   (if (check rcp2 check-cond2)
     (go 2))
   (if (check rsc3 check-cond3)
     (go 1))
   (return *solution*)))
```

Listing 2. Problem solving engine compiled by MULTIS II. for the Belt scheduling system. All variables defined globally represent domain knowledge that must be acquired before the engine becomes functional and the lexical variables represent inputs, outputs and their transformation.

tions and their type" "What is the condition for selecting one order over another?" "What determines selection of workstations for assignment to an order?" These questions along with the list of orders that need to be assigned, produce the knowledge base for scheduling belt production. Notice that the answers for the example questions above do not necessarily lead to rules.

Because the scope of this research was mainly to produce the problem-solving engine, the authors have only partially implemented the knowledge acquisition module. This module instantiates the variables and parameters used by the building blocks with test data so that the user can evaluate the resulting KBS. Nevertheless, this evaluation is still implicit and more usage of the system by real experts is still needed for a more complete evaluation. With the help of the author, however, MULTIS II was able to produce a prototype of the belt scheduling system that produced a solution similar to that shown if figure 4.

5 General Discussion

5.1 Problems Encountered

The example of previous the section covered the different stages of interview in MULTIS II. This illustration should give an eagle-eye view of how one might use MULTIS II and take advantage of its functionality. However, since MULTIS II was only built to test the hypotheses on task ontologies and not on domain ontologies it still needs much work for producing real-world applications. Also, the case-base needs to be expanded to cover more types of scheduling problems. The problem in the example became part of the case base and its prototypicality will increase according to the number of times its GPN is modified to represent future cases.

Another problem that arises with the framework for task ontologies and the system MULTIS II is that of enabling users to generate real-world task descriptions without help from knowledge engineers. The MULTIS II system is just one experiment on how to map the ontology translations between experts and the computer. Even though the GVB gives some capability to the expert to identify the vocabulary for a particular task and its relation to MULTIS II's ontology, it may be extremely difficult in some circumstances to determine a perfect description or even a rough one. MULTIS II is just one of such experiments and there are many other similar efforts (McDermott et al., 1990; Klinker et al., 1991; Marques et al., 1991; Dallemagne et al., 1992; and Marques et al., 1992), (Steels, 1990 and 1992), (Wielinga, Schreiber and Breuker, 1992), (Runkel and Birmigham, 1992), (Hori and Nakamura, 1991 and 1992), and (Puerta and Musen).

The authors argue that these efforts, though making a large contribution, have only helped to identify symbol-level ontologies similar to the building blocks discussed in previous chapters. Consequently, they provide no direct mapping between the expert and the computer because it is made by task analysts (that is people, whose job it is to map one description of the task [e.g., that of the task performer's] onto another description of the task [e.g., that of the programmer's]). The ontologies presented in this paper make an important contribution to solve this problem, however they still need much refinement. One of the authors joined efforts with one of the groups mentioned above, namely McDermott's group at Digital Equipment Corporation and investigated how knowledge engineers make the mapping themselves. As a result of the study, the UnitedWorlds framework, presented in another paper (Tijerino et al. 1992), was proposed as a methodology that analysts can use to bridge that gap, such that the resulting task descriptions help write application programs that fit into the workplace rather than intrude on it. This study was performed as a step to identify what needs be done to formalize the translation process of task performer's descriptions into descriptions useful for generation of application programs. The resulting framework gives insight on what task analysts are or should be asking task performers to generate those descriptions.

5.2 Related Research

Recently, the knowledge acquisition community has grown into different schools of thought. There are subjects of research such as knowledge acquisition from text (Boy 1989 and 1990), by induction (Quinlan 1986 and Mitchell et al., 1986), from primitives (Chandrasekaran, 1986 and McDermott, 1988), and others. More recently the community has tried to coordinate efforts to produce shareable and reusable primitives in the form of ontologies (Neches et al., 1991). This work has evolved from these efforts and from the idea that there must be two different connecting intermediate representation primitive sets, that is, the knowledge-level ontologies and task-level ontologies described in this paper chapters, that can bridge the gap between human and computer representations. This idea is what has made this research different from others.

5.2.1 Chandrasekaran's Generic Tasks

Chandrasekaran (1986) and Bylander and Chandrasekaran (1987) proposed units of search control mechanisms that are very useful to construct inference engines for KBS's 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 problem-solving tasks, are too specialized in the context of knowledge engineering. Therefore, they are very difficult to be understood by the domain expert himself.

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 Problem-solving 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 by knowledge engineers to construct KBS's.

5.2.2 Clancey's Heuristic Classification

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

5.2.3 McDermott's Half-Weak Methods

Introduction of *half-weak methods* by McDermott (1988) contributed greatly to knowledge engineers in building KBS's. However, as Chandrasekaran's generic tasks, it is difficult for an average domain expert to understand his/her own problem-solving process in terms of such half-weak methods. *Half-weak methods*, as their name imply are half way the continuum between *weak methods* and *strong methods*. Weak methods are weakly constrained to task features, i. e., they show weak task dependency, but no domain dependence. Strong methods, on the contrary, show both strong dependency to a particular task in a particular domain. Therefore, *half-weak methods* are those methods that show strong dependency on task, but none on a particular domain, e.g. *heuristic classification*. McDermott has presented examples of half-weak methods such as: (1) *cover-and differentiate*, a method suitable for certain types of diagnostic tasks and (2) *propose-and-revise*, a method suitable for certain types of constructive tasks. These methods have evolved into a family of methods called mechanisms for a wide variety of tasks (Marques et al., 1991).

Again, these so-called *half-weak* methods are similar to MULTIS II'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*. Recently, Klinker et al. (1992) have been working along these lines by introducing a *Cataloguer/Browser*, and idea borrowed for MULTIS II's GVB.

5.2.4 KADS

Breuker and Wielinga (1989) propose to make models of problem-solving 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 framework they proposed, is intended for knowledge engineers, its potential for usability by domain experts seems to be high. Because KADS's knowledge modeling framework uses generic processes and generic vocabularies (though not called that way) to model human problem solving it is extremely similar to the MULTIS II's modeling mechanisms.

6 Concluding Remarks

The major objectives of this work were 1) design a general methodology for knowledge acquisition for KBS construction, 2) identify general and reusable task components for problem solving in the form of task ontologies, 3) outline general guidelines for task analysis interview based on two-level task ontologies, 4) synthesis of KBS's from components, and 5) identify what problems arise from the methodology and how to solve them. At least, this work contributed toward these objectives in 1) by designing MULTIS II, 2) providing shareable and reusable components, 3) outlining the guidelines in the system, 4) synthesizing KBS's as an output of the system, and 5) identifying problems throughout the work.

Overall, this work contributed mostly by introducing the two-level task ontologies and outlining guidelines to map them as intermediate representations between the domain expert and the computer. So far there is no distinction made on two-level ontologies in the research community. This work should be able to lead others in accomplishing this goal to greater extent. The implications of making this mapping between human and machine are of great importance to the development of intelligent computer systems and the augmentation of intelligence of their users. The reason for this is that the computer will be more accessible to all persons not only as a tool, but also as a partner in problem solving. Before this goal can be accomplished, there must be more effort spent in identifying two-level ontologies for not only the tasks mentioned in Chapter 4, but also for a wide range of domains. This might seem too ambitious a goal, but some researchers have already started doing it (Guha and Lenat, 1990).

References

ASTEM (1993). Toward very large knowledge bases for expert system construction assistance. Advanced Software Technology & Mechatronics Research Institute of Kyoto. TR-P048-92.

Bareiss, R. (1989). *Exemplar-based knowledge acquisition*. Chandrasekaran, B. (ed) Academic Press.

Boy, G. A. (1989). The block representation in knowledge acquisition for computer integrated

documentation. In *Proceedings of the 4th KAW.* Banff, Alberta Canada.
Boy, G. A. (1990). Acquiring and refining indices according to context. In *Proc. of the 5th KAW.*
Breuker, J, and Wielinga, B. (1989). Models of expertise in knowledge acquisition. In *Topics in Expert System Design,* G Guida and C Tasso (eds). pp. 265-295.
Bylander, T. and Chandrasekaran, B. (1987). Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition. Inter. Jour. of Man-Machine Studies. Vol. 26, pp. 231-243.
Chandrasekaran, B. (1986) Generic tasks for knowledge-based reasoning: the right level of abstraction for knowledge acquisition, *IEEE Expert,* 1, pp. 23-30.
Clancey, W.J. (1985). Heuristic classification, *Artificial Intelligence,* 27, 3, 1985.
Dallemagne et al. (1992). Making application programming more worthwhile, In *Knowledge Engineering and Cognition,* F. Schmalhofer and G. Strube, eds., Springer-Verlag, Berlin.
Guha, R.V. and Lenat, D.B. (1990). CYC: A Mid-Term Report, *AI Magazine,* 11, 3, pp. 32-59.
Honda, T., Tijerino, Y. A., Kitahashi, T., Mizoguchi, R., Hasegawa, H., Nomura, Y. (1991). An interview system based on a problem solving model: MULTIS – verification of problem solving primitives–. In *Proc. of 5th Annual Conference of the Japanese Society on Artificial Intelligence,* Vol. II, pp. 807-810. (Japanese).
Hori, M, and Nakamura, Y. (1991). Synthesizing scheduling engines based on inference structure. In *Proceedings of the World Congress on Expert Systems,* pp 1215-1222.
Hori, M, and Nakamura, Y. (1992). Methodology for configuring scheduling engines with task-specific components. In Proc. of the JKAW. pp. 215-229.
Klinker, G., Bhola, C., Dallemagne, G., Marques, D. and McDermott, J. (1991). Usable and reusable programming constructs, *Knowledge Acquisition,* Vol. 3, 117-135.
Klinker, G., Marques, D., McDermott, J. Mersereau, T. and Stintson, L. (1992). The active Glossary: taking integration seriously. In Proc. of the 7th KAW. pp. 14-1 to 14-19.
Marques, D., Dallemagne, G., Klinker, G., McDermott, J. and Tung, D. (1991). More data on usable and reusable programming constructs, *Proc. Sixth KAW,* SRDG Publications, Dept. of Computer Sci., Univ. of Calgary, Alberta, Canada, T2N 1N4, 14-1 to 14-19
Marques, D., Klinker, G., Dallemagne, G., Gautier, P., McDermott, J. and Tung, D. (1992). Easy programming - empowering people to build their own applications, *IEEE Expert,* 7, 3, 16-29.
McDermott, J. (1988). Using problem solving methods to impose structure on knowledge, *Proc. of the International Conf. on AI Applications,* pp.7-11, 1988.
McDermott J., Dallemagne G., Klinker G., Marques D. and Tung D. (1990). Explorations in how to make application programming easier. *In Proceedings of 1st JKAW.* 134-147.
Mitchell, T. M., Keller R. M. and Kedar-Cabelli, S. T. (1986. Explanation-Based Generalization: An unifying view. *Machine Learning,* 1, pp. 47-80.
Mizoguchi, R., Tijerino, Y., and Ikeda, M. (1992). Two-level mediating representation for a task analysis interview system. *In Proc. of AAAI-92 Workshop for Knowledge Representation Aspects of Knowledge Acquisition.* San Jose, Ca., 107-114.
Mizoguchi, R., Ogawa, H., Ushioda, Y., Kojima, S., Shindo, S., Masuda, R., Nagahashi, K., Masui, S. and Sakama, C. (1992) Society for Articial Intelligence Technical Report, SIG-F/H/S/I-9201-6(12/4), pp. 41-48.
Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. and Swartout, W. R. (1991). Enabling technology for knowledge sharing. *Ai Magazine.* Vol. 12, No. 3, pp. 36-56.
Newell, A. (1982). The knowledge level, *Artificial Intelligence,* Vol. 18, No. 1, pp. 87-127.
Puerta, A. R., Tu, S. W., and Musen, M. A. (1992). Modelling tasks with mechanisms. Technical Report KSL-92-30, Knowledge Systems Laboratory, Stanford University, Stanford, CA.
Quinlan, J. R (1986). Induction of decision trees, Machine Learning, 1, pp. 81-107.
Runkel, J., and Birmingham, W. P. (1992). Knowledge acquisition in the small. In *Proc. of the 7th KAW. pp.* 22-1 to 22-18.
Steels, L. (1990). Components of Expertise. In AI Magazine. Vol. 11, No. 2, 29-49.
Steels, L. (1992). End-user configuration of applications. In Proceedings of 2nd JKAW. pp. 47-64.
Tijerino, Y. A., Kitahashi, T. & Mizoguchi, R. (1990). A task analysis interview system that uses a problem-solving model. In *Proceedings of The 1st JKAW,* 331-344 .
Tijerino, Y., Kitahashi, T. and Mizoguchi, R. (1991). MULTIS: A knowledge acquisition system based on problem solving primitives, In *Proceedings of 6th KAW.* 32-1 to 32-20.
Tijerino, Y. A., Marques, D., Mizoguchi, R., Klinker, G., Kitahashi, T. and McDermott, J. (1992). A study on task descriptions Usefulness for task knowledge acquisition. In *Proceedings of 7th KAW.* pp. 29-1 to 29-19
Tijerino, Y. A., Kitahashi, T. & Mizoguchi, R. (1993). MULTIS: A knowledge acquisition system based on problem solving primitives. *International Journal of Expert Systems.* 5, 2.
Wielinga, B. J., Schreiber, A. Th., and Breuker, J. A. (1992). KADS: A modeling approach to knowledge engineering, *Knowledge Acquisition,* Academic Press, Vol. 4, No. 1, pp. 5-53.