Pergamon

Expert Systems With Applications, Vol. 9, No. 1, pp. 15-25, 1995
Copyright © 1995 Elsevier Science Ltd
Printed in the USA. All rights reserved
0957-4174/95 $9.50 + .00

0957-4174(94)00044-1

Task Analysis Interview Based on Task Ontology

RIICHIRO MIZOGUCHI, YURI TIJERINO,* AND MITSURU IKEDA

The Institute of Scientific and Industrial Research, Osaka University, Osaka, Japan

Abstract—*In order to realize automated knowledge acquisition, we have to solve many problems such as lack of reusability and sharability of knowledge, which is one of the shortcomings in the current knowledge base technology, to fill the conceptual gap between the computer and domain experts and so on. Recent research activities in knowledge acquisition community are focused on task ontology, because it is expected to contribute a lot to making it easier to elicit expertise from domain experts. The authors have been involved in the research of knowledge acquisition and knowledge reuse. This article is concerned with task ontology and its use in a task analysis interview system MULTIS. We first discuss the knowledge reusability to identify appropriate task ontology. Then, we introduce a two-level mediating representation that contributes to bridging the gap and hence to making the task analysis interview fluent. MULTIS has been implemented in Macintosh Common Lisp.*

1. INTRODUCTION

KNOWLEDGE ACQUISITION SYSTEMS usually require knowledge about the task structure of the problem to know the role of knowledge to acquire, since it guides the acquisition process. This observation shows that task analysis is one of the essential jobs in knowledge acquisition. In MULTIS (Mizoguchi, 1988; Tijerino, 1990, 1991), task analysis is made according to two major steps: (1) rough identification, and (2) detailed analysis. Rough identification of task structure is a classification problem, because the expert system (ES) under consideration is identified as one of the prestored types of ESs such as diagnosis, design, configuration, control, and so forth. This task is not difficult to perform in many applications. On the other hand, detailed task analysis is the major topic in MULTIS. This is not an easy task because domain experts cannot articulate how they perform their tasks.

MULTIS is designed as a task analysis interview system, which interacts with domain experts to identify the detailed task structure based on two level mediating representations (Boose, 1990). After task analysis, MULTIS generates a problem solving engine for the target ES. This article presents the MULTIS approach to task analysis problems and mediating representation based on task ontology, which contributes to bridging the understanding gap between the computer and domain experts. A prototype of MULTIS has been implemented in Macintosh Common Lisp (MacIntosh Computers, Cupertino, CA).

2. KNOWLEDGE SHARING AND REUSE

2.1. Knowledge Decompilation

One of the major shortcomings of the current technology for knowledge base building is lack of reusability and sharability of knowledge (Musen, 1991). This makes it difficult to build knowledge bases, since one always has to build them from scratch. Facilitating knowledge sharing and reuse thus should contribute to making it easier to build knowledge bases. In order to achieve this, we have considered decompilation of expertise. One of the reasons why knowledge in most of the current knowledge bases cannot be reused is that it is an already compiled knowledge tailored for specific problems. This leads to an idea of knowledge decompilation, that is, expertise can be decompiled into several kinds of reusable knowledge (Yamaguchi, 1987). Although knowledge decompilation is a very useful idea and has many implications, we will only discuss task/domain decomposition shown in Figure 1. Expertise can be decomposed into a task-dependent but domain-independent portion and a task-independent but domain-dependent portion. Both portions can be further decompiled into several kinds of knowledge.

Requests for reprints should be sent to Riichiro Mizoguchi, The Institute of Scientific and Industrial Research, Osaka University, 8-1, Mihogaoka, Ibaraki, 567 Japan. E-mail: miz@ei.sanken.osaka-u.ac.jp

* Currently in the ATR Communication Systems Research Laboratories

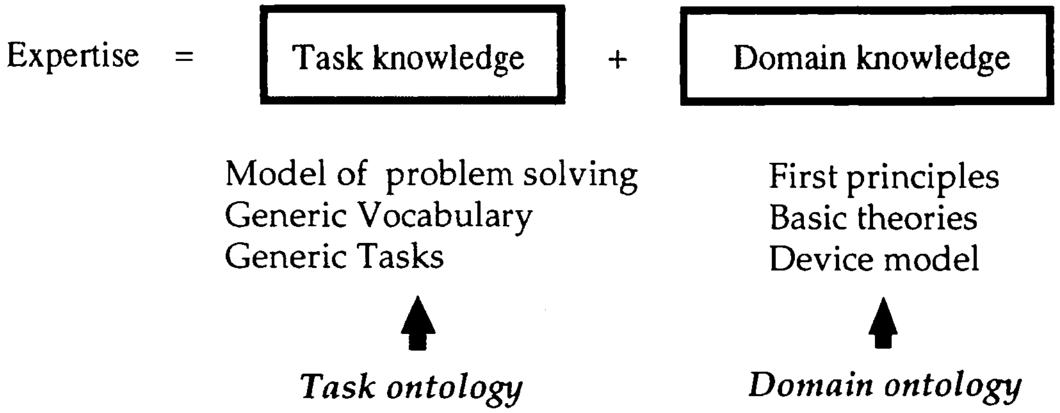


FIGURE 1. Expertise decompilation.

We call the former task knowledge and the latter domain knowledge. Knowledge representation for reuse and sharing requires common vocabulary designed carefully. Both types of knowledge thus require their own ontologies, which play an essential role in our goal. Our major concern in this article is the task knowledge, since it is deeply related to the knowledge acquisition process.

2.2. Task Ontology

Task knowledge is mainly composed of control structures specific to respective tasks. Several models for task structure description have been proposed (Chandrasekaran, 1986; Clancey, 1985; McDermott, 1988), which contribute to providing reusable components for inference engines. Although they discuss their own views of control structure in expert systems, they do not fully discuss ''task ontology'' that is indispensable for task knowledge description. By task ontology, we mean a set of primitives for representation of task structures common to ESs in various domains. To facilitate knowledge reusability, we have to devise appropriate vocabulary for describing the control structure of all expert systems.

Needless to say, it becomes a trivial job to identify such vocabulary if we search for it at a very high abstraction level. But such vocabulary will not be effective, in other words, one cannot gain any benefits by reusing them. So, the main issue is to identify the appropriate level that is general enough in the sense of reusability and effective enough when reused. By vocabulary, we mean not only the one related to control structures or activities but also concepts necessary for describing task structure. In other words, our task ontology should be mainly composed of task-dependent verbs and nouns. We discuss the importance of this later.

Task knowledge is already reusable and sharable in its definition. What should be done next is further analyses of task knowledge to obtain the appropriate task ontology. It would contribute to describe inference engines of existing expert systems as well as future ones in a domain-independent manner. The resulting descriptions can be stored in case data bases which are shared and reused by the prospective expert system developers.

3. SCHEDULING TASK ONTOLOGY

3.1. Mediating Representation

One of the problems interview systems face is the conceptual gap between the domain experts and the system. For the sake of discussion, we assume that domain experts are not necessarily versed on computer literacy and in some cases don't know what an expert system is not to mention interview systems. Similarly, an interview system can not possibly know about all domains. The first thing we have to do is to design an interview system, which can acquire knowledge without knowing the domain in advance. The above discussion contributes to this requirement, because it suggests that an interview system can be designed to have such a case base for task knowledge and to perform task analysis interview based on it.

Determination of who shares the task knowledge is also important, because the user of the sharable knowledge largely affects the abstraction level of its description. As discussed above, we have two kinds of users: one is the domain expert who has to represent their own problem solving processes, and the other is the computer that has to encode the inference engine from the task description. Thus, our goal is to establish such an appropriate level of description of human problem solving activities (task knowledge) that should be intelligible both to domain experts and the computer.

Based on the above consideration, the authors designed a two-level mediating representation called generic processes and building blocks as shown in Figure 2. The former is for domain experts and the latter is for the system. Correspondence between the two is

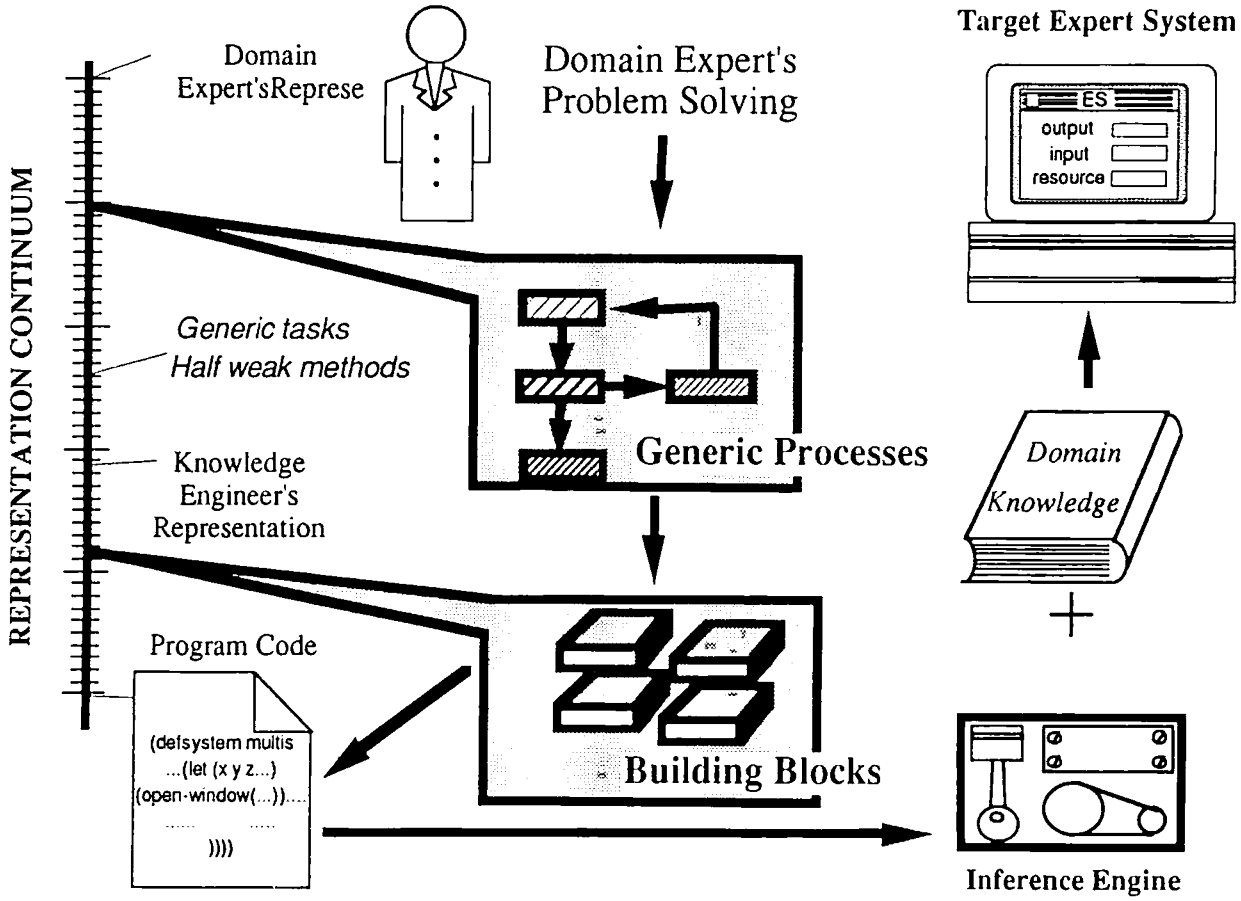


FIGURE 2. Abstraction level of generic processes and building blocks.

arranged in advance. Cases for the case base are represented in terms of the generic processes and other concepts associated with them. Requirements for generic processes are as follows:

(1) Generic processes should be easy for domain experts to understand.

(2) Generic processes should be task specific and domain independent.

And requirements for building blocks are as follows:

(1) It should be easy for the computer to generate codes.

(2) It should be task specific and domain independent.

Next we will explain more about both of these representations.

3.2. Generic Vocabulary

We introduce a new concept called generic vocabulary, which plays a role of basic ontology used for describing generic processes. In order to make the discussion concrete, let us take scheduling ESs as an object of task ontology identification. A scheduling problem can be viewed as a determination task of time and resource (RSC) assignment to schedule recipients (RCP) under the condition that the assignment satisfies the constraints given while optimizing the goal concerned. Note here that the last statement is completely domain-independent. Furthermore, there includes concepts such as RCP, RSC, time, and assignment that play a crucial role in task ontology. Figure 3 and 4 show generic vocabulary identified thus far consisting of generic verbs, generic nouns, generic adjectives, constraint-related vocabulary, and goals. Generic vocabulary acts as a mediating representation, which fills the understanding gap between the domain experts and the computer. Those terms shown in the figures are easily understood by domain experts. In the case of scheduling, the experts have the concept of what they are scheduling (which we call schedule recipient), what to assign to them (which we call schedule resource), under which scheduling conditions are constraints made, how much of the constraints can be relaxed tolerance, and so on. These generic nouns thus indicate important concepts in scheduling tasks. In a similar manner, generic verbs represent fundamental and common activities in scheduling tasks. In the following subsections, we discuss each of them in turn.

3.2.1. *Generic Verbs.* Generic verbs are the ones representing the primitive activities in problem solving processes. In scheduling tasks, assign and select are typical ones. Figure 4 shows generic verbs identified

- Generic verb: (See Fig. 4) (26 in total)
- Generic nouns(40 in total)
  - Schedule Recipient: RCP
    - RCP-GRP
  - Schedule Resource: RSC
    - RSC-GRP
  - Schedule
    - Scedule, Subschedule, Intermediate solution, Final solustion, etc.
  - Schedule representation
    - Gantt chart, Time table, etc.
  - Constraint, Goal, Priority, Data/Information
- Generic adjective, 11 in total
  - Unassigned, Previous, Last, Next, Satisfaying, Violating, etc.
- Constraint-vocabulary(67 in total)
  - 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 in total)
  - Status
    - Maximum, Minimum, Uniform, Continuous
  - Object
    - Load balance, Rate of operation, Efficiency, Idle time, Operation time, etc.

FIGURE 3. Generic vocabulary for scheduling.

for scheduling. They consist of 17 RCP/RSC verbs that mainly take RSC or RCP as their objects, and eight constraint verbs that take constraints.

RCP/RSC verbs represent main activities in the course of problem solving. They are organized according to the generate and test and modify paradigm as shown in Figure 4. For example, typical control flow in scheduling tasks are ''First Pickup an RCP and Select appropriate RSC to the RCP, then Assign the RSC to the RCP. The assignment is checked against the constraints. If constraint violation occurs, then reassignment is performed.''

Constraint verbs represent the activities treating constraints that are also important in scheduling tasks. Typical examples are the Neglect constraint and Relax constraint, which greatly affect the control flow as the RCP/RSC verbs. Every generic verb has at least one computation mechanism called a building block, which is used for problem solving engine generation and discussed in a later section.

- 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 groups
    - 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

FIGURE 4. Generic verbs for scheduling.

3.2.2. *Generic Nouns.* Generic nouns are roughly defined as terms used in conjunction with generic verbs. As described earlier, typical generic nouns are RCP, RSC, and constraint. While instances of RCP and RSC are domain concepts, they are independent of a particular domain and are considered as task-specific concepts. Generally speaking, which domain concept is considered as an instance of which of the two generic nouns, RCP or RSC, is not dependent on the nature of the domain concepts but on the human scheduler's convenience. The only exception of this is job, lot, or order. They are always considered as RCP. For this reason, we organized domain concepts such as job/lot/order, machine, human resource, transportation vehicle, and place separately from generic nouns. Due to the space limitation, description of domain concepts is omitted in this article.

Because instances of RCP and RSC are frequently treated as groups of objects, they have RCP-GRP and RSC-GRP as their subclasses, respectively. The other generic nouns are constraint, goal, priority, schedule, and solution representation. Gannt chart is a typical solution representation used in the job shop scheduling tasks.

3.2.3. *Generic Adjective.* Most of the modifiers appearing in the sentences describing scheduling tasks are represented using constraints that refer to the domain-dependent knowledge. However, we can find domain-independent modifiers that we call generic adjectives. Examples of them are last, next, unassigned, temporal, and so on.

3.2.4. *Constraint-Related Vocabulary.* Needless to say, constraints play essential roles in scheduling tasks. Top level subclasses of constraint-related vocabulary are constraint/condition, constraint adjective, constraint predicate, and attribute. Constraint/condition has two subclasses such as strong constraints, which must be satisfied, and weak ones, which are desirable to be satisfied. Constraint predicates are used to define constraints and include all the predicates defined in temporal logic for describing relations between time intervals. Attributes are components of constraints. Typical examples of them are load, efficiency, frequency, cost, time point, time interval. Time-related concepts play particularly important roles, because many of the constraints are related to time in scheduling. Due date is one of the most important concepts in the time points. Starting and ending time points of RCP are also crucial.

3.2.5. *Goal/Preference.* Problem solving process of scheduling depends largely on goal or preference defined in the problem, which shows goal and preference can characterize the scheduling task. They consist of pairs of an object and its status to satisfy in which the objects are the abovementioned attributes and status is maximum, minimum, uniform, and so on. It is sometimes possible to identify the correlation between some portion of task flow and goal/preference of the task. For example, when the goal is to make the load of a certain resource uniform, the solution process usually includes computation of the load, sequencing the RSC in the order of the load followed by the assignment of the RSC to RCP in this order in the generation phase, and exchange of the RSC of maximum load with one of minimum load in the modification phase. Thus, chunks of control related to respective goals facilitate the reusability of knowledge level parts.

Pickup-RCP
Select-RSC
Assign-RCP-to-RSS
Update-priority

Classify-RCP
Sequence-Rsc
Evaluate-condition
Relax-constraint
....

FIGURE 5. Generic processes for scheduling.

3.3. Generic Process and Generic Process Network

Generic processes are represented in terms of generic vocabulary as follows: generic process = generic verb + generic noun. Typical examples include Pickup-RCP, Select-RSC, Assign-RSC-time-to-RCP, Update-priority, Relax-constraint, and so on. Problem solving process of domain experts is described in terms of generic processes and the result is configured as a network of generic processes, which is called the generic process network (GPN). A GPN can be thought of as task flow defined in terms of reusable components describing meaningful stages of the problem solving process. Figure 5 and 6 show some examples of generic processes and a GPN of a 24-hour operator allocation task, respectively. The task consists of allocating all the operators to one of the four job types (night duty, seminight duty, day duty, and off duty) satisfying the allocation requirements under several constraints and minimizing the load deviation among the operators. The rough image of the solution process is as follows:

(1) Classify the jobs necessary during the time span under consideration according to the job types.
(2) Sequence the job groups in the order of priority (load of the duty types).
(3) Pickup a job type.
(4) Pickup a job from the job type.
(5) Select an appropriate operator available.
(6) Assign the job to the operator.
(7) Update the load information of the operator.
Steps 4 through 6 are repeated until all the jobs are tried in the job type.
(8) Select an operator of maximum load.

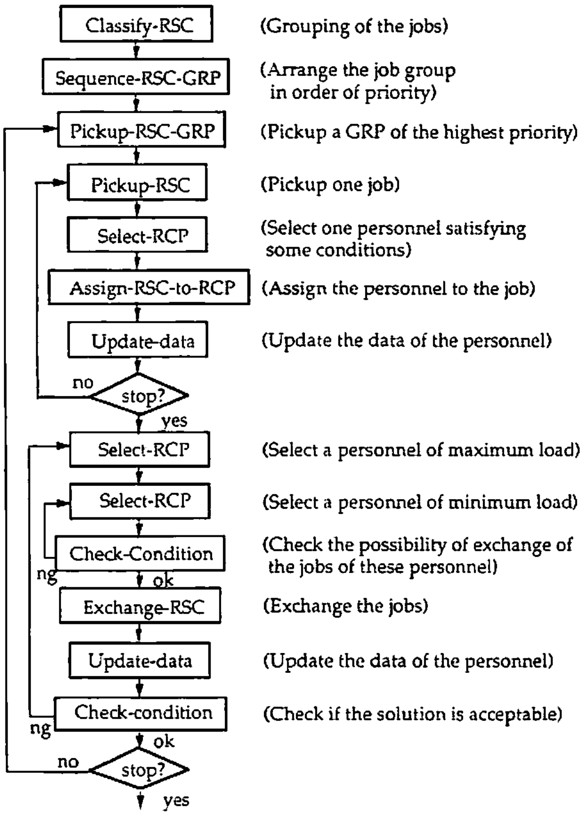

Classify-RSC
(Grouping of the jobs)
Sequence-RSC-GRP
(Arrange the job group in order of priority)
Pickup-RSC-GRP
(Pickup a GRP of the highest priority)
Pickup-RSC
(Pickup one job)
Select-RCP
(Select one personnel satisfying some conditions)
Assign-RSC-to-RCP
(Assign the personnel to the job)
Update-data
(Update the data of the personnel)
no
stop?
yes
Select-RCP
(Select a personnel of maximum load)
Select-RCP
(Select a personnel of minimum load)
Check-Condition
(Check the possibility of exchange of the jobs of these personnel)
ng
ok
Exchange-RSC
(Exchange the jobs)
Update-data
(Update the data of the personnel)
Check-condition
(Check if the solution is acceptable)
ng
ok
no
stop?
yes

FIGURE 6. A generic process network for 24-hr personnel allocation.

(9) Select one of minimum load.

(10) If the exchange of the two is possible, then exchange them.

(11) Update the load information.

(12) Repeat the steps 8 through 11 until the goal minimizing the deviation of the load of operators, is satisfied.

(13) If the job types are exhausted, then terminate. Otherwise, go to step 3.

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, which is programmed by connecting icons of generic processes to each other via the mouse or any other pointing device. The internal representation of GPN is used by the building block compiler (BBC) to generate code of the problem solving engine corresponding to the GPN.

As mentioned earlier, GPNs are stored in the case base for reuse in the development process of future ESs. It is critical for knowledge reuse to clearly discriminate between task-dependent knowledge and domain-dependent knowledge. A GPN is a skeleton of an ES engine and is organized independently of the domain to maximize its reusability while maintaining the task-specific structure of the original ES. These characteristics help minimize the size of the case base. One can reuse the whole GPN, portions of it (such as generator, tester, or modifier), as well as each generic process contained in it. Taking generic vocabulary into account, the reusable objects are organized as a four-level hierarchy in our system.

3.4. Building Blocks and Building Block Networks

Building blocks are symbol level parts readily used as components of problem solving engines of ESs and are realized as abstract programs obtained by analyzing generic processes with respect to the input/output relations, data structure, and knowledge source required. By knowledge source, we mean constraints or conditions necessary in a building block for its execution. Knowledge source is the very domain-dependent knowledge detached from the task knowledge. Some typical examples of building blocks are presented in Figure 7. Generally speaking, every generic verb is indexed to more than one building block. As Figure 7 shows, Select has at least eight building blocks associated with it according to the characteristics of knowledge source. SelectMaxAll, for example, makes a list of all the objects of maximum value obtained by the execution of criterion function and Sequence has two variants depending on whether the given ordering information is total ordering or partial ordering. A rule interpreter can be a building block for every generic verb because it may require a complicated heuristic search for performing the task.

A building block network (BBN) is a network of building blocks representing the problem solving engine of an ES. When composing a BBN, one has to select appropriate building blocks. The selection is not as difficult as it might seem at first because it is determined by the characteristics of the knowledge source, which are easily obtained from the expert through interview.

3.5. 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 to acquire. It is not easy for domain experts to separately articulate these two kinds of knowledge. In our research on task ontology, we are aiming toward realization of clear discrimination between the two in order to make it easier to acquire knowledge from domain experts. One can see this goal is successfully realized in our system from the above discussion.

Another of our principles 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 usually not easy to cope

Assign: make a list of RSC, time, and RCP; (RSC, time, RCP)
input: two objects, Obj1, Obj2 and Obj3
output: (Obj1 Obj2 Obj3)
Select:select object satisfying consition
input: list
output: atom or list
knowledge source: condition, 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
Pick-up: 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
clssifyRule: 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

FIGURE 7. Some examples of building blocks.

with this problem, the above principle helps us identify the right size of building blocks. Furthermore, it makes it easier to acquire domain knowledge because interview could be well-focused 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 because necessary information is given as input information. They are primitive building blocks in this sense.

(2) Select and Evaluate require criterion functions, conditions, or a set of rules for evaluation. Classify requires decision trees, distance or similarity functions, or a set of rules. These knowledge sources partly depend on what object is processed by the building block. For example, Select-RCP in the above 24-hour personnel allocation task, that is, selection of a personnel who has minimum load allocated thus far, requires a criterion function that calculates the load of personnel as a knowledge source. Thus, each knowledge source is specific to a particular object and activity, which shows it satisfies the above principle.

4. MULTIS

This section is concerned with how task analysis is done by MULTIS, which is currently developed using scheduling tasks as examples.

4.1. Task Analysis Methodology

The basic idea used in MULTIS is that a problem solving engine of an ES under consideration can be synthesized from prefabricated parts. For this purpose, it has a library of building blocks corresponding to the components of the problem solving engine. The mechanism for the synthesis is described in section 4.3. One of the major issues in our research is how to enable domain experts to synthesize problem solving

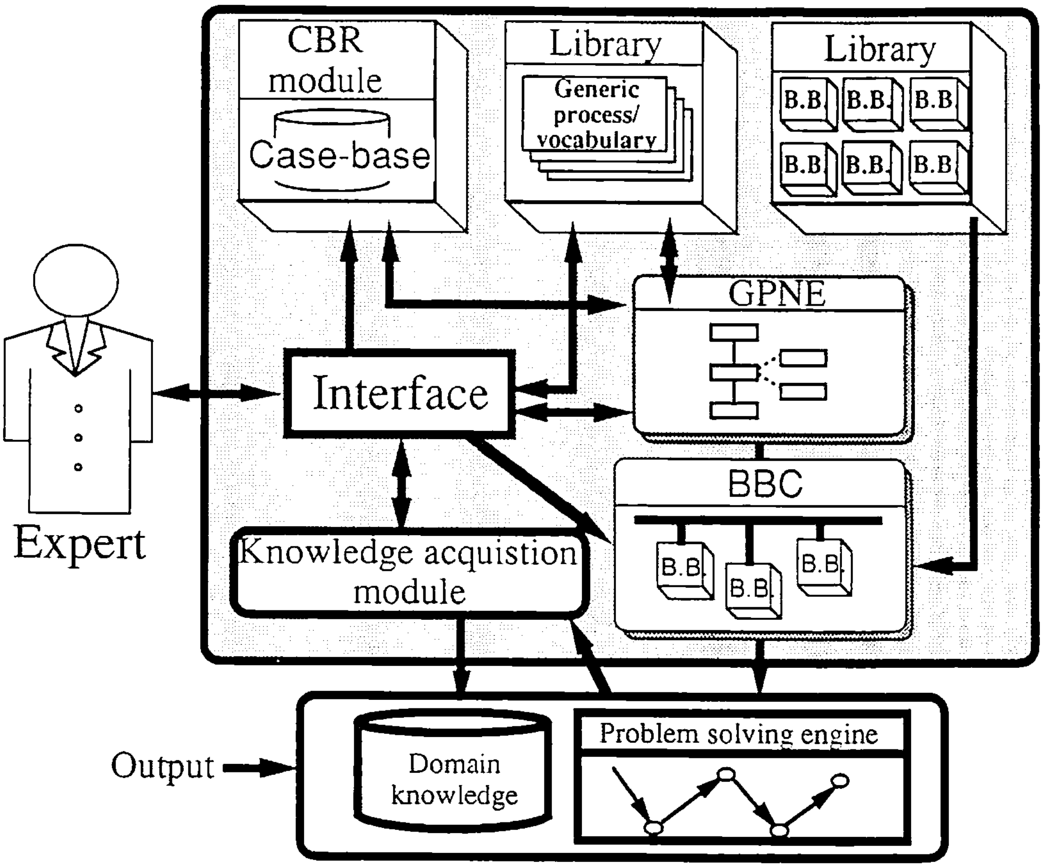


FIGURE 8. Block diagram of MULTIS.

engines for their tasks. Generic vocabulary and generic processes are designed to this end. They are easy for domain experts to understand, so it becomes a relatively easy job to describe their problems in terms of generic vocabulary and generic processes.

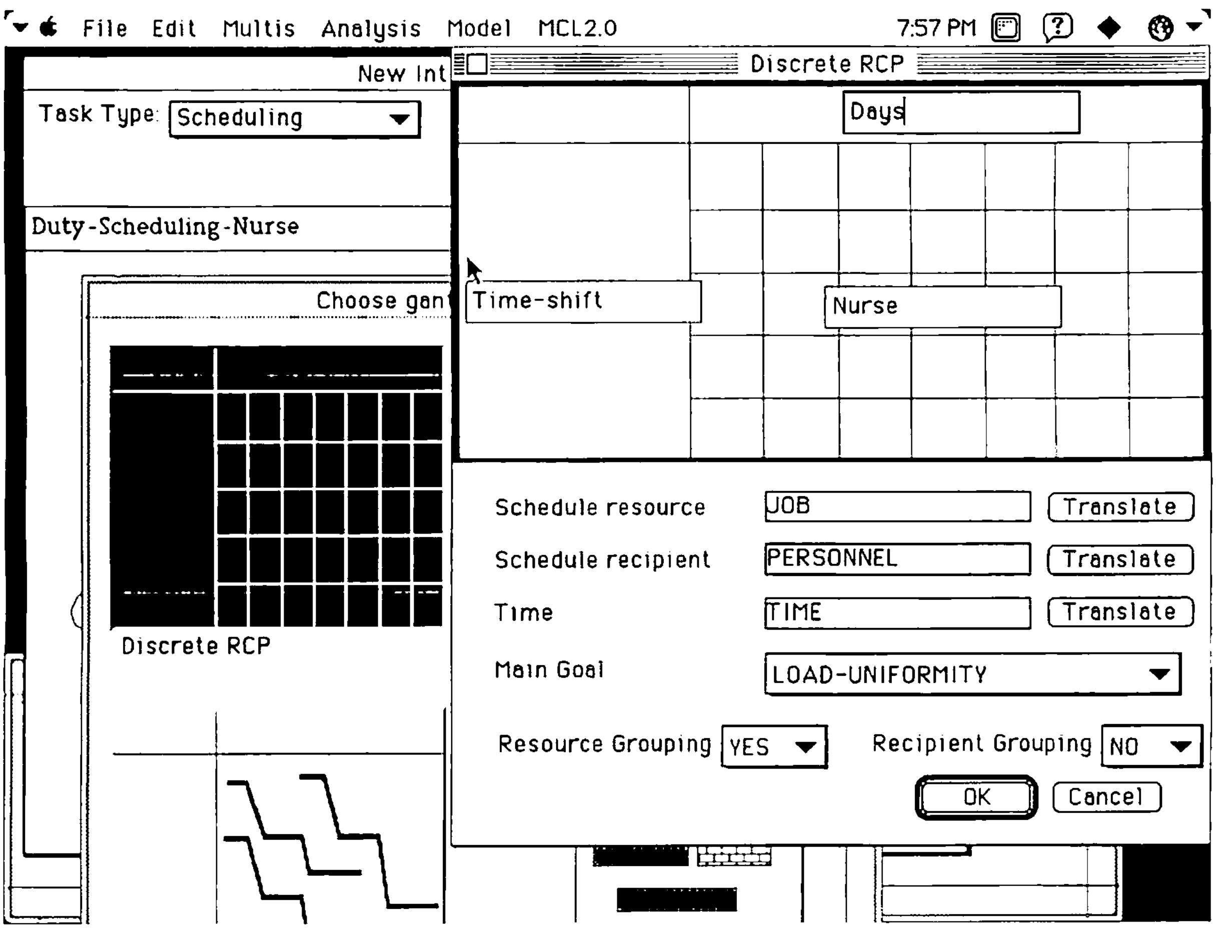
However, even if the vocabulary the interview system uses is intelligible to domain experts, it is not easy for them to write a control structure from scratch. MULTIS uses case-based reasoning (CBR), which presents the domain expert appropriate cases of several ESs' control structures described as GPNs (see Figure 7). 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. GPNs 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 RCPs, and (5) time axis.

MULTIS can present the 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 a third one in terms of the domain concepts under consideration—because 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 obtain their own network.

4.2. The Architecture

Figure 8 shows the block diagram of MULTIS, which consists of seven modules: CBR module, generic vocabulary/process library, building block library, generic process network editor (GPNE), BBC, knowledge acquisition module, and interface module. GPNE builds a GPN through interaction with a domain expert. During the process, it retrieves several cases from the case data base and consults the generic vocabulary library to translate the generic vocabulary contained in the cases into some domain specific concepts. After building GPN, control is passed to BBC, which identifies building blocks necessary for implementing inference engine through interview. This process is described in the next subsection.

MULTIS is implemented in Macintosh Common Lisp. It first tries to obtain several basic domain-specific vocabulary from the solution representation. The domain expert selects from the several sets of solution representations presented such as gantt chart, time table, and so on which are typical representations of scheduling results. The rows, columns, and entries of



Task Type: Scheduling

Duty-Scheduling-Nurse

Days

Time-shift

Nurse

Schedule resource: JOB

Schedule recipient: PERSONNEL

Time: TIME

Main Goal: LOAD-UNIFORMITY

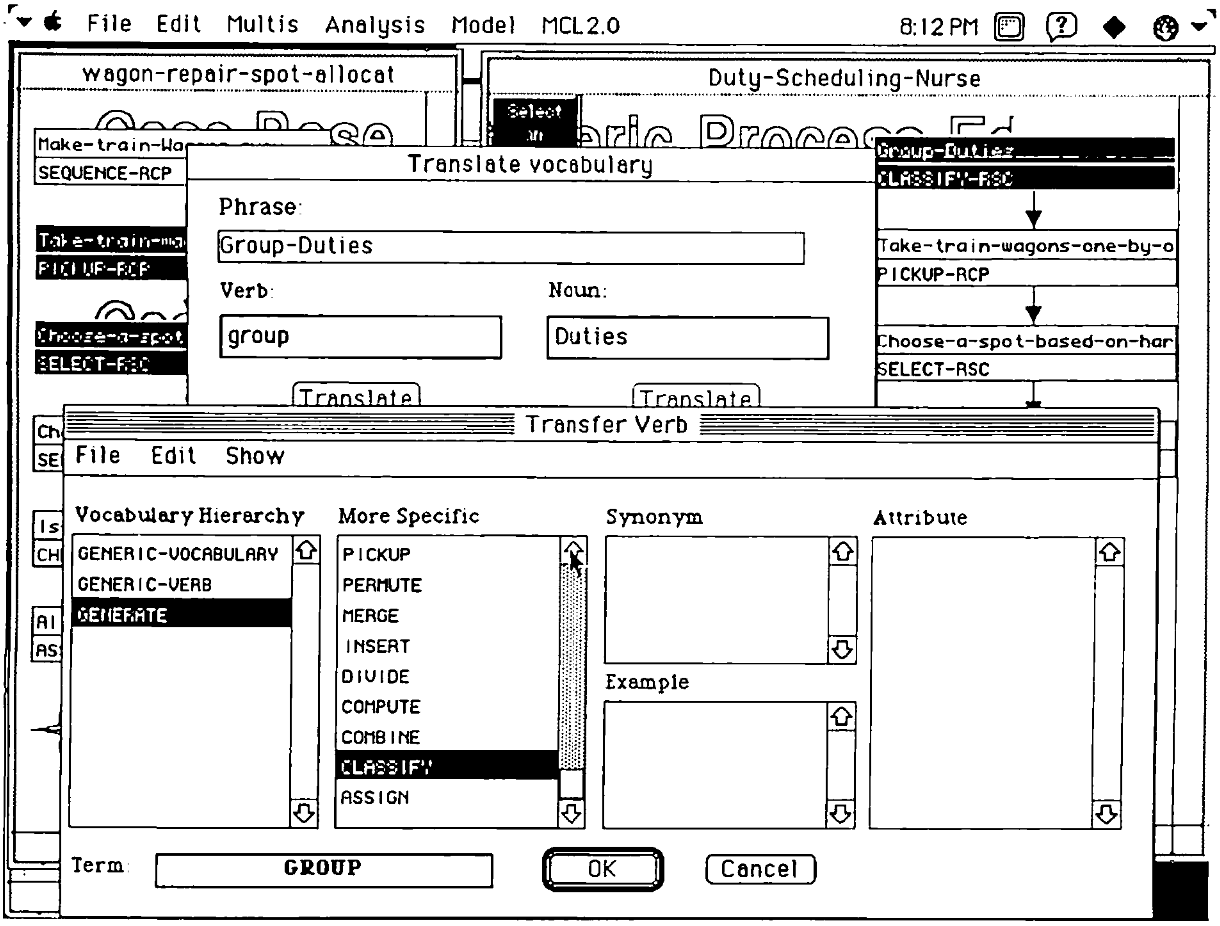
Resource Grouping: YES

Recipient Grouping: NO

FIGURE 9. A screen dump of initial interview by MULTIS.

a solution representation corresponds to time, schedule resources, and schedule recipients, respectively. Thus, the domain-specific terms of these three concepts are easily obtained through a simple interaction as shown in Figure 9. Suppose the ES under consideration is concerned with duty-allocation for nurses, for example. In Figure 9, the left lower-layer window shows the typical solution representations given by MULTIS, the shaded window is selected by the user, and the upper-right window shows the interaction for getting those three domain terms. After obtaining these terms and some other information about treatment of the recipient and resource, MULTIS retrieves several cases from case DB, and interviews with the domain expert to get other information useful for identifying the control structure of the ES.

Figure 10 shows a screen dump of MULTIS interview for GPN editing. The left lower-layer window (case base window) shows a case GPN retrieved. The user can get explanations of the whole process or every GP of GPN shown on the window and reuse arbitrary parts of the it by cut and paste operation using mouse to compose his/her own GPN. The shaded two processes are reused in the GPNE as shown in the right lower-layer window. When the user would like to know the meaning of the generic vocabulary, a browser can be used for generic vocabulary inspection. The browser provides the user with the facility to browse the hierarchy of vocabulary and its examples by which the user easily understands the vocabulary. When the user composes a GPN, he/she first states fragments of the processes in their own terms. In the figure, the user tries to describe their first process named Group-Duties (shaded process in the window). After that, MULTIS requires him to describe the processes in terms of generic vocabulary. During the translation, he consults the browser mentioned above. The middle lower-layer window shows the translation process of the first process with the help of the browser shown in the middle lower window in which his domain-specific verb "group" is translated into generic verb "classify" which is a subclass of GENERATE, which is a subclass of GENERIC-VERB. During the translation, the user easily gets explanation and examples of every vocabulary. Similarly, he translates the noun "duties" into "RSC:resource" as shown in the right lower-layer window. When the domain expert meets a difficulty in understanding the case GPN, MULTIS translates the vocabulary into the terms familiar to the user. Since MULTIS has several domain terms obtained from the



Phrase: Group-Duties

Verb: group

Noun: Duties

Term: GROUP

FIGURE 10. A screen dump of MULTIS interview for GPN construction.

interaction thus far, it can replace the generic vocabulary in the GPN with the domain terms the user uses and show the newly instantiated GPN from which the user easily understands what GPN means and how to modify it to obtain their own GPN.

4.3. Procedure for Generating a Problem Solving Engines With the BBC

This section presents how problem solving engines are synthesized from a GPN. As mentioned above, MULTIS has relationships between generic processes and building blocks. The outline of how MULTIS generates code for an inference engine from the GPN is as follows:

Step 1: Build a skeletal structure of the engine. Each generic process is considered as a function call of the corresponding building block. The order of function calls is arranged as that shown in the GPN.

Step 2: Select appropriate building blocks for each generic process. Interview for the selection of building blocks is performed according to the strategies embedded in respective building blocks.

Step 3: Identify the input and output relations for each building block. Connectivity between building blocks is checked here.

Step 4: Acquire necessary information for each building block such as variable names, attribute of interest, conditions, and so on.

Step 5: Interview of how to back track when an impasse takes place.

Step 6: Generate the codes.

5. CONCLUSION

We have discussed a two-level mediating representation for a task analysis interview system MULTIS. Generic vocabulary and generic processes have been discussed along with examples and their usage in task analysis interview. The generic vocabulary and generic processes have been evaluated through description of nine scheduling ESs presented in literature and proved to have sufficient expressiveness.

One can find a similarity between generic tasks proposed by Chandrasekaran (1986) and our generic process. But the concept of generic tasks is one-level representation and it has no generic vocabulary. Although it is useful as a conceptual parts of inference engines, it cannot be used as a mediating representation. KADS (Wielinga, 1992) and Spark/Burn/FireFighter (McDermott, 1990) have also some ideas similar to ours. Although metaclass and knowledge source in KADS corresponds to generic nouns and generic verbs, they are not tailored enough as task ontology. In summary, they are less task-dependent than MULTIS, which might make the interview less efficient.

The authors have set up a consortium for developing and evaluating MULTIS in ASTEM/RI in cooperation with eight companies. In the consortium, MULTIS has also been evaluated by using several ESs the members were involved in the development, in which our generic vocabulary is shown to have sufficient expressive power as task ontology for scheduling tasks. Future plans include the augmentation of case data base, formalization of generic vocabulary and generic processes, evaluation of interview behavior of the current system, and extension of this approach to other types of ESs.

Finally, we would like to discuss the generality of MULTIS. Note that MULTIS is designed in a highly modular structure. The three libraries for generic vocabulary, GPN, and building blocks are dependent on the target task under analysis but they are passive data referred to by all the modules independent of the target task. This architecture makes MULTIS very general and applicable to analysis of other types of tasks such as design and diagnosis by substituting the three libraries. The only exception to this generality is the interface module part of which is dependent on solution representation of scheduling, that is, gannt chart or time table. One has to modify this task dependency in order to make MULTIS completely task independent, though it is not difficult.

Although this article has only discussed task ontology of scheduling tasks, we can find ontologies of other tasks such as design, diagnosis, control, etc. As one can see, the scheduling task ontology discussed in this article is not a tough problem to design ones for other tasks. For example, ontology for design task is discussed in Mizoguchi (1993). One of the interesting issues is to compare the difference between various tasks through ontologies identified.

REFERENCES

Boose, J. (1990). Knowledge acquisition tools, methods, and mediating representations. *Proceedings of JKAW90* (pp. 25–62). Kyoto, Japan: JKAW.

Chandrasekaran, B. (1986). Generic tasks for knowledge-based reasoning the right level of abstraction for knowledge acquisition. *IEEE Expert*, 1(3), 23–30.

Clancey, W. J. (1985). Heuristic classification. *Artificial Intelligence*, 27(3), 289–350.

McDermott, J. (1988). Using problem solving methods to impose structure on knowledge. *Proceedings of the International Conference on AI Applications* (pp. 7–11). Tokyo, Japan.

McDermott, J., Dallemagne, G., Klinker, G., Marques, D., & Tung, D. (1990). Exploration in how to make application programming easier. *Proceedings of JKAW90* (pp. 134–147). Kyoto, Japan: JKAW.

Mizoguchi, R., Yamaguchi, T., & Kakusho, O. (1988). On a methodology for building expert systems, SIG-KBS-8802. *Japanese Society for Artificial Intelligence* (pp. 11–22) (in Japanese).

Mizoguchi, R. (1993). Knowledge acquisition and ontology. *Proceedings of KB&KS: Building Large-Scale Knowledge Bases and Knowledge Sharing* (pp. 121–128) 1993.

Musen, M. (1991). Dimension of knowledge sharing and reuse. *Knowledge Systems Laboratory* (Report KSL-91-65). Stanford, CA: Stanford University.

Tijerino, Y., Kitahashi, T., & Mizoguchi, R. (1990). A task analysis interview system that uses a problem-solving model, *Proceedings of JKAW90* (pp. 331–344). Kyoto, Japan: JKAW.

Tijerino, Y., Kitahashi, T., & Mizoguchi, R. (1991). MULTIS: A knowledge acquisition system based on problem solving primitives. *Proceedings of 6th Banff KAW* (pp. 32.1–32.20). Banff, Canada.

Wielinga, B., Schereiber, A. T., & Breuker, J. A. (1992). KADS: A modeling approach to knowledge engineering. *Knowledge Acquisition*, 4(1), 5–53.

Yamaguchi, T., Mizoguchi, R., Tao Ka, N., Kodaka, H., Nomura, Y., & Kakusho, O. (1987). Basic design of knowledge compiler based on deep knowledge. *Journal of Japanese Society for Artificial Intelligence*, 2(3), 77–84.