

Task Ontology and its Use in a Task Analysis Interview System

--- Two-level Mediating Representation in MULTIS ---

Riichiro Mizoguchi, Yuri Tijerino and Mitsuru Ikeda

I.S.I.R., Osaka University
8-1, Mihogaoka, Ibaraki, 567 Japan
Email: miz@ei.sanken.osaka-u.ac.jp

Abstract

Automated knowledge acquisition is an ultimate goal of knowledge engineering. To achieve this we have to solve many problems such as lack of reusability and sharability of knowledge, which is one of the serious shortcomings in the current knowledge base technology, to fill the conceptual gap between the computer and domain experts and so on. The authors have been involved in the research of knowledge acquisition and knowledge reuse. This paper 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 which contributes to bridging the gap. Generic vocabulary and building blocks are the major components of the mediating representation. Finally, we present a rough image of how MULTIS works. MULTIS has been implemented in Macintosh Common Lisp and C.

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, 88][Tijerino, 90][Tijerino, 91], 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, since the expert system(ES) under consideration is identified as one of the pre-stored 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, since 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, 90]. After task analysis, MULTIS generates a problem solving engine for the target ES. This paper presents 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 and C.

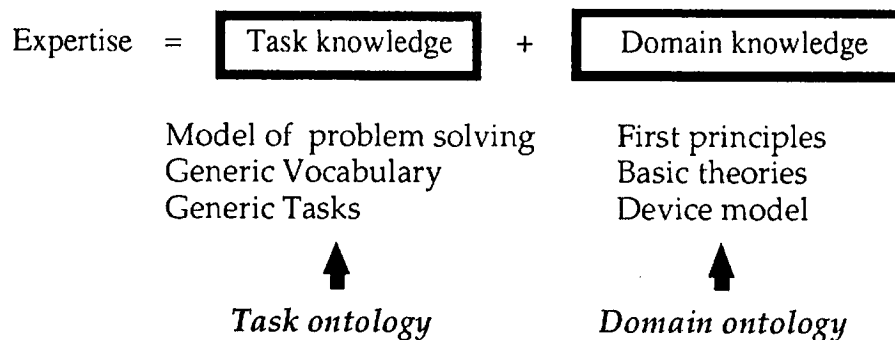


Fig. 1 Expertise decompilation.

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, 91]. 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, 87]. Although knowledge decompilation is a very useful idea and has many implications, we here only discuss task/domain decomposition shown in Fig. 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. 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 paper 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, 86][Clancey, 85][McDermott, 88] 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" which 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 which 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, since 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, since 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 his/her own problem solving processes, and the other is the computer which 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 Fig. 2. The former is for domain experts and the latter is for the system. Correspondence between the two is 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 computer to generate codes.

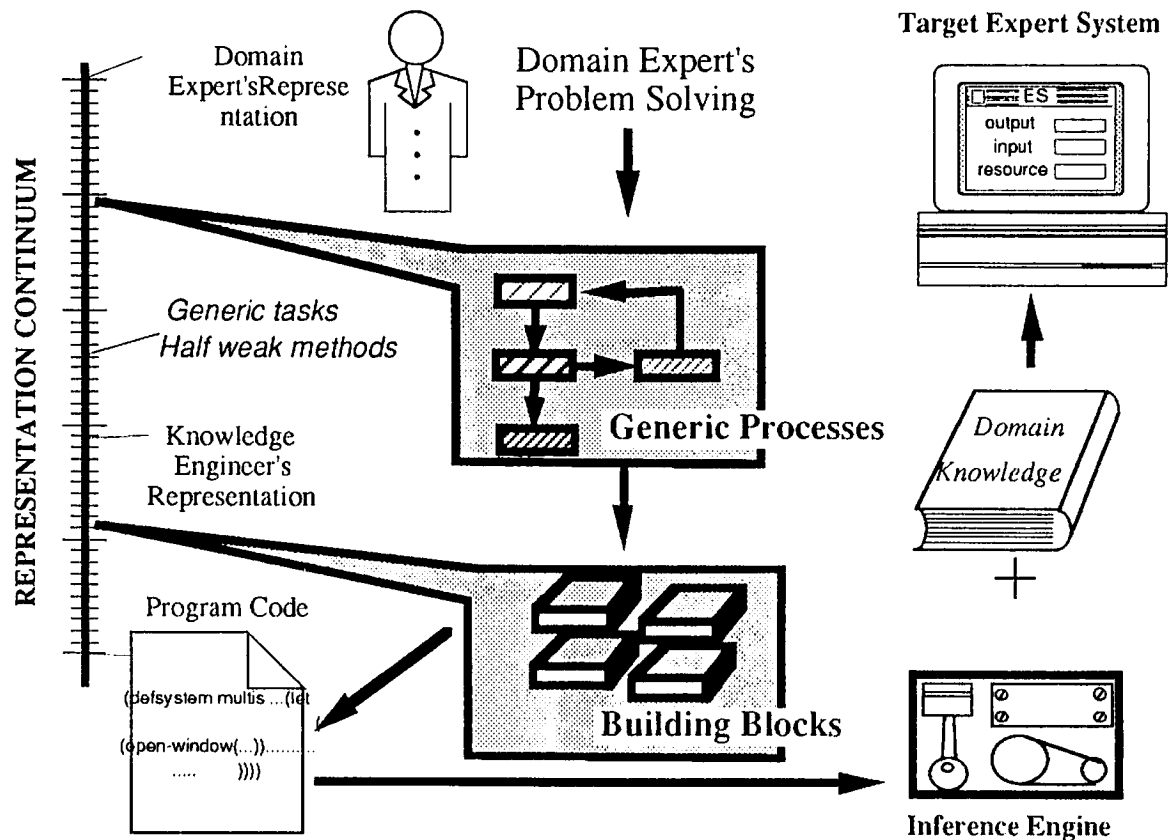


Fig. 2 Abstraction level of generic processes and building blocks.

(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* which play a crucial role in task ontology. Figs. 3 and 4 show generic vocabulary identified thus far which consists 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*, that of what to assign to them which we call *schedule resource*, that of under which conditions scheduling is made *constraints*, that of how much the constraints can be relaxed *tolerance* and so on. These generic nouns thus indicate important concepts in scheduling tasks. In

the 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. Fig. 4 shows generic verbs identified for scheduling. They consist of 17 RCP/RSC verbs which mainly take RSC or RCP as their objects and 8 Constraint verbs which take constraints.

RCP/RSC verbs represent main activities in the course of problem solving. They are organized according to the Generate & Test & Modify paradigm as shown in Fig. 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 which are also important in scheduling tasks. Typical examples are *Neglect* a *constraint* and *Relax* a *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 as discussed on a later section.

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 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 paper.

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*. *Gantt 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 which refer to the domain-dependent knowledge. However, we can find domain-independent modifiers which we call generic adjectives. Examples of them are *last*, *next*, *unassigned*, *temporal* and so on.

3.2.4 Constraint-related vocabulary

Generic verb: (See Fig. 4)

Generic nouns

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

Unassigned, Previous, Last, Next, Satisfaying,
Violating, etc.

Constraint-vocabulary

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

Status

Maximum, Minimum, Uniform, Continuous

Object

Load balance, Rate of operation, Efficiency,

Idle time, Operation time, etc.

Fig. 3 Generic vocabulary 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 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

Fig. 4 Generic verbs for scheduling.

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, since 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 above-mentioned 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 in the modification phase. Thus, chunks of control related to respective goals facilitate the reusability of knowledge level parts.

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 GPN: Generic Process Network. A GPN can be thought of as task flow defined in terms of reusable components that describe meaningful stages of the problem solving process. Figs. 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, semi-night 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.

<i>Classify-RCP</i>	<i>Update-priority</i>
<i>Pickup-RCP</i>	<i>Evaluate-condition</i>
<i>Divide-RCP</i>	<i>Check-condition</i>
<i>Select-RSC</i>	<i>Relax-constraint</i>
<i>Sequence-RSC</i>	<i>Neglect-constraint</i>
<i>Assign-RSC and tiem to RCP</i>

Fig. 5 Generic processes.

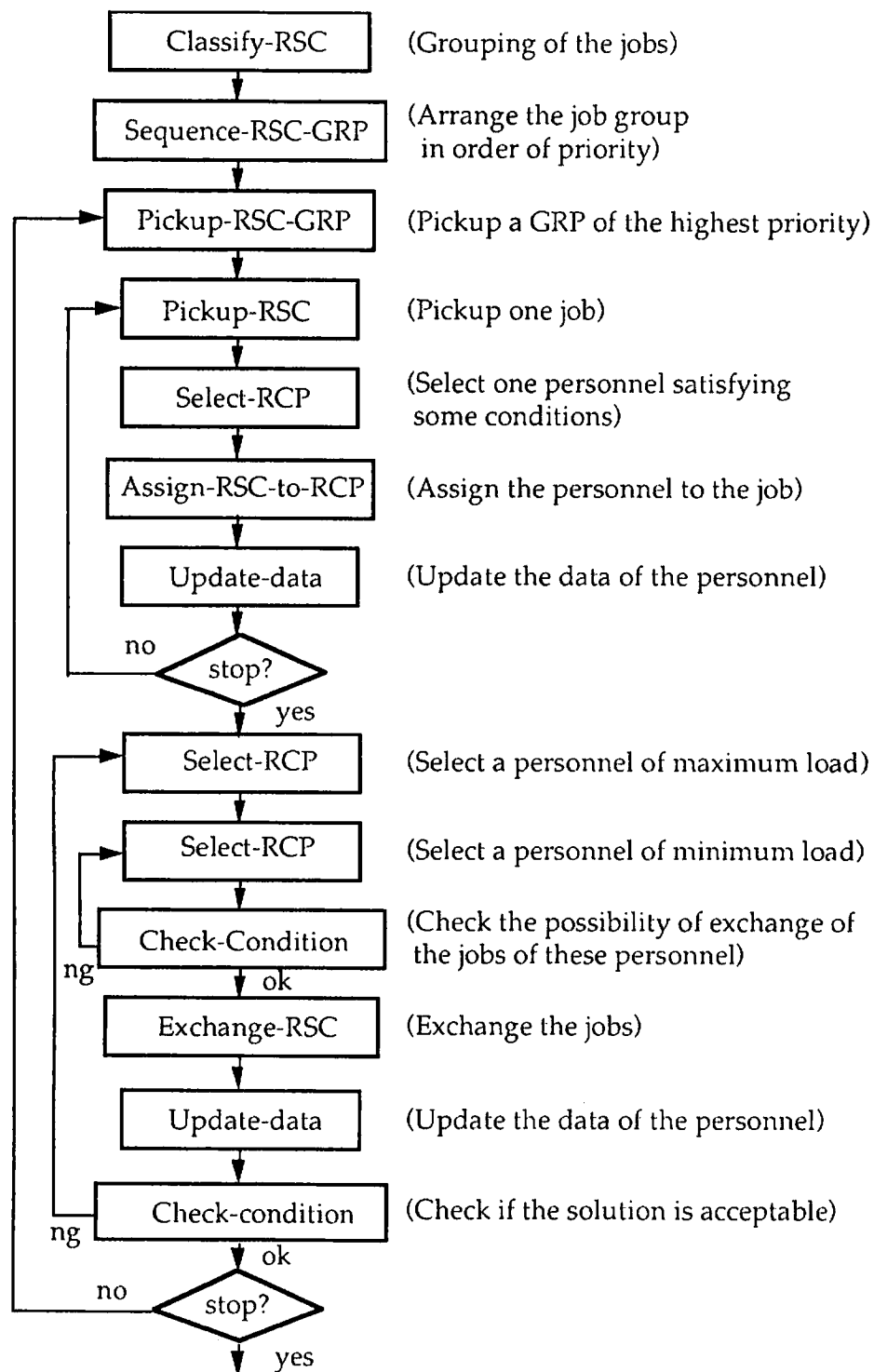


Fig. 6 A generic process network for 24-hour personnel allocation.

(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.

(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, that is, 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 BBC (Building Block Compiler) 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. This 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 Fig. 7. Generally speaking, every generic verb is indexed to more than one building block. As Fig. 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, since it may require complicated heuristic search for performing the task.

BBN: A Building block network 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, since it is determined by the characteristics of the knowledge source which is easily obtained from the expert through interview.

3.5 Task ontology and knowledge acquisition

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 condition
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

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 evaluating 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

Fig. 7 Some examples of building blocks.

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 at 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 with this problem, the above principle helps us identify the right size of building blocks. Furthermore, it makes it 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. 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 which 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 employed in MULTIS is that a problem solving engine of an ES under consideration can be synthesized from pre-fabricated parts. For this purpose, it has a library of building blocks which correspond 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 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.

Even if the vocabulary the interview system uses is intelligible to domain experts, however, it is not easy for them to write a control structure from scratch. MULTIS employs CBR(Case-Based Reasoning) which presents the domain expert appropriate cases of several ESs' control structures described as GPNs (See Fig. 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 user the cases in several ways—one in terms of generic vocabulary, another one in terms of the domain concepts of the particular domain and

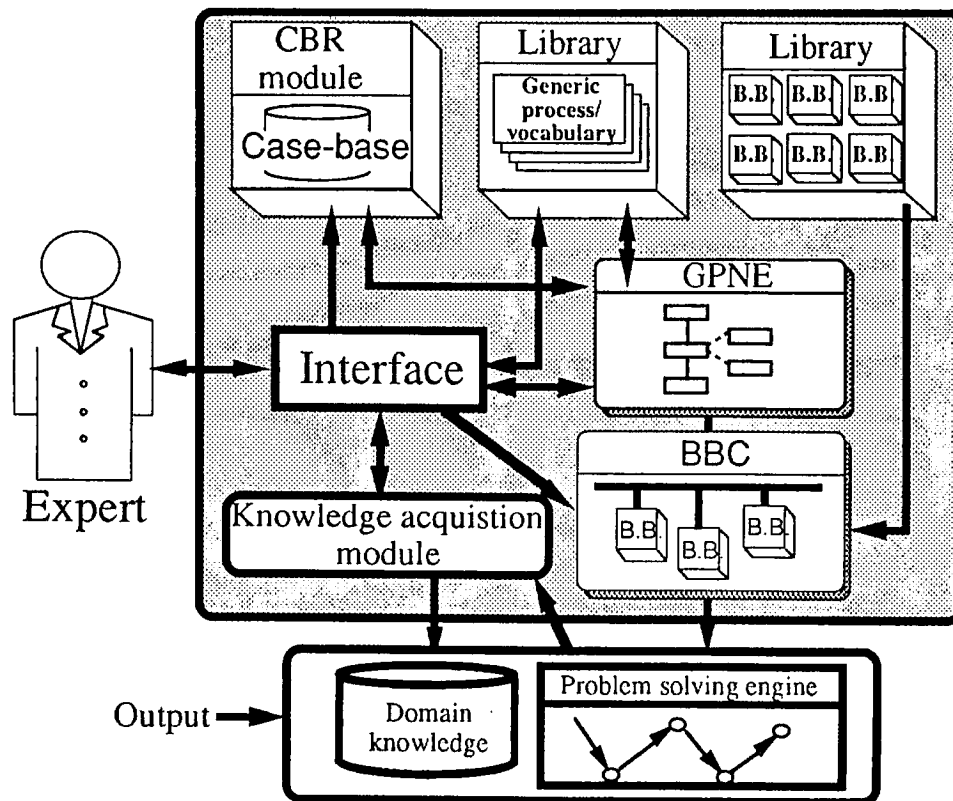


Fig. 8 Block diagram of MULTIS.

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 obtain their own network.

4.2 The architecture

Fig. 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), building block compiler (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 DB and consults 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 first tries to obtain several basic concepts by using the template of a solution representation such as a gantt chart or a time table in which the rows, columns and entries correspond to time, schedule resources and schedule recipients, respectively. Thus, at least these three concepts are easily obtained through a simple interaction. Suppose the ES under consideration is concerned with personnel allocation for 24-hour jobs. In this case, the time table could look as shown in Fig. 9.

	Feb.1st	Feb.2nd	Feb.3rd	Feb.4th
night duty	John	Tom	Bob	Judy
semi-night duty	Geof	John	Susan	Tom
day duty	Tom	Bob	Susie	Beth
off duty	Judy	Geof	Beth	Tom

Fig. 9 An example of a time table of personnel allocation.

From Fig. 9, MULTIS can obtain the concept of "job type" by asking question for example, "What do you call 'night duty,' semi-night duty,' 'day duty' and 'off duty' collectively?" Then, it comes to know the correspondence between schedule-resource and "job type" and that they have four instances. Similarly, it comes to know schedule-recipient is called by the domain expert "personnel" and their instances. After obtaining these concepts in addition to the information of indexes for case retrieval, MULTIS retrieves several cases from the case DB, and interviews with the domain expert to get other information useful for identifying the control structure of the ES. When the domain expert meets a difficulty in understanding the case GPN, MULTIS translates the vocabulary to the concepts familiar to him/her.

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 [Chandrasekaran, 86] 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, 91] and Spark/Burn/FireFighter[McDermott, 90] have also some ideas similar to ours. Although meta-class 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 DB, formalization of generic vocabulary & generic processes, evaluation of interview behavior of the current system and extension of this approach to other types of ESs.

References

- [Boose, 90] Boose, J.: Knowledge acquisition tools, methods, and mediating representations, Proc. of JKAW90, pp.25-62, 1990.
- [Chandrasekaran, 86] Chandrasekaran, B.: Generic tasks for knowledge-based reasoning: the right level of abstraction for knowledge acquisition, IEEE Expert, 1, pp.23-30, 1986.
- [Clancey, 85] Clancey, W.J.: Heuristic classification, Artificial Intelligence, 27, 3, 1985.
- [McDermott, 88] McDermott, J. : Using problem solving methods to impose structure on knowledge, Proc. of the International Conf. on AI Applications, pp.7-11, 1988.
- [McDermott, 90] McDermott, J. et al.: Exploration in how to make application programming easier, Proc. of JKAW90, pp.134-147, 1990.
- [Mizoguchi, 88] Mizoguchi, R. et al.: On a methodology for building expert systems, SIG-KBS-8802, Japanese Society for Artificial Intelligence, pp.xxx-xxx, 1988.
- [Musen, 91] Musen, M. et al.: Dimension of knowledge sharing and reuse, Knowledge Systems Laboratory, Report KSL-91-65, 1991.
- [Tijerino, 90] Tijerino, Y. et al.: A task analysis interview system that uses a problem-solving model, Proc. of JKAW90, pp.331-344, 1990.
- [Tijerino, 91] Tijerino, Y. et al.: MULTIS: A knowledge acquisition system based on problem solving primitives, Proc. of 6th Banff KAW, 1991.
- [Yamaguchi, 87] Yamaguchi, et al.: Basic design of knowledge compiler based on deep knowledge, Journal of Japanese Society for Artificial Intelligence, Vol.2, No.3, pp.77-84, 1987.
- [Wielinga, 92] Wielinga, B. et al.: KADS: A modeling approach to knowledge engineering, Knowledge Acquisition, Academic Press, Vol.4, No.1, pp.5-53, 1992.