Task Ontology for Reuse of Problem Solving Knowledge

Riichiro Mizoguchi, Johan Vanwelkenhuysen, and Mitsuru Ikeda

ISIR, Osaka University

Osaka, Japan

miz@ei.sanken.osaka-u.ac.jp

ABSTRACT

This paper presents a task ontology which plays a crucial role in reusing expertise. We first discuss a few basic issues concerning ontology for knowledge reuse. The first issue is typology of ontology stressing that there is more than one kind of ontology. The second topic is context. In order to enable knowledge sharing and reuse, knowledge should be decompiled into context-dependent and context-independent parts. In problem solving knowledge, the context is determined mainly by task structure. Based on this observation, we propose a task ontology which contributes to making the problem solving context explicit. After discussing these issues we show how to design task ontology. Finally, an overview of a task analysis interview system, MULTIS, based on task ontology is presented to illustrate how an implemented task ontology is used.

1 INTRODUCTION

Ontology is expected to play an important role in knowledge sharing and reuse[16]. However, it seems to the authors that there is no consensus on what an ontology is among the researchers. Although a number of discussions have been made on this topic, we can find difference between them. The first topic discussed in this paper is what an ontology is. The main point is that there is not only one kind of ontology. Typology of ontology is a crucial issue to discuss, though many people are ignorant of this very important point which helps make the discussion on ontology clearer.

The second issue is context. That is, what context should be taken into account when designing ontology. Does knowledge exist independently of any context? Is domain ontology or ontology of some object a candidate which can exist and be recognized independently of context? Our answer is no. There are several kinds of ontologies according to the context given. Explicit representation of the context is therefore critical to ontology design. After the context is made explicit, one can design ontology. From knowledge base technology perspective, knowledge should be considered in some context, that is, in problem solving situation. In contrast to a data base which is usually designed application-independent, a knowledge base is always application-specific in its nature.[1] Consequently, to the extent that we are interested in knowledge rather than data bases, the investigation of ontologies taking into account the context of knowledge application is a necessity.

Then, how to make the context explicit? Our claim is the context can be specified by making problem solving process and its environment explicit. This suggests the importance of task ontology which specifies problem solving process and workplace ontology[22] specifying the environment where the problem solving process is embedded. The former is discussed in this paper and the latter in the accompanying paper[23]

Another issue to discuss in connection with context is "whether sharing and reuse are the same or not". At least, there are two situations of sharing/reuse of knowledge. We do not care about the difference between the meaning of the terms: sharing and reuse, but think it is very important to distinguish the two situations: One concerned with multiple agents with own knowledge bases who ASK one another to solve some problems and the other concerned with a single agent(not necessarily multiple ones) who USEs knowledge in a knowledge base to BUILD another knowledge base. To investigate these two situations gives us stimulating thoughts useful for ontology design. After

---

[1] Knowledge bases in heuristics-based expert systems are full of knowledge specific to their application tasks.

discussing these general issues, we present an example of task ontology for scheduling tasks and the guidelines to ontology design. Finally, a task analysis interview system named MULTIS is described.

2 WHAT IS AN ONTOLOGY?

2.1 Definition

¿From the engineering perspective, ontology plays an important role in knowledge sharing and reuse. We have been involved in the design of ontology aiming at enabling knowledge reuse. Although we can find literature discussing several types of ontologies, many of the papers discuss it as if there exist only one type of ontology. Or, what kind of ontology is discussed is implicit. This makes the situation rather confusing. We have to be careful when we talk about ontology not to forget to specify what ontology we are discussing. In order to clarify what an ontology is, let us go back to its definition. There are several definitions of ontology.
(1)From Philosophy, "Ontology is a systematic explanation of existence".
(2)From AI perspective,"Ontology is an explicit specification of conceptualization" [2]
(3)From knowledge base perspective, "Ontology is a system of concepts/vocabulary used as primitives for building artificial systems"[12]

In this paper, we adopt the third perspective. In other words, we are interested in knowledge for problem solving rather than knowledge in general.

2.2 What makes ontologies different from each other?

To consider how knowledge is used helps us understand what an ontology is. Let us enumerate what manners exist in using knowledge.
(1) Direct methods:
Imagine someone is trying to reuse knowledge in a knowledge base to build another knowledge base for a different purpose. In such a case, he/she has to investigate carefully the knowledge base to know what knowledge is reusable for his/her objective. The methods used in such cases could be called direct methods, since they directly manipulate knowledge.
(2) Indirect methods:
Indirect methods aim at sharing knowledge through communication among the owners of knowledge. An agent asks another agent to solve some problems according to a specified communication protocol. Therefore, these methods are not interested in what knowledge is used for solving a problem and how it is solved. Instead, they are interested only in which agent can solve the problem.
(3) Case-base methods:
This type of methods try to share knowledge through cases stored in case bases. This approach is based on the observation that sharing each piece of knowledge

in a very large knowledge is difficult. This matches new technologies such as case-based reasoning and memory-based reasoning.

The ontology problem is frequently discussed in the context of knowledge sharing and reuse. And sharability and reusability of knowledge are usually discussed at the same time. Because these two words share a lot, they prevent people from understanding the difference between the above first two uses of knowledge. However, discussion on the meaning of these words themselves is of no importance to our purpose. Rather, we would like to name the above three ways of knowledge use.

Although the first two methods are related to each other, they are very different. The former is interested in the details of knowledge in a knowledge base such as how it is organised and the goal and application condition of each piece of knowledge in it, that is, it treats a knowledge base as a "white box", while the latter considers one as a "black box" and is not interested in how the agent solves the problem but interested only in the answer to the problem. This difference affects very much the concept of ontology. We would like to call the former "REUSE" of knowledge, since it is just the same concept as "Program reuse" in software engineering and the latter "SHARING" knowledge.

Case-base methods do not require detailed analysis of cases but vocabulary for indexing them which determine another ontology. These three views on knowledge use and the role of knowledge bases cause differences in ontologies. Knowledge viewed from the direct methods perspective should be organized around how to use knowledge in what kinds of problem solving context. Viewing knowledge from the indirect method perspective, the ontologies should not be concerned with how to solve problems by each agent. The only thing of importance is how to ask a specific agent to get desirable information. So, the ontology is responsible for making the communication fluently. In a case base, how to retrieve appropriate cases is the main issue. So, ontology is necessary for appropriate characterization of cases as indices.

Thus, corresponding to the above three methods, we could list at least three kinds of ontologies which we could call
1)Content ontology for reusing knowledge,
2)Communication(tell&ask) ontology for sharing knowledge, and
3)Indexing ontology for case retrieval.

2.3 Before vs. after ontological commitment

One of the important issues concerning ontology is before/after ontological commitment. This has not been discussed seriously in spite of its importance. After commitment has been made to an ontology, what to do with the ontology is its formal representation which

does not necessarily represent its meaning and syntactic translation from one representation to another. Thus, a language like Ontolingua [2] works well for this purpose.

Before commitment, however, it is not the case, since designers and the potential users have to discuss the meaning of the ontology, otherwise people cannot commit to it. When the ontology is implemented in a computer, not only hierarchical relations but also meaning of each concept has to be represented. In order to represent the meaning of ontology, one needs procedural representation in addition to declarative one. Imagine one tries to build a qualitative model of some device by "reusing" several components, say, pipe, pump, water, etc., defined by another person. Needless to say, these are the basic concepts defined in an ontology of, say, plant domain. In this case, the specifications such as "water is a kind of liquid", "a pipe is a kind of conduit and connected to a pump" are of no use to him/her, since they can nothing to do with the qualitative simulation he/she wants to do. What he/she needs is procedural meaning of them, that is, how the components behave in the simulation. Therefore, when such an ontology is available, he/she does want to know procedural representation of them or procedural semantics of the interpreter which interprets their declarative representation. Thus, "before commitment ontologies" cannot be represented without procedural representation.

Apparently, we are now in the phase of "before commitment", since we have been struggling in designing ontologies to which many people commit. At this very period, to discuss "after commitment ontology" without making the assumption explicit makes the discussion misleading. In this paper, we would like to refer to "Before Commitment ontology" and "After Commitment ontology" as "BC ontology" and "AC ontology", respectively.

2.4 Knowledge level

Another issue to discuss is "knowledge level" which has been playing an important role in various discussions on knowledge representation[13]. The idea of knowledge level also applies to ontology discussion. The knowledge level model of communicating agents suggests that the participants already have made an ontological commitment if their communication is successful[3], in which "tell and ask" are major functions. This situation is depicted in Fig.1. Participants do not need a model of each other because they can communicate according to the protocol determined in advance based on the ontological commitment. This type of communication seems to be reasonable and typical one. However, we can imagine another type of communication among agents(humans) as shown in Fig. 2 where a domain expert and a knowledge engineer are involved in a knowledge acquisition interview.

In this case, ontological commitment is hard to make, since both are not familiar with the other's domain. Furthermore, knowledge engineers are responsible for making the interview fluently by building a model of the domain expert's conceptual representation of expertise. Thus, Communication 1 requires a communication ontology(tell&ask ontology), while Communication 2 requires a content ontology. The readers may notice the knowledge level of Communication 2 is viewed as the knowledge use level [15][21].

2.5 Is ontology like a conceptual schema?

Ontology is often explained by indicating an analogy to conceptual schema in a relational data base[2]. This analogy captures a portion of what ontology means, i.e., ontology can be partly viewed as a formal specification of conceptualization. But there is an ontology which has more implications than conceptual schema. That is, it gives not only specification of the content of knowledge but also defines meaning of the primitive concepts of knowledge. In the communication ontology, the specification ontology may be sufficient. However, content ontology has deeper implications.

In conceptual schema case, one does not have to define what a conceptual schema is or what a relational data base is or meaning of, say, "salary" and "name", since he/she already knows all of them. He/she can use the conceptual schema just using pattern matching. Therefore, one can say conceptual schema is portable. In this sense, however, conceptual schema analogy is somewhat misleading, since it only explains the communication ontology or AC ontology.

For another example, imagine a procedure of solving a set of linear equations. When one wants to share the procedure, specification of its input and output suffices. When one is involved in coding of the procedure, however, specification of input and output does not make sense for him/her. What he/she needs is the solution algorithm, that is, the content of the procedure.

2.6 Is ontology portable?

When one speaks of portable ontology[2], it implies computers can manipulate the ontology as a whole, but the reality is not so. What can computers do with ontology in the destination site? All the programs already existing there do not understand the ontology. It is true every program becomes portable when appropriate language translators are available. The usage of a program is understood just only knowing its function, that is, relations between input and output. In other words, users can use a ported program viewing it as a black box. Therefore, it is easy for users to use the ported program. However, this cannot apply to ontology, since ontology always requires HUMAN INTERPRETATION of its meaning. If one wants to
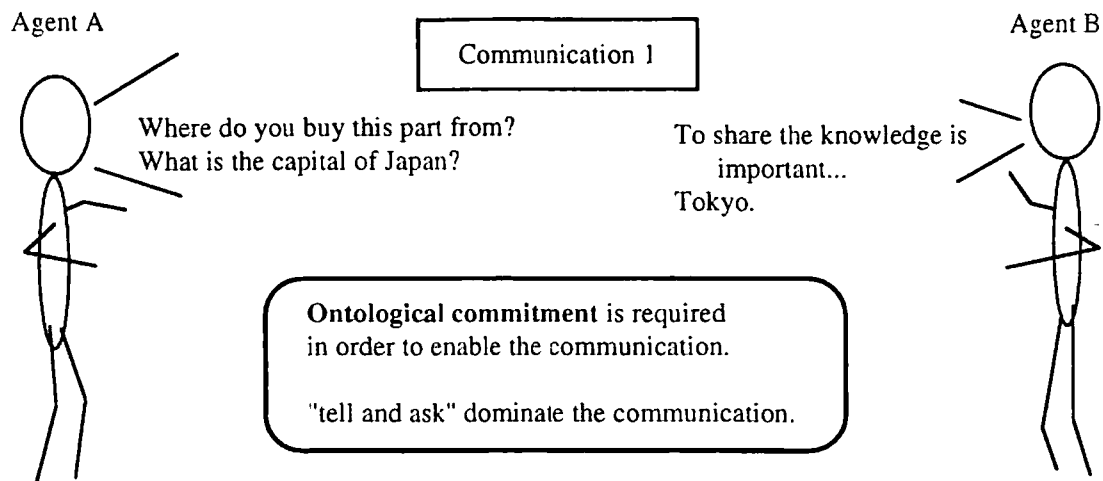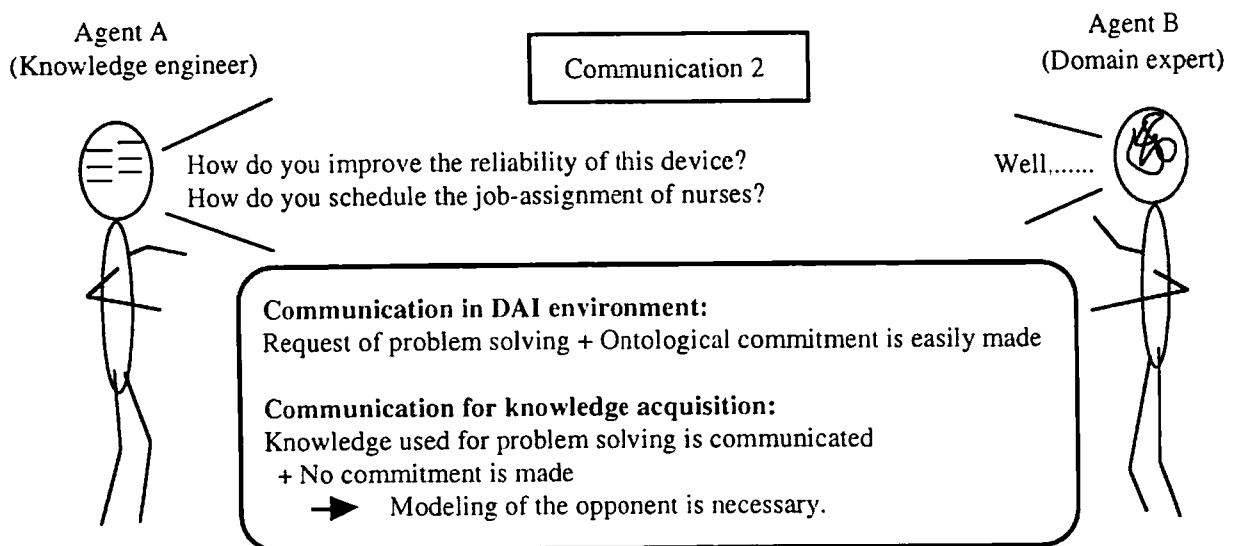
Figure 1: Communication 1(tell&ask).

Figure 2: Communication 2(Knowledge acquisition).

use the ported ontology, he/she first understands it in depth, then he/she codes a new program for utilizing and manipulating the ontology. Thus, all the jobs to be done with ontology is performed not by computers but by humans. This is not the concept of portability.

In general, all the meaning lies in the primitive concepts, the relations and functions between them, and the methods utilizing them. And many of them cannot be represented declaratively. In conclusion, content ontology or BC ontology cannot be portable.

The above discussion is made from the content ontology point of view, that is, not "sharing" but "reuse" view point. How about the communication(tell&ask) ontology or AC ontology? If a user at the ported site knows the meaning of the ontology before hand, like the conceptual schema case, then some programs he/she has already coded could immediately use the ported ontology. Thus, communication ontology could

be portable.

2.7 Data vs. Knowledge

There is considerable similarity between a data base and a knowledge base. However, if they are not completely the same, then what is the major difference between them. Among them, what we would like to point out is context-dependency. Data bases are designed so as to make the application-dependency as small as possible, while knowledge bases are application-specific in their nature. This difference is critical. Data in data bases are less context-sensitive than knowledge, in other words, data can be interpreted as the user wants more independently of any context than knowledge. When he/she applies knowledge in a knowledge base to a different problem, however, one has to pay close attention to the context of problem solving and check applicability of the knowledge(expertise).

Note here, however, there is knowledge which has characteristics similar to data, say, physical units and mathematical formulae[4][8]. These kinds of knowledge is "static" in the sense that it has the same meaning in any context. The other extreme is rules in a rule base which contains heuristic knowledge of a domain expert. They are highly context-dependent. Nevertheless both kinds of knowledge are referred to as "knowledge" which causes a serious confusion, since the former is very similar to data but the latter is not.

Let us show an example which highlights the difference between data and knowledge. Imagine a rule in a rule base. When one is retrieving rules appropriate for the current situation, the rule is viewed as not knowledge but data, since it is the same as data retrieval. After finding an appropriate one, he/she tries to execute the rule, then the rule becomes knowledge, since its execution affects the problem solving context. This shows a clear discrimination between data and knowledge. When one views knowledge from the point of view of communication 1 in Fig. 1, that is, tell&ask communication, it becomes data. When one views it from the communication 2 in Fig. 2 point of view, i.e., one is interested in the content of knowledge, on the other hand, he/she really discusses about "knowledge".

Therefore, one has to be very careful when he/she discusses knowledge in the context of tell&ask communication because if he/she does not make it explicit, readers understand "knowledge" is discussed in that discussion in spite of that not "knowledge" but "data" is discussed there.

2.8 How to use ontology

The last issue related to ontology is how to use it. Gruber states ontology is used for interface between multiple software agents[2]. This shows his standpoint from which he views ontology is one like conceptual schema. There are many researchers taking this view.

In a task analysis interview system, MULTIS, which will be discussed in the last section, we use ontology in two ways such as 1) to facilitate an interaction between domain experts and a computer interviewer for knowledge acquisition, that is, communication 2 by bridging the conceptual gap between the domain experts and computers, and 2) to reuse it to build a problem solving engine which simulates the problem solving behavior of the domain expert. Therefore, we do need content ontology.

2.9 Summary

We have thus far discussed the difference between ontologies and found out several dimensions which contribute to making the characteristics clear:
(1) Sharing vs. reuse
(2) Black box vs. white box. .
(3) Communication 1(tell&ask) vs. communication

2(knowledge acquisition)
(4) After vs. before commitment
(5) Data vs. knowledge

Thus, research interests concerning ontology design largely differ from one another according to how he/she views ontology. When discussing ontology, therefore, we have to make it explicit which ontology we are talking about. As stated earlier, we will confine ourselves to content ontology and BC ontology for problem solving knowledge(expertise) reuse in the rest of this paper.

3 EXPLICATION OF PROBLEM SOLVING CONTEXT

3.1 The knowledge perspective

Knowledge cannot be viewed independently of the context determined by the problem solving process. From the definition of Ontology (1) in section 2 knowledge should be context-independent. But from (2), whether knowledge is independent of the context or not is implicit. From (3), one easily infers that it is dependent on the goal of the system built which apparently depends on the problem solving process the system is involved in. Especially, when we discuss the content ontology, we have to pay much attention to the context issue. That is, ontology design should begin by considering how to make problem solving context explicit.

A human problem solver is considered to be an expert if he can deal with and exploit effectively the many characteristics of his problem solving environment. The emergence of such a behavior is recognized to be the result of adaptation to a history of interactions with the problem solving environment.

Problem solving expertise, in particular, is a product of an on-going process in which a structure on knowledge emerges as an adaptation to the interactions with the problem solver's environment[22]. The knowledge being processed comes from various sources such as domain theory, objects being reasoned about, workplace environment, and so on. The structure allows for effective application of the knowledge in a problem solving situation. Expertise is thus tuned to the specific environment in which problem solving is carried out. Expertise is therefore often referred to as heuristic or compiled knowledge.

Because of the specificity of heuristic knowledge, reuse is limited. For enabling reuse of expertise, a technique of "knowledge decompilation" [11] is widely recognized as being useful. This technique decomposes expertise into several kinds of knowledge, making explicit and justifying the role this knowledge plays in the problem solving process. The importance of knowledge decompilation is that it unravels knowledge content issues (as opposed to representation). We argue that understanding knowledge content is a fundamental issue to allow for knowledge reuse and sharing.

Knowledge decompilation covers many aspects. In this article, we limit ourselves to task and domain decompositions and discuss their contributions to knowledge reuse and sharing.

Roughly speaking, problem solving knowledge can be decomposed into task-dependent and domain-dependent portions. We call the former task knowledge and the latter domain knowledge. Furthermore, task knowledge is deeply related to the environment, called workplace, in which the problem solving takes place. Careful study of workplace is necessary for us to discuss task knowledge which is sensitive to it. All the three kinds of knowledge require their own ontologies to make them reusable and sharable. Ontology for workplace is discussed in another paper[23].

Task knowledge is tightly related to problem solving structures specific to each task. There have been proposed several models for task structure description [1][9] [15][24] which contribute to providing reusable components for inference engines. Although they discuss their own views of problem solving structure in expert systems, they do not fully discuss "task ontology" which is required for task knowledge description. Task ontology provides us with a specification for what objects and relation among them are necessary for performing the task, which helps us design domain ontology.

3.2 Task ontology

"Task" is not the same as "problem" though some people consider contrary. This statement is justified by the fact that one can say "perform a task" but cannot "perform a problem", which shows their inherent difference. Here we consider "a task" as "a sequence of problem solving steps". Therefore, our task ontology necessarily includes "verbs" representing problem solving activities. Methods which is a concept similar to that of task consist of task structure with control. Because control structure has no task-specific concepts, we do not try to design ontology related to methods.

Task ontology[5][10][19] is a system of vocabulary for describing problem solving structure of all the existing tasks domain-independently. It is obtained by analyzing task structures of real world problems. Design of task ontology is done in order to overcome the shortcomings of generic tasks and half weak methods while preserving their basic philosophies. The ultimate goal of task ontology research includes to provide vocabulary necessary and sufficient for building a model of human problem solving processes.

When we view a problem solving process based on search as a sentence of natural language, task ontology is a system of semantic vocabulary for representing meaning of the sentence. The determination of the abstraction level of task ontology requires a close consideration on granularity and generality. Representa-

tions of two sentences with the same meaning in terms of task ontology should be the same. These observations suggest task ontology consists of the following four kinds of concepts:

(1) Generic nouns representing objects reflecting their roles appearing in the problem solving process,
(2) Generic verbs representing unit activities appearing in the problem solving process,
(3) Generic adjectives modifying the objects, and
(4) Other conceptsspecific to the task.

Task ontology for scheduling tasks which will be discussed in detail later, for example, looks as follows:

Nouns: "Schedule recipient", "Schedule resource", "Due date", "Schedule","Constraints", "Goal", "Priority", etc.
Verbs: "Assign", "Classify", "Pick up", "Select", "Relax", "Neglect", etc.
Adjectives: "Unassigned", "The last", "Idle" etc.
Others: "Strong constraint","Constraint predicates", "Constraint satisfaction","Attribute", etc.

Thus, task ontology provides primitives in terms of which we can describe problem solving context and makes it easy to put domain knowledge into problem solving context, since it provides us with abstract roles of various objects which could be instantiated to domain-specific objects. Domain knowledge organized without paying attention to its usage is difficult to find out how to incorporate what portion of it into a specific problem solving process. In the above examples, Schedule recipient and Schedule resource represent the two major objects in the scheduling task domain and its roles. Much of domain-dependent knowledge is incorporated into verbs which are formulated as operation and domain-dependent knowledge. For example, the verb "select" requires some criterion for evaluating objects to "select" appropriate one from them. This will be discussed again later.

4 TYPOLOGY OF ONTOLOGY

Domain ontology is a system of vocabulary for describing the domain. Although detailed discussion on it is omitted here because of space limitation, let us briefly discuss domain ontology. We believe domain ontology is divided into the following three categories:

(1) Object ontology related to objects under consideration in the task. This ontology covers the structure and components of the object.

(2) Activity ontology related to activities taking place in the domain. Verbs play important role in this ontology, however, they are different from those in task ontology. The subjects of the former verbs are objects, components, or humans involved in the activities of interest, while those of the latter are domain experts.

(3) Field ontology related to theories and principles which govern the domain. This ontology contains

primitive concepts appearing in the theories and relations and formulas constituting the theories and principles.

Content ontology includes another category, so called general ontology or common ontology used for representing common sense world discussed, say, in Cyc[7]. It consists of things, events, time, space, causality, behavior, function, etc. The authors have been investigating the last two topics, i.e., behavior and function for years and came up with a new ontology for them[14], though it is omitted here.

Ontology for representing ontology employed in, say, Ontolingua is called meta-ontology. Now, types of ontology are summarized in Fig. 3.

> 1. Content ontology
> Task ontology
> Generic noun
> Generic verb
> Generic adjective
> Others
> Workplace ontology[22]
> Domain ontology
> Object ontology
> Activity ontology
> Field ontology
> General/Common ontology
> Things
> Events
> Time
> Space
> Causality
> Behavior
> Function
> etc.
> 2. Tell&Ask ontology
> 3. Indexing ontology
> 4. Meta-ontology

Figure 3: Typology of ontology.

5 TASK ONTOLOGY DESIGN

This section presents an example of design process of task ontology.

5.1 What is task ontology design?

The purpose of task ontology design includes the following four issues:

(1) To understand in depth of how human experts solve problems.

(2) To extract and organize the vocabulary which characterizes the task.

(3) To identify knowledge for problem solving.

(4) To identify appropriate computational mechanisms.

Problem solving knowledge is tightly coupled with problem solving structure, that is, characteristics of task knowledge. Different knowledge is obtained by different formulations of task models. Thus, task ontology contributes to making problem solving knowledge explicit and extracting domain knowledge necessary for performing the task.

5.2 Guideline of task ontology design

A rough sketch of a procedure for designing task ontology is presented here. There are two methods for ontology design such as top-down and bottom-up methods. The former starts with analysis of existing expert systems, and come up with a set vocabulary which abstract the concepts obtained from the computational mechanisms found in the systems. The latter first analyses the domain experts behavior and come up with hierarchy of vocabulary with computational mechanisms. The following is a bottom-up one.

1) Identify task/subtask hierarchy.

2) Collect simple sentences stating unit problem solving activities in every subtask.

3) Discriminate between the domain-dependent knowledge and task-dependent knowledge by extracting verbs and identifying domain knowledge required by the verbs for its execution.

4) Refine verbs until obtaining those of appropriate grain sizes.

5) Refinement of the verbs is terminated when the domain knowledge required by each verb becomes homogeneous or the activity corresponding to an algorithm is reached. For example, a diagnostic task is viewed as "to identify the faulty part" or "to select the most plausible cause(s) from the sets of possible candidate causes" at the top level. Needless to say, grain size of "identify" and "select" of this example is too large. So, they need further refinement. But, as in nurse time shift scheduling, "to select the nurse of the heaviest load" is of the good grain size, since this "selection" is clearly defined by specifying a criterion function which calculates load of each nurse. When we are interested in the soybean disease diagnosis, on the other hand, the "identification" of the disease of the observed soybean can be interpreted as a classification task. Then, "classify" can be a candidate of task primitive verb, since the classification can be done using decision tree interpretation. The computation mechanism is an decision tree interpreter and domain knowledge is decision tree description of a specific problem.

6) Identify computational mechanism for each verb.

7) Identify nouns and other concepts as objects and components of the knowledge source of the verbs. Many of the nouns are identified as objects of verbs. Other nouns are obtained mainly by analyzing the knowledge sources associated with verbs. For example, knowledge required by "select" includes a criteria for evaluating various objects in which we find cost, precision, reliability, behavior, etc. Analysis of constraints also enables us to find other important concepts including many attributes of objects.

8) Abstract nouns to decrease domain-dependence and obtain task-general concepts which represent roles in the task/subtask.

9) Identify domain-independent adjectives which restrict or denote specific concepts/objects. Domain-dependent adjectives should be formalize as constraints.

The roles of task ontology is summarized as follows:

(1) To provide domain experts with human-readable conceptual primitives in terms of which they can express their way of problem solving.

(2) To enable a translation of the knowledge-level description of the problem solving process into symbol-level executable code.

(3) To specify the problem solving context which contributes to clarify domain knowledge.

(4) To enable us to build a task analysis interview system as will be shown in Section 6 where an overview of MULTIS is discussed as an example. Thus, task ontology is operationalized in such a system.

5.3 Task ontology for scheduling tasks

We have developed a task analysis interview system named MULTIS[10][18][19] which models the task structure of a domain expert based on task ontology. An ontology for scheduling tasks in MULTIS is shown as follows:

Generic nouns
    Schedule Recipient(RCP): (eg.) Job, Order, Driver, Nurse, etc.
    RCP-GRP: Group of RCPs
    Schedule Resource(RSC): (eg.) Line, Machine, Track, Duties, etc.
    RSC-GRP: Group of RSC
    Time slot: Slots of time RCP is assigned to.
        {These three are the basic terms in scheduling tasks, since solution is represented in terms of them.}
    Schedule: Solution of the task
        Schedule, Intermediate solution, Final solution, etc.
    Schedule representation
        Gantt chart[2], Time table, etc.
        {Solution and its representation are separated, since a solution can be

represented in several manners. Form of solution representation gives us very useful information because they have domain specific information such as terminology, types of problems, etc.}
    Constraint/condition: Constraints to be satisfied by the schedule
    Goal: Goal of the scheduling to optimize
    Priority
    Data/Information

Generic adjective
    Assigned/unassigned, Previous, Last, Next, Satisfying, Violating, etc.

Constraint-related vocabulary
    Constraint/Condition
        Strong constraint: Constraints which must be satisfied
        Weak constraint: Constraints which can be relaxed in some cases
    Constraint adjective:
        Maximal, Minimal, Earliest, Latest, Longest, Shortest, Largest, etc.
    Constraint-predicate:
        Equal to, Larger than, Smaller than, Include, Exclude, Overlap, etc.
    Attribute: Components of constraints/ attributes of objects
        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: {Same as that appearing in generic noun}
    Status
        Maximum, Minimum, Uniform, Continuous, etc.
    Object
        Load balance, Rate of operation, Efficiency, Idle time, Operation time, etc.

RSC/RCP verb: {These verbs mainly take RSC or RCP as their objects. The subjects of them are domain experts.}
    Generate: Generates objects to process
        Assign: assign RCP to RSC and time
        Classify: classify objects into groups
        Combine: make tuples of objects
        Compute: obtain values of parameters
        Divide: divide objects into groups
        Insert: insert an object into a list
        Merge: merge some objects

Permute: generate a permutation
Pickup: take an object out of a list
Remove: remove objects from a list
Select: take objects satisfying a condition
out of a list
Sequence: arrange objects in order
Test: Test if an enumeration or search is exhaust
or not{This is used together with the
verb Pick-up}
Check: check object if it satisfies a
constraint/condition
Evaluate: evaluate an object to obtain score
Modify: Modify assignments and data
Reassign
Reassign
Exchange: exchange assignments
Shift left/right: shift assignment to
left/right
Pack-to-the-left
Update: update data

Constraint_verb: Verbs which take constraints as
their objects
Add
Delete
Neglect
Strengthen
Weaken/Relax

Several executable building blocks are associated with every verb in MULTIS. Each building block is defined as a combination of its main computational mechanism and domain knowledge necessary for performing its function. Some examples of specifications of typical building blocks are shown below

Classify: classify objects into some groups
input: list of objects
output: list of lists of object and its category
domain knowledge: decision tree, set of rules,
distance or similarity, etc.
ClassifyDT: classify objects using a decision tree
ClassifyAttr: classify objects of the same
attribute value
ClassifyDist: classify objects based on the
distances among them
ClassifyRule: classify objects by interpreting rules
Select: select object(s) satisfying constraint/condition
input: objects
output: list of object(s)
domain knowledge: condition, criterion function,
set of rules for evaluation
SelectMaxAll: make a list of all the objects of
maximum value calculated according to
the criterion function.

SelectMinAll, SelectMaxOne, SelectMinOne,
SelectMaxpAll, SelectMaxpOne, SelectSatisfyAll
SelectSatisfyOne: select an object satisfying
the constraint/condition
Sequence: arrange objects in order
input: list of objects with ordering information
output: list of ordered objects
domain knowledge: none
SequenceT: arrange objects according to values of
total ordering
SequenceP: arrange objects according to values of
partial ordering

Computational mechanisms of other verbs are omitted here due to the space limitation. Adjectives are also omitted.

The authors had set up a consortium for developing and evaluating scheduling ontology in ASTEM/RI in cooperation with eight companies. In the consortium, MULTIS has been evaluated by describing task structures of several ESs which the members of the consortium were involved in their development. The evaluation shows our task ontology has sufficient expressive power for scheduling task structures. Scheduling systems used for evaluation include four job-shop scheduling systems, three car-dispatching systems, two production planning systems, two human resource allocation systems, and one configuration system.

In appendix, a portion of Ontolingua implementation of our scheduling ontology is shown. As stated above, the implementation of our content ontology in Ontolingua is not very satisfactory, but we think it helps readers understand its formal representation. The whole implementation will be available through WWW soon.

6 MULTIS

6.1 Overview

In MULTIS, ontology is called a system of "generic vocabulary" which consists of generic nouns, generic verbs, generic adjectives, and other task specific concepts. A combination of generic verb and noun such as "verb + noun" is referred to as "generic process". A network of generic processes is called generic process network(GPN) which represents a skeleton of task structure at the knowledge level.

Fig.4 shows the block diagram of MULTIS which consists of seven modules such as CBR, generic vocabulary/process library, building block library, generic process network editor (GPNE), GPN compiler(GPNC), interview engine and interface. GPNE builds a GPN through interaction with a domain expert. During the process, it retrieves several GPNs from case DB and consults generic vocabulary(task ontology) library to translate the generic vocabulary contained in the cases(GPNs) into some domain-specific

---

[2] A gantt chart is a chart used for schedule representation of job-shop scheduling tasks.

concepts. After building a GPN, control is passed to GPNC which identifies building blocks necessary for implementing inference engine and eventually generates an executable code through interview.
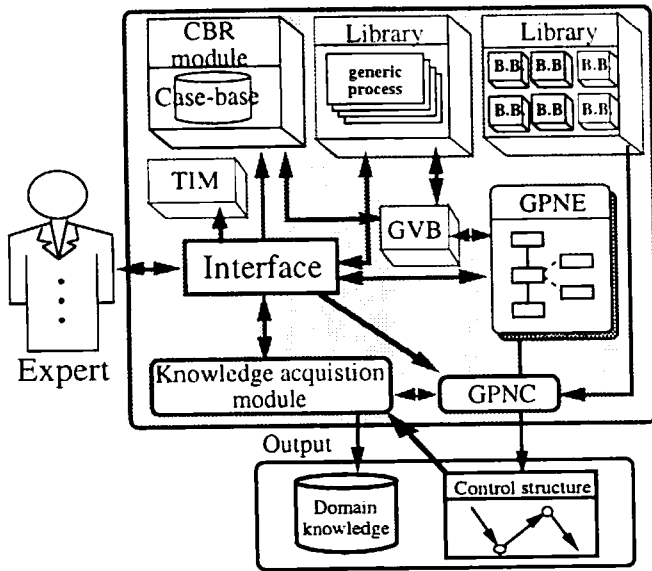

Figure 4: Block diagram of MULTIS

6.2 How it works

MULTIS first tries to obtain several basic concepts by using some templates of schedule representation. The rows, columns and entries of them correspond to time, schedule resources and schedule recipients, respectively. Thus, at least these three domain-specific concepts are easily obtained through a simple interaction.

One of the major issues in MULTIS 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 so 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. Domain experts can build their problem solving models by modifying the GPN retrieved. 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 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.

Fig. 5 shows an example of GPN obtained for a belt scheduling systems discussed in [19]. The upper part of each box includes representation of unit operation in domain terms and the lower part that in generic vocabulary. As described above, this GPN is configured mainly by cut&paste consulting the case base. The GPN shows main flow of the belt scheduling process and implicitly specifies domain knowledge required for each generic verb(see 5.3), which makes it easier to elicit domain knowledge.

After GPN design has been terminated, GPNC compiles it to generate Common lisp code by configuring building blocks during which GPNC asks the user several questions necessary for selecting appropriate building blocks associated with each verb and data flow among them. Backtrack information is also obtained through the interview. To enable this, MULTIS has an ontology for backtracking which is applicable to wide rage of backtracking phenomena observed in scheduling tasks.

6.3 Generality

Finally, we would like to discuss the generality of MULTIS. Note that MULTIS is designed in a highly modular structure. Content of 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 replacing the three libraries with those of these tasks. The only exception to this generality is the interface module part of which is dependent on solution representation of scheduling, that is, gantt chart or time table. We have to modify this task-dependency in order to make MULTIS architecture task-independent.

7 CONCLUSIONS

Wielinga and Schreiber discuss ontological issues in knowledge base technology[25]. They take similar position as Ontolingua approach, that is, they consider ontology as a formal theory of conceptualization in knowledgeable agents. What makes their approach differ from that of Ontolingua is they introduce several types of ontologies reflecting the differences of applications, and hence, task structures. In this sense, we
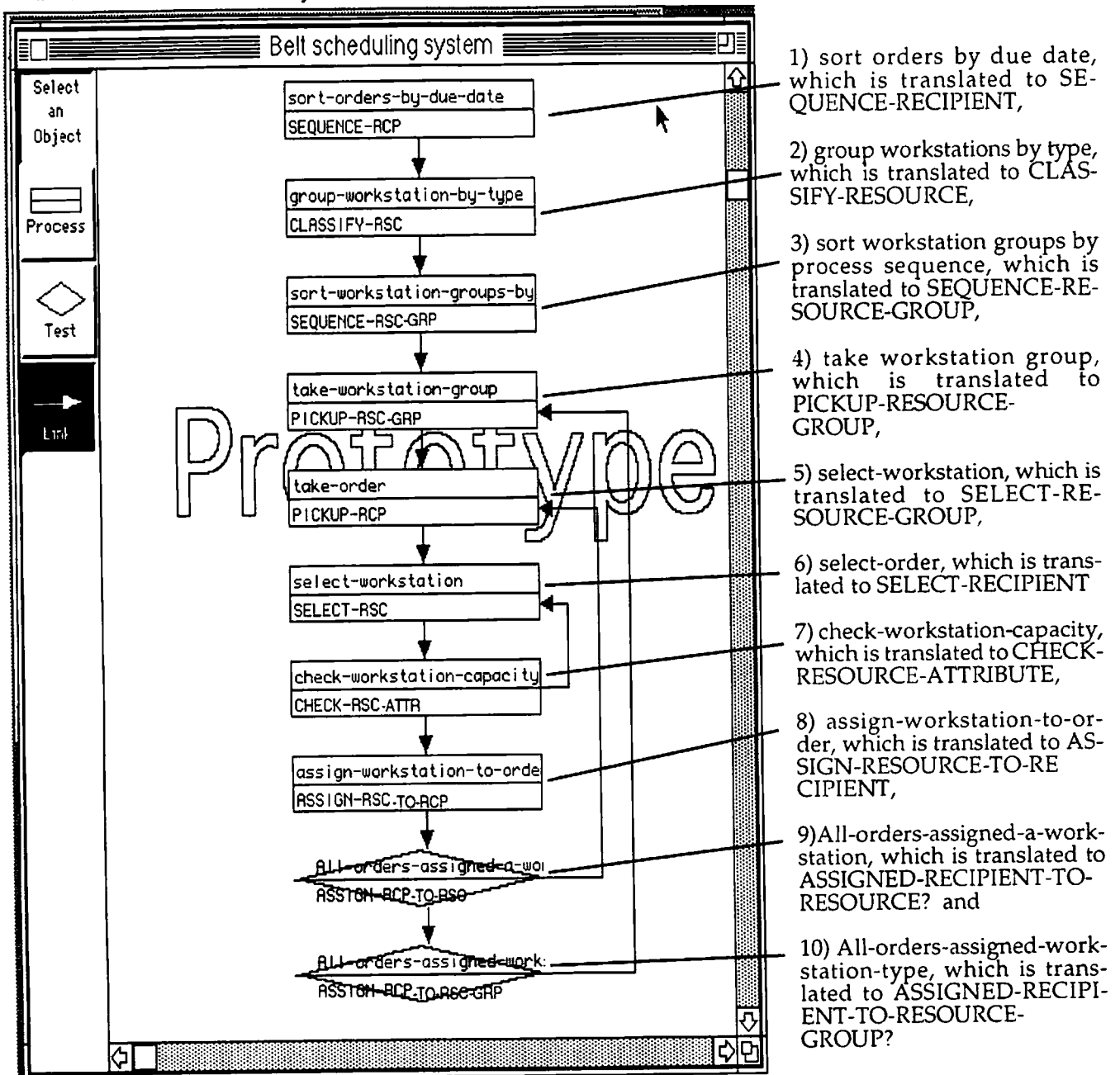
Figure 5: An example of GPN for a belt scheduling task.

share a lot with them. However, they weakly discuss task ontology which is one of the main issues in this paper. Although we adopt task-domain decomposition approach to knowledge modeling like them, we try to ontologize equally both domain-specific knowledge and task-specific knowledge.

We have discussed what an ontology is and pointed out several perspectives useful for understanding it in depth. A task ontology is presented and task analysis interview system named MULTIS has been overviewed to show how task ontology is effectively operationalized. Although the usefulness of task ontology in task analysis interview is verified, how it helps acquire and

organize domain knowledge is not fully evaluated. We plan to augment the expressiveness of task ontology in explaining the relations between organizational characteristics and problem solving process [22].

Acknowledgement

REFERENCES

[1] Chandrasekaran, B.: Generic tasks for knowledge-based reasoning: the right level of abstraction for knowledge acquisition, IEEE Expert, Vol.1, No.3, pp.23-30, 1986.

[2] Gruber, T.: A translation approach to portable ontology specifications, Proc. of JKAW'92, pp. 89-108, 1992.

[3] Gruber, T.: Toward principles for the design of ontologies used for knowledge sharing, TR KSL93-04, Stanford University, 1993.

[4] Gruber, T.: An ontology for engineering mathematics, Proc. of Comparison of implemented ontology, ECAI'94 Workshop, W13, pp.93-104, 1994.

[5] Hori, M., et al.: Methodology for configuring scheduling engines with task-specific components, Proc. of JKAW'92, pp.215-230, 1992.

[6] Karbach, W., M. Linster, A. Voss: Models, methods, roles and tasks: many labels - One idea?, Knowledge Acquisition, 2, pp.279-299, 1990.

[7] Lenat, D. and R. Guha: Building Large Knowledge-Based Systems, Addison-Wesley Publishing Company, Inc., 1990.

[8] Mars, N.: An ontology of measurement, Proc. of Comparison of implemented ontology, ECAI'94 Workshop, W13, pp.153-162, 1994.

[9] McDermott, J.: Using problem solving methods to impose structure on knowledge, Proc. of the Intl. Conf. on AI Applications, pp.7-11, 1988.

[10] Mizoguchi, R. et al.: Task ontology and its use in a task analysis interview systems - Two-level mediating representation in MULTIS -, Proc. of the JKAW'92, pp.185-198, 1992.

[11] Mizoguchi, R.: Expert systems and knowledge base technology, International J. of Computer and Engineering Management, Vol.1, No.2, pp.24-39, 1993.

[12] Mizoguchi, R.: Knowledge acquisition and ontology, Proc. of the KB&KS'93, Tokyo, pp. 121-128, 1993.

[13] Newell, A.: The knowledge level, Artificial Intelligence, Vol.18, No.1, pp.87-127, 1982.

[14] Sasajima, M., et al.: An investigation on domain ontology to represent functional models, Proc. of the Eighth International Workshop on Qualitative Reasoning about Physical Systems, pp.224-233, 1994.

[15] Steels, L. : Components of expertise, IEEE Expert, Vol.11, No.2, pp.28-49, 1990.

[16] Swartout, W. et al.: Knowledge sharing: Prospects and challenges, Proc. KB&KS, Tokyo, pp.95-102, 1993.

[17] Takaoka, Y. et al.: Towards re-use of knowledge - A study of methods regarding substation fault recovery operation support system -, Proc. of WCES'94, Lisbon, Paper-ID-336, 1994.

[18] Tijerino, A. Y. et al: A task analysis interview system that uses a problem solving model, Proc. of JKAW90, pp.331-344, 1990.

[19] Tijerino, A. Yuri, et al.: MULTIS II: Enabling end-users to design problem-solving engines via two-level task ontologies, Proc. of EKAW'93, 1993.

[20] Vanwelkenhuysen, Johan: Participative Conceptual system design of industrial knowledge systems, Technical Report 39-1, Artificial Intelligence Lab., Free University of Brussels, 1993.

[21] Van de Velde, W.: Issues in knowledge level modelling, Tech. Report, AI lab, Free University of Brussels, AI-Memo 93-09, 1993.

[22] Vanwelkenhuysen, Johan and R. Mizoguchi: Maintaining the workplace context in a knowledge level analysis, The Third Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, pp.33-47, 1994.

[23] Vanwelkenhuysen, Johan and R. Mizoguchi: Workplace-Adapted Behaviors: Lessons Learned for Knowledge Reuse, Proc. of KB&KS '95, 1995(Submitted).

[24] Wielinga, B. et al.: The KADS knowledge modelling approach, Proc. of JKAW'92, pp.23-42, 1992.

[25] Wielinga, B. et al.: Reusable and sharable knowledge bases: A European perspective,Proc. of KB&KS'93, Tokyo, pp.103-115, 1993.

Appendix: A portion of Ontolingua implementation of Scheduling task ontology

```
(define-class classify (?classify)
"Classify is one of the generic-verbs. It classifies a list of objects into some categories by using
domain knowledge.The domain knowledge is a relation between input objects and categories. Output is a list
of categories. Each category is a list of a category name and a list of its members"
        :def (exists (?input ?output ?dk)
                (and (verb.name ?classify classify)
                        (verb.input ?classify ?input)
                        (list ?input)
                        (verb.output ?classify ?output)
                        (category-list ?output)
                        (verb.domain-knowledge ?classify ?dk)
                        (constraint ?dk)
                        (forall ?category
                            (=> (and(member ?category ?output)
                                    (category ?category))
                                (forall ?element
                                    (=> (member ?element(second ?category))
                                        (satisfies-constraint
                                            (listof ?element (first ?category))
                                            ?dk))))))))
        :axiom-def (subclass-of classify generic-verb))

;; generic-noun
(define-class rsc (?x)
"RSC is resource used in scheduling."
        :def    (and(individual ?x)
                    (value-type ?x rsc.attributes list))
        :axiom-def (and (subclass-of rsc generic-noun)
                        (subclass-partition
                            rsc
                            (setof resource
                                human-rsc
                                material-rsc
                                vehicle-rsc
                                facility-rsc
                                space-rsc
                                duty-rsc
                                ))))

;; generic-adjective
(define-relation un-assigned (?un-assigned)
        :axiom-def (subrelation-of un-assigned generic-adjective)
        :iff-def (not (assigned ?un-assigned)))

(define-relation assigned (?assigned)
"A kind of generic-adjective. Solution of scheduling is represented as a set of assignments in 'assignment-set'"
        :axiom-def (subrelation-of assigned generic-adjective)
        :iff-def (or
                    (and    (rsc ?assigned)
                            (exists (?time ?rcp ?assignment-set)
                                (and (assignment-set ?assignment-set)
                                    (member (listof ?assigned ?time ?rcp)
                                        ?assignment-set))))
                    (and    (time-unit ?assigned)
                            (exists (?rsc ?rcp ?assignment-set)
                                (and (assignment-set ?assignment-set)
                                    (member (listof ?rsc ?assigned ?rcp)
```

```
                              ?assignment-set))))
        (and    (rcp ?assigned)
                (exists (?rsc ?time ?assignment-set)
                     (and (assignment-set ?assignment-set)
                          (member (listof ?rsc ?time ?assigned)
                                  ?assignment-set)))))
```