Ontology for Modeling the World from Problem Solving Perspectives

Riichiro Mizoguchi, Mitsuru Ikeda,
Kazuhisa Seta, and Johan Vanwelkenhuysen*
ISIR, Osaka University
* Currently, INRIA, France
miz@ei.sanken.osaka-u.ac.jp

Abstract

We discuss some basic issues concerning ontology along with concrete examples. After a brief general discussion, typology of ontology is discussed. Based on the top-level classification into four categories such as content ontology, communication ontology, indexing ontology, and meta-ontology, further classification of content ontology is discussed in detail. The basic philosophy behind the typology is to view knowledge from problem solving perspectives, since the world is too complicated to model without any specific objectives. Among the several categories of ontologies, we investigate a task ontology which characterizes the problem solving context which in turn characterizes domain knowledge. Concrete examples of task ontology are described. A portion of Ontolingua implementation is shown in appendix. We finally mention how task ontology is used by a task analysis interview system, MULTIS in a sophisticated manner.

1. Introduction

There have been many discussions on ontology in the literature and various ontologies are becoming available these days. The authors have discussed meta-level specification and classification of ontology in[Mizoguchi, 1995]. In this paper, in order to make ontology discussion clearer, we classify ontologies in more detail.

The world is too complicated to model without an explicit goal. Particularly, when the purpose of the modeling is engineering one, we do need some specific goal to get useful ontology. Our research goal includes in-depth understanding of knowledge for real-world problem solving. In other words, we are interested in the expertise which domain experts have. A domain expert has his/her own model of the small world he/she is involved in. Therefore, a model of expertise reflects the structure of the world viewed from the domain expert who solves problems in it.

We have been involved in the design of a task ontology, while some other people are trying to model a domain without consideration on problem solving performed by domain experts. What is a difference between the two approaches to ontology research? How many types of ontologies exist? What methodology is available for ontology design? We try to answer these questions in this paper. Another topic is how to use ontology. Which agent, a human or a computer, use an ontology and in what purpose each agent uses an ontology are discussed. After a brief general discussion, typology of ontology is discussed in section 3. Section 4 describes a design method called AFM: Activity-First Method. Finally, how ontology is used in MULTIS, a task analysis interview system, is illustrated.

2. General discussion

2.1 Use of ontology

When someone designs something, its prospective use has to be explicit, otherwise the design would not be effectively done. How about ontology design? It seems to the authors how to use ontology has not been fully discussed. Who uses ontology is the first issue to discuss. Which agents are the users of ontology, humans or computers? This difference is critical. If humans are the only users of ontology, then formal representation does not make sense very much, since readability and expressiveness of formal representations are lower than informal

representation in natural language. The authors do not think formal representation is much less ambiguous than natural language because the former has primitives whose meaning cannot be defined in principle. In spite of its inherent ambiguity, natural language has been used as a communication medium by humans, which weakly supports that natural language could be sufficient for representing ontology for humans.

The formal representations make sense when they are used by computers. The formally-represented ontology is currently used not as a set of ontology but a set of ***data*** by computers. It only enables computers read them and show them in a network or hierarchy. It is the shallowest use. Or we could say one cannot say formally-represented ontology is used by computers. As far as the authors know, the only exception is the work by Fox et al. who develop a prover which does reasoning on formally-represented ontology[Gruninger, 1994].

The next issue is how to use ontology. The authors believe one can say a computer uses ontology only if it uses it ***"as ontology"***. Ontology is frequently used as a vocabulary in terms of which agents communicate with one another. However, this usage of ontology is just as a set of labels of concepts. Meaning of concepts or the formally-represented ontology are not used inside computers. In this sense, we could say usage of ontology is not matured enough up to now. In the knowledge acquisition systems, however, ontology plays more critical role and the meaning of concepts has to be extensively used to make the interview effective and to generate code of the problem solving engine by interpreting the domain expert's statements in terms of ontology. Basically, we can translate formally-defined ontology into internal representation necessary for performing the knowledge acquisition interview. This topic is discussed in section 5.

2.2 Context

Distinguishing between knowledge and data is crucial in ontology design. Among several differences, what we would like to point out is context-dependency. Data bases are designed so as to make the application-independency as small as possible, while knowledge bases in expert systems are application-specific in their nature. This difference is critical. Data can be interpreted more context-independently than knowledge. When one applies knowledge in a knowledge base to different problem solving, however, he/she has to pay close attention to the context of problem solving and check validity of the knowledge(expertise).

Note here, however, this characterization is not crispy. There is knowledge which has characteristics similar to data, say, physical units and mathematical formulae[Mars, 1994][Gruber 1994]. These kinds of knowledge is said to be "static" in the sense that their meaning is very stable. The other extreme is rules in a rule base which contains heuristic knowledge of a domain expert. They are highly context-dependent in both of the run-time context and design-time context, i.e., the goal of the expert system. Nevertheless both kinds of knowledge are referred to as "knowledge" which sometimes causes a serious confusion, since the former is very similar to data but the latter is not.

On the other hand, when a piece of knowledge is viewed as a target of retrieval, we can say it is a data. When someone tries to use it for solving a problem, however, it is not a data any more but a knowledge which should be carefully interpreted according to the problem solving context.

The above discussion shows the importance of the specification of the context in which knowledge is used. There could be 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 the knowledge base technology perspective, knowledge should be considered in a problem solving situation. Then, the next issue is how to make the context explicit. Our claim is the context can be specified by making the problem solving process explicit using **Task Ontology** which is discussed in section 3.3 and workplace ontology[Vanwelkenhuysen, 1994, 1995b] specifying the environment where the problem solving process is embedded.

While the above discussion has been made from a problem solving perspective, there is another viewpoint to ontology design without any specific task in mind. Some people try to model objects or organizations as they are. For example, a nuclear power plant is composed of a reactor, primary cooling system, an intermediate heat exchanger, secondary cooling system, steam generator, power generator, etc. Therefore, he/she could identify these subsystems which are further composed of a number of components of smaller grain size and design ontology according to them. The resulting ontology seems objective and independent of any context. However, the reality is not so. The developer of such an ontology does need some criteria to decide the depth of modeling, i.e., the smallest size of the components, necessary attributes of each components, and degree of approximation of formula or quantitative relations among parameters of components. He/she may have to go down to quantum physics. Chemical characteristics of water may be neglected in some cases. Even if the object of interest is a man-made artifact, not all the phenomena are well known, which frequently requires approximation to obtain an analytical expression among several parameters. And there are sometimes multiple alternatives of doing approximation. Thus, context specification is critical in organizing knowledge, and hence designing ontology.

2.3 How to capture ontology

Imagine a set of sentences explaining how to solve a problem. Viewing the sentences are equivalent to the problem solving process, the task ontology is a system of vocabulary for representing meaning of each sentence. The determination of the abstraction level of a task ontology requires a close consideration on granularity and generality. Representations of two sentences with the same meaning in terms of task ontology should be the same. These observations suggest the 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 concepts** specific to the task under consideration.

Ontology design in our research is done based on this idea.

3. Typology of ontology

3.1 Top level classification

In the previous paper, we discussed what an ontology is and came up with a top level classification which is summarized here[Mizoguchi, 1995]. According to the several dimensions such as sharing/reuse of knowledge, before/after commitment, black box/white box, etc., ontologies can be classified into the following four categories:

1) Content ontology
2) Communication(tell&ask) ontology
3) Indexing ontology
4) Meta ontology

Content ontology is one for reuse of knowledge. By reuse, we mean the same meaning as in "software reuse", that is, computers have something to do with content of the vocabulary in an ontology. By sharing, we mean knowledge is used indirectly by agents who ask an agent who owns the knowledge to solve problems. Researchers interested in content ontology view a knowledge base as a white box, while those interested in communication ontology view one as a black box. Indexing ontology is one for associating indices with each case. Indices should be carefully designed to enable effective retrieval of cases. Meta-ontology is one for representing ontology employed in, say, Ontolingua. In the following subsections, we discuss content ontology in detail.

3.2 Finer classification

The world is so complicated that we cannot easily design ontology useful for reuse of knowledge in practical problem solving. Thus, we try to model expertise that domain experts have, since it represents the model of the world built by domain experts through their rich experience in solving real world problems. Roughly speaking, the world consists of a number of domains each of which consists of a lot of objects, concepts, agents, activities, theories/principles, and domain experts performing problem solving tasks there, etc. Objects include power plants, circuit boards, patients, etc., concepts productivity, cost, power supply, complaints, etc., agents truck drivers, nurses, etc., theories/principles newton's and Kirchhoff's laws, and domain experts plant operators, circuits designers, troubleshooters, and doctors. An object might be related to multiple tasks each of which at least one domain expert is involved in. For example, we easily find three types of tasks such as design, diagnosis, and job-shop scheduling. Further, a domain expert performs a task within an environment(workplace) which surrounds him/her and the object and determine several requirements to his/her performance. The difference between domain experts and agents are ambiguous in some cases. To avoid confusion, we define domain experts as humans whose expertise one wants to model and agents are those for whom one is not interested in how they do their jobs.

According to this characterization of the world, we first identify three types of ontologies such as workplace ontology, task ontology, and domain ontology. Domain ontology is further divided into object ontology, field ontology, and activity ontology. These three types of ontologies are task-independent. The above discussion on context and task knowledge suggests another type, that is, task-dependent domain ontology.

Content ontology includes another category, so called general ontology or common ontology used for representing common sense world discussed, say, in Cyc[Lenat, 1990]. 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[Sasajima, 1995], though it is omitted here. Now, content ontology is summarized in Fig. 1.

Content ontology

Workplace ontology[Vanwelkenhuysen, 1994]

It is an ontology for workplace which affects task characteristics by specifying several bourdary conditions which characterize and justify problem solving behavior in the workplace. Workplace and task ontologies collectively specify the context in which domain knowledge is interpreted and used during the problem solving.

Examples(Circuit troubleshooting): *Fidelity/Efficiency/Precision/High reliability/etc.*

Task ontology[Mizoguchi, 1992, 1995][Hori, 1994][Wielinga, 1993]

Task ontology is a system of vocabulary for describing problem solving structure of all the existing tasks domain-independently. It does not cover the control structure but do components or primitives of unit inferences taking place during performing tasks. Task knowledge in turn specifies domain knowledge by giving roles to each objects and relations between them.

Examples(Scheduling tasks): *Schedule recipient/schedule resource/goal/constraint/select/assign/classify/remove/relax/add/etc.*

Domain ontology

Task-dependent ontology[Mizoguchi,1995]

A task structure requires not all the domain knowledge but some specific domain knowledge in a certain specific organization. We call this special type of domain knowledge T-domain ontology because it depends on the task.

T-Domain ontology

Examples(Job-shop scheduling): *Job/order/line/due date/machine availability/load/cost/nurse/vehicle/etc.*

Task-independent ontology

Because object and activity ontologies are related to activities, we call them activity-related ontology and field ontology activity-independent ontology.

Activity-related ontology

This ontology is related to activities taking place in the domain and is designed having simulation of the domain activity in mind such as enterprise ontology. There are two major activities exist in a domain. One is behavior of an object and the other is organizational or human activities. Verbs play an important role in this ontology, however, they are different from those in task ontology. The subjects of the former verbs are objects, components, or agents involved in the activities of interest, while those of the latter are domain experts.

Object ontology[Vanwelkenhuysen,1995a]

This ontology covers the structure, behavior and function of the object.

Examples(Circuit boards): *component/connection/line/chip/pin/gate/bus/state/role/etc.*

Activity ontology(Enterprise ontology)[Gruninger,1994]

Examples: *use/consume/produce/release/state/resource/commit/enable/complete/disenable/reenable/etc.*

Activity-independent ontology

Field ontology

This ontology is related to theories and principles which govern the domain. It contains primitive concepts appearing in the theories and relations, formulas, and units constituting the theories and principles.

Units[Mars,1994]

Examples: *mole/kilogram/meter/ampere/radian/etc.*

Engineering Math[Gruber,1994]

Examples: *Physical quantity/physical dimension/unit of measure/Scalar quantity/linear algebra/physical component/etc.*

General/Common ontology

Examples: *Things/Events/Time/Space/Causality* [Lenat,1990]
Behavior/Function [Sasajima,1995], etc.

Communication(tell&ask) ontology

Indexing ontology

Meta ontology

Fig. 1 Typology of content ontology.

Typology of ontology is not an exclusive classification, since it mainly concerns the view points from which we investigate the world. Some of the components in the typology are shared by multiple ontologies.

3.3 Task ontology

A task ontology is designed in order to overcome the shortcomings of generic tasks and half weak methods, i.e., inflexibility caused by their large grain size, while preserving their basic philosophies, i.e., modeling at the right level of abstraction. In the process of knowledge acquisition(KA), they are considered as templates of task structure which could guide the KA process[Wielinga,1992]. As it is easily imagined, however, they are not very useful when the template does not match the target task. This motivated MULTIS project [Tijerino,1990][Mizoguchi,1992][Tijerino,1993] which aims at developing multi-task KA interview system based on task ontology which enables reconfiguration of task templates preserving role-limiting functions of task models with a finer grain size to increase flexibility. In MULTIS, a combination of a generic verb and a generic noun such as "verb + noun" is referred to as "generic process". A network of generic processes is called a generic process network(GPN) which represents a skeleton of task structure at the knowledge level and serves as a task template. A case base stores a number of GPNs of existing expert systems for prospective use as templates of task structures. The ultimate goal of the task ontology research includes to provide all the vocabulary necessary and sufficient for building a model of human problem solving processes at the knowledge level.

Task ontology for scheduling tasks, for example, are as follows:

Generic Nouns: *Schedule recipient*, e.g., *jobs/nurses/etc.*
Schedule resource, e.g., *machines/work-shift/etc.*,
Due date/Constraints/Goal/Priority/etc.
Generic Verbs: *Assign/Classify/Pick up/Select/Relax/Neglect/etc.*
Generic Adjectives: *Unassigned/The last/available/etc.*
Others: *Strong constraint/Constraint predicates/Constraint attributes/etc.*

A 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 the domain-dependent knowledge is incorporated into verbs which are formulated as an operation and a set of domain-dependent knowledge. Several executable building blocks are associated with every verb in the task ontology. Each building block is defined as a combination of its main computational mechanism and domain knowledge necessary for performing its function. Two examples of specifications of typical building blocks are shown below.

Classify: classify objects into some categories.
input: a set of objects
output: a set of lists of an 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 into the same class.
ClassifyDist: classify objects based on the distances among them.
ClassifyRule: classify objects by interpreting rules.

Select: select object(s) satisfying a constraint/condition specified from a set of objects.
input: a set of objects
output: a set of object(s)
domain knowledge: condition, criterion function, set of rules for evaluation, etc.
SelectMaxAll: make a set 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.

One of the most important characteristics of the task ontology is that meanings of verbs are defined at the symbol level, that is, at least one executable code is associated with each verb to enable semi-automatic generation of a problem solving engine for the target task. A task ontology for scheduling tasks has been

implemented in Ontolingua some of which are shown in Appendix.

3.4 T-domain ontology

A T-domain ontology specifies what types of domain concepts are relevant to a specific task. Thus, T-domain ontology is organized according to the requirements made by the task ontology. Let us show some examples of a T-domain ontology in the job-shop scheduling tasks.

Schedule recipient : *job/order/lot/etc.*
Schedule resource: *machines/line/materials/operators/etc.*
Schedule representation: *gantt chart* composed of lines/machines for the vertical axis, allocation time for horizontal axis, and jobs/orders for the entries.
Constraint: *Jobs should be finished before due dates/Availability of the machine A should be greater than X/Maximal load of machine B is Y/etc.*
Constraint attribute: *tardiness/idle time/cost/productivity/availability/etc.*
etc.

Constraint attributes used to define a constraint specify what attributes of the objects are necessary for the task. This is one of the advantages of task-driven domain analysis. Fig. 2 shows some portion of task ontology and T-domain ontology for scheduling tasks.

Generic vocabulary for scheduling is composed of five types of concepts such as *generic verbs, generic nouns, generic adjectives, constraint-vocabulary, and goal.* As described in 2.3, *generic nouns* consist of *schedule recipient(RCP), schedule resource(RSC), schedule, goal,* and *data/information.* As is implicitly shown, *generic nouns* can be objects of *generic verbs*. This is why *goal, constraint, and attribute* appear twice in the hierarchy. Five types of links are used in Fig. 2. Unlabeled links are is-a links. Generic verbs are composed of *RSC/RCP verbs* and *constraint verbs*. The former take *schedule resource* and *schedule recipient* as their objects, while the latter constraints. Twenty six verbs have been identified for scheduling tasks in all. As mentioned earlier, each verb has several executable building blocks which represent its meaning.

The task structure is symmetric with respect to *schedule resource* and *schedule recipient*. Which object to take as a *schedule resource* or a *schedule recipient* is up to the domain expert. Therefore, many of T-domain terms are subordinated by both of the two. As defined in Appendix, *schedule resource* and *schedule recipient* are defined as objects put on the row and entry of the schedule representation such as gantt chart or time table, respectively. *Job* and *order* are exceptions. They are definitely *schedule recipients.*

RSC-GRP and *RCP-GRP* stand for grouped *RSC* and *RCP*, respectively. *Goal* is a goal of the scheduling task to be achieved. Typical goals include *to minimize the tardiness*, *to maximize the efficiency*, etc. This implies *goal* is composed of *status* and *object* as shown in the figure. *Schedule*(solution) and *schedule representation*(solution representation) are clearly discriminated. The former is defined as content of the latter(vessel). We define *generic adjective* as domain-independent modifier and constraint as domain-dependent one. For example, *assigned* and *the last* are generic adjectives and something "of the heaviest load" is taken as a constraint.

The shaded portion represents the T-domain ontology. A task ontology provides domain-independent vocabulary each term of which can be instantiated by T-domain ontology according to the subsumption relations. Actually, GPNs written in terms of the task ontology can be translated into the T-domain terminology.

3.5 Workplace ontology[Vanwelkenhuysen,1995b]

Workplace ontology is a vocabulary which captures characteristics of the workplace in which problem solving is performed; these characteristics (e.g., reliability of resources, limitations imposed by organizational processes) have a recognized effect on the structure of the problem solving process. A portion of workplace ontology for troubleshooting is given in the Fig. 3.

We have investigated the relationships between these concepts and task structures and come up with some observations concerning how they affect the tasks. Furthermore, we also obtained some guidelines for modification and refinement of predefined task structures to adapt to the target task of interest. See the literature for details[Vanwelkenhuysen,1995a].

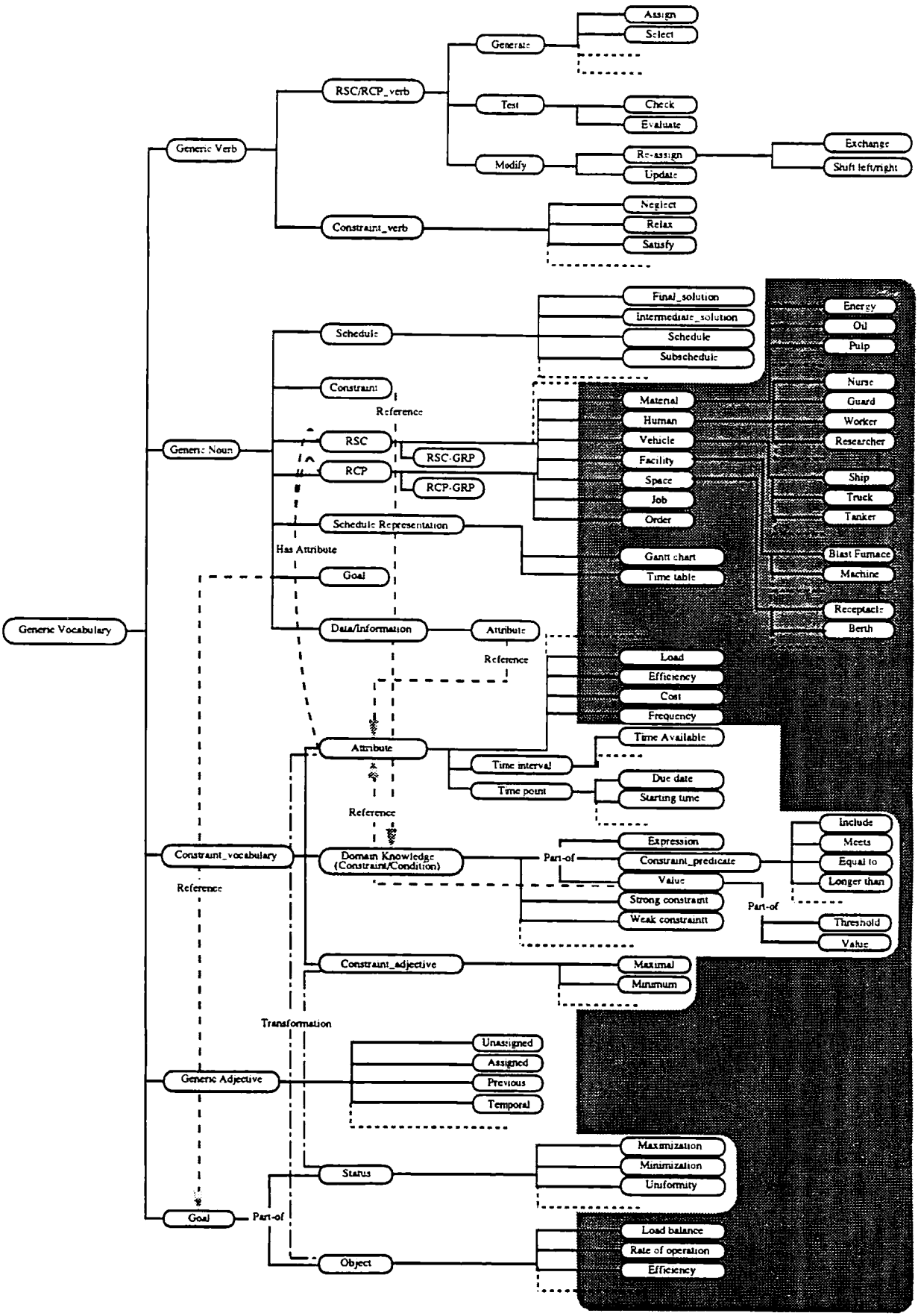

Fig. 2 A portion of task and T-domain ontologies for scheduling tasks. The shaded portion represents T-domain ontology. Unlabeled links are is-a links.

Organizational processes: this category of workplace ontology concerns characteristics of product life-cycle processes performed by the organization under focus.

Component testing/Assembly/Acceptance test/Repair/etc.

High reliability/Low accuracy/Low cost/etc.

Nature of the diagnostic task: This category of workplace ontology concerns characteristics of the diagnostic problem solving task as an activity integrated in an organization.

Critical life cycle phase: a characteristic that indicates a degree of importance with respect to the organization's business goals.

Exponential/Marginal cost propagation: A characteristic that indicates an exponential/ marginal grow of cost through the product life-cycle with respect to the organization's business goal in case an error is produced during the problem solving process.

Performance requirements: this category of workplace ontology concerns attributes of processes.

Fidelity/Efficiency/Precision/etc.

Nature of the device being reasoned about:

Stable/Fragile design/Short product life-cycle

Expected faults and fault characteristics:

Intermittent/Catastrophic failures

Structural faults/Functional component fault/Connection faults/Timing faults

Business goals:

Low-cost manufacturing/High product quality

Fig. 3 Some examples of the workplace ontology.

4. Task ontology design

This section presents a design methodology of task ontology. Although it is tuned for task ontology design, it is applicable to design of other activity-related ontologies. Expertise modeling is viewed as design of a simulator of domain expert's. Simulation consists of behaviors which are represented as verbs at verbal level, which suggests the modeling process be driven by analysis of verbs first followed by that of nouns(objects). We here propose a methodology named activity-first method(AFM) for ontology design mainly for activity-related ontologies including the task ontology. A justification to this approach is drawn from characteristics of natural language in which verbs governs the other constituents in the sentence structure. A rough sketch of AFM is presented here.

After collecting simple sentences describing problem solving activities in the task of interest, verbs are identified, and then, discrimination between the domain-dependent knowledge and task-dependent knowledge is done. As was discussed in section 3.3, a verb is modeled as a domain-independent computational mechanism associated with domain knowledge necessary for the execution. The discrimination is done by identifying domain knowledge required by the verbs for its execution based on the verb model.

Next, verbs are refined until obtaining those of appropriate grain sizes. The clear discrimination between task and domain knowledge, abstraction level, and grain size are the critical. The first is done relatively easy in the above phase. The last two are not independent of each other. A general guideline to ontology design is that the result should be general enough for reuse and specific enough for characterizing the task structure.

Refinement of a verb is terminated when the domain knowledge required by it 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 because they include chunks of procedures which specify how to ***"identify"*** and ***"select"***. So, they need further refinement. But, as in work shift scheduling of nurses,***"to select the nurse of the heaviest load"*** is of a good grain size, since this ***"selection"*** can be clearly defined by specifying a criterion function which calculates the 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 simple classification task because all the necessary symptoms are observed before hand. 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. If the symptoms have to be carefully observed one by one according to the plausible causes and context, further refinement is required.

Then, computational mechanism for each verb is identified. This phase should be performed in parallel with the above phase. Meaning of each verb is its computational mechanism. The simpler it is, the better.

Computational mechanism is essential for configuring the problem solving engine of the target task. A close attention should be paid not to design a mechanism of larger grain size containing domain knowledge within its code. Other domain-specific concepts are identified by investigating the objects and domain knowledge of verbs.

5. How ontology is used in MULTIS

One of the goals of MULTIS project is to develop a task analysis interview system which enables domain experts or junior knowledge engineers to build executable task models. MULTIS employs cut&paste-based friendly interface and CBR(Case-Based Reasoning). MULTIS with scheduling task and T-domain ontologies first tries to obtain several basic domain-specific terms by using the template of schedule representation such as gantt chart, time table, etc. Rows, columns and entries of them corresponds to time, schedule resources and schedule recipients, respectively. Thus, at least three domain-specific concepts are easily obtained through a simple interaction.

MULTIS can present the user cases(GPNs, see section 3.3) in several ways; one in terms of task ontology, another in terms of the domain concepts(T-domain ontology) of the particular domain by consulting the subsumption relations between task ontology and T-domain ontology shown in Fig. 2. The translation between task ontology and domain concepts is made very smoothly. This helps domain experts understand what the GPN is and how to modify cases to build their own network. This GPN modification process can be augmented by providing justification and modification guidelines based on workplace ontology discussed above. After GPN design has been terminated, GPN compiler compiles it to generate a Common lisp code by configuring building blocks by asking 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 range of backtracking phenomena observed in scheduling tasks.

6. Conclusion

We have discussed typology of ontology together with some examples of them. A methodology for activity-related ontology design is also discussed. Task ontology of MULTIS has been evaluated by describing task structures of several expert systems which evaluators, members of a 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.We are currently developing a support system for building reusable knowledge bases based on knowledge decompilation and ontology design. A preliminary evaluation of the supporting methodology using real substation operation task have shown it is very promising[Takaoka,1994]. Ontology designed from the problem solving perspective is not very general, though it enables very efficient organization of knowledge. This implies the methodology is inefficient for designing ontology of something which has nothing to do with problem solving. Future work includes to refine the ontology implementation and to augment the task ontology to include other types of task such as diagnosis and design.

References

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

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

[Gruninger, 1994] Gruninger, R. and M. Fox: The design and evaluation of ontologies for enterprise engineering, Proc. of Comparison of implemented ontology, ECAI'94 Workshop, W13, pp.105-128, 1994.

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

[Hori, 1994] Hori, M. and Y. Nakamura: Reformulation of problem-solving knowledge via a task-general level, Proc. of the Third Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop: JKAW'94, pp.1-15, 1994.

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

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

[Mizoguchi, 1992] Mizoguchi, R. et al.: Task ontology and its use in a task analysis interview systems -- Two-level mediating representation in MULTIS --, Proc. of the Second Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop: JKAW'92, pp.185-198, 1992.
[Mizoguchi, 1993] Mizoguchi, R.: Knowledge acquisition and ontology, Proc. of the KB&KS'93, Tokyo, pp. 121-128, 1993.
[Mizoguchi, 1995] Mizoguchi, R. and Johan Vanwelkenhuysen: Task ontology for reuse of problem solving knowledge, Proc. of KB&KS'95, pp.46-59, 1995.
[Nishida, 1993] Nishida, T. and H. Takeda: Towards the knowledgeable community, Proc. of the KB&KS'93, pp.157-166, 1993.
[Sasajima, 1995] Sasajima, M., et al.: FBRL: A Function and behavior representation language, Proc. of IJCAI-95, 1995(to appear).
[Takaoka, 1994] Takaoka, Y. et al.: Towards a methodology for identifying ontologies for building reusable knowledge bases. Proc. of the Third Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop: JKAW'94, pp.49-63, 1994.
[Tijerino, 1990] Tijerino, A. Y. et al: A task analysis interview system that uses a problem solving model, Proc. of JKAW'90, pp.331-344, 1990.
[Tijerino, 1993] 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.
[Vanwelkenhuysen, 1994] Vanwelkenhuysen, Johan and R. Mizoguchi: Maintaining the workplace context in a knowledge level analysis, The Third Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop: JKAW'94, pp.33-47, 1994.
[Vanwelkenhuysen, 1995a] Vanwelkenhuysen, J. and R. Mizoguchi: Ontologies and guidelines for modeling digital systems, Proc. of KAW'95, Banff, 1995.
[Vanwelkenhuysen, 1995b] Vanwelkenhuysen, J. and R. Mizoguchi: Workplace-Adapted Behaviors: Lessons Learned for Knowledge Reuse, Proc. of KB&KS '95, pp.270-280, 1995.
[Wielinga, 1992] Wielinga, B. et al.: The KADS knowledge modelling approach, Proc. of JKAW'92, pp.23-42, 1992.
[Wielinga, 1993] 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 the task ontology for scheduling tasks is shown. The total number of lines is about two thousands and five hundred. Some classes are introduced according to the implementation level requirements. ?$xxx stands for a variable taking a set for its value. Domain-knowledge and constraint/ condition are synonyms.

```
;;Generic Noun
(define-class generic-noun (?generic-noun)
  :axiom-def
    (and
     (subclass-of
      generic-noun
      generic-vocabulary)
     (subclass-partition
      generic-noun
      (setof schedule
             constraint
             rsc
             rcp
             schedule-representation
             goal-gn
             data/information))))

;;RSC
(define-class rsc (?rsc)
  :axiom-def
    (and
     (subclass-of rsc generic-noun)
     (subclass-partition
      rsc
      (setof material-rsc
             human-rsc
             vehicle-rsc
             facility-rsc
             space-rsc
             others)))
  :def
    (and
     (individual ?rsc)
     (exists ?sch-rep
       (and
         (schedule-representation ?sch-rep)
         (schedule-representation.vertical
          ?sch-rep
          ?$rsc)
         (member ?rsc ?$rsc)
         (value-type ?rsc rsc-attribute
          attr-value)))))

(define-class facility-rsc (?f-rsc)
  :axiom-def
    (and
     (subclass-of facility-rsc generic-
           noun)
     (subclass-partition
      facility-rsc
      (setof machine-rsc
             blast-furnace-rsc
             tank-rsc
             oscilloscope-rsc
             others)))
  :def
```

```
    (and
     (value-type
      ?f-rsc
      facility-rsc.production-efficiency
      prod-efficiency)
     (value-type
      ?f-rsc
      facility-rsc.maximal-production
      max-prod)
     (value-type
      ?f-rsc
      facility-rsc.production-time
      time-unit)
     ...))

;;RCP
(define-class rcp (?rcp)
  :axiom-def
    (and
     (subclass-of rcp generic-noun)
     (subclass-partition
      rcp
      (setof material-rcp
             human-rcp
             vehicle-rcp
             facility-rcp
             space-rcp
             job-rcp
             order-rcp
             others)))
  :def
    (and
     (individual ?rcp)
     (exists ?sch-rep
        (and
          (schedule-representation
           ?sch-rep)
          (schedule-representation.contents
           ?sch-rep
           ?$rcp)
          (member ?rcp ?$rcp)
          (value-type ?rcp
                      rcp.attribute
                      attr-value)))))

;assignment
(define-class assignment (?assignment)
  :axiom-def
    (subclass-of assignment generic-noun)
  :def
    (and
     (value-type
      ?assignment
      assignment.rcp rcp)
     (value-type
      ?assignment
      assignment.rsc rsc)
     (value-type
      ?assignment
      assignment.time-unit time-unit)))

(define-class assignment-set (?$ass-set)
  :def
    (and
     (set ?$ass-set)
     (forall ?assignment
        (=> (member ?assignment
                    ?$ass-set)
            (assignment ?assignment)))))

;;Generic Verb
(define-class generic-verb (?generic-verb)
  :axiom-def
    (and
      (subclass-of
       generic-verb
       generic-vocabulary)
      (subclass-partition
       generic-verb
       (setof rsc/rcp-verb
              constraint-verb)))
  :def
    (and
      (value-type
       ?generic-verb
       verb.function v-function)
      (value-type
       ?generic-verb
       verb.input  tuple-set)
      (value-type
       ?generic-verb
       verb.output tuple-set)
      (value-type
       ?generic-verb
       verb.dk domain-knowledge)
      (verb.function
       ?generic-verb ?v-function)
      (verb.input
       ?generic-verb ?$input-tuple)
      (verb.output
       ?generic-verb ?$output-tuple)
      (= (v-function.input ?v-function)
         ?$input-tuple)
      (= (v-function.output ?v-function)
         ?$output-tuple)))

(define-class v-function (?v-function)
  :def
    (and
      (set ?v-function)
      (forall ?tuple
         (=> (member ?tuple ?v-function)
             (list ?tuple)))
      (forall (?tuple1 ?tuple2)
         (=> (and
               (member ?tuple1 ?v-function)
               (member ?tuple2 ?v-function)
               (= (butlast ?tuple1)
                  (butlast ?tuple2)))
               (= (last ?tuple1)
                  (last ?tuple2))))))

(define-class tuple-set (?tuple-set)
  :def (and
         (set ?tuple-set)
           (forall ?tuple
              (=> (member ?tuple
                          ?tuple-set)
                  (list ?tuple)))))

(define-function v-function.input
           (?v-function) :-> ?$input-tuple
  :lambda-body
    (cond ((instance-of
            ?v-function v-function)
           (setofall (first ?tuple)
                     (member  ?tuple  ?v-fun
ction)))))

(define-function v-function.output
           (?v-function) :-> ?$output-tuple
  :lambda-body
    (cond ((instance-of
            ?v-function v-function)
           (setofall (last ?tuple)
                     (member  ?tuple  ?v-fun
ction)))))

(define-class domain-knowledge (?dk)
  :axiom-def
    (and (subclass-of
          domain-knowledge
          constraint/condition)
         (subclass-of
```

```
          domain-knowledge
          constraint-vocabulary)))

(define-class rsc/rcp_verb (?rsc/rcp_verb)
  :axiom-def
    (and
     (subclass-of
      rsc/rcp_verb
      generic-verb)
     (subclass-partition
      rsc/rcp_verb
      (setof generate-verb
             test-verb
             modify-verb))))

(define-class generate-verb (?gv)
  :axiom-def
    (and
     (subclass-of
      generate-verb
      rsc/rcp_verb)
     (subclass-partition
      generate-verb
      (setof assign
             classify
             select
             others))))

(define-class assign (?assign)
  :axiom-def
    (subclass-of assign generate-verb)
  :def
    (and
     (verb.function ?assign ?assign-f)
     (forall ?tuple
        (exists
           (?input-tuple ?output-tuple
            ?cell)
           (=> (member ?tuple ?assign-f)
               (and
                (= (first ?tuple)
                   ?input-tuple)
                (= (last  ?tuple)
                   ?output-tuple)
                (= (nth ?input-tuple 1)
                   ?rcp)
                (= (nth ?input-tuple 2)
                   ?rsc)
                (= (nth ?input-tuple 3)
                   ?time-unit)
                (rcp ?rcp)
                (rsc ?rsc)
                (time-unit ?time-unit)
                (cell ?cell)
                (cell.rsc ?cell ?rsc)
                (cell.time-unit
                 ?cell
                 ?time-unit)
                (unassigned ?rcp)
                (available ?cell)
                (= (nth ?output-tuple 1)
                   ?assignment)
                (assignment ?assignment)
                (assignment.rcp
                 ?assignment ?rcp)
                (assignment.rsc
                 ?assignment ?rsc)
                (assignment.time-unit
                 ?assignment ?time-unit))))
)))

(define-class select (?select)
  :axiom-def
    (subclass-of select generate-verb)
  :def
    (and
     (verb.function ?select ?select-f)
     (verb.dk ?select ?dk)
     (forall ?tuple
      (exists (?input-tuple ?output-tuple)
         (=> (member ?tuple ?select-f)
             (and
              (= (first ?tuple)
                 ?input-tuple)
              (= (last  ?tuple)
                 ?output-tuple)
              (= (nth ?input-tuple 1)
                 ?input)
              (set ?input)
              (= (nth ?output-tuple 1)
                 ?output)
              (set ?output)
              (subsetof ?output ?input)
              (satisfies-constraint
                (listof ?input ?output)
                ?dk)))))))

;;generic-adjective
(define-relation assigned (?assigned)
  :axiom-def
     (subrelation-of
      assigned
      generic-adjective)
  :def
    (and
     (rcp ?assigned)
     (exists (?$assignment-set ?assignment)
        (and
          (assignment-set ?$assignment-set)
          (member ?assignment
                  ?$assignment-set)
          (assignment.rcp ?assignment
                          ?assigned)))))

(define-relation available (?available)
  :axiom-def
      (subrelation-of  available generic-
adjective)
  :def
     (and
       (cell ?available)
       (forall ?$assignment-set
          (not (exists ?assignment
            (and (assignment-set
                  ?$assignment-set)
                 (member ?assignment
                         ?$assignment-set)
                 (cell.rsc ?available
                           ?cell-rsc)
                 (cell.time-unit
                  ?available
                  ?cell-time-unit)
                 (assignment.rsc
                  ?assignment
                  ?cell-rsc)
                 (assignment.time-unit
                  ?assignment
                  ?cell-time-unit)
                   ))))))
```