Task ontology: Ontology for building conceptual problem solving models

Mitsuru IKEDA[1] Kazuhisa SETA[1] Osamu KAKUSHO[2]
Riichiro MIZOGUCHI[1]

Abstract. We have investigated the property of problem solving knowledge and tried to design its ontology, that is, Task ontology. The main purpose of this paper is to illustrate a Conceptual LEvel Programming Environment (named CLEPE) as an implemented system based on Task ontology. CLEPE provides three major advantages as follows. (A) It provides human-friendly primitives in terms of which users can easily describe their own problem solving process (descriptiveness, readability). (B) The systems with task ontology can simulate the problem solving process at an abstract level in terms of conceptual level primitives (conceptual level operationality). (C) It provides ontology author with an environment for building task ontology so that he/she can build a consistent and useful ontology. In this paper, firstly we briefly introduce the concept of task ontology. Secondly, CLEPE and its design principle is described. In CLEPE, one can represent his/her own problem solving knowledge and realize th*e conceptual-level execution.*

1. Introduction

Knowledge becomes usable and useful only when it fits to the use-context. This is the justification of the expert system technology whose power relies heavily on heuristic knowledge or expertise of domain experts rather than objective knowledge like domain theory. Expert system technology has obtained high performance at the cost of non-reusability of knowledge and low productivity of the knowledge base development. As is well-known, overcoming such difficulties has been the major goal in knowledge engineering community for years. The key ideas include decomposition of expertise into two kinds of knowledge: Task-dependent knowledge and domain-dependent knowledge. The early work on the former is found in [Chandrasekaran 86], [Clancey 85], [McDermott 88] followed by KADS[Wielinga 92], Protege [Puerta, 1992] and MULTIS[Mizoguchi 92],[Tijerino 93a] , [Mizoguchi 95]. The authors have proposed the concept of task ontology[Mizoguchi 92] to formalize the knowledge for problem solving domain-independently. Similar ideas have discussedextensively in knowledge acquisition community.

The term "task ontology" can be intepreted in two ways: (1) Task-subtask decomposition together with task categorization such as diagnosis, scheduling, design, etc. and (2) An ontology for specifying problem solving processes*. The latter shares the word usage with "domain ontology" which means an ontology of a domian and specifies concetps and relations appearing in a domain of interest. "Domain ontology" does not mean domain-subdomain decomposition. Likewise, by "task ontology", we here mean the latter, that is, an ontology specifying concepts and relations appearing in a task of interest.

Roughly speaking, one could imagine task ontology is a union or unification of diagnostic task ontology, control task ontology, design task ontology, scheduling task ontology and so on. It may be right. But, we do not discuss such a unified ontology in this paper, instead, we discuss the issue with a specific task in our mind in order to make the ontology more task-specific than "method ontology"[Heist 97][Chandrsekaran,98] which deals with rather general concepts independently both of domain and task.

One of the issues related to task-domain decomposition is: Although the decomposition contributes to reuse of knowledge, it is not easy to find the appropriate place in the problem solving process where each piece of domain knowledge should be put. Our task ontology research was initiated taking such an issue into account from the beginning. It includes two kinds of concepts related to activity and role of an object to which an activity applies. The latter represents the role which the object bound to plays during the problem solving process. The advantage of task ontology is that it specifies not only skeleton of the problem solving process but also context where domain concepts are used. This characteristic gives several remakable utility discussed below.

MULTIS, a task structure acquisition interview system, has been built based on task ontology [Mizoguchi 92][Tijerino 93] . 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. However, in MULTIS project, we just showed the possibility of task ontology but the whole ideas have not been formalized well. The formalization is needed badly to attain our research goal. The following four issues seem to be helpful in attempting sketch out our research on task ontology engineering.

A. Clarify the area of task ontology by specifying task sharing between a computer and a human.
B. Build task ontology.
C. Specify the relation between task ontology and general ontology.
D. Design a framework to bridge the gap between the model described based on the ontology and the runnable computational model.

The goals of our research on task ontology are to make problem solving knowledge explicit and exemplify its availability through the development of CLEPE: Conceptual LEvel Programming Environment. CLEPE provides three major advantages as follows. (A) It provides human-friendly primitives in terms of which users can easily describe their own problem solving process (descriptiveness, readability). (B) The systems with task ontology

[1] *ISIR Osaka University, 8-1, Mihogaoka, Ibaraki, Osaka, 567, Japan*
[2] *Faculty of Economics and Information Science, Hyogo University, 2301, Sihn-zaike, Hiraoka-machi, Kakogawa, Hyogo, 675-01, Japan*

We know the term "problem-solving ontology" is better than the term "task ontology" as a terminological point of view. But, we followed the convention word useage in the knolwedge-based systems community in which we call diagonsis, design, etc. a task.

can simulate the problem solving process at an abstract level in terms of conceptual level primitives (conceptual level operationality). (C) It provides ontology author with an environment for building task ontology so that he/she can build a consistent and useful ontology. In this paper we firstly discuss the basic issue on the concept of task ontology and then describe the design principle of CLEPE as a form of ontology use.

2. Conceptual Level Programming Environment

An ontology explicitly represents the meaning of concepts and the relation among them. To obtain a sophisticated ontology, we need a methodology for ontology construction. And we also need to demonstrate its effective use to convince people. Our final goal is to build an integrated enviroment for building and use of an ontology. As a first step, the goals of this research is mainly concerned with task ontology. In this context, CLEPE has been designed for both of development and use of ontology. From one aspect it is an environment to build the task ontology, and from the other it is an environment to describe one's problem solving knowledge in terms of the ontology.

The main role of task ontology author is to analyze the problem solving knowledge and to build the task ontology which can be easily acceptable to end-users . To support the ontology author's work, CLEPE provides Task Ontology representation Language (named TOL) and an environment for editing and browsing the ontology.

It is a quite time consuming work for end-users to describe their own problem solving processes in a rigid form. To lighten the load of end-users, it is important for task ontology to reflect their common conceptual understanding of problem solving. On the other hand, from computers standpoint, the description of the problem solving process should be rigid enough to specify the computational semantics. We may say that this conflict is a common problem of programming languages for end-user(s). The key to the problem is to shift task ontology close to users and to embody the function to fill the gap between end-users and computers. CLEPE has the ability to make up for the deficit of user's description and to reconstruct rigid specification of the computation.

2.1 Task Ontology

Now let us go into the detail of task ontology. Roughly speaking ontology is composed of two parts, that is, taxonomy and axioms. Taxonomy is a hierarchical system of concepts and axioms are established rules, principles, or laws among the concepts. From the viewpoint of the ontology use, axioms specify the competence of ontology. In other words, a class of the questions to which the answers can be derived from the axiom specifies the competence of the ontology.

Following the analogy of natural language processing, we can easily understand the role of task ontology as a system of semantic features to represent the meaning of the problem solving description. The advantages of the integration of task ontology into CLEPE is as follows:

A. Task ontology provides human-friendly primitives in terms of which users can easily describe their own problem solving processes (descriptiveness, readability).
B. The system can simulate the problem solving processes at the conceptual level and show users the execution process in terms of conceptual level primitives (conceptual level operationality).
C. The systemtranslates problem solving knowledge into symbol level code (symbol level operationality).

For the moment, it may be useful to look more closely at the functional feature of task ontology. Here, let us introduce three models M(A), M(B), and M(C), which embody the functions A, B, and C listed above, respectively. According to the analogy of natural language again, M(A) corresponds to the lexical level of natural language, M(B) is an internal model of intended meaning represented by the sentences, and M(C) has a capability to simulate the dynamic, concrete story implied by the sentences.

From now on, M(A), M(B) and M(C) are called "lexical level model", "conceptual level model", and "symbol level model", respectively. Lexical level model mainly deals with the syntactic aspect of the problem solving description, and conceptual level model captures conceptual level meaning of the description. Symbol level model corresponds to runnable program and specifies the computational semantics of the problem solving.

Table 1 shows a configuration of task ontology. Task ontology is composed of three layers. The top layer is called lexical level ontology (TO/K-L) in terms of which M(A) is represented. The middle layer is called conceptual level ontology (TO/K-C) which specifies

Table 1. Configuration of task ontology

Ontology	Vocabulary	Axiom
Task Ontology (TO)	**Vocabulary for representing problem solving process**	**Execution model based on correspondence between TO/K and TO/S**
Knowledge Level Ontology (TO/K)	**Vocabulary for representing conceptual level execution process**	**Correspondence between TO/K-L and TO/K-C (pragmatic meaning),** **Conceptual level execution model**
LexicalLevel Ontology (TO/K-L)	**Generic vocabulary (generic noun, generic verb, generic adjective,etc.)** **Nodes and links constitute GPN**	**Syntactic rules in generic process** **Meaning of generic process (syntactic meaning)**
Conceptual Level Ontology (TO/K-C)	**Vocabulary stands for objects and task activities**	**Effects as meaning of activity**
Symbol Level Library (TO/S)	**(Program components at symbol level)**	**Axiom related to execution process of task based on symbol level computational semantics**

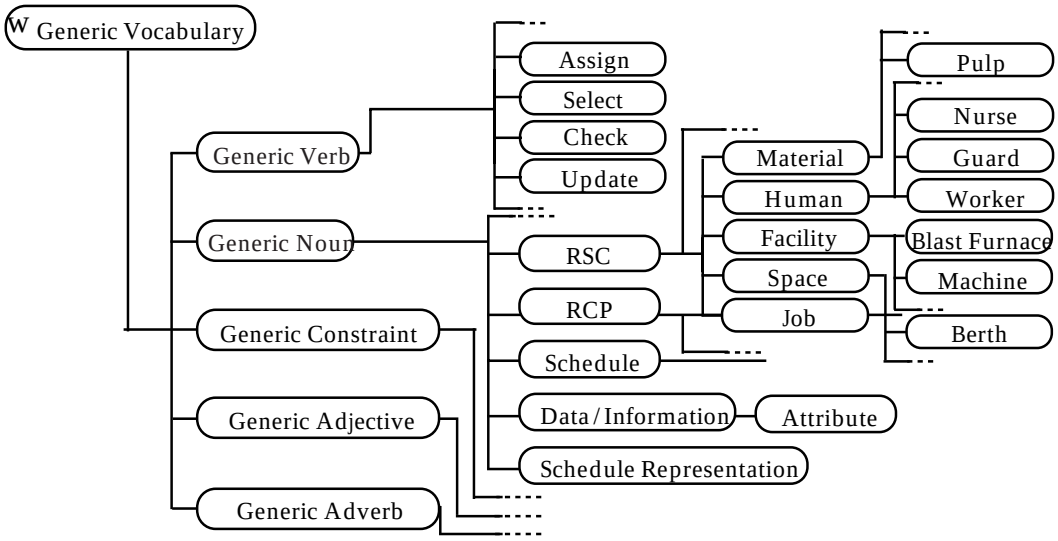

Figure 1. Part of lexical level ontology

computational semantics of M(B). Knowledge level ontology is composed of general terms for these two ontologies. Lexical level ontology specifies the language in terms of which end-users externalize their own knowledge of the target task, while conceptual level ontology is an ontology which represents the contents of knowledge in their minds.

Figure 1 shows a hierarchy of lexical level ontology. All the concepts of lexical level ontology are organized into word classes, such as, generic verb, generic noun, generic adjective etc. The determination of the abstraction level of task ontology requires a close consideration on granularity and generality. Representation of two sentences with the same meaning in terms of lexical level ontology should be the same. These observations suggest lexical level 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 words specific to the task.

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

Generic Nouns: Schedule recipient, Schedule resource, Due date, Solution representation, Constraints, Goal, Priority,
Generic Verbs: Assign, Classify, Pick up, Select, Relax, Neglect
Generic Adjective: Unassigned, The last, etc.
Others: Strong constraint, Constraint predicates, Constraint attributes, etc.

In the conceptual level ontology, the concepts to represent our perception of problem solving are organized into generic concept class, such as, activity, object, status, and so on. There are some relations among the two worlds, i.e., lexical world and conceptual world. Intuitively generic verb, generic noun, and generic adjective in lexical world correspond to activity, object, and status in the conceptual world, respectively. TO/S is a collection of symbol level CLOS code fragments. 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 two major objects in the scheduling task domain and its roles. One of the most important characteristics of task ontology is that meanings of verbs are also defined at the symbol level, that is, at least one executable code is associated with each verb to enable semiautomatic generation of runnable problem solving engine for the target task.

Figure 2 shows an image of interface for users. The network in the figure is called Generic Process Network (GPN). GPN represents user's problem solving process in terms of lexical level ontology. Each node of the GPN is separated into two parts. The upper part represents a concrete process in terms of natural language, and the lower is a generic process which is a task ontology translation of the upper part. Generic processes are represented in terms of generic terms which are defined on the lexical level task ontology (TOK/-L). Basically the generic process has a form as follows:

Generic process = Generic verb + Generic noun

Typical examples includes classify-schedule_resource(RSC), sequence-schedule_recipient(RCP), pickup-RCP, select-RSC, assign-RSC_schedule_RCP, update-priority, relax-constraint, and so on. A generic term, which acts as component of generic process, is the smallest concept of lexical level task ontology concepts.

The author of Generic Process Network (GPN) firstly inputs the upper part of GPN node and then translates it into generic process. The link of GPN represents the control flow of problem solving. A GPN can be thought of as task flow defined in terms of general, reusable component that describes meaningful stages of the problem solving process.

A GPN does not represent data flow explicitly, though it manages it implicitly and presents it to the user during the interpretation process to verify the correctness of the GPN built. The GPN is used by the GPNC (Generic-Process-Network Compiler) to generate code of the problem solving process.

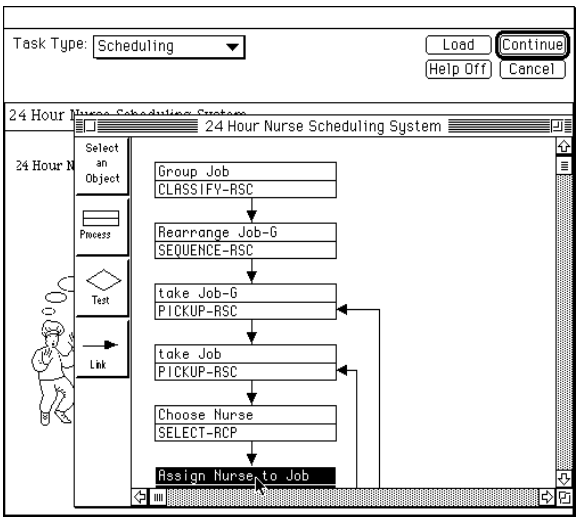

Fig.2 A screen image of CLEPE

2.2 Task ontology representation

We have been developing a language TOL, a Task Ontology representation Language. Before going into TOL specification, a few remarks should be made concerning general requirements for ontology representation language.

When we build an ontology, it is important to see the target world from a viewpoint of one's purpose. In our research on task ontology, there are two viewpoints, that is, task-type independent one and task-type specific one. Task type is a kind of categorization of tasks, for example, scheduling task-type, bookkeeping task-type and so on. In general, conceptual recognition of problem solving and the vocabulary used for describing it largely depend on task-types. Hence we cannot build human friendly ontology if we ignore the task-type specific characteristics. On the other hand, we have to note the task-type independent viewpoint is also very important to capture general problem solving concepts and to make problem solving knowledge more reusable.

We divide task ontology into two types, that is, task specific ontology (Task-S) and core task ontology (C-Task). Task specific ontology is an explicit description of task types specific characteristics. Core task ontology provides ontology authors with a set of task type independent primitives to build task specific ontology.

To embody these two viewpoints, TOL should allow ontology authors to represent relations between task specific ontology and core task ontology explicitly. To put the matter simply, core task ontology specifies the semantics of task specific ontology. We think separation of core task ontology and task specific ontology is a key to task ontology engineering. In this paper, we use the terms OA(C-Task) and OA(Task-S) to refer to the core task ontology author and task specific ontology author, respectively.

2.3 Examples of task ontology description

The correspondence relation between a concept of conceptual level ontology and one of lexical level ontology, as has been suggested, is important for task ontology definition.

Take "a set of assignments" from scheduling task-type as an example.

Our conceptual understanding about "a set of assignments" in a problem solving context is that it runs through GPN changing its status successively, for example, "not completed" at first, then "completed but inappropriate" and finally "appropriate". When we externalize this understanding, we would choose one word from three different words, that is, "partial solution", "temporary solution", and "final solution", according to the status of the object.

The point of this example is that, the single conceptual level object could be represented by more than one different lexical level terms in GPN.

The upper part of Figure 3 shows a definition of a generic noun class of "temporary solution" included in lexical level ontology. Note that the reserved keywords of lexical level ontology and conceptual level ontology definition are specified by core task ontology. "Define-Tol-noun", for example, is a reserved word used for generic-noun class definition. The necessary slots in the body of Define-Tol-noun are also specified by C-Task ontology. By ":cor-object", for example, correspondence between lexical level ontology and conceptual level ontology needs to be specified in the body of Define-Tol-noun form. "temporary-solution" is defined as a generic-noun of lexical level task ontology. The meaning of the body of the definition is "it is a subclass of assignment-set (:class-hierarchy), the class of the corresponding object should be O-assignment-set and the status of the object should be S-temporary (completed but inappropriate)".

On the other hand, the lower parts of Figure 3 shows the definition of the object of conceptual level task ontology. In the definition, we specify class hierarchy (:class-hierarchy), permanent property of the object (:object-spec), and set of status constraints by which the status of the object in a certain task context can be represented (:status-spec). "O-assignment-set" is defined as a class of conceptual level task ontology objects. The specification of the class is divided into two categories, that is,: object-spec and :status-spec. :object-spec specifies permanent property of objects, while :status-spec represents a list of state in which the object would get in a certain problem solving context.

```
(Tol-noun temporary-solution (?t-sol)
  :class-hierachy (subclass-of temporary-solution assignment-set)
  :cor-object (?O-ass-set :constraints (instance-of ?O-ass-set O-assignment-set))
  :status-spec ((S-temporary ?O-ass-set)))


(Tol-object O-assignment-set (?$O-ass)
  :class-hierachy (subclass-of assignment-set object)
  :object-spec (?$O-ass :constraints
                              (forall ?O-ass
                                  (=> (member ?O-ass ?$O-ass)
                                      (instance-of ?O-ass O-assignment))))
  :status-spec (?status-spec :constraints
                                   (member ?status-spec
                                          ((not (s-temporary ?$O-ass)) (s-partial ?$O-ass)
                                           (s-temporal ?$O-ass)(s-optimal ?$O-ass)))))
```

Figure. 3 Definition of a noun gtemporal-solution and object assignment-setÅh

We think that systematic organization of task ontology presented thus far could be a basic framework of ontology construction and use.

3. Design principle

CLEPE is a comprehensive environment on which two types of authors, that is, ontology authors and GPN authors, can work. The work of ontology authors is to write the task ontology definition in terms of TOL, as shown in Figure 3. In this section, however, we discuss the design principle of CLEPE only from GPN authors' point of view. In CLEPE, GPN author can describe his/her problem solving knowledge and observe the execution process in terms of plain words. We will discuss object flow analysis and conceptual level execution from functional aspects in the following so that readers can concretely capture what implicitness the system permits and how it deals with them.

To provide a human friendly environment for describing problem solving knowledge, computers need to be as close as possible to humans so that they can interpret the implicitness in problem solving knowledge.

Let us take an example of the implicitness in problem solving description.

The lack of human's consciousness of the objects to which a process takes effect is a source of the implicitness. When a GPN author puts a generic verb into a generic process, its input and output objects should be bound into the input and the output of the generic process, respectively. However the bindings cannot be always specified by a GPN author explicitly. For example, in case of a check process to check the termination condition of the loop for sequential scan of a set, input/output objects are often omitted in the description, because it is quite obvious for a GPN author, that is, "until the set is exhausted". This is a typical example of the lack of a human's consciousness of problem solving. They know it but don't write it explicitly. Having respect for user's consciousness of problem solving is a key to human friendliness of CLEPE.

Loss of the information caused by human's unconsciousness is compensated by the axiom of knowledge-level task-ontology (TO/K). In the case of the example above, CLEPE can derive the termination condition from the axiom on the pragmatics relation between pickup and check.

To make the implicit explicit, CLEPE analyzes GPN and try to reconstruct the object flow intended by a GPN author. The process is called object flow analysis. CLEPE employs a focus model for object flow analysis. Focus model models a context of anaphoric reference among objects based on syntactic information, effects of the verb, properties of noun, and structure of a GPN. Figure 4 shows a GPN and a focus model. Each focus represented by shaded ellipse in the Figure includes some objects created by prior processes. The third shaded ellipse from the top shows that an assignment is generated by the process 6, that is, "assign-RCP-to-RSC" and consists of a scheduling-resource(rsc) picked up by the process 4 and a scheduling recipient (rcp) selected by the process 5. Appearance and disappearance of objects depend on GPN structure. Focus model in the Figure illustrates that the objects created inside of the loop is disappeared outside and the assignment-set is created as an output of the whole loop.

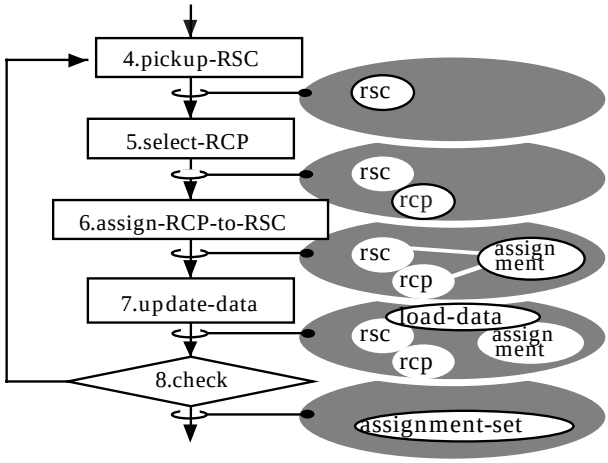

Figure.4 A part of GPN and corresponding focus

Once the GPN is built by users, CLEPE interprets it on the assumption that he/she completely agrees with ontological commitment. However, there might be a gap between the interpretation and the user's intention because the agreement is partial. In such a case, we have no choice but to expect the user to revise the GPN. To support the user's work, CLEPE provides conceptual level execution of GPN.

Advantages of conceptual level execution are as follows: (1) A user can recognize the difference between the meaning intended by him/her and system's interpretation. (2) A user and system can reach an agreement on the problem solving description more explicitly. In 4.2 we will give a detailed description of the conceptual level execution.

Ontology, in general, is an agreement between users and systems. Thus, the goal of ontology author is to build an ontology which can be easily accepted by most users. But in practice, we cannot ignore the gap between the meanings which users read into the terms and the semantics rigidly defined by ontology. As has been suggested, the gaps have to be ultimately filled in by users. It follows from what we discussed thus far that we should realize the existence of the gap and implement the function to support user's work of filling the gap. We think the function is essential to ontology engineering.

4. Conceptual Level Programming Environment

-Construction of CLEPE-

CLEPE supports both ontology authors who construct ontology and GPN authors who describe GPNs using the ontology.

In Figure 5, ontology author (OA) is arranged above side and GPN author (GPNA) left side. Thin planes stand for languages, e.g. the base and the left side correspond to the description level of CLOS.

Core task (C-Task) ontology is the task ontology independent of task types. Ideally Core task ontology should be constant and an ontology authors concentrate on building the task specific (Task-S) ontologies for new task types. CLEPE provides ontology authors with functions of editing and browsing both core task ontology, task specific ontology and symbol level ontology, because our current research interests include to fix the boundary between the three ontologies.

The main work of a GPN author are as follows: (1) To describe their own problem solving, (2) to make sure that his/her problem solving knowledge represented by GPN is correctly interpreted by the system, (3) to modify GPN if necessary.

Figure 6 shows the module structure of CLEPE. In this figure, rectangles and ellipses stand for the functional modules and data, respectively. Arrows linking modules stand for the data flow. In the following, we explain each module briefly.

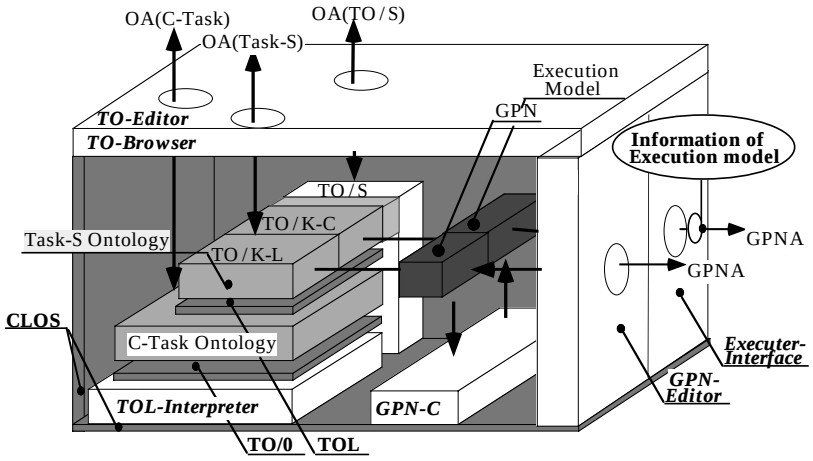

Figure.5 Overview of CLEPE

TOL-Parser parses the task ontology described with TOL.

Ontology Manager manages ontology base and deals with the requests for the class information or instance generation.

GPN-Parser parses a GPN

Model Generator generates conceptual level execution model by referring the object flow analyzed.

Working Memory Manager manages all the data related to the object flow analysis.

Constraint Generator generates syntactic or pragmatic constraints as results of object flow analysis.

Anaphoric Resolution Engine identifies anaphoric reference among objects based on the focus and constraints generated.

Focus Manager manages the focus which plays a key role in object flow analysis.

Executor interprets conceptual level execution model and shows the execution process to users.

In preparation for interpretation of the GPN written by a GPN author, CLEPE reads task ontology description represented with TOL. The task ontology description is translated into internal form by TOL-Parser and stored into ontology base. Ontology Manager manages the ontology base and deals with the requests related to the ontology made by other modules, for example, inquiries for class information, creation of a class instance and so on. Once a GPN author completes editing his/her own GPN, CLEPE initiates the GPN interpretation process. The functional modules in shaded portion of Figure 6 takes an internal form of GPN from GPN-Parser and generates the conceptual-level execution model. We call the shaded portion ARM: Anaphoric Resolution module. We adopt a focus model as a basic framework of anaphoric resolution. Focus manager updates the focus model dynamically based on the constraints generated by the other modules. Two types of constraints, that is, local constraints and global constraints, are generated in different manners. For each generic process, local constraints are composed based on the syntactic structure of the generic process and the ontological meaning of each word. After all the local constraints for the whole GPN are generated, global constraints are synthesized along the structure of the GPN by anaphoric resolution engine. It tries to find the consistent correspondence relations among the objects appeared in the GPN based on the local constraints and the latest focus model. Focus represents a set of objects which can be accessible from a certain generic process. Focus manager interprets constraints newly added into WM and infer whether the presence of objects referred to by the constraints could change or not. Once a consistent anaphoric relation

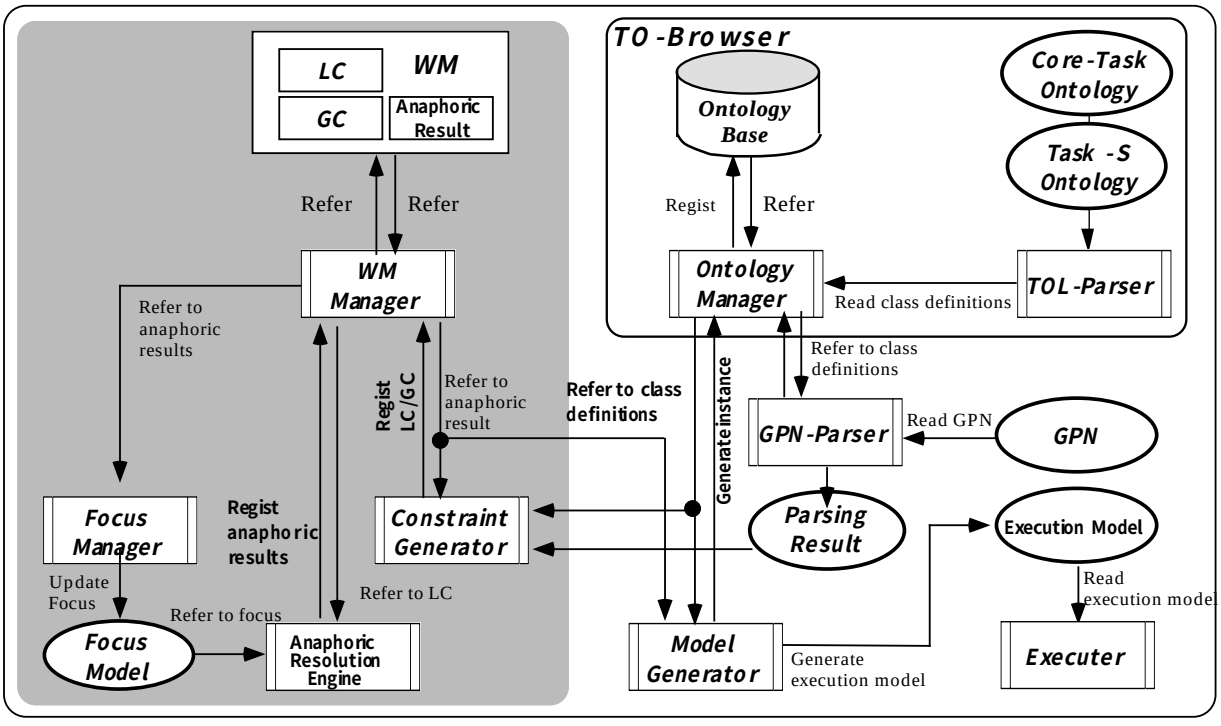

Figure. 6 Module structure of CLEPE

is established by ARM, the GPN is translated into conceptual level execution model. A GPN author can run the execution model with executor.

In the following sections, we describe TOL and conceptual level execution.

4.1 TOL: a language for describing task ontology

Figure 5 shows a hierarchy of ontology description language. TOL/0 at the bottom of the hierarchy provides description primitives for ontology author and defines semantics of upper-layer language.

Therefore all the semantics of task ontology described with TOL is specified ultimately at the level of TOL/0. To say concretely, TOL/0 specifies the meaning of generic concepts for describing problem solving knowledge and provides some primitives for constructing core task ontology.

Core task ontology author [OA(C-Task)] specifies the lexical entities and conceptual ones using TOL/0 primitives, e.g. Define-Tol-Core-Lexical-Word, Define-Tol-Core-Concept, etc. By reading the specification of core task ontology into CLEPE, a set of conceptual primitives at the TOL level is introduced. Task specific ontology author [OA(Task-S)] specifies the concepts appearing in the target task type with TOL. For example, the semantics of the verb "assign," which appears in the scheduling task type, is defined at TOL-level by using Define-Tol-Verb.

4.2 Conceptual level execution

Once CLEPE established the consistent object flow of a GPN by object flow analysis, a conceptual level execution model is generated. Object flow specifies the history of each object run through the problem solving model, for example, "when an object appears and disappears," or "how the status of the object changes". Since the history is described at knowledge-level task ontology (TO/K) level, a GPN author could easily understand the execution process of his/her own GPN. Related to this, there are some differences between an object constraint and a status constraint. The former is static and the latter is dynamic. Static membership is a permanent property of object while dynamic constraint depends on task context. For instance, the membership of a solution in the solution class is permanently true. On the other hand, a dynamic constraint of the optimality of a solution depends on the context of object flow. In the conceptual level execution model, an object is generated based on static membership and its history is represented by dynamic constraints. So execution at the level of conceptual level is defined as a history of the changes of objects. These two constraints are explicitly separated in the definition of task ontology.

Figure 7 shows a rough image of a conceptual level execution. The left side of the figure represents a problem solving knowledge (TO/K-L model) about a 24 hour job assignment task. In the TO/K-L description, the problem solving knowledge is described with a set of human friendly primitives. The right side shows the conceptual level execution model corresponding to the problem solving knowledge. Fragments headed by :sc and :dc are constraints inferred by object flow analysis. :sc and :dc fragments correspond to static membership and a dynamic constraint, respectively. The transition from the input objects to the output one of assign process shows that the output object is an instance of the assignment class and composed of the two objects which are the output of the "pick-up" process and one of the "select" process. In terms of the conceptual level vocabulary, we could say the role of "assign" process is to bind the "picked-up job" and "selected nurse" together and produce a new assignment. The assignment set in the rectangle represents the output, "partial solution," of loop structure.

One might say "I can't find any difference between the execution model itself as a result of object flow analysis and the conceptual level execution." The difference would be more clearer by considering the competence of "execution". The major difference between the model and execution is that the model captures the descriptive and static aspect of task structure, while the execution captures the substantial and dynamic aspect based on conceptual level computational semantics. At any time point during the execution, user can make inquiries about any event of the execution, for example,

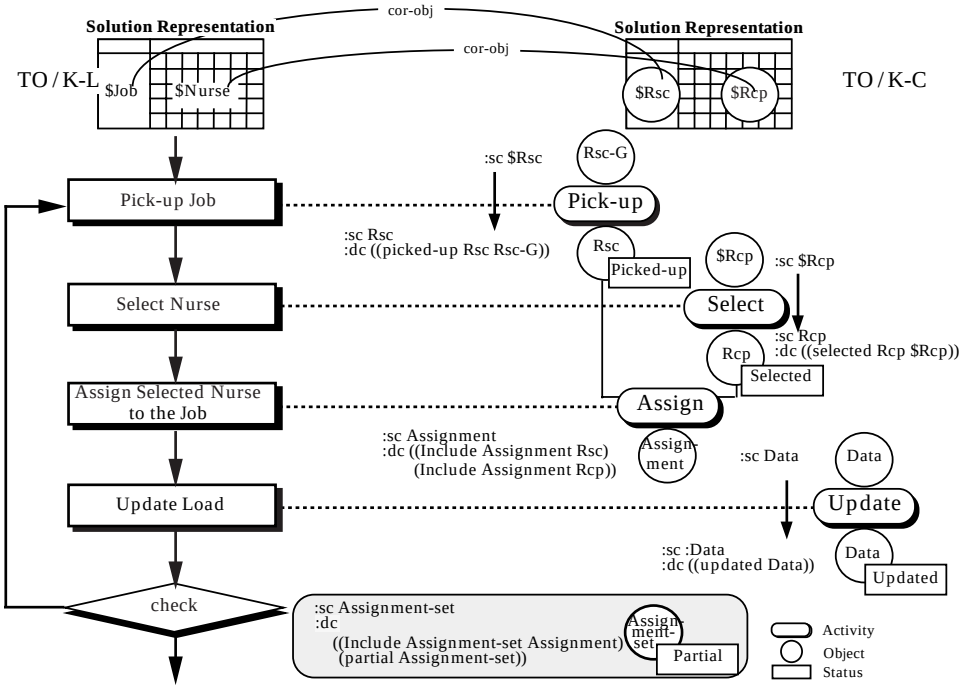

Figure 7 An image of conceptual level execution

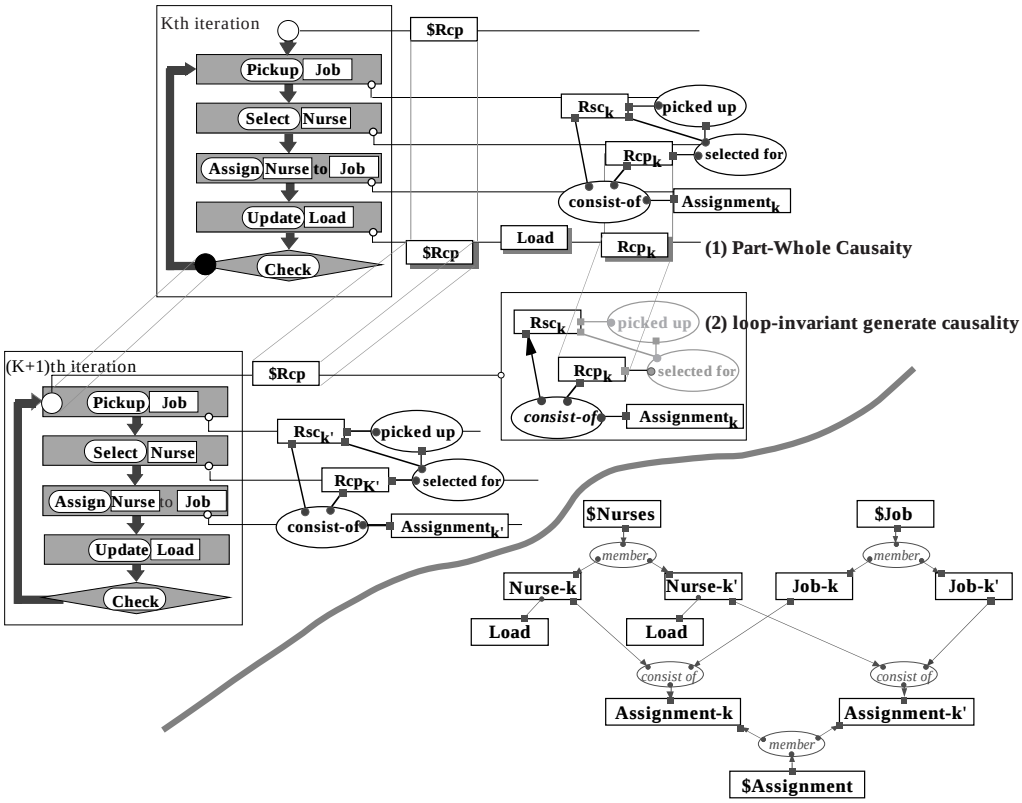

Figure 8 Examples of Problem Solving Causality

"What types of objects still remain (after running the pickup process) ?" or "What objects are generated now (after nth execution of the loop) ?" and system can answer the inquiries in the right situation.

By keeping the continuity from the symbol level program code to conceptual level model, CLEPE can give the conceptual level execution about the execution result at the symbol level.

5 Competence of Conceptual Model of Problem Solving

Task ontology consists of a variety of axioms which play the important role to realize most functions of CLEPE. Because of space limitation, here we will take up some of the axioms needed for conceptual level execution and show an example of conceptual problem solving model and its competence.

Conceptual level execution is a function which provides the trace information of execution process of GPN in the appropriate abstract level. The function reduces the load of the end-users' work while they are debugging the GPN. In general, an end-user using a conventional programming environment often feels uncomfortable, because the abstract level of the trace information such as real data is too low for them to matched it off against their understanding of problem solving. On the other hand, conceptual level execution provides end-users with the conceptual level information which can be easily mapped off against their understanding of the intended behavior of GPN. In the following, we introduce the concept of problem solving causality which plays most important role for generating appropriate information about the behavior of GPN .

The information provided by the conceptual level execution mainly concerns on how objects and the relation among them change during problem solving. An idea of 'version' of objects is introduced as a source of the information. Change of version represents when and how the change of an object or relation happened. Furthermore, changes of version are propagated over the model, for example, the version change of part of an object is propagated to the whole object. It reflects how end-users recognize the changes of objects in domain world. An important point, here, is that all the changes happened in domain world should not be reported to end-users, because too much information would bather them. Instead, the report should include only the information really useful for end-user to grasp problem solving behavior clearly. Problem solving causality is a set of axioms needed to realize this summarization function. Here, we show "(1) part-whole causality" and "(2) Loop-invariant generate causality" as examples.

Figure 8 shows object flow model (partially) and domain model corresponding to it.

In object flow model, all the effects of an activity at each step of GPN are represented. In the domain model which corresponds to the given task flow model, changes of domain objects caused by the activities and the changes of relations among the objects are also represented in terms of 'version.'

In figure 8, we discuss the relation among task flow and changes of domain world objects taking the part-whole causal relation as an example. Let's focus on the causal relation between the update process in kth iteration of the loop and the select process (which selects a nurse with minimum load from a set of nurses) in (k+1)th iteration. When update process updates the load data of the nurse who is assigned to a job in assign process in kth iteration, we can say the version of the nurse changes. In addition to this, the set of nurses including the nurse also changes its status. This is the case that the change of the part is propagated to the whole through the component-of relations among objects specified in the domain world. However, whether this propagation should be reported to end-users or not is a matter for argument on problem solving causality. Problem solving

causality answers the question based on whether the change is important or not from problem solving viewpoint. In this case, it is important because the change of the set of nurse guarantees correctness of the input to the select process in the succeeding iteration. Thus, when the select process is executed in (k+1)th iteration of the loop, conceptual level execution shows end-users that the input object of the select process is identical to one in kth iteration and the load data of all its members are appropriately updated by update process in kth iteration. By representing the changes of objects caused by task execution in terms of 'version of the domain object,' it is possible for CLEPE to explain the behavior of problem solving at arbitrary time in terms of appropriate expression. For example, concerning the roles of select process at every loop iteration, "the select process, through all the loop iteration, selects a nurse with minimal load from the set of the nurses whose load are adequately updated by update process in the last itelation of the loop."

Another example is the one we call loop-invariant generate causality. In preparation, let us consider the life of a relation among objects. In Figure 8, we can see two types of relations, that is, loop-temporal relations and permanent relations. 'Picked_up' binary relation for the Jobk is an example of loop-temporal relation. This appears when pickup process outputs the Jobk in kth iteration of the loop and disappears when the iteration is completed. The same thing is true for 'Selected for' relation for the Nursek.

The life span of the 'Picked_up' relation in the Pickup-Check loop structure is specified by the axiom of task ontology. The version maintenance function of CLEPE sets up the life span of each instance of the relation based on the axiom. In case of the 'Selected_for' relation, it becomes more complex. In the axiom related to Select process, there is no specification for the life span of the relation. Instead, the general axiom of task ontology says that "if a conceptual entity depends tightly on the other conceptual entities, the life span of them should be same as a general rule." Following the principle, 'Selected_for (Nursek,Picked_up(Jobk))' should disappear at the same time that 'Picked_up(Jobk) disappears, because Nursek is Selected_for 'Picked_up(Jobk).' Here, however, we should notice that the generic relations, Pickup_up(*) and Selected_for(*,*), form an invariant structure through the iteration of the loop.

On the other hand, the consist-of relation among Assignment, Job and Nurse is an example of permanent relation. There are two kinds of permanencies in our task ontology, that is, the problem solving permanency and the problem permanency. The former means that a conceptual entity remains throughout problem solving but disappears when completion. The latter mean that an entity never disappears to represent the results of problem solving as the consist-of relation in our example.

The life of each conceptual entities appearing in problem solving processes is maintained by the version management mechanism of CLEPE. The problem solving causality is specified in terms of the relation among the version changes of the conceptual entities.

'Loop-invariant generate causality' is specified as "if a portion of problem solving model generates permanent conceptual entities from loop-invariant ones, there may exist loop-invariant generate causality.*" In our case, the causal relation extracted by the causality is that "Assign process generates an Assignment which consists of Picked-up Job and Nurse Selected for the Job in each iteration of the loop.*

The Assignment is added to the Assignment-set which is the solution to the given problem.

As we can see in the above two examples, the problem solving causality can extract a meaningful set of relations from the large number of relations on the problem solving model. Without it, end-users would be bored with an incontinent talk of meaningless relations.

6 Capturing the Problem Solving Model

Problem solving causality is built in the task ontology as general relation among problem solving processes and objects. As we can see in the examples of the previous section, by using the problem solving causality, dynamics of problem solving processes is presented to end-users as not only the time series of computational operations but also meaningful causal relations among the changes of objects with keeping the correspondence between problem solving processes and domain concept. The presentation would be well acceptable to end-users because it appropriately reflects the epistemic characteristic of their understanding of the problem solving process. Thus, we could say that problem solving causality is one of the most important parts of task ontology as user model. In this section, we will exemplify the roles of the problem solving causality as a static user model.

6.1 Problem Solving Causality

Problem solving causality is causal relation among the parts of problem solving model. Once the object flow model corresponding to the given GPN is identified, CLEPE tries to find out causal relation underlying the problem solving model based on the ontology of problem solving causality and then build a conceptual level execution model. When CLEPE provides end-users with the trace information of conceptual level execution, the problem solving causality plays an important role as a basic agreement among end-users and CLEPE to share the common understanding of the problem solving process. The major role of the causality, in general, is to assign the role meaningful from problem solving viewpoint to each of the objects referred by the relation as arguments.

In general, the causal relation underlying a problem solving model is quite complicated and entangled. If one tries to draw the figure to show the causal relation of a certain problem solving model, he/she will find that it is too complicated to draw it on one plane. Thus, it is quite difficult for end-users to describe the causal relation explicitly by themselves, even if it is obvious for them. So, in order for end-users and CLEPE to share the common understanding of the problem solving model, we cannot expect that end-users to express their intention by themselves as input to CLEPE. Instead, CLEPE accepts rather simple description of problem solving process, such as GPNs, and then reconstructs the object flow model and reveals the causal relation underlying it based on task ontology. In practice, of course, the reconstruction task is not easy one even with aid of task ontology. To overcome the difficulty, CLEPE interacts with end-users to reveal end-users' real intentions of GPN. Nevertheless, if there still remains some gap between end-users' intention and CLEPE's understanding, CLEPE provides end-users with the conceptual level execution function and expects them to adapt (debug) their problem solving description to task ontology by themselves. In this sense, we believes that the conceptual level execution function, together with the reconstruction function of problem solving causality is indispensable one for end-user programming environment.

As we discussed in the previous section, it is desirable that the problem solving causality would be presented to end-user in domain-oriented manner, because end-users prefers domain-oriented representation to task-oriented one in general. To cope with domain oriented property of end-users' consciousness for the problem solving, the task-oriented representation of the causality, which is specified in task ontology, needs to be translated into domain-oriented representation. To embody such a hybrid representation, we introduced a task-domain binding (TD-binding) mechanism which act as glue to integrate the domain concepts into task context. For example, in Figure 1, we can say that the nurse of a domain concept

is integrated into the task context and assigned the role as a scheduling recipient (RCP). In this case, TD-binding binds the domain concept, nurse, and the task concept, RCP, together, and serves either meaning of nurse or one of RCP in compliance with requests.

6.2 Roles of Problem Solving Causality

Figure 5 shows an overview of conceptual level programming environment CLEPE. The processes represented by arrows marked with (1) (2) and (3) are end-user's work for describing GPN, object flow model construction, and conceptual level execution respectively. All the processes are supported by task ontology, for example, lexical task ontology specifies syntax of GPN description in (1), conceptual task ontology and TD-binding specifies meaning of the object flow model in (2). In (3), as we discussed thus far, problem solving causality plays important role. In the next section, we would like to focus attention on what concepts we should prepare as axiom of problem solving causality to realize the function of (3) and how CLEPE uses them .

Problem solving causality is specified as axioms among concepts of task ontology . In the conceptual level execution, CLEPE explicitly presents the process of how the domain objects changes through problem solving based on the causality. Thus end-users can easily understand the behavior of their own GPN by observing the conceptual level execution.

We skip detailed explanation of the axiom because of space limitation, however, instead, rough classification of the causality is shown below.

I. *Activity causality* which captures functional relation among activities, e.g. the activity as supplier provides the other activities as consumers with input objects.
II. Object causality which captures how objects are generated, e.g. relation among material objects (input objects of an activity) and product objects (output objects of the activity)
III. *Control causality* which captures causal relationship among activities based on control structure, e.g. loop structure.

Causality I, II are adapted to the model based on object flow model and III is based on the control structure of GPN. We report the problem solving causality in detail in other papers.

We explain the validity of task context through some examples of conceptual level execution and illustrate what a human friendly environment of execution is provided for end-users.

7. Concluding remarks

In this paper, we proposed a conceptual level programming environment based on task ontology. The system can answer for the continuity from the conceptual level description to problem solving and runnable code.

Both COMMET workbench as an embodiment of Components of Expertise [Steels 90] and SBF (Spark, Burn, Firefighter) [Mcdermott 88][Yost 95] are practical and sophisticated systems for end-users programming. However, knowledge level analysis has not been done in a systematic manner and ontological commitment assumed is not presented explicitly.

The most related work to our research is TOVE (Toronto Virtual Enterprise) project headed by Mark S.Fox [Fox 93],[Gruninger 95]. The idea concerning task ontology is almost same as ours. The important difference is that we think a great deal of the lexical aspect of ontology.

We are currently implementing CLEPE based on the design principle presented in this paper .

References

[Chandrasekaran 86] Chandrasekaran,B. Generic tasks for knowledge based reasoning: the right level of abstraction for knowledge acquisition, IEEE Expert, Vol.1, Not.3, pp.23-30, 1986.

[Chandrasekaran 98] Chandrasekaran, B. et. al. Ontology of Task and Methods, ECAI98 Workshop on Applications of Ontologies and Problem solving Methods, 1998.

[Clancey 85] Clancey, W.J. Heuristic classification, Artificial Intelligence, Vol.27, No.3, pp.289-350, 1985

[DeBellis 95] Michael DeBellis: User-Centric Software Engineering,IEEE EXPERT, Februrary (1995).

[Fischer 96] Fisher, G.: Seeding, Evolutionary Growth and Receeding: Constructing, Capturing and Knowledge in Domain-Oriented Environment, Domain Knowledge for Interface System Design,Chapman&Hall, pp.1-16 (1996)

[Fox 93] Fox, M.S., Chionglo, J., Fadel, F.: A Common-Sense Model of the Enterprise, Proc. of the Industrial Engineering Research Conference (1993)

[Mizoguchi 95] Mizoguchi, R., et. al., Task ontology for Reuse of Problem Solving Knowledge, Proc. of 2nd International Conference on Very L:arge-Scale Knowledge Bases, Tnschede, The Netherland, pp.46-59 (1995)

[McDermott 1988] McDermot. J. Using problem solving methods to impose structure on knowledge. Proceedings of the International Conference on AI Applications, pp. 7-11,(1988)

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

[Puerta, 1992] Puerta, A.R, et al., A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools, Knowledge Acquisition, 4, pp.171-196, 1992.

[Seta 96] Kazuhisa Seta, Mitsuru Ikeda, Osamu Kakusho, Riichiro Mizoguchi: Design of a Conceptual Level Programming Environment Based on Task Ontology, Proc. of Successes and Failures of Knowledge Based Systems in Real World Applications, p.p. 11-20 (1996).

[Steels 90] Components of Expertise, AI Magazine, Vol.11, No.2, pp. 28-49 (1990).

[Tijerino 90] Tijerino, A.Y. et al, (1990). A task analysis interview system that uses a problem solving model, Proc. of JKAW'90, pp.331-344

[Tijerino 93a] Tijerino A.Y. et al., MULTIS II : Enabling End-Users to Design Problem-Solving Engines Via Two-Level Task Ontologies, Proc. of EKAW '93, pp. 340-359, (1993)

[Tijerino 93b] Tijerino.A, Ikeda, M., Kitahashi, T., and Mizoguchi, R. (1993). A Methodology for Building Expert Systems Based on Task Ontology and Reuse of Knowledge, Journal of Japanese Society for Artificial Intelligence, Vol.8, No.4, pp.476-487 (in Japanese)

[Van Heijest 97] Van,Heist et.al., Using explicit ontologies in KBS development, Internatinal Journal of Human-Comupter Studies, Vol 47, pp.183-292 (1997)

[Wielinga 92] Wielinga, B.J.KADS: A modeling approach to knowledge engineering, J. of Knowledge Acquisition, Vol.4, No.1, pp.5-53 (1992)

[Yost 94] G.R.Yost et.al.:The SBF Framework,1989-1994: From Applications to Workplaces, Proc. of the EKAW-94, pp.318-339 (1994).