Design of a Conceptual Level Programming Environment Based on Task Ontology

Kazuhisa SETA*1 Mitsuru IKEDA*1 Osamu KAKUSHO *2 Riichiro MIZOGUCHI *1

{seta,ikeda,miz}@ei.sanken.osaka-u.ac.jp

kakusho@humans-kc.hyogo-dai.ac.jp

*1 ISIR Osaka University

Tel. +81 6 879 8416, Fax. +81 6 879 2123

*2 Faculty of Economics and Information Science, Hyogo University

Tel. +81 794 24 0052, Fax. +81 794 26 2365

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 the conceptual-level execution.

***Keywords*:** task ontology, conceptual level programming environment, ontological commitment

1. Introduction

An expectation that the concept of ontology would play a very important role to realize knowledge sharing and reuse is now widely shared by many researchers in the field. At the current state of the art, however, we only have made one step forward towards methodology and theory of ontology engineering [Mizoguchi 96].

Roughly speaking, subject of ontology research could be divided into two categories. One is domain ontology and the other is task ontology. Domain ontology is a system of the concepts organized independently of its usage. On the other hand, task ontology is a system of vocabulary for describing problem solving structure of all the existing tasks domain-independently. Task ontology provides us with a view of the world situated by problem solving context. It makes the meaning of the problem solving knowledge explicit. One can describe ones own problem solving knowledge using task ontology. Needless to say, the boundary between them is quite vague. However, when we closely look at the usage of task ontology, the distinctive feature of it becomes clear.

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 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

In general, the aim of ontology research is to explicitly represent the meaning of concepts and the relation among them. To attain the goal, sophisticated methodology for ontology construction and use is strongly desired.

Our final goal is to build a methodology for building ontology. As a first step, the goals of this research are to lay a foundation for building task ontology and to illustrate its use. To the ends, CLEPE has two aspects. 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.

When designing CLEPE we should notice that there are two types of users. One is an *ontology author* (OA)

who builds ontology of problem solving, and the other is an end-*user* who describes his/her own problem solving using the ontology. Generally ontology specifies an agreement between a user and a system on conceptualization. The main role of OA is to analyze the problem solving knowledge and to build the task ontology which can be naturally acceptable to end-users . To support OA'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 somewhat rigid form. To lighten the load of end-users, it is important for task ontology to reflect their common perception of problem solving. On the other hand, from computers standpoint, the description of problem solving 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 the deficit of user's description and to reconstruct rigid specification of the computation.

The remarkable features of CLEPE are summarized in the following.

I. End-users can describe their own problem solving using human friendly primitives.

II. End-users can observe task execution process and debug their own description at the conceptual level.

III. Continuity from user's description of problem solving to computational semantics is maintained.

2.1 Task Ontology

Now let us go into the detail of task ontology. Generally ontology is composed of two parts, that is, taxonomy and axioms. Taxonomy is an ordered 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 [Mizoguchi 96]. Task ontology is an ontology of the target task organized from the viewpoint of problem solving [Tijerino 93].

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 system makes problem solving knowledge runnable by translating it 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 sentences 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

Ontology	**Taxonomy**	**Axiom**
Task Ontology (TO)	Vocabulary for representing execution process of task	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 compose of GPN	Syntactic rule 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 Ontology (TO/S)	Program components at symbol level (BB)	Axiom related to execution process of task based on symbol level computational semantics

Table1. Configuration of task o

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.

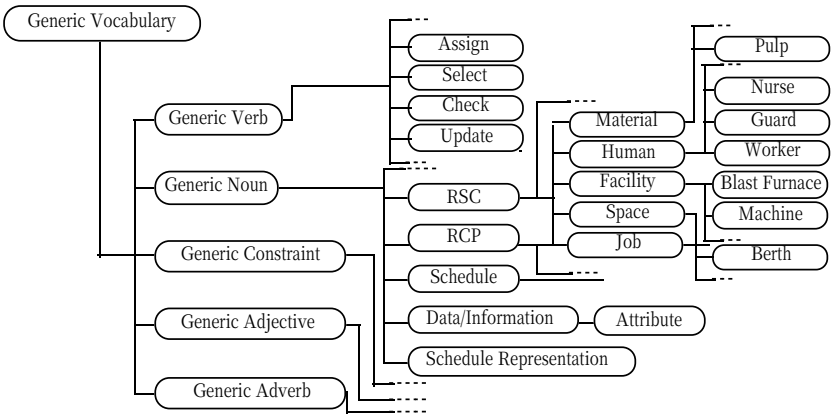

Fig.1 A hierarchy of terms in TO/K-L (Par

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 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. 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.

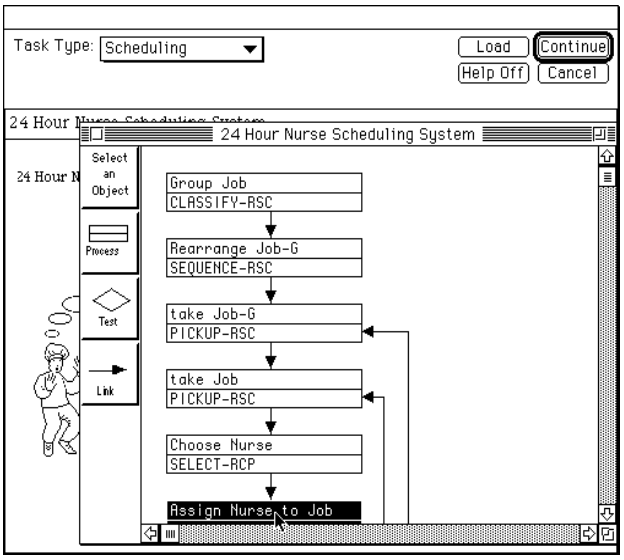

Fig. 2 A screen image of

Figure 2 shows an image of interface for users. The network in the figure is called Generic Process Network (GPN). GPN represents users 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 TO/K-L translation of the upper part. A generic term, which acts as component of generic process, is the smallest concept of TO/K-L concepts. The author of GPN (GPNA) 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.

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, book-keeping 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 (Task-S) ontology and Core Task (C-Task) ontology. Task-S ontology is an explicit description of task types specific characteristics. C-Task ontology provides OA with a set of task type independent primitives to build Task-S ontology.

To embody these two viewpoints, TOL should allow OA to represent relations between Task-S ontology and C-Task ontology explicitly. To put the matter simply, C-Task ontology specifies the semantics of Task-S ontology.

We think separation of C-Task ontology and Task-S 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 C-Task ontology author and Task-S ontology author, respectively.

2.3 Examples of task ontology description

The correspondence relation between TO/K-C concept and TO/K-L one, 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”, “temporal solution”, and “final solution”, according to the status of the object.

The point of this example is that, the single TO/K-C object could be represented by more than one different TO/K-L terms in GPN.

The upper part of Fig. 3 shows a definition of a TO/K-L generic noun class of “temporal solution”. Note that the reserved keywords of TO/K-L and TO/K-C definition are specified by C-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 TO/K-L and TO/K-C needs to be specified in the body of Define-Tol-noun form. “*temporal-solution*” is defined as a generic-noun of TO/K-L. 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-temporal (completed but inappropriate)”.

On the other hand, the lower parts of Fig. 3 shows the definition of the concept of object (TO/K-C). 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 TO/K-C 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.

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, OA and GPNA, can work. In this section, however, we discuss the design principle of CLEPE only from GPNA’s point of view, because of space limitation. In CLEPE, GPNA can describe his/her problem solving knowledge and observe the execution process in terms of plain words. We discuss object flow analysis and conceptual level execution from functional aspects as follows so that readers can concretely capture what implicitness the system permits and how it deals with them.

3.1 Implicit description in a problem solving knowledge

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 GPNA 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 GPNA

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


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

Fig. 3 Definition of a noun "temporal-solution" and object

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 GPNA, 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 GPNA. 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. 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.

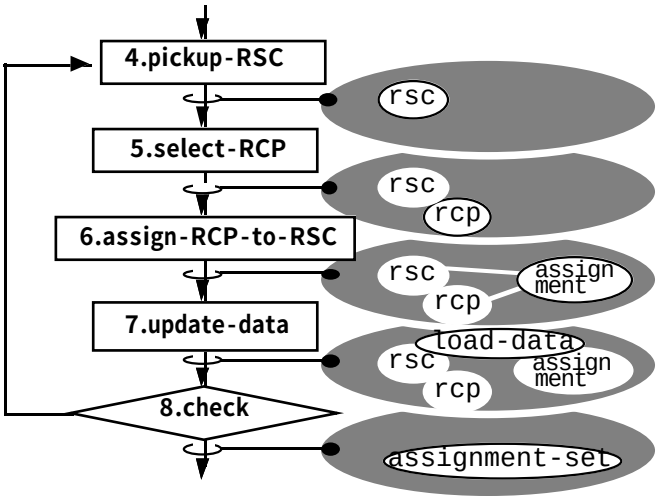

Fig. 4 A part of GPN and correspond

3.2 Conceptual level execution

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 OA 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 OAs who construct ontology and GPNAs who describe GPNs using the ontology.

In figure 5 which shows the overview of CLEPE, OA is arranged above side and GPNA left side.

Thin planes stand for languages, e.g. the base and the left side correspond to the description level of CLOS.

C-Task ontology is the task ontology independent of task types. Ideally C-Task ontology should be constant and an OA concentrates on building the Task-S ontologies for new task types. CLEPE provides OAs with functions of editing and browsing both C-Task ontology and Task-S ontology, because our current research interests include to fix the boundary between the two ontologies.

The main work of a GPNA 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.

- *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 GPNA, 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 GPNA completes editing his/her own GPN, CLEPE initiates the GPN interpretation process. The functional modules in shaded portion of Fig. 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 is established by ARM, the GPN is translated into conceptual level execution model. A GPNA can run the execution model with executor.

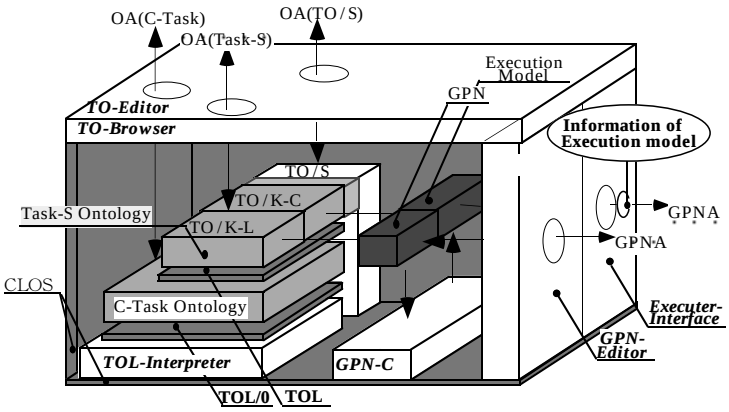

Fig. 5 Overview of CI

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 OA 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 C-Task ontology.

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 C-Task ontology into CLEPE, a set of conceptual primitives at the TOL level is introduced. 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 TO/K level, a GPNA 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

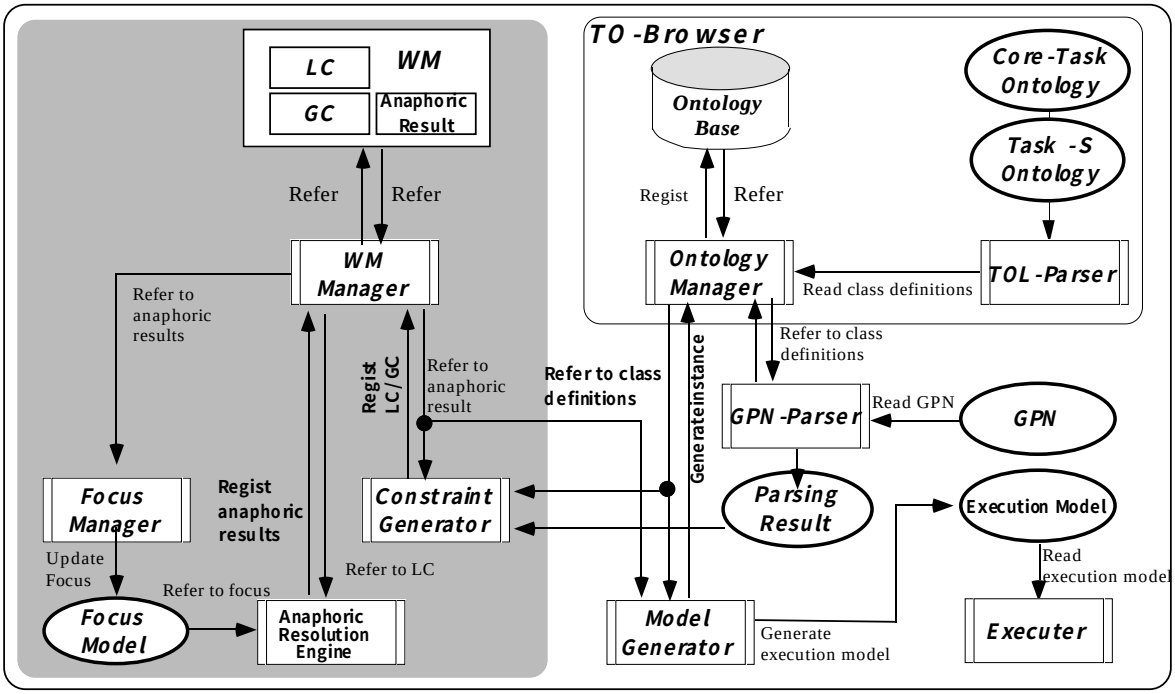

Fig. 6 Module structure of

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 TO/K-C 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) 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, "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. 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) [Mcdernott 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

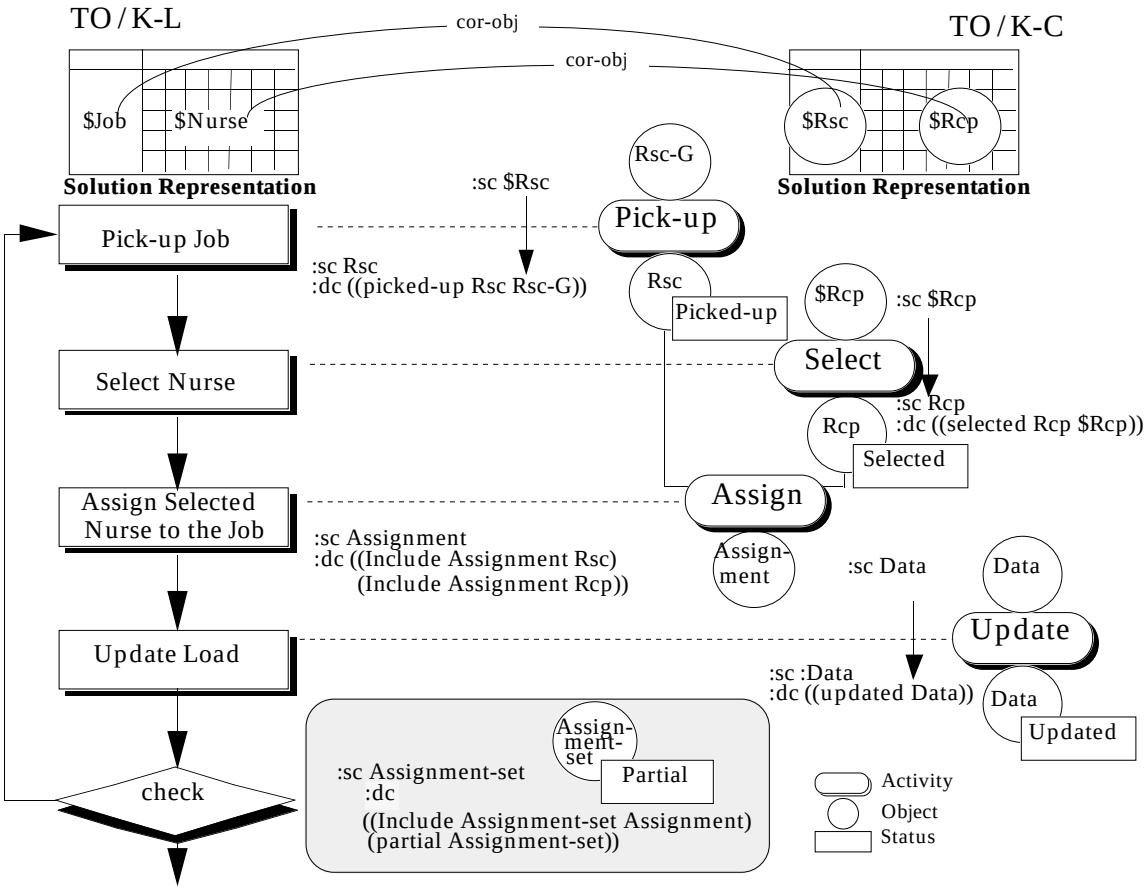

Fig. 7 An image of conceptual level

[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 .

Acknowledgment

The authors are grateful to Yasuyuki Hara and Teruyuki Shima for their valuable discussions and supports.

References

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

[Gruninger 95] Gruninger , M. and Mark Fox, M.S. : Methodology for the Design and Evaluation of Ontologies, *Proc. of IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing* (1995).

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

[McDermott 88] McDermott, J. : Using Problem Solving Methods to Impose Structure Knowledge, *Proc. of the Int. Conf. on AI Applications*, pp. 7-11 (1988).

[Mizoguchi 95] R. Mizoguchi, J. Vanwelkenhuysen, and M. Ikeda: Task Ontology for Reuse of Problem Solving Knowledge, Second International Conference on Building and Sharing of Very Large-Scale Knowledge Bases, *Proc of the KB & KS-95*, pp.46-59 (1995).

[Mizoguchi 96] R. Mizoguchi., M. Ikeda : Towards Ontology Engineering, *Proc. of the ECAI-96* (1996). (submitted)

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

[Tijerino 93] Tijerino.A, M. Ikeda, T. Kitahashi, R. Mizoguchi: 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 (1993). (in Japanese)

[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).