In Anthony Jameson, Cécile Paris, and Carlo Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97.* Vienna, New York: Springer Wien New York. © CISM, 1997. Available on−line from http://um.org.

Capturing a Conceptual Model for End-User Programming: Task Ontology As a Static User Model

Kazuhisa Seta[1], Mitsuru Ikeda[1], Osamu Kakusho[2], and Riichiro Mizoguchi[1]

[1] Institute of Scientific and Industrial Research, Osaka University, Japan
[2] Faculty of Economics and Information Science, Hyogo University, Japan

Abstract. To realize a human friendly conceptual level programming environment, it is very important to build a static user model based on the analysis of what concepts are most important for end-users when performing the task and which concepts of a problem solving specification could be out of their awareness. We have investigated a task ontology for building the static user model. Putting task ontology on the basis of a Conceptual LEvel Programming Environment, CLEPE, provides three major advantages: 1. It provides human-friendly primitives in terms of which users can easily describe their own problem solving process (descriptiveness, readability). 2. The systems with the ask ontology can simulate the problem solving process at an abstract level in terms of conceptual level primitives (conceptual level operationality). 3. It provides the ontology author with an environment for building a task ontology so that he/she can build a consistent and useful ontology.

1 Introduction

In practice, it is really hard to develop an automatic problem solving system that can cope with the variety of problems we expect to be solved by computer systems. The main reason is that the knowledge needed for solving the problems varies considerably depending on the properties of the problems. This implies that we should realize the well-known fact, sometimes ignored, that users have more knowledge than computers. From this point of view, the importance of user-centric systems (DeBellis, 1996) is now widely recognized by many researchers.

An end-user programming environment, as an incarnation of the philosophy, provides end-users with a variety of functional components which stand for the concepts appearing in the target task and allows them to build their own problem solving models in terms of those components. In such an environment, end-users can easily describe their knowledge by using the components.

To realize an end-user programming environment, the environment should 1. adaptively evolve according to changes in requirements and changes in the target world that it deals with (Fischer, 1996); 2. have a framework for explicitly representing the computational semantics of the components provided for end-users; 3. have a framework for end-users to easily and smoothly externalize the problem solving knowledge in their mind in computer-readable form; and 4. have the capability to interpret the description and generate the runnable problem solving model with both rigid computational semantics and high cognitive fidelity.

Our research on task ontology concerns all of the above requirements for end-user programming environments. In principle, task ontology is a systematic definition of the concepts appearing in the end-users' understanding of problem solving. In this sense, it can be viewed as a static model of the end-users.

We expect that, in terms of ontology, the environment will be able to capture the end-users' conceptual model of problem solving on the level of abstraction and provide them with useful programming guidance.

Our research project aims at developing a task ontology (static user model) embedded in the environment which satisfies all of the four requirements above. In this paper, however, we will concentrate on the last two, 3 and 4, from the viewpoint of user modeling.

2 CLEPE: An Environment for End-User Programming

The Conceptual LEvel Programming Environment (CLEPE) that we have been developing has two aspects. One is an environment for the ontology author to build a task ontology as a static user model and the other is one for the end-users to describe problem solving knowledge based on the task ontology. What functionality the environment should present to ontology authors is also an important issue. Because of space limitations, however, we omit this topic. In the following discussions, we assume that a task ontology as a static user model has already been built in the environment by the ontology author, and we focus on the issues concerning the availability of the static user model from the viewpoint of its use.

The target tasks of CLEPE are rather routine tasks, such as scheduling and salary calculation. Currently, we have been investigating a variety of scheduling tasks, whose goal is to find an assignment of scheduling recipients to scheduling resources that satisfies given constraints, e.g., to assign nurses to all the jobs (night duty, semi-night duty, day duty) taking fairness into consideration.

In CLEPE, end-users describe their own problem solving knowledge in a diagrammatic representation with a constrained set of natural language sentences. Then they can verify the system's interpretation of the description using the conceptual level execution functionality of CLEPE. The

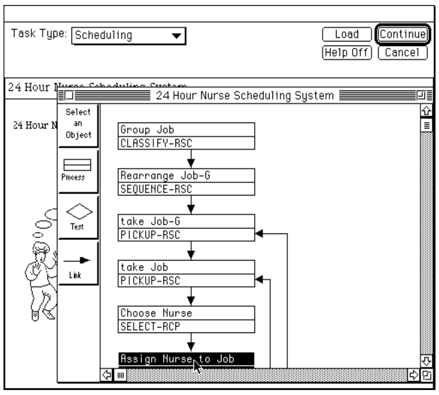

Figure 1. A screen image of CLEPE.

continuity from the diagrammatic representation to conceptual computational semantics is one of the key features of CLEPE, which originated in our precedent project MULTIS (Tijerino and Mizoguchi, 1993).

To realize such an environment in which the end-users can 1. describe their own problem solving processes using the concepts of which they are most conscious, 2. understand the execution result of the model on an appropriate level of abstraction and 3. debug it at the conceptual level, it is necessary to analyze 1. what concepts are most important for the end-users when performing the task, 2. which concepts of problem solving specification might be out of their awareness and 3. what kind of troubles end-users would face in the environment and what their causes might be.

2.1 Design Principle

It is quite a time-consuming task for the end-users to describe their own problem solving processes with reasonable precision. To lighten the load of the end-users, it is important for a task ontology to reflect their common perception of problem solving. On the other hand, from the computer's standpoint, the description of the problem solving should be rigid enough to specify the computational semantics. We could say that this conflict is a common problem of program
ming languages for end-users. The key to the problem is to shift task ontology close to the end-users and to embody the function to fill the gap between end-users and computers. Let's look at the typical situation. The structure of the problem solving process is roughly divided into control structure and data structure. In general, we can expect that the control structure of the problem solving knowledge in the end-users' mind has something in common with the control structure in the externalized description of it. On the other hand, the conceptual structure of the domain world perceived by the end-users could be different from the data structure appearing in the description, because they are not aware of the operational concepts needed for problem solving. So CLEPE allows the end-users to describe their own problem solving control structures in terms of domain activities and domain objects which are familiar to them.

The key features of CLEPE can be summarized as follows: 1. The end-users can describe their own problem solving processes in terms of human friendly primitives. 2. The end-users can observe the task execution process and debug their own description at the conceptual level. 3. The continuity from the user's description of problem solving to computational semantics is maintained.

2.2 Framework for Describing Problem Solving

Figure 1 shows an image of the interface for end-users. The network in the figure is called a Generic Process Network (GPN). GPN represents the end-users' problem solving knowledge in terms of lexical level task ontology (to be explained later in Section 3). Each node of the GPN, called a *generic process*, is separated into two parts. The upper part represents a domain process in terms of natural language, and the lower is a generic process which is a lexical level task ontology translation of the upper part. Each generic process is a combination of a verb and a noun:

Generic Process (GP) = Generic Verb + Generic Noun

Figure 1 shows a GPN for a 24-hour nurse allocation task. A node represents a generic process and a link represents the control flow of problem solving. The goal of the task is to allocate the nurses to the jobs under some constraints. In the top node in Figure 1, for example, “Grouping the Jobs” is a domain process and “Classify Scheduling-Resource (RSC)” is the corresponding generic process.

In the following, with the abstract scenario, we describe how CLEPE supports the end-users’ work. The scenario is roughly divided into three phases.

Phase 1: Description of problem solving knowledge (by end-users). In this phase, the work of an end-user is to specify his/her own problem and then to compose a GPN to solve it. First, the end-user is asked to fill in some fields in the visual workplace for a problem specification. Then, to lighten their initial load, CLEPE retrieves a set of similar task cases from the GPN library based on the specification and shows them to end-users (Tijerino and Mizoguchi, 1993). End-users can refer to or reuse them to describe their own problem solving knowledge.

During the composition of a GPN, an end-user inputs the domain process in the upper field of a GPN node and then translates it into the generic process in the lower field. In this phase, lexical level task ontology plays an important role in providing vocabulary for the description of generic processes. The end-user selects the appropriate terms from the vocabulary shown in the lexical level task ontology browser and composes the generic process sentences.

It is important that lexical level task ontology be acceptable to both end-users and the environment, because its role is to lay the foundation for them to share the syntax and the semantics of the common language (GPN) to represent problem solving knowledge. Thus, from the viewpoint of user modeling, we could say the lexical level task ontology should be designed by taking the linguistic properties of the end-users’ problem solving description into consideration. On the other hand, from the viewpoint of the end-user programming environment, the interpretation mechanism to clarify the semantics of the end-user’s description should be available. In CLEPE, the semantics of GPN is explicitly represented as a conceptual level execution model based on conceptual level task ontology. In the following phase, CLEPE interprets the end-user’s GPN and generates the corresponding conceptual execution model.

Phase 2: Making up for the incompleteness of a problem solving description (cooperative work of an end-user and CLEPE). 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 easily make up the incompleteness in a problem solving description, if it exists. Let us take an example of the incompleteness in a problem solving description.

The lack of the human’s awareness of the objects to which a process applies is a source of the incompleteness. When end-users put a generic verb into a generic process, input objects and output objects should be bound into the input port and the output port of the generic process, respectively. However the bindings cannot always be specified by an end-user explicitly. For example, in the case of a check process to check the termination condition of the loop for a sequential scan of a set, input/output objects are often omitted in the description, because the condition is quite obvious to an end-user: “until the set is exhausted”. This is a typical example of the lack of a human’s awareness of problem solving. They know it but don’t write it explicitly. The respect for a user’s awareness of problem solving is a key to the end-user friendliness of CLEPE.

To make up for the incompleteness, CLEPE analyzes the GPN and tries to reconstruct the object flow intended by the end-user. The mechanism is called *object flow analysis* (Seta et al., 1996). In this phase, CLEPE generates the object flow model (shown in Figure 2 in Section 4). Conceptual level ontology (an explanation will appear in Section 3) plays an important role in this phase. It specifies the meaning of task concepts. Based on the meaning, CLEPE checks the consistency of object flow models, for example, the consistency of changes of objects through the object flow. Once inconsistency or incompleteness of a problem solving description is identified, CLEPE tries to make up for it based on the conceptual level ontology by interacting with the end-user.

Phase 3: Conceptual level execution (by CLEPE). Once a GPN has been built by an end-user, CLEPE interprets it on the assumption that the user completely agrees with the static user model. However, there might be a gap between the interpretation and the end-user's intention, because the agreement is only partial. In such a case, we have no choice but to expect the end-user to revise the GPN. To support the user's work, CLEPE provides the functionality of conceptual level execution of a GPN.

The advantages of conceptual level execution are the following: 1. The end-user can recognize the difference between the meaning intended by him/her and the system's interpretation. 2. The end-user and the system can reach an agreement on the problem solving description more explicitly. Conceptual level ontology plays an important role in the conceptual level execution. It specifies the meaning of activities and objects and provides the framework for representing the changes of objects which are caused by activities. In this phase, the system can explain the role of each process in the task flow and the history of changes of each object. The explanation is generated based on the conceptual level execution model built by object flow analysis in Phase 2. For example, CLEPE explains that the role of the assign process (shaded in Figure 1) is to generate an assignment which consists of a picked-up job and the nurse selected for the job in each iteration of the loop. In Sections 4 and 5, we will discuss the conceptual level execution in more detail.

3 Task Ontology as a Static User Model

From now on, we focus on the role of task ontology as a static user model embedded in the end-user programming environment. The aim of user modeling research, in general, is to enable smooth communication between humans and computers. The approach to this goal can be roughly divided into two phases: static analysis and dynamic adaptation. Static analysis is the process of analyzing the end-user's epistemological conceptual structure of problem solving prior to the environment design. The goal of dynamic adaptation is to change the conceptual structure embedded into the environment and to become well suited to the individual properties of end-users when it is in operation. It seems that most attention has been paid to the latter phase in the area of user modeling without any particular reason. However, in the domain of programming, it is very difficult to dynamically capture the individual properties of a programmer as a dynamic user model because the programming task is too complex. Thus, a static user model, instead, is expected to play a more important role than a dynamic one. We believe that the environment becomes more supportive for end-users by integrating a well-defined task ontology as a static user model, because it can capture their intentions adequately with the aid of the ontology. Furthermore, we believe the integration makes dynamic user modeling easier, because it lays the foundation for analyzing and representing individual properties of programming behavior. This is the underlying philosophy of our research.

Task ontology is a systematic definition of conceptual structure to represent the end-user's awareness of various kinds of problem solving activities. It serves as a kind of meta-definition of a problem solving description language, GPN in our case. In this sense, it plays a role similar to that of the Meta Object Protocol (MOP) for the CLOS language. By using MOP, one can adjust the computational semantics of CLOS to his/her own objectives. Task ontology, on the other hand, specifies the conceptual meaning of the concepts appearing in GPNs. This means that the epistemic fidelity of GPN depends largely on how deeply we can capture the end-user's conceptual structure of problem solving in a task ontology. Once the task ontology is embedded in the environment, it provides some functions for the end-users; those are consistency checking of a GPN, conceptual level execution of it, and its compilation in the runnable computer program. The guidance presented to the end-users based on the functions can be appreciated by them when the task ontology is designed well enough to capture the end-user's conceptual structure of problem solving. By integrating task ontology into the environment, three major advantages mentioned in Section 2.1—descriptiveness/readability, conceptual level operationality, and symbol level operationality—can be realized.

A task ontology is composed of two layers. The top layer is called *lexical level ontology* and the bottom layer is called *conceptual level ontology*. 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 knowledge in their minds. 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 the end-user's perception of problem solving are organized into generic concept classes such as *activity*, *object*, *status*, and so on. Intuitively generic verb, generic noun, and generic adjective in the lexical world correspond to activity, object, and status in the conceptual world, respectively.

End-users can describe their problem solving knowledge using the words defined in the lexical level ontology and the system can interpret end-users' intentions in the description based on conceptual level ontology.

Generally an ontology is composed of two parts: taxonomy and axioms. Taxonomy is an ordered system of concepts, and axioms are established rules, principles, or laws relating the concepts.

4 Advantages of a Conceptual Model of Problem Solving

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

Conceptual level execution is a function which provides the trace information on the execution process of a GPN on the appropriate level of abstraction. 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 level of abstraction of the trace information such as real data is too low to allow them to match it against their understanding of the problem solving. On the other hand, conceptual level execution provides end-users with conceptual level information which can be easily mapped onto their understanding of the intended behavior of the GPN. In the following, we introduce the concept of *problem solving causality*, which plays an important role for generating appropriate information about the behavior of a GPN.

The information provided by the conceptual level execution mainly concerns how objects and the relations among them change during problem solving. An idea of a *version* of objects is introduced as a source of the information. A 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, a change in a part of an object is propagated to the whole object. It reflects how end-users recognize the changes in objects in the domain world. An important point here is that all of the changes that happen in the domain world should not be reported to end-users, because too much information would bother them. Instead, the report should include only the information really useful for the end-user to grasp the 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 2 shows an object flow model (partially) and a domain model corresponding to it. The lexical level model (GPN) depicted in the Figure 2 is a simplified version of the one in Figure 1.

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

In Figure 2, we discuss the relation between task flow and changes in domain world objects, taking the part-whole causal relation as an example. Let's focus on the causal relation between the *update* process in the *k*-th iteration of the loop and the *select* process (which selects a nurse with minimum load from a set of nurses) in the (*k*+1)th iteration. When the update process updates the load data of the nurse who is assigned to a job in the assign process in the *k*-th 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 an example where the change in a 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 concerning problem solving causality. Problem solving causality answers the question based on whether the change is important or not from the problem solving viewpoint. In this case, it is important, because the change in the set of nurses guarantees the correctness of the input to the *select* process in the succeeding iteration. Thus, when the *select* process is executed in the (*k*+1)th iteration of the loop, conceptual level execution shows end-users that the input object of the *select* process is identical to the one in the *k*-th iteration and that the load data of all of its members are appropriately updated by the update process in the *k*-th iteration. By representing the changes in objects caused by task execution in terms of versions of the domain objects, it is possible for CLEPE to explain the behavior of problem solving at arbitrary times using appropriate expressions. For example, CLEPE explains the roles of the *select* process at every loop iteration as follows: "The *select* process, in each the loop iteration, selects a nurse with minimum load from the set of the nurses, whose loads were adequately updated by the update process in the last iteration of the loop."

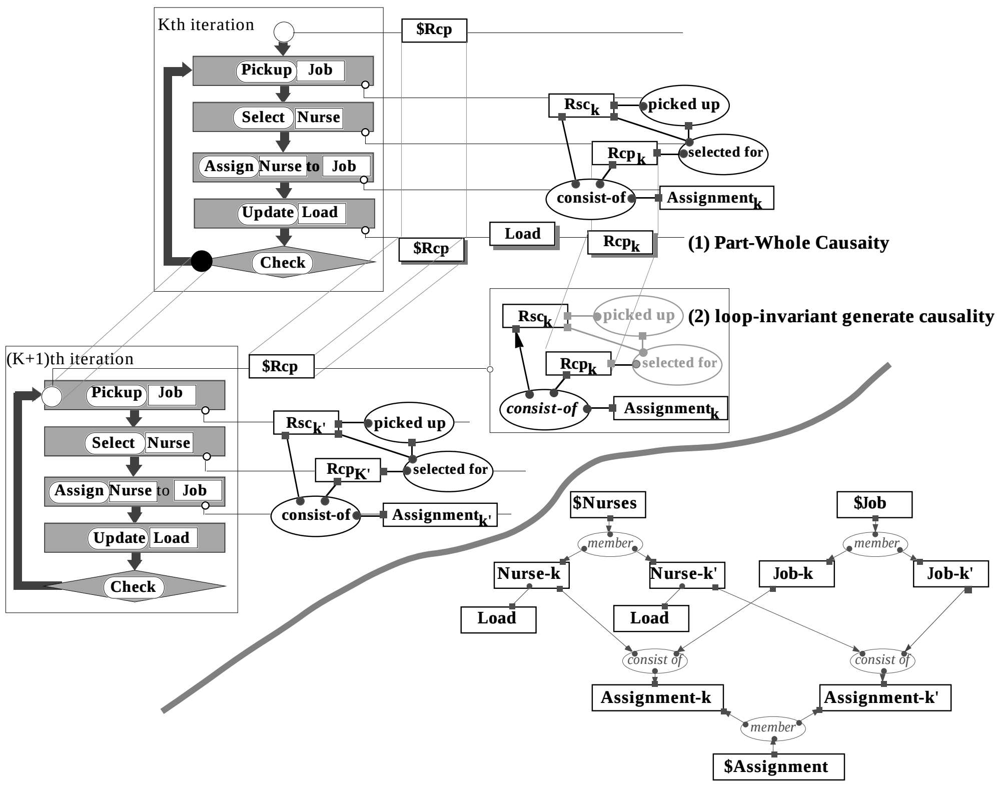

Figure 2. Examples of problem solving causality.

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 2, we can see two types of relations, that is, loop-temporal relations and permanent relations. The "Picked_up" binary relation for Job-K is an example of a loop-temporal relation. This fact becomes evident when the pickup process outputs Job-K in the *k*-th iteration of the loop and disappears when the iteration is completed. The same thing is true for the "Selected for" relation for Nurse-k.

The life span of the "Picked_up" relation in the Pickup-Check loop structure is specified by the axiom of the 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 the *select* process, there is no specification for the life span of the relation. Instead, the general axiom of task ontology says "If a conceptual entity depends tightly on the other conceptual entities, their life spans should be same." Following this principle, "Selected_for (Nurse-k, Picked_up(Job-K))" should disappear at the same time that "Picked_up(Job-K)" disappears, because Nurse-k is Selected_for "Picked_up(Job-K)". Here, however, we should notice that the generic relations, Picked_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 a permanent relation. There are two kinds of permanencies in our task ontology; *problem solv-*

ing permanency and *problem permanency*. The former means that a conceptual entity remains throughout problem solving but disappears upon completion. The latter means that an entity never disappears if it is needed to represent the results of problem solving, as with the “consist-of” relation in our example.

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

Loop-invariant generate causality is specified as follows: “If a portion of the 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 this: “The assign process generates an Assignment which consists of Picked-up Job and Nurse Selected for the Job in each iteration of the loop.” Assignment is added to the Assignment-set which is the solution to the given problem.

As we have seen in the above two examples, the problem solving causality can extract a meaningful set of relations from the large number of relations in the problem solving model. Without it, end-users would be bored with verbose reports about insignificant relations.

5 Capturing the Problem Solving Model

Problem solving causality is built into the task ontology as a 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, the dynamics of problem solving processes are presented to end-users not only as the time series of computational operations but also as meaningful causal relations among the changes in objects which keep the correspondence between problem solving processes and domain concepts. The presentation should be acceptable to end-users because it appropriately reflects the epistemic properties 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 a user model.

Problem solving causality is a set of causal relations among the parts of a problem solving model. Once the object flow model corresponding to the given GPN is identified, CLEPE tries to find out the causal relations underlying the problem solving model based on the ontology of problem solving causality and then to 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 a meaningful role to the objects from the problem solving viewpoint.

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

As we discussed in the previous section, it is desirable that the problem solving causality be presented to the end-user in domain-oriented manner, because end-users prefer domain-oriented representations to task-oriented ones in general. To cope with the domain-oriented property of end-users' awareness of problem solving, the task-oriented representation of the causality, which is specified in the task ontology, needs to be translated into a domain-oriented representation. To embody such a hybrid representation, we introduced a task-domain binding (TD-binding) mechanism which acts as glue for integrating the domain concepts into the 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 of a scheduling recipient (RCP). In this case, TD-binding binds the domain concept, *nurse*, and the task concept, *RCP*, together, and serves either the meaning of nurse or the one of RCP in compliance with requests.

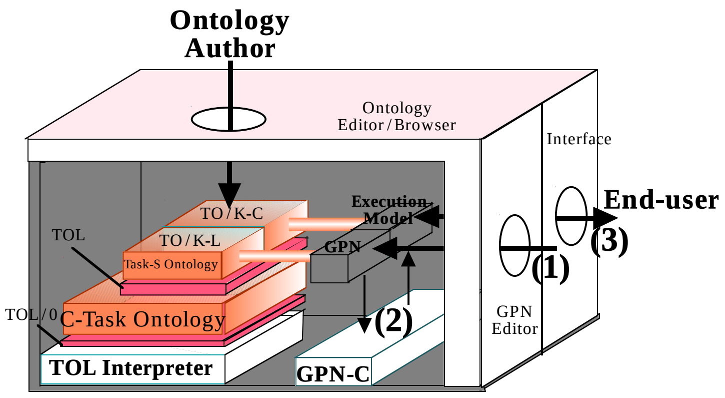

Figure 3. Overview of CLEPE. (TOL: Task ontology representation language. TO/K-L: Lexical level ontology. TO/K-C: Conceptual level ontology. GPN-C: GPN compiler, which translates a GPN into a conceptual level execution model.)

Figure 3 shows an overview of the conceptual level programming environment CLEPE. The processes represented by arrows marked with (1), (2), and (3) are the end-user's work of describing the GPN, object flow model construction, and conceptual level execution respectively. All the processes are supported based on task ontology, for example, lexical level task ontology specifies the syntax of the GPN description in (1), and conceptual level task ontology and TD-binding specify meaning of the object flow model in (2). In (3), as we discussed thus far, problem solving causality plays important role.

6 Concluding Remarks

In general, the design of an end-user programming environment depends largely on the goal of the end-users' task. For example, the goal of KIDSIM end-users is to learn new concepts by observing the behavior produced by the programs that they have written (Cypher and Smith, 1995). In an application of KIDSIM, for example, schoolchildren are expected to learn both elementary physics and programming concepts by observing the physical phenomena simulated by their own programs. Direct manipulation techniques and programming by demonstration (PBD) act as the key technologies of KidSim to help learners to understand the relation between their programs and observed behavior. The educational goal of a KidSim application and the target concepts of learning are implicitly embedded into the application by the authors. Then the learners are expected to find out the concepts, which are carefully hidden in the microworld, by themselves through trial-and-error experiments. This is a form of discovery learning. Therefore, the implicitness of the application author's educational intention is beneficial from the educational viewpoint. On the other hand, where the end-user's goal is simply software development, as with CLEPE, the implicitness is rather undesirable, because the conceptual structure of the target task, task ontology in our case, is hopefully shared by ontology authors and end-users, as we have discussed in this paper. Thus, we could say that the explicitness of the task ontology as a static user model is the distinctive feature of CLEPE compared to KidSim.

The DODE (Domain Oriented Design Environment) architecture developed by G. Fischer and the TOVE (TOronto Virtual Enterprise) project are closely related to our research.

The DODE project (Fischer, 1996) has developed a human friendly framework in which a wide variety of domain concepts can be integrated, for example, network design, voice dialog design, kitchen design, and so on.

The advantages of their research include the following: (1) To facilitate software development, they realize the collaborative environment based on the analysis of all of the stakeholders' activities. (2) They propose a theory, called SER (Seeding, Evolutionary growth and Reseeding), to capture the software evolution process explicitly. End-users can adjust their own workbench designed on the basis of the theory to dynamic change of the environment, e.g., software requirements, evolution of the domain, and so on.

In our research, the framework corresponding to this theory is called "ontology swapping". In the framework, an end-user can select a task-type-specific ontology (Seta et al., 1996) from a task ontology library. A task-type is a kind of categorization of tasks, for example, the scheduling task-type, the book-keeping task-type, the demand analysis task-type and so on. As shown in Figure 3, task-type specific (Task-S) ontology is placed at the top of ontology hierarchy and specifies the descriptive primitives provided for end-users. By swapping the ontology, the end-user programming environment can be adjusted for the end-users' target task.

The TOVE (Fox et al., 1993) ontology and ours share a similar goal in the sense that both approaches aim at formulating the conceptual meaning of a variety of types of activities in the target world as ontologies. The TOVE project has built a practical ontology of enterprise activities and developed a framework which can answer a variety of questions about dynamic aspects of the activities based on it. The difference between TOVE and our research is that TOVE provides an environment in which end-users can directly manipulate the rather complex data model (corresponding to our conceptual level execution model) based on the ontology, while CLEPE respects human friendliness and adopts the simple description scheme (GPN) which reflects the end-users' awareness of problem solving. In addition, CLEPE also provides the conceptual level execution function based on task ontology to promote ontological agreement between end-users and CLEPE.

From the viewpoint of transferability of the architecture, most of the functional modules of CLEPE are basically independent of the task domain. A task-type-specific ontology as a static user model is transferable over the domains of the same task-type, e.g., the scheduling task-type-specific ontology can act as a static user model for the 24-hour nurse allocation problem, the vehicle allocation problem, and so on. Only the workplace is not transferable in any sense, because it tightly depends on a certain task-type. As we have discussed, the "ontology swapping" mechanism is a key technique for enhancing the flexibility of the architecture. In the CLEPE project, we have a plan to integrate a wide variety of task-type-specific ontologies into a task ontology library to enhance the flexibility of CLEPE.

To verify the utility of task ontology from the viewpoint of user modeling, we asked thirteen scheduling experts to compose GPNs to represent their problem solving knowledge for their own real world scheduling problems. After some trial-and-error, all of them completed their description. And we got an entirely favorable response from them. After the development of CLEPE is completed, we will perform a more careful empirical study to evaluate the utility of the whole architecture.

References

Cypher, A., and Smith, D. C. (1995). KidSim: End user programming of simulations. In *Proceedings of Computer Human Interaction '95,* 27-34.

DeBellis, M. (1995). User-centric software engineering. *IEEE Expert* 10:34-41.

Fisher, G. (1996). Seeding, evolutionary growth and reseeding: Constructing, capturing and evolving knowledge in domain-oriented environment. In *Domain Knowledge for Interface System Design*. London: Chapman & Hall. 1-16

Fox, M. S., Chionglo, J., and Fadel, F. (1993). A common-sense model of the enterprise. In *Proceedings of the Industrial Engineering Research Conference.*

Seta, K., Ikeda, M., Kakusho, O., and Mizoguchi, R. (1996). Design of a conceptual level programming environment based on task ontology, In *Proceedings of Successes and Failures of Knowledge Based Systems in Real World Applications,* 11-20.

Steels, L. (1990). Components of expertise. *AI Magazine* 11(2):28-49.

Tijerino, A. Y., and Mizoguchi, R. (1993). MULTIS II : Enabling end-users to design problem-solving engines via two-level task ontologies, In *Proceedings of the European Knowledge Acquisition Workshop '93,* 340-359.

Yost G., Klinker, G., Linster, M., Marques, D., and McDermott, J. (1994). The SBF Framework,1989-1994: From applications to workplaces, In *Proceedings of the European Knowledge Acquisition Workshop '94,* 318-339.