Construction and Deployment of a Plant Ontology

Riichiro Mizoguchi, Kouji Kozaki, Toshinobu Sano and Yoshinobu Kitamura

ISIR, Osaka University
8-1 Mihogaoka, Ibaraki, Osaka, 567-0047 Japan
{miz, kozaki, sanop, kita}@ei.sanken.osaka-u.ac.jp

Abstract. Although the necessity of an ontology and ontological engineering is well-understood, there has been few success stories about ontology construction and its deployment to date. This paper presents an activity of ontology construction and its deployment in an interface system for an oil-refinery plant operation which has been done under the umbrella of Human-Media Project for four years. It also describes the reasons why we need an ontology, what ontology we built, what environment we used for building the ontology and how the ontology is used in the system. The interface has been developed intended to establish a sophisticated technology for advanced interface for plant operators and consists of several agents. The system has been implemented and preliminary evaluation has been done successfully.

INTRODUCTION

Ontological engineering[1] is a successor of knowledge engineering which has been considered as a technology for building knowledge-intensive systems. Although knowledge engineering has contributed to eliciting expertise, organizing it into a computational structure, and building knowledge bases, AI researchers have noticed the necessity of a more robust and theoretically sound engineering which enables knowledge sharing/reuse and formulation of the problem solving process itself. Knowledge engineering has thus developed into "ontological engineering" where "ontology" is the key concept to investigate.

Although the necessity of an ontology and ontological engineering is well-understood, there has been few success stories about ontology construction and its deployment to date. This paper presents an activity of ontology construction and its deployment in Oil-refinery plant which has been done under the umbrella of Human-Media Project for four years.

Human Media project, which is a MITI(Japanese Ministry of International Trade and Industries) funded national project, is intended to invent an innovative media technology for happier human life in the coming information society in 21st century. It is something an integration of the three representative media such as Knowledge media, Virtual media and Kansei media. "Kansei" is a Japanese term which roughly means the sixth or seventh sense sensing for satisfaction, comfort, beauty, softness, etc. Our ontology construction activities have been done in the project named

"Development of a human interface for the next generation plant operation" running as a subproject of Human Media project.

The interface for oil-refinery plant operation has been developed intended to establish a sophisticated technology for advanced interface for plant operators and consists of Interface agent: IA, Virtual plant agent: VPA, Semantic information presentation agent: SIA, Ontology server: OS and Distributed collaboration infrastructure: DCI. The last two are mainly for issues related to system building, while the first three are related directly to interface issues. OS has been developed employing ontological engineering as a key knowledge media technology[2].

This paper presents the reasons why we need an ontology, what ontology we built, what environment we used for building the ontology and how the ontology is used in the system. The next section discusses a short introduction to ontology. Section 3 explains the role of the plant ontology. Detailed description of the plant ontology we built is given in Section 4. Section 5 presents the explanation of its use in the entire interface system we are developing. Section 6 describes an ontology development environment, Hozo, we developed and used for building the plant ontology. Hozo has been extensively used in many other projects in our group and the next version is being developed based on our experience and uses' feedback. Section 7 discusses the related work followed by conclusion.

WHAT IS ONTOLOGICAL ENGINEERING AND WHAT IS AN ONTOLOGY?

Roughly speaking, ontologies consist of **task ontology**[3][4] which characterizes the computational architecture of a knowledge- based system which performs a task and **domain ontology** which characterizes the domain knowledge where the task is performed. By a task, we mean a process like diagnosis, monitoring, scheduling, design, and so on. In our context, operation is a task. The idea of task ontology which serves as a theory of vocabulary/concepts used as building blocks for knowledge-based systems[3][4][5][6] might provide us with an effective methodology and vocabulary for both analyzing and synthesizing knowledge-based systems. An ontology is understood to serve as a kernel theory and building blocks for content-oriented research.

Why ontology instead of knowledge? Knowledge is domain-dependent, and hence knowledge engineering which directly investigates such knowledge has been suffering from rather serious difficulties, such as domain-specificity and diversity. Further, much of the knowledge dealt with in expert systems has been heuristics domain experts have, which makes knowledge manipulation more difficult. However, in ontological engineering, we investigate knowledge in terms of its origin and elements from which knowledge is constructed. An ontology reflects what exists out there in the world of interest or represents what we should think exists there. Hierarchical structure of concepts and decomposability of knowledge enable us to identify portions of concepts sharable among people. Exploitation of such characteristics makes it possible to avoid the difficulties knowledge engineering has

faced with. The following is an enumeration of the merits we can enjoy from an ontology:

1. *A common vocabulary.* The description of the target world needs a vocabulary agreed among people involved.
2. *Explication* of what has been often left implicit. In all of the human activities, we find presuppositions/assumptions which usually are left implicit. Any knowledge base built is based on a conceptualization possessed by the builder and is usually implicit. An ontology is an explication of the very implicit knowledge. Such an explicit representation of assumptions and conceptualization is more than a simple explication.
3. *Systematization* of knowledge. Knowledge systematization requires well-established vocabulary/concepts in terms people use to describe phenomena, theories and target things under consideration. An ontology thus contributes to providing a backbone for the systematization of knowledge.
4. *Standardization.* The common vocabulary and knowledge systematization bring us more or less standardized terms/concepts.
5. *Meta-model functionality.* A model is usually built in the computer as an abstraction of the real target. And, an ontology provides us with concepts and relations among them which are used as building blocks of the model. Thus, an ontology specifies the models to build by giving guidelines and constraints which should be satisfied. This function is viewed as that at the metalevel.

THE ROLE OF A PLANT ONTOLOGY

Any intelligent system needs a considerable amount of domain knowledge to be useful in a domain. The amount of knowledge necessary often goes large, which sometimes causes difficulties in the initial construction and maintenance phases. As described above, one of the methods we adopted to cope with such problems is ontological engineering. The plant ontology makes contributions in our system in many respects described above. Roughly speaking, the essential contribution of an ontology is making shared commitment to the target plant explicit, and hence terminology is standardized within the community of agents. By agents, we also mean human agents, operators, to share such a fundamental understanding about the plant. This enables the system to communicate with operators using the terms stored in Ontology server: OS. It is the second major role of OS in the current implementation of the interface system which is discussed below in Section 4.

In message generation, we need to pay maximal attention to word selection to make operators' cognitive load minimum in message understanding. After an intensive interview with domain experts, we found human operators use different terms to denote the same thing depending on context. When we first noticed this fact, domain experts apologized for this seemingly random fluctuation of word usage, since they did not know the reason why they use terms that way and they were used to collaboration with computer engineers who do not like neat adaptation and tend to compel their idea of "this is what a computer can do, so accept it". They kindly declared that they would soon determine a unique label for each thing. But, we were

different from such computer engineers. Instead of accepting their proposal, we carefully analyzed the way of their word usage and finally came up with that it is not random except a few cases. Many of the wording have good justifications which have to be taken care of in the message generation. The way of doing so is described in 5.2.

The reasons why we employed distributed collaboration architecture with multiple agents include making the whole system robust and easy to maintain. As is well known, however, these merits are not free. We need a well-designed vocabulary for describing message content as well as a powerful negotiation protocol. Although the latter is of importance, it is out of the scope of this paper.

DCI is responsible for enabling collaborative problem solving by multiple agents with the help of OS. It is one of the key factors that domain-dependent knowledge be isolated in OS so that DCI can be as general as possible.

PLANT ONTOLOGY

We built a plant ontology which consists of several hierarchical organizations of concepts such as *operation task ontology*, *plant components, plant objects, basic attributes* and *ordinary attribute*. The key issue in design of an ontology is clear distinction essential categories from peripheral or view-dependent concepts.

Operation Task Ontology

The major constituents of a task ontology are concepts of action done by the task performer, operators in our case, and concepts of the role which domain objects play in the task performance. This is the key issue of task ontology. That is, a task ontology reveals the problem solving context in a task of interest to specify the roles the domain objects play. Without this, it is left implicit that how domain concepts should be organized under a specific task.

Activity Concepts: Operation of a plant consists of monitoring the behavior of the plant, diagnosing abnormal states if any and operating devices to recover from such states. Thus, the three actions, ***monitor***, ***diagnose*** and ***operate*** are the top level category of action part. Under these, we also identified ***enumerate, list up, decide, predict, etc.***

There are two kinds of hierarchies of concepts organization. *is-a* hierarchy and *part-of* hierarchy. For example, sub-concepts of ***operate*** in the *part-of* hierarchy would be ***recognize***(the state), ***predict***(the near future), ***identify***(the causes), and ***decide*** (operation to take). On the other hand, sub-concepts of ***reason*** in the *is-a* hierarchy, are ***predict and retrospectively reason*** and its super is ***think.*** In the *is-a* hierarchy case, properties of the super concepts are inherited to the sub-concepts. In ***reason*** case, its [input : output] roles, [state : state], are inherited by ***predict and retrospectively reason*** but specialized to [current state : future state] and [current state : causal state].
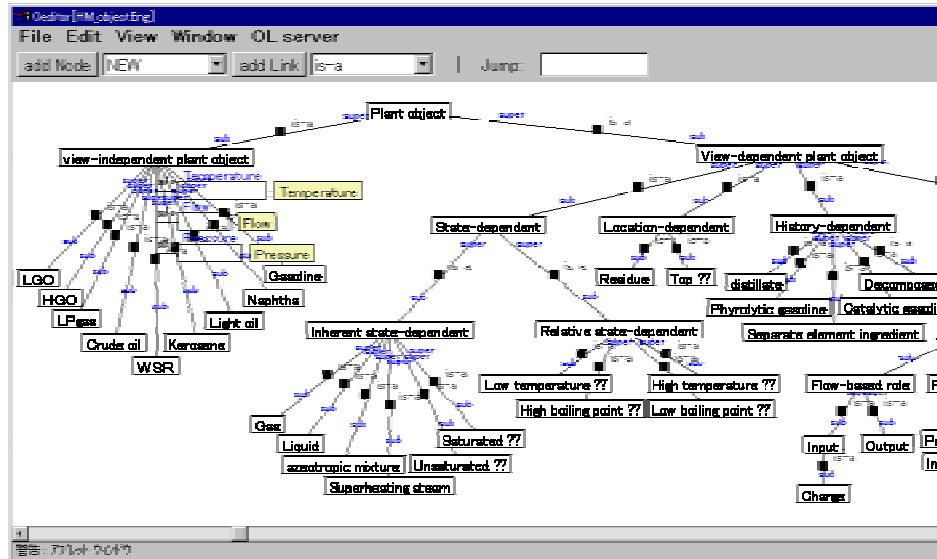
Fig. 1. A Portion of Plant Object is-a Hierarchy in the Plant Domain Ontology

Role Concepts: Major task-specific role concepts include state of operation, *abnormal state, candidate cause, countermeasure operation* etc. When a state of operation is recognized as abnormal, then it comes to be called *abnormal state*. *Near future state* is a state predicted from an *abnormal state* as the *current state*. A *cause* of a fault has its sub-states: a *candidate cause* which is an inferred causal state and a cause which is a real cause.

Domain Ontology

There exist two major things in the plant domain: *Plant components(devices)* and *plant objects* to be processed by the *devices*. Domain concepts also have role concepts like task ontology. To say precisely, many of the domain concepts are role concepts. The first things we have to do when designing a domain ontology is discrimination of roles concepts from essential categories(or basic concepts), i.e., view- or context-independent concepts. Let us first take *plant object*. The top-level categories of *plant object* are *view-independent object* and *view-dependent object*. The former includes LP gas, gasoline, naphtha, etc. which are categories persistent in any situation. The latter includes *tower-head ingredient, liquid, distillate, input, intermediate product, raw material, fuel*, etc. All are view- or context-dependent. The major task needed was categorization of such dependency. Fig. 1 shows a portion of *plant object* *is-a* hierarchy. The top-level categories of *view-dependent plant object* are *state-dependent, location-dependent, history-dependent and role-dependent* objects. *state-dependent* objects has *inherent state-dependent* and *relative state-dependent* objects
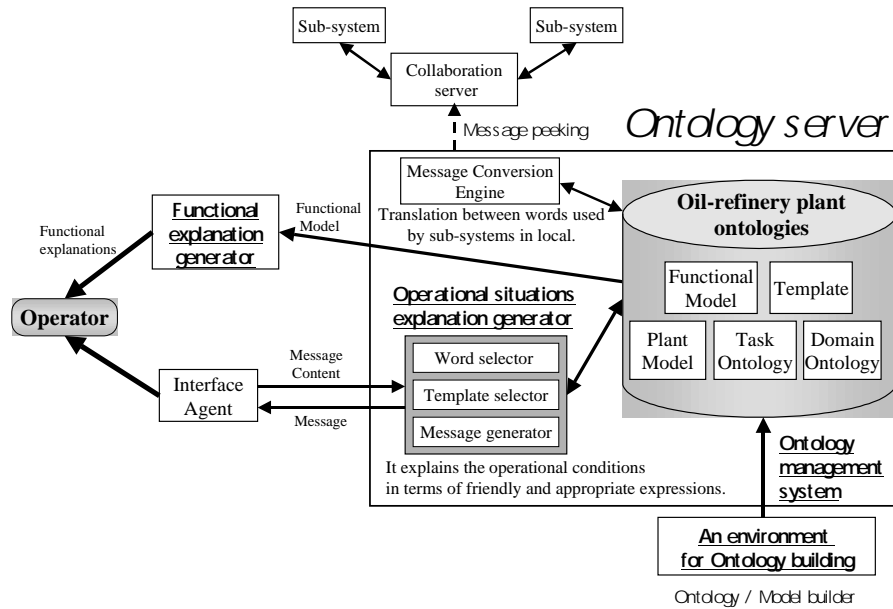
Fig. 2. Block diagram of Ontology server

as its sub-concepts. The former includes *liquid, gas, superheating steam* etc. and the latter *low temperature ingredient*, *low boiling point ingredient*, etc.

   Attribute also needs careful treatment. Most of the attributes people think so are not true attribute but *role attribute*. Let us take an example of *height*. It is a *role attribute* whose *basic attribute* is *length. Height, depth, width* and *distance* are *role attributes.* Just like a man is called a husband when he has got married. The true attribute is called *basic attribute*. Examples of *basic attribute* include *length, area, mass, temperature, pressure, volt, etc*. Role attribute includes *height, depth, input pressure, maximum weight, area of cross section, etc.* Needless to say, these attributes are also decomposed into several sub-concepts.

   We finally built an ontology which contains about 500 concepts which are approved by the domain experts and the coverage is around the normal pressure fractionator of a full-scale refinery plant.

   Another kind of domain ontology is necessary to build a model of a plant as an active artifact. That is an ontology of function and behavior which is task-independent. This ontology is used for deeper understanding of the dynamic characteristics of an artifact. Although it is interesting, it is omitted in this paper because of the space limitation [8][9].

USE OF THE ONTOLOGY

System overview of Ontology Server: OS

Ontology server has several functions in the interface system.
(1)To store the plant ontology for use of representing message contents.
(2)To build a model of the target plant shared by other agents
(3)To generate natural language messages presented to operators according to appropriate word selection
(4)To answer questions about the structure of the plant
(5)To translate among vocabularies local to each of the agents.

In general, the major contribution of OS to the whole system is to standardize concepts about the target plant each agent has as well as vocabulary used by them. Fig. 2 shows the block diagram of OS where several types of plant ontologies are stored and two functional modules such as explanation generator and message generator are also shown. OS has a stylized application protocol interface supported by DCI, by which any agent can communicate with OS and can inspect the ontology in it.

Appropriate word selection

After intensive discussion with domain experts, we found a remarkable fact that they sometimes use multiple names to denote the same entity. Let us take an example shown in Fig. 3 in which two controllers exist: Level controller (LC29) and flow controller (FC29). Both controllers use the same control valve as an actuator. It is a typical example of cascaded control. LC29 takes care of the liquid level of the overhead drum which contains

Fig. 3. Cascaded control of
LC and FC

reflux(Naphtha). And FC29 is in charge of controlling the flow of Naphtha coming out of the overhead drum. The control valve is called "Level adjustment control valve" and "Naphtha extraction flow control valve" depending on which controller the operator focuses on. The problem, however, is that the focused controller is seldom explicit. So, the system has to infer where is the focus on and to trace the focus shifts during the course of operation. Further, the system has to know the term generation mechanism to attain the maximal flexibility of the system. Concerning the labels of each concept, we only ask the model builder of the target plant to write the default name of things appearing in the plant. The role name which will be attached to
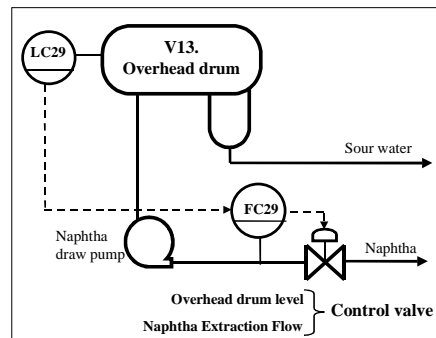
things are basically generated by the system according to the context identified. This makes system building easier. Especially, plant objects have various role names which the objects play from the viewpoint of the device which processes them. For example, *input material* is a typical role name which many of the plant objects could play.

The system has two kinds of rules such as **Focus tracing rules** and **Role name generation rules.** The former is for tracing the focal point to identify the context and the latter for "name" generation of the target entity. The philosophy of name generation is to associate a default name with each entity and rules are invoked when names other than the default one should be used. Note that roles names are generated adaptively to the topological structure of the plant.

While the rule application, knowledge about each concept/term is obtained by consulting the ontology and the plant model which has been built by connecting components generated by instantiation of corresponding concepts in the ontology.

Focus tracing rules

Assume the system is asked to choose an appropriate term for things(attributes, name, etc.) of an object identified by its unique identifier. Let us call the object CO: Current Object. The rule are divided into four cases according to what things of CO is:

Case 1: An attribute of a plant object(not a device)
(a) If CO is input or output of the device focused previously, then the focused device is the same as the last focused device.
(b) If the last focused device is either a controller or a control valve and CO is equal to that the controller measures, then the device the controller measures is the new focused device.
(c) If the task is decision of countermeasure, then the new focused device is the entire plant.
(d) Otherwise, no focused device exists.

Case 2: An attribute of a device
(a) If a controller which measures the attributes exists, then the new focused device is the controller.
(b) Otherwise, the new focused device is the same as the last focused device.

Case 3: A control valve(CO is a control valve itself)
(a) If CO is the same as the target object of the last focused device, the focused device is the same as the last focused device.
(b) Otherwise, the device which operates CO is the new focused device.

Case 4: others
(a) In all cases where device names are concerned other than the cases (2) and (3), CO is the new focused device.

Role name generation rules

The philosophy is to associate a default name with each thing and rules are invoked when names other than the default one should be used.
(1) For plant objects
    (a) If there is a focused device, then
        (a-1) if the focused device has role names of its input/output objects, then CO name is the role name of the input/output object of the focused device.
        (a-2) otherwise, the CO name is "<focused device> input/output".
    (b) Otherwise, default name of CO is used.
 (2) For a controller and a control valve
    (a) If the focused device measures an attribute of another device, then the name of the CO is "<the device which focused device controls>" + "<the attribute>" + ["control valve" or "controller"](e.g., overhead drum level controller).
    (b) If the focused device measures an attribute of a plant object and if the device controlled by the focused device has a role name of its input/output object, then CO name is "<the role name>" + "<the attribute>" + ["control valve" or "controller"]. If there is no role name, then the CO name is "<the controlled device>" + ["inlet" or "outlet"] + "<the attribute>" + ["control valve" or "controller"](e.g., desalter inlet temperature controller).
 (3) For a thing other than a control valve or a controller
    (a) If an attribute of CO is concerned, then return "<the object which the focused device measures>" + "<the attribute>".
    (b) Otherwise, CO name is the name of the focused device.

Template design of messages

A message should be easily understood by operators in any situation. So, its syntactic form should be highly stylized and cannot be long. We collected a lot of sample messages with the help of domain experts and classified the syntactic forms into the following seven types:

(1) Warning
(2) Near future prediction
(3) Candidate cause presentation
(4) Diagnosis result presentation
(5) Justification of decisions
(6) Countermeasure presentation
(7) Justification of countermeasures

Each type has a few templates for covering a small amount of variations. Each template is specified using concepts in the ontology. Message construction algorithm can be simple for the above reason. In fact, a simple blank filling method is employed. Templates are described in terms of intermediate categories contained in the ontology

Example: OS receives a message content from Interface Agent which monitors the plant. A message content is represented in a list as follows: (<Template type> <non-terminal symbols>*)

J: [警告 [VFC29, MV] 全開]
E: [Warning [VFC29, MV] full throttle]

J: 警告:ナフサ抜き出し流量コントロールバルブが全開です
E: Warning: Naphtha extraction flow control valve is in full throttle.

J: [原因候補1 [VFC29] スティック]
E: [Candidate cause1 [VFC29] stick]

J: 原因候補：リフラックスドラム液レベルコントロールバルブ
がスティック

Implementation and evaluation

OS has been implemented in Java and Lisp. The ontology developed has been implemented in Ontology editor discussed in Section 6. The number of concepts in the ontology is about 500. We did a full-scale experiment to evaluate the system. Domain experts developed seven scenarios of various faults with countermeasures for them and many types of messages to operators. The templates we built cover all the messages. We picked up all of the messages to evaluate the word selection and message generation functions of OS and confirmed all of the message contents sent from IA through LAN were successfully translated into the desirable messages in terms of appropriate terms. Fig. 4 shows an example screen dump of the prototype system . The lower half is a portion of the plant model we built and the left upper window is a window of IA and right one is of OS. The window configuration is arranged for demonstration. The plant model generated using the ontology is shared by all the agents. The domain experts evaluated the performance of the system favorably.

AN ENVIRONMENT FOR ONTOLOGY DEVELOPMENT AND ITS USE

The environment, named "Hozo", used for building the plant ontology and model, is composed of graphical interface, editor and ontology/model server in a Client-Server architecture. It is implemented in Java and editor is implemented as Java applet so that it can work as a client through the Internet. Hozo manages ontologies and models for each of the developers. Users can read and copy all the ontologies and models stored in Hozo, but cannot modify any developed by others. Models are built by

choosing and instantiating concepts in an ontology and by connecting the instances. After consistency checking of the model using axioms defined in the ontology, the model is ready for use by other agents. The model built is available in a Lisp format which is sound and machine interpretable.

Editor

The editor interface is composed of the following five parts:
1. **Browsing panel** displays an ontology/model graphically
2. **Definition panel** where definition of concepts and relations is done.
3. **Term list panel** lists all the concepts contained in the ontology alphabetically and is used for concept retrieval.
4. **Menu bar** for selecting tools
5. **Tool bar** for selecting commands

The editor can be viewed as a kind of two-dimensional language in the sense that users can develop an ontology by defining concepts with appropriate relations such as *is-a*, *part-of*, etc. in the browsing panel. User can define their own relations. Consistency of the ontology and of the model built from the ontology is
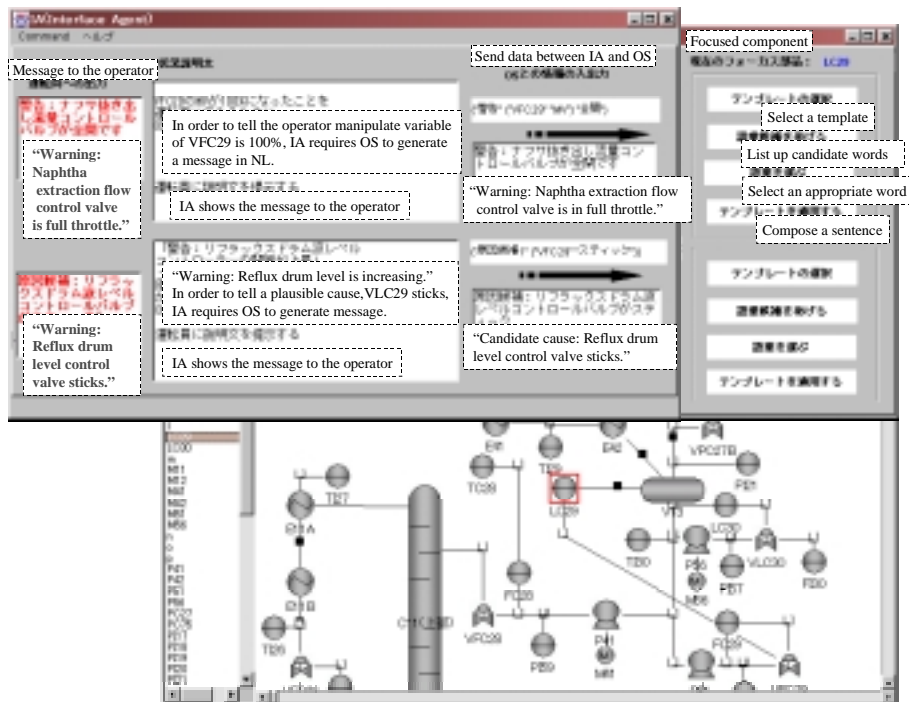

Fig. 4. Screen dump of a message generation demonstration

automatically guaranteed in terms of the subsumption and *instance-of* relations. Attributes are defined in the definition panel for each concept with the visible inheritance history.

Browsing panel

Browsing panel has two modes: Tree mode and Network mode. The former displays the ontology in a hierarchical structure using only *is-a* relations between concepts. In the latter mode, all the relations including those user defined are shown in a network structure. In the both modes, slots of a concept, usually representing its attributes and parts, can be shown by request. A number of mouse operations for manipulating trees and networks are available.

Definition panel

Definition panel allows users to read and define concepts and relations they designate in the Browsing panel. Items for edition are as follows:

label: It denotes a label denoting the concept being defined.
axiom: Constraints which has to be satisfied by all of its instances.
def: Informal definition in natural language
slot:(kind, name(label), class constraint, value)

"kind" denotes kinds of content of the slot such as *attribute* and *part* etc. "name" is a label of the slot, "class constraint" denotes what class should the "value" belong to and "value" is a value in the case that "kind" is *attribute* or a *part* in the case that "kind" is *part*. These are used for consistency checking by the server, and hence, property inheritance and consistency between subsumption and *instance-of* relations are checked by the server.

Model construction

A model is built using the ontology developed and is used by other agents. As described in the above, Hozo guarantees the compliance of the model with the ontology. All the necessary operations such as instance making, connecting them, deletion/ addition of instances, etc. for building and manipulating a model are done by built-in mouse operations in the browsing panel. During the model construction, users are given all the classes defined in the ontology to instantiate. So, the browser works as a model browser/editor. When the user selects a class to instantiate, its definition is displayed in the class-browsing panel. All the instances built properly inherit properties of all of their super-concepts. In fact, the model used in OS has been built like this and shared by all of the other agents as an ontology-compliant model.

Other functions

Managing and overriding inheritance

Managing *is-a* inheritance is done by Hozo automatically. When necessary, overriding the inherited value is allowed to specialize it more.

Fig. 5. A snapshot of the plant ontology definition

GIF attachment

A node, which denotes concept(class and instance), is usually displayed as a square. If necessary, GIF can be attached to it to make the instances real just like we did in our project which is shown in Fig. 4.

Output in a text format

The ontology and model can be output not only in a Lisp format for computer interpretation but also in a text format with indentation. XML version will be ready soon.

Implementation

The current version of Hozo has been implemented in Java(JDK1.1) and been used for two years not only by our lab members but also by some researchers outside. The following are some example ontologies developed thus far:

1. Ontology for Learning support systems
2. Ontology for Authoring systems
3. Ontology for Instructional design
4. Ontology of fault in diagnostic tasks
5. Ontology of function in artifact design

Fig. 5 shows a snapshot of the plant ontology definition about *Component* in a Network view mode. A node has its super concept to its left and has several slots for attribute and its part definition. Components are classified according to their functions because function is an essential property of them. Controller inherits "control function" from its super concept "Component for control" and has specialized

information such as its "control object is an instance of actuator" is written in the slot. While a slot is defined as "PV(Process Variable) takes a "value" in "Controller", that of its sub-concept, "Flow controller", is defined as "PV takes a "value of amount of flow" by overriding it.

Hozo is available at the URL: http://www.ei.sanken.osaka-u.ac.jp/oe/oe.html

Future work

We have identified some room to improve Hozo through its extensive use. The first topic is about effective guidelines for ontology development that is badly needed by developers. Because many of the existing guidelines are those similar to Software development guidelines, we need neater one, that is, one which can help users distinguish between classes and roles, identify appropriate relations and build a proper abstraction hierarchy of classes. Although this topic is the most important, it is out of the scope of this paper.

Other topics include basic functions which support neat representation of an ontology. Especially, we extended Hozo with respect to treatment of "Relation" and "Role" concepts. The following is the summary of extension:

- On the basis of that most of the things are composed of parts and that those parts are connected by a specific relation to form the whole, we introduced "Wholeness concept" and "Relation concept". The former is a conceptualization of the whole and the latter of the relation. Typical example is "married couple(fufu in Japanese)" and "fufu relation". Hozo will provide functions to manage the correspondence between these two different conceptualizations of the same entity. Needless to say, Hozo gives an identity to every relation.
- Description framework of roles: Role is specified in various ways. One of the typical way is specification in a *part-of* relation, like "husband" in "fufu", that is, husband is a role in a wholeness concept of a married couple(fufu).
- Sophisticated display of *part-of* relations and its editing
  The current Hozo has only one *part-of* relation which is transitive, but the next version will introduce several *part-of* relations some of which are not transitive.
- Augmentation of the axiom definition and the language

The latest version of Hozo has been currently implemented and will be open soon. Compatibility to existing ontology representation method like OIL[10] will be considered.

RELATED WORK

ONTOGENERATION[11] explains the content of ontology in Spanish based on two kinds of ontologies: domain and linguistic ontologies. Message generation in our system is not special. Templates are designed by observing sample messages provided by the domain experts. Sentence structure of the message is highly stylized because of

the small number of templates. Its unique feature exists in context-sensitive word selection rather than sentence generation itself.

Our view of an ontology is based mainly on its use in building a well-founded model, that is, we think meta-model functionality is of the most important. This contrasts well with that of Guarino's idea of top-level ontology design[7]. The idea of task ontology shares a lot with Common KADS[4][5].

Several ontology development methodologies have been proposed[12][14]. Both of them present users guidelines to follow together with sophisticated tools like those employed in the conventional knowledge acquisition process. Most of the tools are based on a frame-based knowledge representation language with an additional functionality for writing axioms. Hozo is similar to them in that sense, but is different from them in some respects: (1) It is essentially a GUI-based language. An ontology is defined through the graphical interface based on *is-a* and *part-of* hierarchies. (2) It has special functionalities for ontology design such as treatment of role concepts whose instatiation is done differently from that of a basic concept. (3) Clear discrimination among, **role**(husband role), **role-holder**(husband) and **basic concept**(man) is done to treat "Role" properly. (4) In the next version of Hozo, several kinds of *part-of* relations are to be introduced such as *component-part-of* which is the most common *part-of* relation, *material-part-of*, and so on[13]. (5) It does not allow multiple inheritance because most of the use of multiple inheritance in knowledge representation are inappropriate from ontology point of view.

Hozo shares an idea of ODE of METHONTOLOGY[14] in that it generates machine code of the ontology defined in a more informal way.

CONCLUSION

Plant ontology and its roles in the interface system for oil-refinery plant operation have been discussed. The plant ontology design has been almost completed and its utility has been demonstrated in message generation with appropriate word selection. The future work until the end of the project, March in 2001 includes design of a negotiation ontology for use of flexible and powerful collaboration among agents through sophisticated negotiation. Task ontology will be used for specifying functionality of each agent and hence for helping negotiation task ontology design. This topic is one of the main topics in our research plan.

We also discussed an environment for ontology development, Hozo, used for the development of our plant ontology. It was informally evaluated by domain experts and they gave favorable comments. They found utility of Hozo in making their knowledge explicit and in operationalizing it and want to use it in the daily activity. Hozo has been extensively used in many projects to develop various ontologies. Its revised version will be ready soon.

Acknowledgement

REFERENCES

[1] Mizoguchi, R. A Step towards Ontological Engineering, National Conference on AI of JSAI, AI-L13, 1998, http://www.ei.sanken.osaka-u.ac.jp/english/step-onteng.html.

[2] R. Mizoguchi, A. Gofuku, Y. Matsuura, Y. Sakashita, M. Tokunaga, Human Media Interface System for the Next Generation Plant Operation, Proc. of 1999 IEEE Int'l Conf. On SMC, Tokyo, October, pp.630-635, 1999.

[3] R. Mizoguchi, et al., Task Ontology for Reuse of Problem Solving Knowledge. KB&KS '95, pp.46-59, 1995

[4] V. R. Benjamins, B. Wielinga, J. Wielemaker, and D. Fensel : Brokering Problem-Solving, Knowledge at the Internet. In Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Workshop (EKAW-99), D. Fensel et al. (eds.), Lecture Notes in Artificial Intelligence, LNAI 1621, Springer-Verlag, May 1999

[5] G. van Heijst, A. Th. Schreiber, and B. J. Wielinga: Using explicit ontologies in KBS development. International Journal of Human-Computer Studies, 46(2/3):183–292, 1997.

[6] B. Chandrasekaran, J. R. Josephson, and R. Benjamins, What are ontologies, and why do we need them?, IEEE Intelligent Systems, Vol.14, No.1, pp.20-26, 1999

[7] Guarino, Nicola: Some Ontological Principles for Designing Upper Level Lexical Resources. Proc. of the First International Conference on Lexical Resources and Evaluation, Granada, Spain, 28-30 May 1998.

[8] M. Sasajima, Y. Kitamura, M. Ikeda, and R. Mizoguchi, FBRL:A Function and Behavior Representation Language, Proc. of IJCAI'95, pp.1830-1836, 1995.

[9] Y. Kitamura and R. Mizoguchi, Meta-Functions of Artifacts, The Thirteenth International Workshop on Qualitative Reasoning (QR-99), Scotland, June 6-9 1999.

[10] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. Van Harmelen, M. Klein, S. Staab, and R. Studer: OIL: The Ontology Inference Layer, to appear. http://www.ontoknowledge.com/oil.

[11] G. Aguado, et al., ONTOGENERATION; Reusing domain and linguistic ontologies for Spanish generation, Proc. of the ECAI 98 Workshop on Applications of ontologies and problem-solving methods, pp.1-10, 1998.

[12] M. Uschold and M. Gruninger, ONTOLOGIES: Principles, Methos and Applications, J. of Knowledge Engineering Review, Vol.11, No.2, 1996.

[13] Mizoguchi, R. et al., Foundation of ontological engineering – An ontological theory of semantic links, classes, relations and roles --, J. of JSAI, Vol. 14, No.6, pp.1019-1032, 1999(in Japanese).

[14] Lopez, M.F., Gomez-Perex, A. et al., Building a chemical ontology using Methontology and the ontology design environment, IEEE Intelligent Systems, Vol.14, No.1, pp.37-46, 1999.