An Ontology Editor in Hozo - Treatment of "Role" and "Relationship" -

Kouji Kozaki[a], Yoshinobu Kitamura[a], Mitsuru Ikeda[a], and Riichiro Mizoguchi[a]

[a] *The Institute of Scientific and Industrial Research, Osaka University*
8-1 Mihogaoka, Ibaraki, Osaka, 567 -0047 Japan
Tel: +81-6-6879-8416, Fax: +81-6-6879-2123, E-mail:{kozaki, kita,ikeda, miz}@ei.sanken.osaka-u.ac.jp

Abstract

A methodology of ontology design and a computer system supporting ontology design are needed. Our research goals include development of a methodology for ontology design and a its support environment. Although several systems for building ontologies have been implemented, they do not consider ontological theory very much. We discuss how to apply the "role-concept" and "relationship" in our environment, named Hozo, for creating and using ontologies. We present the architecture, functionalities of its modules, its interface and the some experiences on the design and use of ontologies.

Keywords:

ontology, ontology development system, role, relationship

Introduction

Recently, an ontology is expected to contribute to knowledge sharing and reuse[Mizoguchi 98]. It is, however, difficult to develop a well-organized ontology because the principles of ontology design are not clear enough. Therefore, a methodology for ontology design and a computer system supporting ontology design are needed. Our research goals are development of a methodology for ontology design and supporting environment based on the methodology.

Building an ontology requires a clear understanding of what can be concepts with what relations to others. An ontology focuses on "concepts" themselves rather than "vocabulary", and its design is not the problem of how to represent but that of identifying the inherent conceptual structure. For example, "a bicycle wheel" is recognized as different concepts such as "a rear wheel" and "a driving wheel" according to the context. To take another example, we found human operators use different terms to denote the same device depending on the context in a plant operation system. Furthermore, "a man" can be called "a husband", "a father" and "an employee". The difference between these concepts is discriminated based on the ontological theory of role-concept[Mizoguchi 99]. Similar problems have been discussed by some researchers[Guarino 98, Sowa 95].

Although several systems for building ontologies have been developed to date, they were not based on enough consideration of an ontological theory. We argue that a fundamental consideration of these ontological theories is needed to develop an environment for developing an ontology. Most of the previous ontologies, which are represented in frame-based languages, don't clearly deal with such concepts that need deep ontological investigation. Therefore we begin with a fundamental consideration of an ontological theory. We discuss mainly "role concept" and "relationship", and consider how these ontologically important concepts should be treated in our environment. On the basis of the consideration we have designed and have developed an environment for building and using ontologies, named "Hozo". This paper presents an outline of the functionality of Hozo. We focus on how it treats relations and roles on the basis of fundamental consideration.

The next section outlines the architecture of Hozo. Section 3 discusses a *role-concept* in *part-of* relation and the treatment of the *role-concept* in Hozo. In section 4 we introduce *a wholeness* concept* and *a relation concept.* Section 5 presents the implementation of Hozo and examples of its use. Next we discuss the related work following by conclusions and some future work.

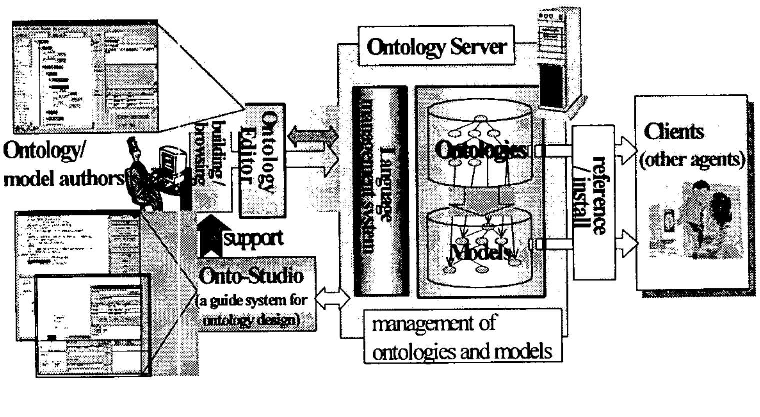


Fig.1. The architecture of Hozo

An environment for building ontologies

Hozo

We have developed an environment, named "Hozo[**]", for building ontologies based on fundamental ontological theories. "Hozo" is composed of "Ontology Editor", "Onto-Studio" and "Ontology Server"(Fig.1).

The ontology editor provides users with a graphical interface, through which they can browse and modify ontologies by simple mouse operations. This system manages properties between concepts in the *is-a* hierarchy. The Onto-Studio is based on a method of building ontologies, named AFM (Activity-First Method) [Mizoguchi 95], and it helps users design an ontology from technical documents. The ontology server manages the built ontologies and models.

Because the architecture is implemented in Java and the ontology editor is an applet, it can work as a client through Internet. Hozo manages ontologies and models considering who is its developer. For each ontologies in Hozo, its author can define and modify it, and the other users can only read and copy it. It lets share ontologies among users without explicit version control.

Models are built by choosing and instantiating concepts in the ontology and by connecting the instances. Hozo also checks the consistency of the model using the axioms defined in the ontology. The ontology and the resulting model are available in different formats (Lisp, Text, XML/DTD) that make it portable and reusable.

Ontologies which Hozo builds

An ontology is composed of concepts and relations which are necessary for representing the world of interest. *Is-a* relation is the most basic relation. It represents a super-sub concept relation. A sub-concept inherits everything from its super-concept.

An ontology reflects what exists out there in the world of interest or represents what we should think exists there. As result, an ontology provides us concepts and relationships which are used as building blocks of the model. It also provides guidelines for building models and constraints which the models should satisfy.

The definition of a concept is composed of the following items:

label : It denotes its name.

super: The name of a super-concept.

axiom: Constraints which have to be satisfied by all of instances of the concept.

def: Informal definition in natural language.

part-concept: Parts which constitute the concept.

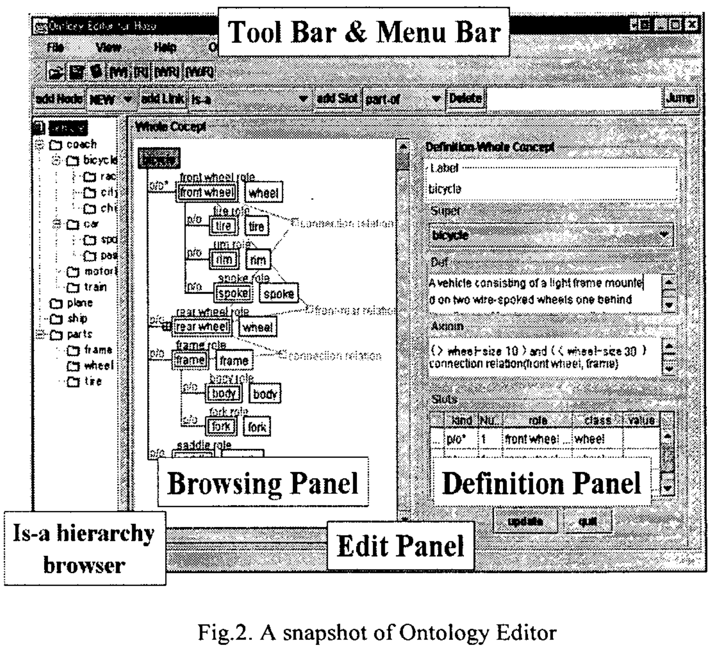
attribute: Attributes of the concept.

A relation between a part-concept and a whole-concept is represented by a *"part-of"* relation. That between an attribute and its concept is represented by an *"attribute-of"* relation. The axiom contains constraints which part-concepts or attributes should satisfy, and relations among the part-concepts. For example, these are constraints on the part-concept such as "any teachers must have a teaching certificate" in a "school", and "the size of wheels are from 10 inch to 30 inch" in a "bicycle". Another example is a constraint on the relation such as there must be a connection relation between a wheel and a frame in a "bicycle". The language for representing these axioms is under development.

Ontology Editor

The ontology editor provides users with a graphical interface, through which they can browse and modify ontologies by simple mouse operations. This interface consists of the following four parts (Fig.2):

1. ***Is-a* hierarchy browser** displays the ontology in a hierarchical structure according to only *is-a* relations between concepts.
2. **Edit panel** is composed of a *browsing panel* and a *definition panel*. The former displays the concept graphically, and the latter allows users define a selected concept in the *is-a* hierarchy browser.
3. **Menu bar** is used for selecting tools
4. **Tool bar** is used for selecting commands



Definition-Whole Concept

Label: bicycle

Super: bicycle

Def: A vehicle consisting of a light frame mounted on two wire-spoked wheels one behind

Axiom: (> wheel-size 10) and (< wheel-size 30) connection relation(front wheel, frame)

Slots

	kind	Nu.	role	class	value
...	p/o*	1	front wheel ...	wheel	

Fig.2. A snapshot of Ontology Editor

---

[*]We coined this term to represent our original idea of "the whole".

[**]"Ho" is a Japanese word and means unchanged truth, laws or rules in Japanese, and we represent "ontologies" by the word. "Zo" means to build in Japanese.

The *Is-a* hierarchy browser displays the ontology in a hierarchical structure according to only *is-a* relations between concepts. Using the *is-a* hierarchy browser, users can select concepts and modify the *is-a* relations. It does not treat the multiple inheritance because we consider that most of the uses of the multiple inheritance in knowledge representation are inappropriate from the ontological point of view. This issue is discussed in section 3.

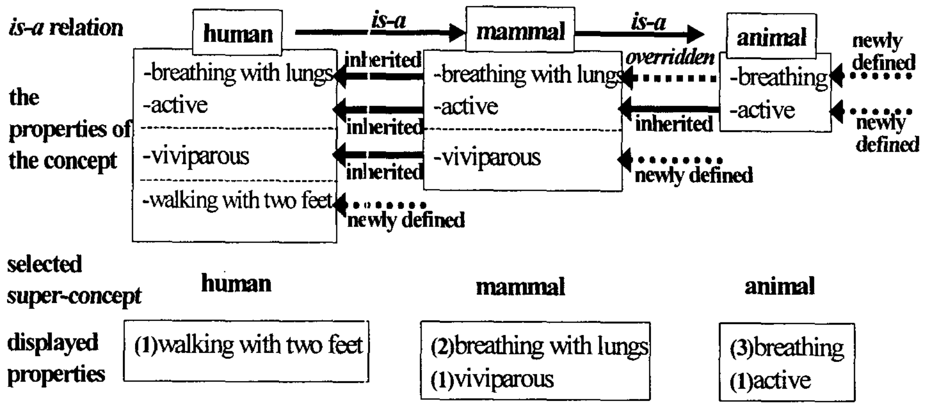


Fig.3. An example of properties displayed in the definition panel

The Edit panel displays the definition of the concept that is selected in the *is-a* hierarchy browser, and allows users to edit it. It is composed of a *browsing panel* and a *definition panel* (Fig.2). The browsing panel graphically displays *part-concepts* that constitute the selected concept. The definition panel allows users to read/write the definition of the concept selected in the browsing panel.

The browsing panel has two display modes: *tree mode* and *network mode*. In the tree mode, the concept is displayed in a hierarchical structure using *part-of* relations between *part concepts*. In the network mode, all relations - including a user defined ones - are shown in a network structure. In the both modes, slots of the concept, usually representing its attributes, can be shown by request. A number of mouse operations for manipulating trees and networks are also available.

The definition panel displays a detailed definition of the concept selected in the browsing panel. Users can read the definition of the concepts and define them. The items for editing are described in section 2-2.

The elements of definitions can be categorized into two types.

1. Newly defined definitions in the designated concept.
2. Inherited definitions from its super-concepts.

The list of super-concepts is shown by request. When users select a concept from the list, the definition panel displays the slot definition inherited from the selected concept. Moreover, the properties of the concept displayed in the definition panel are classified into three categories.

(1) Newly added properties in the concepts.

(2) Properties which are defined by overriding the definitions inherited from its super-concepts.

(3) Properties which are overridden in its sub-concepts

The panel offers color and font facilities for distinguishing them. Fig.3 shows an example of these categories for a concept "human".

Ontology Server

The ontology server provides several functions the ontology editor uses in the course of ontology development. It has 26 functions, necessary for ontology definition for example "define-concept", "add-slot", "get-super-class" and so on. During the building processes the ontology server also checks the consistency of the model using the axioms defined in the ontology. When there is any violation of the axiom the system sends error-message to the user. The ontologies and the model which are built based on them are stored in the Ontology Server and accessible from other systems by the following three ways.

1. **Common access through network:** Users can access the ontologies and models through Internet using the ontology editor.
2. **Translation into different formats**: The ontology server can translate the ontologies and models into different formats (Lisp, Text and XML/DTD) that make them portable and reusable.
3. **Access using API**: The operational functions which the ontology server provides are opened to the public as API. Using the API other systems can use all the functionalities of the ontology server.

A role concept in a part-of relation

Basic concept, role concept and role holder

A *part-of relation* represents a whole-part relation between the *whole-concept* and the *part-concept* which constitutes the *whole-concept*. For example, <"a wheel" ***part-of*** "a bicycle"> represents a relation between a bicycle and a wheel which is a component of the bicycle. The major semantics of the *part-of* relation is that it specifies that when an instance of a whole-concept is created, instances of its all part-concepts are also created.

The current version of Hozo has only a kind of *part-of* relation which is transitive. But in the next version we have planed to consider several kinds of *part-of* such as *component-part-of* which is the most common *part-of* relation, *material-part-of*, and so on [Mizoguchi 99]. Some of the *part-of* relations are not transitive.

Let us consider "a front wheel", in order to investigate the *part-of* relation. One may describe <"a front wheel" ***part-of*** "a bicycle"> also. A question now arises: how are "a

a part-role concept	*a basic concept*	*a role holder*
(a teacher role)	(a human)	(a teacher)
[R1] name	[B1] name	[R1] name
[R2-1] age(>22)	[B1] age	[R2-1]age(>22)
	[B2] height	[B2] height
	[B2] weight	[B2] weight
[R2-2] subjects		[R2-2] subjects
[R2-2] the length of employment		[R2-2] the length of employment
[R2-2] certificate		[R2-2] certificate

Fig.4. An example of part-concept definition

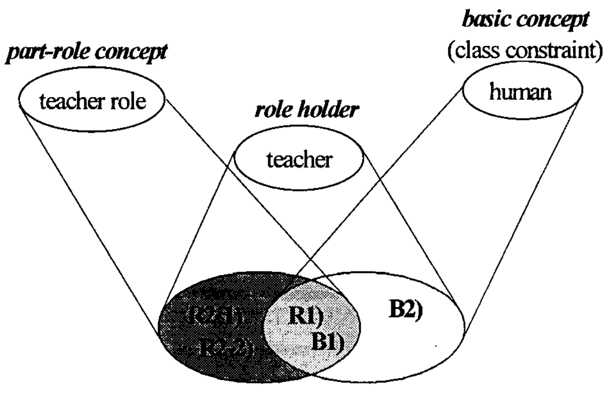


Fig.5. The relationship among definitions

wheel" and "a front wheel" different from each other?

"A front wheel" is not a mere label on "a wheel" because it has more information than "a wheel". Then, does <"front wheel" *is-a* "wheel"> hold? Some may answer "yes". It is, however, inappropriate from the ontological point of view.

John Sowa introduces the *firstness* and the *secondness* of concepts[Sowa 95]. The former is roughly defined as a concept which can be defined without mentioning other concepts. Examples include ion, a man, a tree, etc. The latter is roughly defined as a concept which cannot be defined without mentioning other concepts. Examples include wife, husband, student, child, etc. Concepts of the *secondness* type except artifacts are called ***role-concepts***. Based on his theory, we identified three categories for a concept. That is, a *basic concept*, a *role-concept*, and a *role holder*.

A ***role-concept*** represents a role which a thing plays in a specific context and it is defined with other concepts. On the other hand, a ***basic concept*** does not need other concepts for being defined. An entity of the basic concept that plays a role such as husband role or wife role is called a ***role holder***. There are various *role-concepts* such as roles dependent on the relation and those dependent on a task, etc. In this paper, we concentrate on role-concepts, which appear in the context of the *part-of* relation.

A *part-concept* in the *part-of* relation is composed of three conceptual elements.

Role-concept: A concept representing a role dependent on the *whole-concept*.

Class constraint: A constraint on the class to which the instance playing the role belongs.

Role holder: An entity of a *basic concept* which is holding the role.

The *class constraint* refers to the *basic concept* which is defined elsewhere. Then an instance that satisfies the *class constraint* plays the role and becomes the *role holder*. For example in "a bicycle", its wheel plays the role as a front wheel ("a front wheel role") or a role that steers its body ("a steering role"), which is defined as a *role-concept*. A wheel that plays these roles is called "a front wheel" and "a steering wheel", respectively, which are *role holders*.

The relationship between these concepts

A *role-concept* inherits some properties from a *basic concept* as its *class constraint*. Therefore properties of a *basic concept* are divided into two categories, in the context of a *part-role concept* definition, described as follows.

B1: properties which are inherited by the *role-concept.*

B2: properties which are not inherited by the *role-concept.*

The properties of a *part-role concept* are divided into the following categories:

R1: properties which are inherited from the *basic concept.*

R2: properties which are added in the *role concept.* They are divided into two.

R2-1: added constraints on properties which are inherited from the *basic concept.*

R2-2: new properties which are not defined in the *basic concept.*

Contents of ***B1*** and ***R1*** are absolutely equal. ***R2-1*** overrides parts of ***R1*** (***B1***). The definition of a *role holder* is a sum of that of a *part-role concept* (***R1*** and ***R2***) and that of a *basic concept* (***B1*** and ***B2***), and it is the sum of ***R1*** (***B1***), ***R2*** and ***B2***.

For example, Fig.4 shows definitions of a *role-concept* "teacher role", a *basic concept* "human" and a *role holder* "teacher". In this example, the definition of "teacher role" has "name" as ***R1*** inherited from the *basic concept* "human". The definition of the teacher role includes constraints on "age"(***R2-1***) such as "any teacher must be over twenty two years old". It represents a constraint on the *basic concept* that can play the "teacher role". In addition to these, the teacher role has some additional attributes (***R2-2***) such as "the subjects that the teacher teaches", "the length of employment" and "certificate". Although in this example we simply define the concept using only attributes, the definition of other definition elements such as part-concepts and axioms are defined in the same manner.

Fig.5 shows relationships among the definitions of three concepts of our example. In this figure the top circles represent the three concepts, and the bottom circles represent sets of their properties. This figure tells us properties of the *role holder* "teacher" includes whole properties of the *basic concept* "human".

Inheritance between the *role holder* and the *role-concept* is formally equal as the inheritance relation of an *is-a* relation. As mentioned in section 3.1, however, a *role holder* is not a *sub-concept* of a *basic concept*, and it is such a concept that a *basic concept* plays the role. So, the inheritance is different from a multiple inheritance of *is-a* relation from ontological point of view. We will discuss the difference in the following paragraph.

There have been a lot of discussions about multiple inheritance. In software engineering, they are focused on formal issue such as how it is represented and how it is implemented in the software. However from ontological viewpoint our approach focuses on not its "representation" but its "content", that is, how we should understand the target world. It is important to note the difference between "representation" and "content".

Let us consider a typical example of multiple inheritance. "Mr. Smith" is an instance of "a human" and that of "a teacher" too. Using multiple inheritance of is-a relation, it can be represent that "Mr. Smith" is an instance of the class "a human who is a teacher" which is a sub-concept of both "a human" and "a teacher".

This representation, however, causes some problems in the following cases.

- Even if "Mr. Smith" retires and stop to be an instance of "a teacher", he will have been an instance of "a human".
- When "Mr. Smith" dies and stops to be an instance of "a human", the instance of "a teacher" will disappear as well.

In these cases, the semantics of *is-a* or *instance-of* relation is inconsistent. On the other hand if the semantics of both relation is strictly unified, the representation is ontologically inappropriate.

This problem can be represented as follows by using the three conceptual elements, that is, a *role-concept*, a *class constraint*, and a *role holder*, which are mentioned in section 3.1.

- "Mr. Smith" is an instance of a basic concept "human".
- And it plays a role concept "teacher role", and then it becomes a role holder "teacher".

This example shows that the confusion of the "relation between a role concept and a role holder" with the *is-a* relation causes the problem. Our environment makes it possible to distinguish these relations explicitly. Guarino discusses similar problems as *is-a overloading* and categorizes them into five types[Guarino 98]. This discussion is so important in the fundamental study of ontology that we will investigate it in further detail.

Treatment of the role concept

In the browsing panel a *part-of* relation and a part-role concept are represented by such a manner that is shown in Fig.6a. A diagram of a part-concept is composed of three parts. Each of them represents 1) a *role-concept*, 2) a *class constraint*, and 3) a *role holder*. A symbol besides a link connecting a whole-concept and a part-concept denotes kinds of relation ("p/o" denotes *part-of* relation, and "a/o" denotes *attribute-of*) and a numeral represents the number of part-concepts(or attributes). Fig.6b shows a wheel, which is referred in the *class constraint*, plays "a front wheel role", and the wheel becomes a *role holder* "a front wheel".

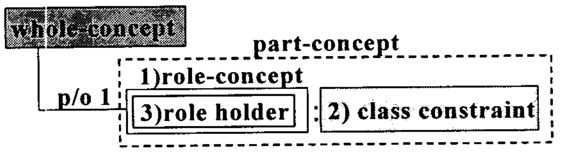
When users select a rectangle in the browsing panel, the definition panel allows them to read and define concepts they designate. In order to treat part-concepts, we prepare two kinds of definition panels for *basic concepts* and *part concepts*. The panel for basic concepts displays contents discussed in section 2-3. That for part concepts displays definition of part-concepts, shown in Fig.7. At the top of the panel, the label of a *role concept*, a *class constraint* and a *role holder* are shown. At the bottom of this panel, definitions of these three concepts are shown on a tabbed panel. Users can switch the following three views to read and edit the definition.

Part view: The panel displays definition of the *role-concept*. It allows users to ***add a new definition*** and ***constraints on properties inherited from a basic concept.***

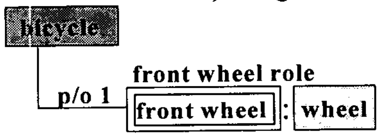
Basic view: The panel displays definition of a *basic concept* referred to in the class constraint. It allows users only to ***select inheritable properties to the role concept.***

Full view: The panel displays the definition of the *role holder*. It allows users ***only to read*** the definitions.

The users' editing process thus consists of two steps, to select inheritable properties in the basic view and to edit properties in the part view. These views are also switched when users click rectangles represented in the browsing panel corresponding to these three views.

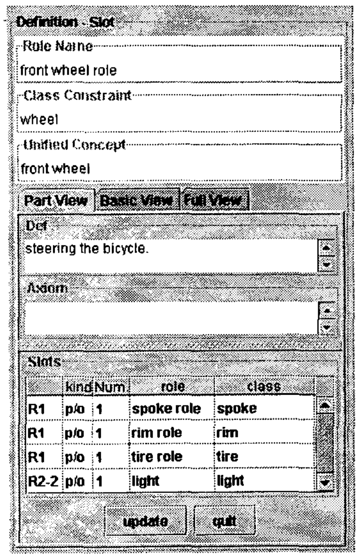


a) a legend



b) an example of "bicycle"

Fig.6. A legend of *part-of* relation



Role Name: front wheel role

Class Constraint: wheel

Unified Concept: front wheel

Def: steering the bicycle.

Axiom:

Slots

	kind	Num	role	class
R1	p/o	1	spoke role	spoke
R1	p/o	1	rim role	rim
R1	p/o	1	tire role	tire
R2-2	p/o	1	light	light

Fig.7. The definition panel for part-concepts

Relation concept

Relation concept and wholeness concept

There are two ways of conceptualizing a thing. Consider a "brothers" and a "brotherhood". "The Smith brothers" is a conceptualization as a *concept*, on the other hand "brotherhood between Bob and Tom" is conceptualized as a *relation*. On the basis of the observations 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 is that of the relation. In the above example, the "brothers" is a *wholeness concept* and the "brotherhood" is a *relation concept*.

A ***wholeness concept*** contains *part-concepts*, which compose the whole-concept, as its parts. And the relation between the *wholeness concept* and the part-concept is a *part-of* relation. So the three concepts, that is, a *part-role concept*, a *class constraint*, and *role holder* discussed in section 3, appear. On the other hand, a ***relation concept*** does not contain *participating concepts*. And the relation between the *relation concept* and the participating concept is not a *part-of* but a *participant-in* relation. A participating concept in the *participate-in* relation is composed of three conceptual elements, that are a *role-concept*, a *class constraint*, and a *role holder*, in the same way as a part concept in *part-of* relation. *Relation concepts* also have *is-a* relations and *super/sub concepts*.

Because a *wholeness concept* and a *relation concept* are different conceptualizations derived from the same thing, they correspond to each other. The *role-concepts* in a *wholeness concept* and those in a *relation concept* are the same. Theoretically, every thing that is a composite of parts can be conceptualized in both perspectives as a *wholeness concept* and a *relation concept*. In fact, there are three types of concepts according to the strength of relationship perspectives:

Wholeness concept perspective is stronger: e.g. artifacts like a bike, a desk, etc.

While a bike is composed of wheels, handlebars, a saddle, etc., it is rare that the relationship of them is conceptualized, say, these parts are in "a bike relation (a relation among parts composing a bike)".

Both perspectives are natural: brother / brotherhood, married couple / marital relationship, parent and child / parent-child relationship, etc.*

Relation concept perspective is stronger: front-rear relation, human relation, etc.

While front-rear relation is a common concept, a wholeness concept "things in a front-rear relation" is rarely conceptualized.

A use for relation concepts in an axiom

As mentioned in section 2.2, the axiom of concepts contains constraints which part- concepts or attributes should satisfy, and relations among part-concepts. A *relation concept* is used to represent the constraint on relations such that there must be a relation between instances of part-concepts in the model. Then some part-concepts play multiple roles, a role in the *wholeness concept* and that in the *relation concept*.

For example, let us consider the *wholeness concept* "a family". "A family" is represented as a *wholeness concept* which is composed of the part-concepts such as "a man" playing "a father role", "a woman" playing "a mother role", and "a human" playing "a child role". A user defines an axiom that there must be "a marital relationship" between the father and the mother in the family. Then the man playing the father role in this family plays the "husband role" also. Furthermore, when "a parent-child relationship" between the man and the child is described, this man plays the "parent role" in the parent-child relationship as well.

Treatment of relation concepts and wholeness concepts

On the basis of the consideration in the previous section, the environment for ontology development must manage the correspondence between a *wholeness concept* and a *relation concept*. The ontology editor displays *wholeness concepts* and *relation concepts* on separate panels. In Fig.8, the left panel shows the *wholeness concepts* and the right panel shows the *relation concepts*. We outline how these two concepts are defined in our editor.

1. A user describes "a married couple (*fufu* in Japanese)" as a *wholeness concept*. Then he/she defines *a husband role* and *a wife role*.
2. The user designates the *wholeness concept* representing a married couple in the left panel, and selects a command ***to define a relation concept***

* All concepts are valid in Japanese

corresponding to the *wholeness concept.*

3. A *relation concept* "a marital relationship" is defined semi-automatically. At the same time *a husband role* and *a wife role* are defined by sharing the definitions of *the husband role* and *the wife role* which is defined in 1.
4. The user describes a *wholeness concept* "family" which consists of *a father*, *a mother*, and *children.*
5. Next the user selects *part-concepts* representing *a father* and *a mother* in the family. Then he/she chooses *a marital relationship* as *a kind of relation* in the tool bar, and selects a command ***to add a relation*** between them.
6. An axiom that there must be a marital relationship between a father and a mother in a family is thus added in the definition of a family.
7. Then according to the definition of a marital relationship a man playing the father role plays the husband role, and a woman playing the mother role plays the wife role as well.
8. The user edits definitions of each role-concepts in the definition panel.

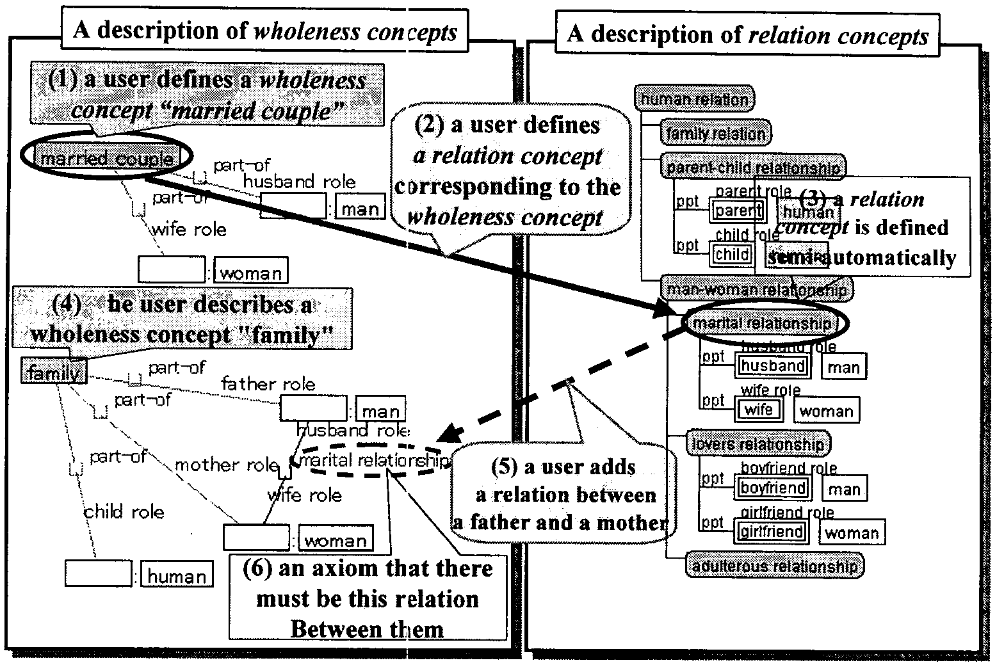

A description of *wholeness concepts*
A description of *relation concepts*
(1) a user defines a *wholeness concept "married couple"*
(2) a user defines *a relation concept* corresponding to the *wholeness concept*
(3) a *relation concept* is defined semi-automatically
(4) the user describes a wholeness concept "family"
(5) a user adds a relation between a father and a mother
(6) an axiom that there must be this relation Between them
married couple
part-of
husband role
wife role
man
woman
family
father role
mother role
child role
human
marital relationship
human relation
family relation
parent-child relationship
parent role
child role
parent
child
man-woman relationship
husband
wife
lovers relationship
boyfriend role
girlfriend role
boyfriend
girlfriend
adulterous relationship
ppt

Fig.8. A description of a relational concepts

As discussed in section 4.1 *role-concepts* in a *wholeness concept* "married couple" and those in a *relation concept* "marital relationship" are derived from the same entity. Therefore Hozo keeps the "husband role" in a married couple and the "husband role" in a marital relationship to have the same definition. And the "husband role" is referred to in the "family" (in above 7), which has the same definition as the "husband role" shown above.

When users edit definition of a concept having multiple roles like husband and father, in the definition panel they select a *role-concept* to edit from a list of the *role-concepts* which the *basic concept* can play.

Implementation and application

The current version of the ontology editor for Hozo has been implemented in Java (JDK1.3) and been used for four years not only by our lab members but also by some researchers outside. The following are some example ontologies developed thus far:

1. A plant ontology in the interface system for oil-refinery plant operation[Mizoguchi 00]
2. Task ontology of learning support systems[Jin 99]
3. An ontology of learning goal in CSCL[Inaba 00]
4. An integrated ontology for defining collaborative learning experiences[Barros 01]

Here we give more detail the plant ontology. The plant model contains a remarkable fact that multiple names are used to denote the same entity. Let us take an example shown in Fig.9 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 reflux (Naphtha). And FC29 is in charge of controlling the flow of Naphtha coming out of the overhead drum . The control valve is called by different name depending on which controller the operator focuses on.

In Hozo, this example is represented that the basic concept "control valve" plays multiple roles depending on the context. Fig.10 shows a snapshot of the plant ontology definition about *Controller.* "Flow Controller" and "Level Controller" inherit control function from its super concept "Controller" and have "Valve" as a class constraint "Target Component of Operation" slot, which is a specialization of "Actuator". In "Flow Controller" the valve plays "Flow-Control role" which depends on "Flow-Control relation", and it becomes the role-holder "Flow-Control Valve". And in "Level Controller" the valve plays "Level-Control role" which depends on "Level-Control relation", and it becomes the role-holder "Level-Control Valve". "Flow-Control relation" and "Level-Control relation" are relational concepts obtained by

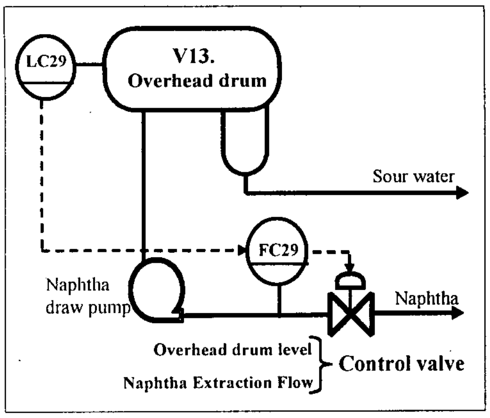
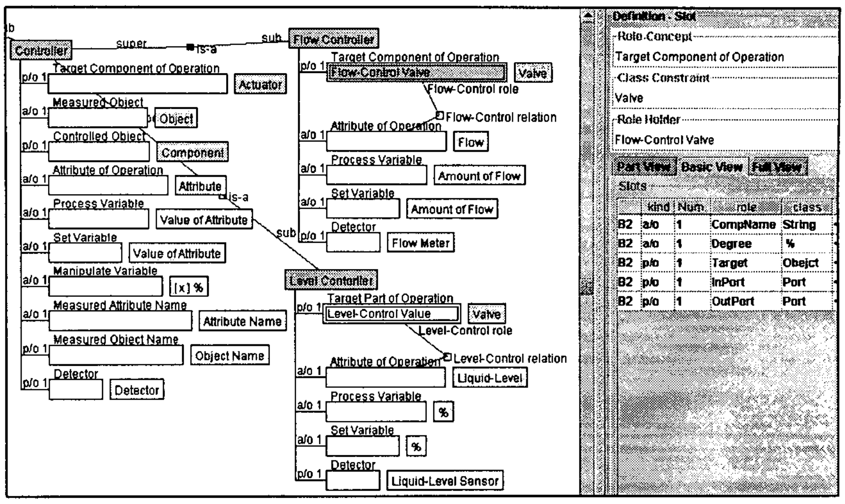


Fig.9. Cascaded control of LC and FC



Role-Concept
Target Component of Operation
Class Constraint
Valve
Role Holder
Flow-Control Valve

Slots

	kind	Num	role	class
B2	a/o	1	CompName	String
B2	a/o	1	Degree	%
B2	p/o	1	Target	Obejct
B2	p/o	1	InPort	Port
B2	p/o	1	OutPort	Port

Fig.10. A snapshot of the plant ontology definition

conceptualizing the functions of component as a relation function. In the instance model which is built based on this ontology, an instance of "Flow Controller"(FC29) and an instance of "Level Controller"(LC29) share the same valve. The valve plays multiple roles, and it is recognized as a different role holders according to the context. This example shows Hozo can treat the change of recognitions by introducing the role-handling technique based on the ontological theory.

Related Work

Our view of an ontology is based mainly on its use in building a well-founded model, that is, we think meta-model functionality of an ontology is the most important. This contrasts well with that of Guarino's idea of top-level ontology design[Guarino 98].

Hozo shares an idea of ODE of METHONTOLOGY[Lopez 99] in that it generates machine code of the ontology defined in a more informal way.

Several ontology development environments have been already developed[Farquhar 96, Swartout 96, Mahalingam 99, Domingue 98]. 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. Clear discrimination among a ***role-concept*** (husband role), a ***role-holder*** (husband) and a ***basic concept*** (man) is done to treat "Role" properly.
2. Management of the correspondence between a ***wholeness concept*** and a ***relation concept***.
3. It does not allow multiple inheritance of *is-a* relation because most of the use of multiple inheritance in knowledge representation are inappropriate from ontological point of view.

Conclusion and Future work

We discussed an environment for ontology development, Hozo, concentrating mainly on how its ontology editor treats *role-concepts* and *wholeness/relation concepts*. Hozo is designed based on a fundamental consideration of an ontological theory. 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 would like to use it in the daily activity. Hozo has been extensively used in many projects to develop various ontologies.

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 a lot 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 important, it is out of the scope of this paper. Other topics include basic functions which support neat representation of an ontology. The following is the summary of the extension:

- 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.
- Ontological organization of various role-concepts
- Augmentation of the axiom definition and the language

References

[Barros 01] Barros, B., Mizoguchi, R., and Verdejo, F.: A Platform for Collaboration Analysis in CSCL: An ontological approach, Proceedings of AIED01, San Antonio, Texas, May 19-23 2001

[Domingue 98] Domingue, J.: Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web, Proceedings of the 11th Banff Knowledge Acquisition Workshop., 1998

[Farquhar 96] Farquhar, A., Fikes, R. and Rice, J.: The Ontolingua Server: a Tool for Collaborative Ontology Construction, Proceedings of the 10th Banff Knowledge Acquisition Workshop, 1996

[Guarino 98] Guarino, N.: 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.

[Inaba 00] Inaba, A., Thepchai, S., Ikeda, M., Mizoguchi, R., and Toyoda, J.: An overview of "Learning Goal Ontology", Proc. of ECAI2000 Workshop on Analysis and Modeling of Collaborative Learning Interactions, pp.23-30, Berlin, Germany, 2000

[Jin 99] Jin, L., Chen, W., Hayashi, Y., Ikeda, M., Riichiro Mizoguchi, R., Takaoka, Y., Ohta, M.:An Ontology-Aware Authoring Tool - Functionalstructure and guidance generation -, Proc. of AIED'99

[Lopez 99] 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.

[Mahalingam 99] Mahalingam, K. and Huhns, M.: Java Ontology Editor (JOE) TUTORIAL, "http://www.engr.sc.edu/research/CIT/demos/java/joe/", 1999

[Mizoguchi 95] Mizoguchi, R., Ikeda, M., Seta, K. and Vanwelkenhuysen, J.: Ontology for Modeling the World from Problem Solving Perspectives, Proc. of IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing, pp. 1-12, 1995.

[Mizoguchi 98] 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.

[Mizoguchi 99] 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).

[Mizoguchi 00] Mizoguchi, R., Kozaki, K., Sano, T., and Kitamura, Y.: Construction and Deployment of a Plant Ontology, 12th International Conference on Knowledge Engineering and Knowledge Management, Juan-les-Pins, French Riviera, October, 2000.

[Sowa 95] John F. Sowa: Top-level ontological categories, International Journal of Human and Computer Studies, 43, pp.669-685, 1995

[Swartout 96] Swartout, B., Patil, R., Knight, K. and Russ, T.: Toward Distributed Use of Large-Scale Ontologies, Proceedings of the 10th Banff Knowledge Acquisition Workshop, 1996Engelmore, R., and Morgan, A. eds. 1986. *Blackboard Systems*. Reading, Mass.: Addison-Wesley.