A Note on Treatment of Incomplete Information in Object–Oriented Rough Sets

Yasuo Kudo
Department of Computer Science and Systems Engineering
Muroran Institute of Technology
Mizumoto 27-1, Muroran 050-8585, Japan
E-mail: kudo@csse.muroran-it.ac.jp

Tetsuya Murai
Graduate School of Information Science and Technology
Hokkaido University
Kita 14, Nishi 9, Kita-ku, Sapporo 060-0814, Japan
E-mail: murahiko@main.ist.hokudai.ac.jp

***Abstract*— We consider to treat incomplete information in the framework of object–oriented rough set models proposed by Kudo and Murai. The object–oriented rough set model treats structural hierarchies among objects based on is-a relationship and has-a relationship, however, treatment of incomplete information about objects is not illustrated. In this paper, we introduce null value objects to the object–oriented rough set model, which illustrate incompleteness of objects that comes from "absence of parts". Moreover, we extend Kryszkiewicz's tolerance relation to apply to the object–oriented rough sets.**

I. Introduction

Pawlak's Rough set theory [3], [4] provides a basic framework of set-theoretical approximation of concepts and reasoning about data. Treatment of incomplete information is one of the most interesting extensions of rough set theory. Various proposals have been made about semantics of unknown value that illustrate incomplete information in the framework of rough sets (for example, [6], [1], [7]). According to Stefanowski and Tsoukiàs [7], interpretations of unknown values are distinguished in the following two semantics:

- the "missing value" semantics (unknown values allow any comparison)
- the "absent value" semantics (unknown values do not allow any comparison)

Kudo and Murai have proposed object–oriented rough set models [2]. The object–oriented rough set model is an extension of the "traditional" rough set theory by introducing object–oriented paradigm used in computer science, and the object–oriented rough set model illustrates hierarchical structures between classes, names and objects based on is-a and has-a relationships. However, in [2], all objects are assumed to have all parts completely, therefore treatment of incompleteness of objects is not illustrated.

In this paper, we consider to treat incomplete information in the framework of object–oriented rough set models. Thus, we introduce null value objects to the object–oriented rough set model, which illustrate incompleteness of objects that comes from "absence of parts". Moreover, we extend Kryszkiewicz's tolerance relation [1] to apply to the object–oriented rough sets.

II. Object–Oriented Rough Sets

We briefly review object–oriented information systems proposed by the authors [2]. First, we describe the concept of class, name and object. Next, we illustrate well-defined structures as a basic framework of the object–oriented rough set model. Moreover, we introduce equivalence relations based on "equivalence as instances". Note that the contents of this section are entirely based on [2].

A. Class, Name, Object

An *object–oriented information system* consists of the following three structures: a *class structure* $\mathcal{C}$, a *name structure* $\mathcal{N}$ and a *object structure* $\mathcal{O}$, respectively:

$$\mathcal{C} = (C, \ni_C, \sqsupseteq_C),\ \ \mathcal{N} = (N, \ni_N, \sqsupseteq_N),\ \ \mathcal{O} = (O, \ni_O, \sqsupseteq_O).$$

Each $X \in \{C, N, O\}$ is a finite non-empty set such that $|C| \leq |N|$ ($|X|$ is the cardinality of X). Each element $c \in C$ is called a *class*. Similarly, each $n \in N$ is called a *name*, and each $o \in O$ is called an *object*. The relation $\ni_X$ is an acyclic binary relation on X, and the relation $\sqsupseteq_X$ is a reflexive, transitive, and asymmetric binary relation on X. Moreover, $\ni_X$ and $\sqsupseteq_X$ satisfy the following property:

$$\forall x_i, x_j, x_k \in X,\ x_i \sqsupseteq_X x_j,\ x_j \ni_X x_k \ \Rightarrow\ x_i \ni_X x_k. \quad (1)$$

The class, name and object structures have the following characteristics, respectively:

- The class structure illustrates abstract data forms and those hierarchical structures based on part / whole relationship (has-a relation) and specialized / generalized relationship (is-a relation).
- The name structure introduces numerical constraint of objects and those identification, which provide concrete design of objects.
- The object structure illustrates actual combination of objects.

Two relations $\ni_X$ and $\sqsupseteq_X$ on $X \in \{C, N, O\}$ illustrate hierarchical structures among elements in X. The relation $\ni_X$ is called a *has-a relation*, which illustrates part / whole relationship. $x_i \ni_X x_j$ means "x_i has-a x_j", or "x_j is a part of x_i". For example, $c_i \ni_C c_j$ means that "the class c_i has a

class c_j", or "c_j is a part of c_i". On the other hand, the relation $\sqsupseteq_X$ is called an *is-a relation*, which illustrates specialized / generalized relationship. $x_i \sqsupseteq_X x_j$ means that "x_i is-a x_j". For example, $\sqsupseteq_C$ illustrates relationship between superclasses and subclasses, and $c_i \sqsupseteq_C c_j$ means that "c_i is a superclass of c_j", or "c_j is a subclass of c_i".

B. Well–Defined Structures

Each object $o \in O$ is defined as an instance of some class $c \in C$, and the class of o is identified by the *class identifier* function. The class identifier id_C is a *p-morphism* between $\mathcal{O}$ and $\mathcal{C}$ (cf. [5], p.142), that is, the function $id_C : O \longrightarrow C$ satisfies the following conditions:

1. $\forall o_i, o_j \in O, o_i \ni_O o_j \Rightarrow id_C(o_i) \ni_C id_C(o_j)$.
2. $\forall o_i \in O,\ \forall c_j \in C,\ id_C(o_i) \ni_C c_j \Rightarrow \exists o_j \in O$ s.t. $o_i \ni_O o_j$ and $id_C(o_j) = c_j$,

and the same conditions are also satisfied for $\sqsupseteq_O$ and $\sqsupseteq_C$. $id_C(o) = c$ means the object o is an instance of the class c.

The object structure $\mathcal{O}$ and the class structure $\mathcal{C}$ are also connected through the name structure $\mathcal{N}$ by the *naming function* $nf : N \longrightarrow C$ and the *name assignment* $na : O \longrightarrow N$. The naming function provides names to each class, which enable us to use plural instances of the same class simultaneously. On the other hand, the name assignment provides names to every objects, which enable us to identify objects by names.

Formally, the naming function $nf : N \longrightarrow C$ is a surjective p-morphism between $\mathcal{N}$ and $\mathcal{C}$, and satisfies the following *name preservation constraint*:

- For any $n_i, n_j \in N$, if $nf(n_i) = nf(n_j)$, then $H_N(c|n_i) = H_N(c|n_j)$ is satisfied for all $c \in C$,

where $H_N(c|n) = \{n_j \in N | n \ni_N n_j, f(n_j) = c\}$ is the set of names of c that n has. The requirement that nf is a surjective p-morphism means that there is at least one name for each class, and structures between names reflect all structural characteristics between classes. The name preservation constraint requires that, for any class $c_i, c_j \in C$ such that $c_i \ni_C c_j$, and any name $n \in N$ with $nf(n) = c_i$, all names of the parts of c are uniquely determined. Thus, the number of names of c_j is fixed as $m = |H_N(c_j|n)|$, and we can simply say that "the class c_i has m objects of the class c_j".

On the other hand, the name assignment $na : O \longrightarrow N$ is a p-morphism between $\mathcal{O}$ and $\mathcal{N}$, and satisfies the following *uniqueness condition*:

- For any $x \in O$, if $H_O(x) \neq \emptyset$, the restriction of na into $H_O(x)$: $na|_{H_O(x)} : H_O(x) \longrightarrow N$ is injective,

where $H_O(x) = \{y \in O \mid x \ni_O y\}$ is the set of objects that x has. $na(x) = n$ means that the name of the object x is n. The uniqueness condition requires that all distinct parts $y \in H_O(x)$ have different names.

We say that $\mathcal{C}$, $\mathcal{N}$ and $\mathcal{O}$ are *well-defined* if and only if there exist a naming function $nf : N \longrightarrow C$ and a name assignment $na : O \longrightarrow N$ such that

$$id_C = nf \circ na, \tag{2}$$

that is, $id_C(x) = nf(na(x))$ for all $x \in O$.

In this paper, we concentrate well-defined class, name and object structures. In well-defined structures, if a class c_i has m objects of a class c_j, then any instance o_i of the class c_i has exactly m instances $o_{j1}, \cdots, o_{jm}$ of the class c_j [2]. This good property enables us the following description for clear representation of objects. Suppose we have $o_1, o_2 \in O$, $n_1, n_2 \in N$, and $c_1, c_2 \in C$ such that $o_1 \ni_O o_2$, and $na(o_i) = n_i$, $nf(n_i) = c_i$ for $i \in \{1, 2\}$. We denote $o_1.n_2$ instead of o_2 by means of "the instance of c_2 named n_2 as a part of o_1".

In object–oriented information systems, attributes and values of "traditional" information systems are special cases of classes and objects. Attributes are defined as classes with no parts, and the set of attributes AT is defined as follows:

$$AT = \{c \in C \mid c \not\ni_C c',\ \forall c' \in C\}. \tag{3}$$

Values are defined as instances of attributes. We call such instances of attributes *value objects*. Note that we assume that we can compare the "values" of null value objects of the same class.

Example 1: We use the same settings with examples of the object–oriented rough set model in [2]. Let $\mathcal{C} = (C, \ni_C, \sqsupseteq_C)$ be a class structure with $C = \{\mathsf{PC},\ \mathsf{DeskTopPC},\ \mathsf{2CPU-DTPC},\ \mathsf{CPU},\ \mathsf{Memory},\ \mathsf{HDD},\ \mathsf{Clock},\ \mathsf{MSize},\ \mathsf{HSize}\}$, and we have the following relationships:

Is-a relation:
$\mathsf{DeskTopPC} \sqsupseteq_C \mathsf{PC}$,
$\mathsf{2CPU-DTPC} \sqsupseteq_C \mathsf{DeskTopPC}$,
$\mathsf{2CPU-DTPC} \sqsupseteq_C \mathsf{PC}$,
$\cdots$.

Has-a relation:
$\mathsf{PC} \ni_C \mathsf{CPU}$, $\mathsf{PC} \ni_C \mathsf{Memory}$,
$\mathsf{Memory} \ni_C \mathsf{MSize}$, $\mathsf{HDD} \ni_C \mathsf{HSize}$,
$\cdots$.

Suppose moreover that Clock, MSize, and HSize are attributes. By the property (1), these relations illustrate connections between classes, for example, "2CPU-DTPC is-a PC" and "PC has-a CPU" imply "2CPU-DTPC has-a CPU".

Next, let $\mathcal{N} = (N, \ni_N, \sqsupseteq_N)$ is a name structure with $N = \{\mathrm{pc},\ \mathrm{desk\_top\_pc},\ \mathrm{2cpu\_dtpc},\ \mathrm{cpu},\ \mathrm{cpu2},\ \mathrm{memory},\ \mathrm{hdd},\ \mathrm{clock},\ \mathrm{msize},\ \mathrm{hsize}\}$ and the following relationships:

Is-a relation:
$\mathrm{desk\_top\_pc} \sqsupseteq_N \mathrm{pc}$,
$\mathrm{2cpu\_dtpc} \sqsupseteq_N \mathrm{desk\_top\_pc}$,
$\cdots$.

Has-a relation:
$\mathrm{desk\_top\_pc} \ni_N \mathrm{cpu}$
$\mathrm{2cpu\_dtpc} \ni_N \mathrm{cpu}$, $\mathrm{2cpu\_dtpc} \ni_N \mathrm{cpu2}$,
$\mathrm{cpu} \ni_N \mathrm{clock}$, $\mathrm{memory} \ni_N \mathrm{msize}$,
$\cdots$.

Moreover, suppose we have a naming function $nf : N \longrightarrow C$

such that

$$nf(\mathrm{pc}) = \mathsf{PC},\ nf(\mathrm{desk\_top\_pc}) = \mathsf{DeskTopPC},$$
$$nf(\mathrm{2cpu\_dtpc}) = \mathsf{2CPU-DTPC},$$
$$nf(\mathrm{cpu}) = nf(\mathrm{cpu2}) = \mathsf{CPU},$$
$$nf(\mathrm{memory}) = \mathsf{Memory},\ nf(\mathrm{hdd}) = \mathsf{HDD},$$
$$nf(\mathrm{clock}) = \mathsf{Clock},\ nf(\mathrm{msize}) = \mathsf{MSize},\ nf(\mathrm{hsize}) = \mathsf{HSize}.$$

Note that we have $H_N(\mathsf{CPU}|\mathrm{2cpu\_dtpc}) = \{\mathrm{cpu}, \mathrm{cpu2}\}$, and $H_N(\mathsf{Clock}|\mathrm{cpu}) = H_N(\mathsf{Clock}|\mathrm{cpu2}) = \{\mathrm{clock}\}$. Thus, for example, 2CPU-DTPC class has two objects of the CPU class, called "cpu" and "cpu2", respectively, one object "memory" of the Memory class, and one object "hdd" of the HDD class.

Finally, let $\mathcal{O} = (O, \ni_O, \sqsupseteq_O)$ be an object structure with the following is-a and relationships:

Is-a relation:
$x \sqsupseteq_O x,\ \forall x \in O,$ and pc3 $\sqsupseteq_O$ pc1, pc3 $\sqsupseteq_O$ pc2.

Has-a relation:
pc$i$ $\ni_O$ ci, pc$i$ $\ni_O$ mi, pc$i$ $\ni_O$ hi, $i \in \{1,2,3\}$,
pc3 $\ni_O$ c4,
ci $\ni_O$ 2.4GHz, $i \in \{1,3,4\}$, c2 $\ni_O$ 3.0GHz,
m1 $\ni_O$ 512MB, mi $\ni_O$ 1GB, $i \in \{2,3\}$,
hi $\ni_O$ 40GB, $i \in \{1,2\}$, h3 $\ni_O$ 80GB.

Moreover, let $na : O \longrightarrow N$ be the following name assignment:

$$na(\mathrm{pc1}) = na(\mathrm{pc2}) = \mathrm{desk\_\ top\_\ pc},$$
$$na(\mathrm{pc3}) = \mathrm{2cpu\_dtpc},$$
$$na(\mathrm{c1}) = na(\mathrm{c2}) = na(\mathrm{c3}) = \mathrm{cpu},\ \ na(\mathrm{c4}) = \mathrm{cpu2},$$
$$na(\mathrm{m1}) = na(\mathrm{m2}) = na(\mathrm{m3}) = \mathrm{memory},$$
$$na(\mathrm{h1}) = na(\mathrm{h2}) = na(\mathrm{h3}) = \mathrm{hdd},$$
$$na(2.4\mathrm{GHz}) = na(3.0\mathrm{GHz}) = \mathrm{clock},$$
$$na(512\mathrm{MB}) = na(1\mathrm{GB}) = \mathrm{msize},$$
$$na(40\mathrm{GB}) = na(80\mathrm{GB}) = \mathrm{hsize}.$$

We define the class identifier $id_C : O \longrightarrow C$ by $id_C = nf \circ na$.

This object structure $\mathcal{O}$ illustrates the following situation: There are three objects pc1, pc2 and pc3. pc1 and pc2 are instances of the DeskTopPC class, and pc3 is an instance of the the 2CPU-DTPC class. pc1 and pc2 have one instance of the CPU class, c1 = pc1.cpu and c2 = pc2.cpu, respectively. On the other hand, pc3 has two instances of the CPU class, c3 = pc3.cpu and c4 = pc3.cpu2, respectively. Moreover, each pc$i$ ($i = 1,2,3$) has just one instance mi of the Memory class, and just one instance hi of the HDD class. Each cpu has its clock (2.4GHz or 3.0GHz), each memory has its size (512MB or 1GB), and each hard disk drive has its size (40GB or 80GB).

C. Indiscernibility Relations in the Object – Oriented Rough Sets

All equivalence relations in object–oriented rough set models are based on the concept of equivalence as instances. In [2], to evaluate equivalence of instances, an equivalence relation $\sim$ on O are recursively defined as follows:

$$\begin{array}{l} x \sim y \iff \\ \quad x \text{ and } y \text{ satisfy the following two conditions:} \\ \quad 1.\ id_C(x) = id_C(y), \text{ and,} \\ \quad 2.\ \left\{ \begin{array}{ll} x.n \sim y.n,\ \forall n \in H_N(na(x)) & \text{if } H_N(na(x)) \neq \emptyset, \\ Val(x) = Val(y) & \text{otherwise,} \end{array} \right. \end{array} \tag{4}$$

where $H_N(na(x))$ is the set of names that $na(x)$ has. $Val(x)$ is the "value" of the "value object" x. Because C is a finite non-empty set and $\ni_C$ is acyclic, there is at least one class c such that c has no other class c', that is, $c \not\ni_C c'$ for any $c' \in C$. We call such class c an *attribute*, and denote the set of attributes by AT. For any object x, if $id_C(x) = a$ and $a \in AT$, we call such object x a *value object* of the attribute a. The value object x as an instance of the attribute a represents a "value" of the attribute.

$x \sim y$ means that the object x is equivalent to the object y as an instance of the class $id_C(x)$. Using the equivalence relation $\sim$, we propose the following binary relation with respect to a given subset $B \subseteq N$ of names as follows:

$$\begin{array}{l} x \sim_B y \iff \\ \quad x \text{ and } y \text{ satisfy the following two conditions:} \\ \quad 1\ \ B \cap H_N(na(x)) = B \cap H_N(na(y)), \text{ and,} \\ \quad 2\ \forall n[n \in B \cap H_N(na(x)) \Rightarrow x.n \sim y.n]. \end{array} \tag{5}$$

$x \sim_B y$ means that x and y are equivalent as instances of the class $id_C(x)$ in the sense that, for all $n \in B \cap H_N(na(x))$, x and y have equivalent instances of the class $id_C(x.n)$. Equivalence classes $[x]_{\sim_B}$ by $\sim_B$ are usually defined. Note that, in the "traditional" rough set theory, all equivalence classes concern the same attributes. On the other hand, each equivalence class of the object–oriented rough set model may concern different classes. In particular, if $B \cap H_N(na(x)) = \emptyset$, the equivalence class $[x]_{\sim_B}$ is the set of objects that are not concerned any class $nf(n), n \in B$ at all.

Example 2: This example is continuation of example 1. Suppose $B = \{\mathrm{cpu}\}$. Using the equivalence relation $\sim$ defined by (4), we construct the equivalence relation $\sim_B$ by (5), and the resulted equivalence classes by $\sim_B$ are as follows:

$$[\mathrm{pc1}]_{\sim_B} = \{\mathrm{pc1,pc3}\},\ [\mathrm{pc2}]_{\sim_B} = \{\mathrm{pc2}\},$$
$$[\mathrm{c1}]_{\sim_B} = O - \{\mathrm{pc1}, \mathrm{pc2}, \mathrm{pc3}\}.$$

The equivalence classes $[\mathrm{pc1}]_{\sim_B}$ and $[\mathrm{pc3}]_{\sim_B}$ correspond to the set of personal computers that have "2.4GHz CPU" and the singleton set of the personal computer that has "3.0GHz CPU", respectively. On the other hand, $[\mathrm{c1}]_{\sim}$ represents the set of objects that have no CPU.

III. Incomplete Information in the Object–Oriented Rough Sets

In this section, we extend the object–oriented rough set model to treat incomplete information. There are many kinds of interpretation of incomplete information, however, we concentrate incompleteness of information about objects that comes from "absence of parts". Informally, absence of parts illustrates the following situation: Suppose we have two classes

c_i and c_j such that c_j is a part of c_i, and an instance o_i of the class c_i, however, there is no "actual" instance $o_j \in O$ such that o_j is an instance of c_j and also is a part of o_i. For example, "a personal computer that its CPU was taken away". has no instance of CPU class, even though any instance of DeskTopPC class should have one instance of CPU class. To illustrate incompleteness we mentioned the above, we introduce *null value objects* into the object–oriented rough set model.

A. Null Value Objects

We introduce null value objects and an incomplete object structure to illustrate "absence of parts" as follows.

Definition 1: Let NO be a finite non-empty set. An *incomplete object structure* $\mathcal{IO}$ is the following triple:

$$\mathcal{IO} = (O \cup NO, \ni_I, \sqsupseteq_I), \tag{6}$$

where O is the (finite and non-empty) set of objects, NO is a finite set such that $O \cap NO = \emptyset$, the relation $\ni_I$ is an acyclic binary relation on $O \cup NO$, and the relation $\sqsupseteq_I$ is a reflexive, transitive, and asymmetric binary relation on $O \cup NO$. Moreover, $\ni_I$ and $\sqsupseteq_I$ satisfy the property (1) and the following condition:

$$\forall x \in NO, \forall y \in O \cup NO, x \not\ni_I y. \tag{7}$$

We call each object $x \in NO$ a *null value object*. On the other hand, each object $y \in O$ is called an *actual objects*. We intend that null value objects have the following characteristics:

1) All null value objects have no objects.
2) Each null value object is an instance of some class.

The characteristic 1) means that each null value object is a special case of value objects, and it corresponds to "null value". The characteristic 2) means that each null value object is also an object of some class. This intends that we can compare null value objects and any other (null value) objects if and only if these objects are instances of the same class.

B. Well-defined structures with null value objects

To illustrate the above characteristics of null value objects, we refine the definition of the class identifier. However, we can not directly extend the domain of the class identifier id_C to $O \cup NO$ with keeping id_C a p-morphism, and therefore we need to weaken the definition of p-morphism.

Definition 2: Let $\mathcal{IO} = (O \cup NO, \ni_I, \sqsupseteq_I)$ and $\mathcal{C} = (C, \ni_C, \sqsupseteq_C)$ be an incomplete object structure and a class structure, respectively. We call a function $id_C : O \cup NO \longrightarrow C$ a *class identifier of incomplete objects* if id_C satisfies the following conditions:

1. $\forall o_i, o_j \in O \cup NO, o_i \ni_I o_j \Rightarrow id_C(o_i) \ni_C id_C(o_j)$.
2. $\forall o_i \in O,\ \forall c_j \in C,\ id_C(o_i) \ni_C c_j \Rightarrow \exists o_j \in O \cup NO$ s.t. $o_i \ni_I o_j$ and $id_C(o_j) = c_j$,

and the same conditions are also satisfied for $\sqsupseteq_I$ and $\sqsupseteq_C$.

$id_C(o) = c$ means that the (null value) object $o \in O \cup NO$ is an instance of the class c. Note that the condition 2 is weakened from the condition of p-morphism to agree with the characteristic of null value objects by (7).

We also need to extend the domain of the name assignment na to $O \cup NO$ as follows.

Definition 3: Let $\mathcal{IO} = (O \cup NO, \ni_I, \sqsupseteq_I)$ and $\mathcal{N} = (N, \ni_N, \sqsupseteq_N)$ be an incomplete object structure and a name structure, respectively. We call a function $na : O \cup NO \longrightarrow N$ a *name assignment* if na satisfies the following conditions:

1) na satisfies the condition 1 and 2 appeared in Definition 2.
2) na satisfies the following *uniqueness condition*:
   - For any $x \in O$, if $H_I(x) \neq \emptyset$, the restriction of na into $H_I(x)$: $na|_{H_I(x)} : H_I(x) \longrightarrow N$ is injective,

   where $H_I(x) = \{y \in O \cup NO \mid x \ni_I y\}$ is the set of "actual" and "null value" objects that x has.

$na(x) = n$ means that the name of the object x is n.

Similar to the case of "complete" object–oriented rough set model, we introduce a naming function $nf : N \longrightarrow C$ as a p-morphism between $\mathcal{N}$ and $\mathcal{C}$ that satisfies the name preservation constraint. Moreover, we say that $\mathcal{C}$, $\mathcal{N}$ and $\mathcal{IO}$ are *well-defined* if and only if there exist a naming function $nf : N \longrightarrow C$ and a name assignment $na : O \cup NO \longrightarrow N$ such that $id_C = nf \circ na$, that is, $id_C(x) = nf(na(x))$ for all $x \in O \cup NO$. Hereafter, we concentrate well-defined incomplete object, name and class structures.

Now, we can explain incompleteness by "absence of parts" correctly. Suppose that any instance x of a class c_i should have m objects of a class c_j, that is, there are m names $n_1, \cdots, n_m$ for the class c_j and m instances $x.n_1, \cdots, x.n_m$ of the class c_j such that $x \ni_O x.n_j$ $(j = 1, \ldots, m)$. Here, if we have $x.n_k \in NO$ for some name n_k, the notion $x \ni_O x.n_k$ illustrates that, even though any instance of c should have m "actual" objects of c_j, there are just $m-1$ objects of c_j as parts of x, and there is no "actual" object that corresponds to $x.n_k$. This situation illustrates "incompleteness" of the object x as an instance of c_i, which is triggered by "absence of parts" of x. Note that there are exactly m instances of c_j as parts of x, and therefore, constraints about design of objects introduced by the name structure are satisfied.

Example 3: Let $\mathcal{O}$, $\mathcal{N}$ and $\mathcal{C}$ be the object structure, the name structure, and the class structure used in example 1, respectively. Moreover, let $id_C : O \longrightarrow C$, $na : O \longrightarrow N$ and $nf : N \longrightarrow C$ be the class identifier, the name assignment and the naming function used in example 1, respectively. We introduce an incomplete object structure $\mathcal{IO} = (O \cup NO, \ni_I, \sqsupseteq_I)$ based on $\mathcal{O}$ as follows:

Let $NO = \{\mathrm{nc1}, \mathrm{nc2}, \mathrm{nc3}\}$ be a set of null value objects with the following id_C and na:

$$id_C(\mathrm{nc1}) = id_C(\mathrm{nc2}) = id_C(\mathrm{nc3}) = \mathsf{CPU},$$
$$na(\mathrm{nc1}) = na(\mathrm{nc2}) = \mathrm{cpu},\ \ na(\mathrm{nc3}) = \mathrm{cpu2}.$$

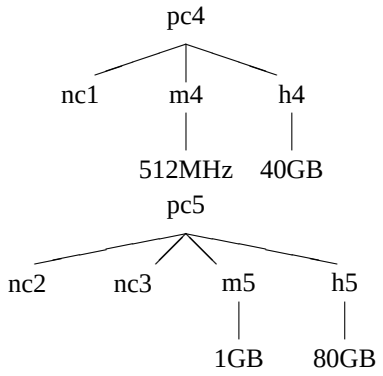

Fig. 1. Has-a relation $\ni_I$ between actual and null value objects

We also add objects with the following names to O:

$$na(\text{pc4}) = \text{desk\_top\_pc}, \quad na(\text{pc5}) = \text{2cpu\_dtpc},$$
$$na(\text{m4}) = na(\text{m5}) = \text{memory},$$
$$na(\text{h4}) = na(\text{h5}) = \text{hdd},$$
$$\cdots .$$

We define the is-a relation $\sqsupseteq_I$ on $O \cup NO$ as follows. Fig. 1 illustrates the has-a relation $\ni_I$ about newly added actual objects and null value objects:

$$x \sqsupseteq_I x, \ \forall x \in O \cup NO, \text{ and,}$$
$$\text{pc}i \sqsupseteq_I \text{pc}j, \ (i, j \in \{1, 2, 4\}),$$
$$\text{pc}i \sqsupseteq_I \text{pc}j, \ (i, j \in \{3, 5\}),$$
$$\text{pc3} \sqsupseteq_I \text{pc}i, \ \text{pc5} \sqsupseteq_I \text{pc}i. \ (i \in \{1, 2, 4\}).$$

The incomplete object structure $\mathcal{IO}$ illustrates the following situations about newly added actual objects and null value objects: There are two personal computers pc4 and pc5 with some lack of parts, respectively. pc4 is an instance of the DestTopPC class, thus pc4 should have one CPU as a part. Similarly, pc5 is an instance of the 2CPU-DTPC class, thus pc5 should have two CPUs as parts. However, both pc4 and pc5 have no CPU as its parts.

IV. Tolerance Relations in Object–Oriented Rough Sets

In this section, we extend the tolerance relation proposed by Kryszkiewicz [1] to apply object–oriented incomplete information systems. According to Stefanowski and Tsoukiàs [7], the tolerance relation corresponds to "missing value" semantics that unknown values allow any comparison. Thus, we think that extended tolerance relations in object–oriented incomplete information systems are suitable for treating incompleteness by "absence of parts".

Definition 4: Let $B \subseteq N$ be a non-empty subset of names. A *object–oriented tolerance relation* T_B on $O \cup NO$ is a binary relation defined as follows:

$$xT_By \iff$$
$$x \text{ and } y \text{ satisfy the following two conditions:}$$
$$1 \ \ B \cap H_N(na(x)) = B \cap H_N(na(y)), \text{ and,} \quad (8)$$
$$2. \ \forall n[n \in B \cap H_N(na(x)) \Rightarrow \text{ either } x.n \sim y.n$$
$$\text{or } x.n \in NO \text{ or } y.n \in NO],$$

where $\sim$ is an equivalence relation on O defined by (4).

We call the relation T_B defined by (8) an *object–oriented tolerance relation*. Similar to the tolerance relation [1], it is easy to check that the relation T_B defined by (8) is reflexive and symmetric, but not necessary transitive.

We intend that the tolerance relation T_B defined by (8) treats "absence of parts" in the sense that, if $x.n$ is a null value object of the class $id_C(x.n)$, $x.n$ is regarded to be equivalent to any instance of the class $id_C(x.n)$. This means that, if we have xT_By, we can make x and y be equivalent, that is, $x \sim_B y$, by removing all null value objects from both x and y, and setting "actual" parts to make $x.n \sim y.n$ for all $n \in B \cap H_N(na(x))$.

For any subset $X \subseteq O$ of "actual" objects, we define the B-lower approximation $\underline{T_B}(X)$ and B-upper approximation $\overline{T_B}(X)$ as the same manner with [1], respectively:

$$\underline{T_B}(X) = \{x \in O \mid T_B(x) \subseteq X\}, \quad (9)$$
$$\overline{T_B}(X) = \{x \in O \mid T_B(x) \cap X \neq \emptyset\}, \quad (10)$$

where $T_B(X) = \{y \in O \mid xT_By\}$. We call the set $T_B(x)$ the tolerance class of x. B-lower approximation $\underline{T_B}(X)$ is the set of objects $y \in O$ such that we can make y be equivalent to all objects $x \in X$. On the other hand, B-upper approximation $\overline{T_B}(X)$ is the set of objects y such that there is at least one object $x \in X$ such that we can make x and y be equivalent. Similar to [1], it is not hard to check that B-lower approximation $\underline{T_B}(X)$ and B-upper approximation $\overline{T_B}(X)$ satisfy the following properties.

Proposition 1: For any non-empty $X \subseteq O$ and any non-empty $B, B_1, B_2 \subseteq N$, the following properties are satisfied:

1) $\underline{T_B}(X) \subseteq X \subseteq \overline{T_B}(X)$.
2) $B_1 \subseteq B_2 \Longrightarrow \underline{T_{B_1}}(X) \subseteq \underline{T_{B_2}}(X)$.
3) $B_1 \subseteq B_2 \Longrightarrow \overline{T_{B_1}}(X) \supseteq \overline{T_{B_2}}(X)$.

Example 4: Let $B_1 = \{\text{cpu}, \text{cpu2}\}$ be a subset of names, and $X = \{\text{pc2}, \text{pc3}\}$. We construct the object–oriented tolerance relation T_{B_1} by (8), and the obtained tolerance classes are as follows:

$$T_{B_1}(\text{pc1}) = \{\text{pc1}, \text{pc4}\},$$
$$T_{B_1}(\text{pc2}) = \{\text{pc2}, \text{pc4}\},$$
$$T_{B_1}(\text{pc3}) = \{\text{pc3}, \text{pc5}\},$$
$$T_{B_1}(\text{pc4}) = \{\text{pc1}, \text{pc2}, \text{pc4}\},$$
$$T_{B_1}(\text{pc5}) = \{\text{pc3}, \text{pc5}\},$$
$$T_{B_1}(\text{c1}) = O - \{\text{pc}i \mid 1 \leq i \leq 5\}.$$

Therefore, we have the following B_1-lower and B_1-upper approximations of X:

$$\underline{T_{B_1}}(X) = \emptyset,$$
$$\overline{T_{B_1}}(X) = \{\text{pc2}, \text{pc3}, \text{pc4}, \text{pc5}\}.$$

On the other hand, if we set $B_2 \stackrel{\text{def}}{=} B \cup \{\text{memory}\}$, we have the following tolerance classes by the tolerance relation T_{B_2} as follows:

$$T_{B_2}(\text{pc1}) = \{\text{pc1}, \text{pc4}\},$$
$$T_{B_2}(\text{pc2}) = \{\text{pc2}\},$$

$$
\begin{aligned}
T_{B_2}(\text{pc3}) &= \{\text{pc3}, \text{pc5}\},\\
T_{B_2}(\text{pc4}) &= \{\text{pc1}, \text{pc2}, \text{pc4}\},\\
T_{B_2}(\text{pc5}) &= \{\text{pc3}, \text{pc5}\},\\
T_{B_2}(\text{c1}) &= O - \{\text{pc}i \mid 1 \leq i \leq 5\}.
\end{aligned}
$$

By these tolerance classes, we have the following B_2-lower and B_2-upper approximations of X:

$$
\begin{aligned}
\underline{T_{B_2}}(X) &= \{\text{pc2}\},\\
\overline{T_{B_2}}(X) &= \{\text{pc2}, \text{pc3}, \text{pc4}, \text{pc5}\}.
\end{aligned}
$$

This illustrates an example of the situation of the property 2) and 3) in Proposition 1.

V. Discussion

We have treated object–oriented incomplete information systems and tolerance relations in object–oriented rough sets, which provide an extension of object–oriented rough sets to treat incomplete information by "absence of parts".

The main idea of this paper is to introduce null value objects that enable us to treat incompleteness of objects and constraints about design of objects simultaneously. Moreover, introduction of null value objects provides flexibility of representation in a sense that name structures also illustrate "possibility" of the numbers of actual parts. As mentioned in section III-B, constraints about design of objects are satisfied in object–oriented incomplete information systems. Thus, if the name structure describes that any instance x of a class c_i has exactly m instances of a class c_j as parts of x, there are exactly m instances of c_j within k actual objects and l null value objects as parts of x, where $k + l = m$ and $0 \leq k, l \leq m$. Therefore, we can regard that the name structure also determines the maximum number of "actual" parts that an object *can* have, instead of determining the number of objects that an object *should* have in object–oriented "complete" information systems

We think that the object–oriented tolerance relation captures characteristics of incompleteness by "absence of parts". This is because, as mentioned in section IV, the "original" tolerance relation provides "missing value" semantics that unknown values allow any comparison. We consider that, in object–oriented incomplete information systems, missing value semantics illustrates the situation that unknown values of some class c allow any comparison with instances of c, which captures an important characteristic of "absence of parts" as an interpretation of null values. Consideration of other interpretations of null values in object–oriented incomplete information systems are future works.

VI. Conclusion

In this paper, we have proposed null value objects to represent incomplete information that comes from "absence of parts" in the object–oriented rough set model. Moreover, similar to [1], we have introduced the object–oriented tolerance relation that arrows incomplete information illustrates by null value objects.

More refinement of theoretical aspects of null value objects and the object–oriented tolerance relation, and extension to treat other kinds of incompleteness, for example, similarity relations proposed by Stefanowski and Tsoukiàs [7], are future works.

References

[1] M.Kryszkiewicz: Rough Set Approach to Incomplete Information Systems, *Information Science*, Vol. 112, pp. 39–49, 1998.

[2] Y. Kudo and T. Murai: A Theoretical Formulation of Object–Oriented Rough Set Models, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 10, No. 5, 2006 (to appear).

[3] Z. Pawlak: Rough Sets, *International Journal of Computer and Information Science*, Vol. 11, pp. 341–356, 1982.

[4] Z. Pawlak: *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publisher, 1991.

[5] S. Popkorn: *First Steps in Modal Logic*, Cambridge University Press, 1994.

[6] R. Słowiński and J. Stefanowski: Rough Classification in Incomplete Information Systems. *Mathematical Computing Modeling*, Vol 12, No. 10–11, pp. 1347–1357, 1989.

[7] J. Stefanowski and A. Tsoukiàs: Incomplete Information Tables and Rough Classification, *Computational Intelligence*, Vol. 17, No. 3, pp. 545–565, 2001.