Translation from Transactions of IPSJ

Representation of Descriptive Name and the Resolution Method with a Semantic Network Structure

FUMIKO KOUDA* and HIDEHIKO TANAKA*

We propose a new name-resolution method suitable for descriptive names, for use in novel name-management systems.

In conventional systems, some naming scheme uses attributes on administration as an identifier of an object. However, there is no naming system where proper attributes of each object are used as identifiers. This kind of naming scheme is also important to users. Therefore, we adopt the definition of a descriptive name from IS7498-3, and consider such a naming scheme as indicating proper attributes of a computer resource.

Our purpose is to review how to represent proper attributes in descriptive names and to construct a new resolution mechanism suitable for them. We have paid attention to denotation classes of attributes, degree of generality in an attribute sort, relation among attributes within an object or between objects and ellipsis or order of attributes in a description. We have analyzed their interrelations and clarified a rule among attributes: they could be ordered partially based on a degree of generality level in each attribute sort.

Considering the resemblance between the partial ordering among attributes and a semantic network, called KL-ONE, we constructed a name resolution algorithm for the descriptions of proper attributes by using the structure of KL-ONE. In this algorithm, we have introduced some new concepts, for treating various kinds of attributes.

This enables us to treat descriptive names of proper attributes with various forms as a key element of computer systems.

1. Introduction

We propose a new name-resolution method for descriptive names which denote degree of meanings and ambiguity of descriptions, and also work as identifiers. We do this by discussing their representational forms, analyzing their properties and by proving theorems. Our proposed resolution method is quite different from ordinary ones because of considering the meanings and ambiguity of names in it.

Conventional naming systems use administration-dependent names for computer resources. These names are reflected on the logical or physical structures and have hierarchy forms. For example, Clearinghouse [9] has a three-layered structure which is composed of a sequence of organization names, location and identifier. Grapevine [1] has a two layered structure. Hierarchy names have a fixed order of their elements and every element must be included. Moreover, we have to know their element values correctly.

Sollins [13] showed categories for names from the consideration of names in ordinary human life and proposed a name management system for distributed processings. Attention is paid to the five aspects for the comparison criteria; they are "assignment", "resolution", "usage", "ambiguity or uniqueness" and "degree of meanings."

According to the criteria which she showed, a conventional distinct name is assigned concretely, and it cannot be abbreviated. So, it is easy to guarantee uniqueness. Resolution of the name is comparatively easy because it uses tree-structured directories. However, there is a restriction when using such names; if we do not know it correctly, we cannot access the resource which is denoted by the name.

This problem occurs because of the fact that naming is regarded as a tool only for administration and no consideration is paid to users' standpoint. Another need is necessary to distinguish objects. A solution to this problem is to consider both ambiguity and degree of meanings in the notion of names. If we could use some description of an object as an identifier, then we would refer to that object even when we did not know the distinct name correctly.

This is a translation of the paper that appeared originally in Japanese in Transaction of IPSJ, Vol. 31, No. 9 (1990), pp. 1397–1409.

*Department of Electrical Engineering, the Faculty of Engineering, University of Tokyo, 3-1, Hongo 7-chome, Bunkyo-ku, Tokyo, 113 Japan.

A descriptive name is known as a name which distinguishes objects by describing their functional properties. Since description is not restricted to a fixed form, descriptive names are able to show various levels of descriptions, such as levels of ambiguity or meanings. Therefore, it is important to be able to treat descriptive names in naming schemes. As the basis of our consideration, we adopt the definition of a descriptive name from IS 7498 part 3 (Naming and Addressing), and continue our discussion.

In the next section, the requirements of descriptive names and the policy of name resolution for descriptive names are described. Section 3 discusses information bases for descriptive names. Section 4 analyzes name resolution for descriptive names theoretically and introduces a new algorithm for it. Examination and discussion are in section 5. Section 6 concludes this paper.

2. Representational Requirements and the Management of Descriptive Names

2.1 Consideration for Representation of Descriptive Names

The definition of IS 7498 says, 'A name that identifies a set of one or more objects by means of a set of assertions concerning the properties of the objects of the set.' According to the definition of IS 7498-3, a descriptive name is a name which identifies objects by describing their properties or attributes. This definition defines no concrete representation nor the scope of properties of objects. So, we will consider the representation of descriptive names under the following conditions within the area of computer resources.

From the standpoint of using computer resources, the following information is important because it explains attributes of resources: 1) the functionality or roles of the resources and 2) feasibility of accessing resources, that is, to be able to use it permanently or temporarily.

An example of an identifier using attributes is X.500 of CCITT [8]. It treats attributes necessary for managing resources, such as those of a nation or an organization. The property of an object itself is out of the consideration in X.500. On the other hand, our proposal treats properties which an object possesses.

There are many ways to represent functionality of computer resources. Let us consider how they are described.

In the description of processing functionality, there are cases that describe attributes in general and without mentioning details, or the cases which describe specifications according to the system structure. The former description can be applied widely even in a heterogeneous environment and can be used in common, but the latter cannot. From this fact, it is necessary to consider in the descriptions different degrees of generality. The notion to indicate a class is called **attribute type**, and each value in the class and each specification of a generality level is called an **attribute value**. Let us consider a set of files, as an example. If we need to look at the active functional file, we will write 'filetype is active.' Otherwise if we need a set of file commands, then we will write 'filetype is a command.' Furthermore if we want to know only a set of built-in commands, then we will write 'filetype is a built-in command.' In these cases, the attribute type is *filetype*, and attribute values are *active*, *command* and *built-in command*, respectively. Attribute values show different levels of generality.

Candidates of attribute types for users or files are *access right, version, code, functionality*, etc. Their attribute values are defined according to their generality degrees. Profile [11] calls an attribute type a **tag** and uses *name, address, phone, mail, login, home* and so on.

Consideration of temporality is also necessary. Some properties of an object always hold and some do not.

In case of having dependencies among properties, they are to be included in the description of properties. The relations are observed as both intrarelation among attributes for an object and interrelation among attributes for several objects. An example of an interrelation is that of a version of a compiler, and feasible files, when we perform compiling. Otherwise, if two properties have no dependency to each other, they hold separately in an object. We call this an "and" relation, in the meaning that both properties hold in the same object.

On the other hand, it is important to offer some way to represent flexibility of description, namely, to select necessary attributes for users or to leave out some constituting elements which are not needed by them. This is because it is necessary to provide users with a way to describe the objects which they really need to access or refer.

2.2 A Descriptive Name and its Indicated Objects

For a descriptive name to work as an identifier, it must be clarified what objects are indicated in the representation of the descriptive name. When we use a descriptive name, we think principally of its meanings and we don't necessarily aim at unique identification. Therefore we will take a position that if a descriptive name refers to several objects, then we adopt all these objects as the resolution.

There is flexibility of representation of descriptive names. Hence, we focus on the problems how the indicated objects change according to the generality levels of description, or under what conditions the different descriptions, such as ellipsis or different order of descriptive elements, denote the same objects.

2.3 Forms for Descriptive Names

The expressional forms of attribute description are usually presented as a relation between a type and its value, and represented by {attribute type, operator, attribute value}. For example, Profile uses the combination of the form 'tag=value'. X.500 has the form, such as {C=GB, L=Winslow, O=Graphic services, CN=Laser Printer}. Yellow-pages service by L. L. Peterson [10] has the form of '⟨type, operator, value⟩.' The operator is used not only '=', but also '<' and '>', as for example 'time>3 msec.' Such expressions are also possible in X.500 by using the filter parameters.

Our concern is to investigate how to describe the proper attributes for computer objects and how to resolve them elegantly.

Here, we mainly consider basic forms of descriptive names. We define these forms as a set of attribute tuples of type and value with equality, denoting by {attribute type=attribute value}, and having the description of the relation between them. If the attribute tuples have no relation among themselves, that is, they have an "and" relation, then they are put arbitrary and sequentially. The order or ellipsis of elements are arbitrarily selected. If attribute tuples have a relation of dependency or inclusion to each other, then their relation is described explicitly, using some functions.

The extended forms of descriptive names have predicate expressions in addition to the basic forms. These can describe contingent expressions. These forms are beyond the scope of this paper.

An example of a descriptive name is as follows:
{file_type=command, functional_description=print, input=data_file, output=print_device, class=user, capability=access_mode, access_check (class, file), function (description, input, output), etc.}

One of them is described such as:
{file_type=built_in_command, functional_description=lzrprint, access_check (class:subject, file:object), function (description:print, input:data_file, output:laprl), etc.}

2.4 Determination of Resolving Method for Descriptive Names

A name-resolution method is a mechanism which makes a link to an object from a given name. It uses a management table, which is in generally called a *directory* and which works as an information base. We think that constructing a directory and a name-resolution method influence each other and it is expected that a name-resolution depends on strategies how to make the directory table.

From the consideration of requirements of descriptive names, we have found out that the following conditions are necessary as a directory for resolving descriptive names: the directory must treat both various levels of generalization degree of attributes, and relations among descriptive elements; it must recover the elements which are omitted; and it must be independent of descriptive sequence.

Some of the name-resolution methods are referred to briefly in the following paragraphs, before determining the design.

A name resolution method for distinguished (distinct) names usually uses conversion mappings, context mappings or tree-structured directoires. Any of them maps from a distinguished name to another one. If necessary, intermediate names are used for the conversion, and this finally reaches an object. Objects are found by tracing conversion of names.

For example, **i-node** in the UNIX operating system has a mapping table which is composed of a tuple {address, identifier}. The context mapping proposed by R. W. Watson [16] uses levels of identifiers. A level of an identifier is mapped to the next by using **context** and **mapping functions**. The last level indicates the desired object.

Comer [3] showed a name-resolution mechanism on the basis of his model of distributed processing which is composed of **environments** and **links**. Names have several levels. In the name-resolution, not only names but also their related environments change. Name conversion uses the **qualified name** which comprises a name and context. The context shows a mapping between names. A primitive name is reduced by using **naming networks** which indicates the relation between contexts.

Attribute-used naming requires interpreting attributes on the way of resolution so that it is insufficient to convert identifiers as described above. A facility of interpreting attributes is necessary.

For example, X.500 uses as a directory a tree-structured DIB (Directory Information Base) which is composed of a set of DIT (Directory Information Tree). DIT comprises a set of nodes, each of which consists of a tuple of attribute type and attribute values. DIB tree shows different categories in each node. Name resolution in X.500 starts from the root of DIB, and checking the category in each node, goes down the tree. Distinguished names used for X.500 reflect these tree structures.

Terry [14] treats structure-free name-management by separating the activities of choosing names, selecting storage sites and resolving object names. Names are resolved through a name resolution-tree. Its algorithm searches authority attributes, by checking **clustering conditions** within contexts. That is, resolving a name is a matter of binding names within contexts until the authoritative name servers for the named objects are discovered.

Profile treats attribute-based naming services. It uses a database as a mapping facility from attributes to **principals**, which are the basic objects for the architecture. The database and a suit of interpret functions are used to resolve the attribute-based names to principals. Debray [4] discusses a theoretical framework for reasoning about naming systems related to Profile.

These are resolutions for attribute-based naming. In the case of Terry attributes are regarded as character strings. Therefore it resembles the case of ordinary distinct name resolution. X.500 tree has a different category at each node of the tree. As the result of this structure shows, the resolution of a descriptive name is influenced by the order of elements. And since the relation between nodes has different meanings, there is some difficulty to determine the place to add new elements or modify these. Furthermore, there is no way to show the inheritance relation of attributes within a node. Using a database for the resolution in Profile is free from the order or ellipsis of elements in descriptive names. However, it does not show abstraction level of attributes clearly and the relations between descriptive elements are ambiguous. Therefore it needs functions to interpret them.

An inheritance relation is observed in a class of attribute descriptions each with the same type. It is constructed based on the abstraction levels of attributes. Since inheritance is important among the set of descriptive names, we consider that the resolution method should be able to deal with inheritance relation. The resolution method must have mapping facility as well. From this consideration, we choose a semantic network as the resolution method in order to resolve descriptive names. Our requirements for the semantic network are to have the same category in all nodes, to be able to represent inheritance between functions of nodes and relations between the functions, to have the facility of tree-structure, and to work as a database.

Semantic networks were first defined by Ross Quillian in 1968. A semantic network is a graph containing a set of nodes connected by links and it shows concepts, properties or relations among them, by corresponding a node to a concept, and a branch to a link. The higher one goes up the network, the more the concepts are generalized. A leaf node indicates an individual concept.

The graph may have the inheritance property among subsuming concepts. It may be a multiple inheritance.

A semantic network can be used as a classifier from its structure.

In the early days, semantic networks had disadvantages of representation of both exception and default. Even today, various approaches are tried for both of them, such as Shastri [12] and Touretzky [15]. Among them, KL-ONE [2] [5] has a well-formed conceptual structure with inheritance and is considered to have no exception nor default value in the structure.

KL-ONE is divided into two different formalisms: one for assertion and one for description. The descriptional part of KL-ONE comprises mainly concepts as nodes, and links among concepts, and provides subsumption and inheritance facilities. The components of a concept are its subsuming concepts (superConcepts) and its local internal structure expressed in *roles* and *structural descriptions*, which express the interrelations among the roles. A role acts like a type and it has its filler concept to express the role more precisely.

Generic concepts are distinguished from individual ones. In general, a generic concept is regarded as a class and an individual concept as an instance, in the sense of an object-oriented language. A local structure of a concept is inherited via subsuming links, which is called superClink, from its super concept (superConcept) to its subconcepts (subConcept). KL-ONE does not allow cancellation of inherited description. KL-ONE is based on the idea of *structured inheritance networks*. We think that the description part of KL-ONE has well depicted this structure. Therefore we shall call the part *SINet* in this paper.

The assertional part of KL-ONE deals with coreference, existence, etc. It uses the notion of *Nexus, Description Wire* and *Context* to express a description. This part can depict the notion, such that 'a Vulcan named Spock is the First Officer of the Enterprise.'

2.5 Correspondence of Descriptive Names to the Information Base(SINet)

The concepts in the description part of KL-ONE have a structure which represent the composition of roles, fillers and the relations among them. Since roles in this structure are regarded as showing types of the related fillers, roles can be mapped to attribute types, and their related fillers to attribute values. Then, a pair of attribute tuples from a given descriptive name can map to the role-filler pairs, and the relation between attribute types is represented by the relation between roles, such as structural description in SINet.

The correspondence above determines the place of the attributes in a descriptive name in SINet. Then, the concepts which satisfy the given descriptive name will be derived from a set of role-filler pairs corresponding to these attributes in SINet. Details will be discussed in section 4.

3. Structure of a Semantic Network and a Way to Use It

3.1 Structure of SINet in KL-ONE

In this section, we review briefly the facilities of the description part of KL-ONE and try to interpret its structure for the usage of our resolution method. We use the notation of the concept in attention, the superconcept of it, and the subconcept of it by *pcon, supcon* and *subcon*, respectively. And when we denote a set of subconcepts and superconcepts of the concept in attention, we begin with a capital letter as in *Subcon* and *Supcon*, respectively.

(1) In KL-ONE, as described in the previous section, a set of concepts is divided in two classes: **generic concepts** and **individual concepts.** There is no more detailed classification. We observe that the category of the generic concepts in KL-ONE mixes two different no-

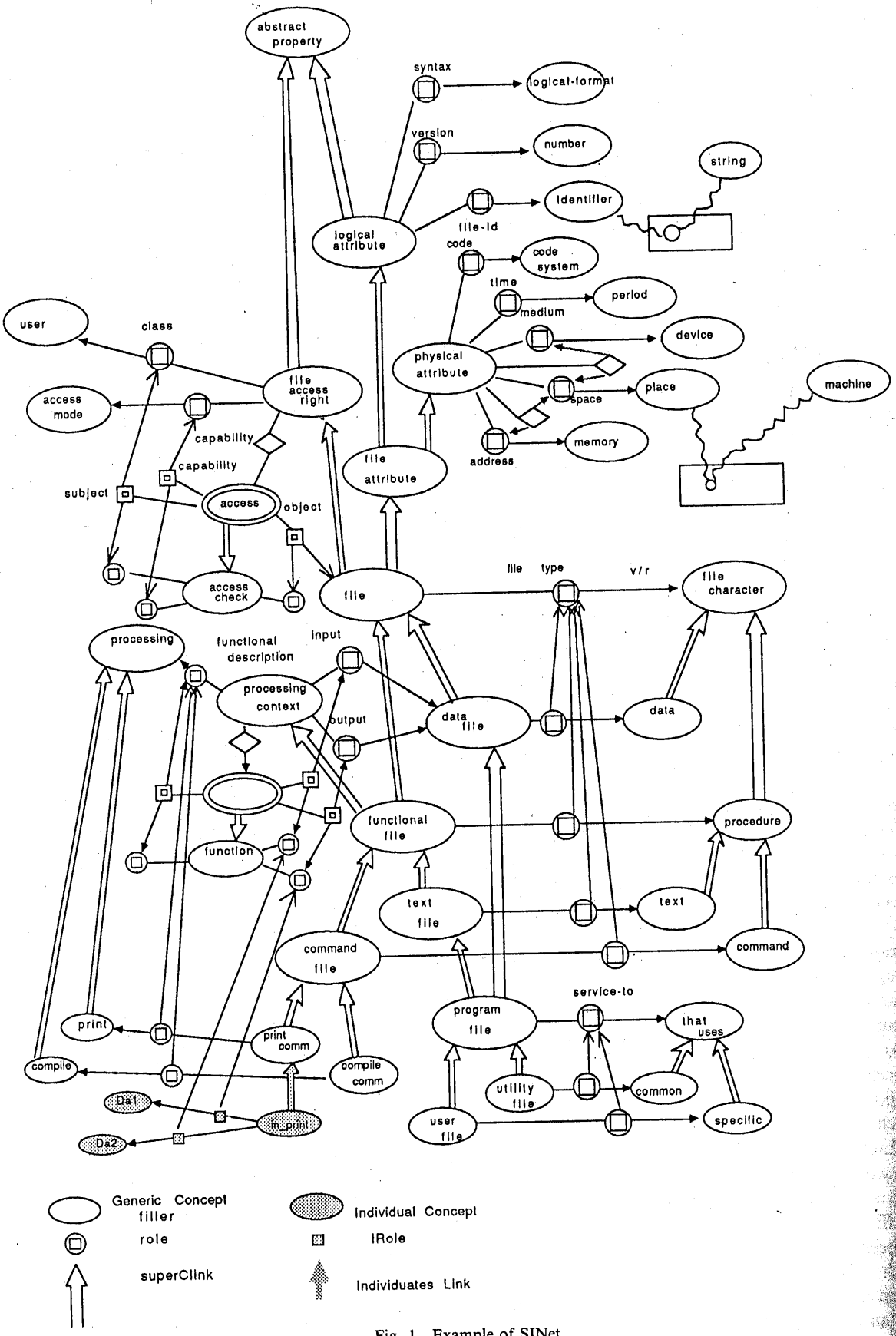

Fig. 1 Example of SINet.

tions of entity and of abstraction uniformly. Our purpose is to make up descriptive names for computer resources. To perform this, the concepts which indicate entities are important. Therefore we shall classify the generic concepts into two: entity concepts and abstract concepts. Hereafter, we shall mainly consider entity concepts. We call the links which connect entity concepts on some condition a **trunk** and focus the consideration on the trunks. We use a file-trunk as an example (see Fig. 1).

(2) An inclusion relation is shown between the concepts connected by a link. Two different relations are offered from the KL-ONE.

a) **restriction**: When a concept (*pcon*) connects to its several subconcepts, the filler-value corresponding to some role in *pcon* becomes more concrete in its each subconcept. This means that a restriction makes some property of the concept classified by describing its filler's value more specificly and disjointly in its each subconcept.

b) **conjunction**: When a concept or several concepts connect to a subconcept, the subconcept inherits all the role-fillers which all its superconcepts have.

(3) A role-filler pair of a concept shows an attribute which the entity corresponding to the concept has.

Let us consider a trunk which includes the concepts with a single role value. And let us restrict the consideration to the concepts on the trunk and a set of fillers related to the role on these concepts. If a concept (*pcon*) is placed at the upper part of the trunk in SINet, the value of the filler which *pcon* has via the role appears more abstract than the filler value for every subconcept of *pcon* which resides in the lower part of the trunk. Hence, we shall introduce a partial ordering for a role based on the ordering of 'abstractness' of filler values. Partial ordering is determined by the connective relations among concepts; if a concept is connected to its subconcepts by the links, then we define the partial ordering as the order that the role-filler pair for the former concept comes before the role-filler pairs for the latter subconcepts. We shall denote this ordering by $\leq$ or $<$ or $=$, and denote a role and a filler by r_i and f_{ijk}, respectively. In these notations, i shows sort of a role, j an ordering position, and k number of the sort in the position j. If attention is on the concept *pcon* and its role-filler pair is denoted by (r_i, f_{ijk}), then the role-filler pairs for the subconcepts *Subcon* are denoted by $(r_i, f_{ij+1\,k+m})$, $(m=1, \ldots, n)$, where n is the number of the subconcepts.

The properties of this partial ordering are observed as follows:

a) This partial ordering consists of a set of role-filler pairs, where there is a single role value and multiple filler values related to the role. The beginning of this ordering is the place where a role occurs newly. The last element is the pair on some individual concept. The ordering may have several last elements because of branches.

b) If the link between two concepts is a conjunction, then the corresponding fillers keep the same value. In this case the partial ordering is

$$(r_i, f_{ijk}) = (r_i, f_{ij+1\,k+1})$$

c) If the relation between the two concepts is a restriction and there are n branches there, then **case (i)** if the fillers of the r_i are explained more concretely, the partial ordering is

$$(r_i, f_{ijk}) < (r_i, f_{ij+1\,k+m})$$

where m is from 1 to n, or **case (ii)** if the role r_i is not related to the restriction, that is, the restriction comes from the other role in the concept, then the fillers for r_i remain unchanged. That is,

$$(r_i, f_{ijk}) = (r_i, f_{ij+1\,k+m})$$

holds, where m is from 1 to n. It means that each branch has the same role-filler pair.

d) In the case of multiple inheritance, different fillers with the same role may be inherited from the different superconcepts. Since multiple inheritance allows a subconcept to have these attributes, the subconcept may have these fillers for a role. That is, when two role-filler pairs are (r_i, f_{ijk}) and (r_i, f_{imn}) the role-filler pair for the subconcept is $(r_i, f_{ijk} \cup f_{imn})$.

We re-interpret the structure of SINet as follows: since each role has own partial ordering, that is a trunk, SINet is composed of different partial orderings. The complexity of this structure stems from the facts that each of the partial ordering begins with different point in SINet and each partial ordering has different branches. These facts produce a phenomenon that ordering structure for the partial ordering for a role may have different forms from linking connections among concepts.

(4) Individual concepts have relation to the assertion part of KL-ONE. It is possible to use the identifier of an individual concept as a distinguished name. From this fact, we consider that the identifiers of individual concepts relate to the naming in the ordinary name management facilities.

Note: We have not yet considered the identifiers for generic concepts. So, we regard the distinct name given to a generic concept as a **generic name** (defined in IS7498-3) for it.

3.2 Partitioned Expression of the SINet

In general, the role-filler structure for some concept always passed on to its subconcepts in SINet. Attributes for a concept are increased by a new role or modified by a restriction of a role in order to explain it more detail. In this way concepts in the lower part of SINet have more descriptions than those in the upper parts.

In this subsection we propose a way to diminish these repetitious descriptions by analyzing the connective relations between a concept and its subconcepts or between a concept and its superconcepts. This is not the main

part of our paper, though. But this structure is expected to make an information base for resolving descriptive names.

Our way is to divide a trunk of SINet into a set of components. Each component corresponds to a concept in the trunk of SINet, and has modified descriptions about roles and fillers on the concept and link relations between the concept and its super/sub concepts. Every component is expected to work as a unit of the information base for our name-resolution method. That is, in the case of a generic concept a unit of the concept in attention (*pcon*) consists of: 1) the links between its superconcepts or its subconcepts; if a link connects several superconcepts to *pcon*, this shows multiple inheritance; otherwise, if a link connects between a superconcept and *pcon* then this shows conjunction. 2) the description of modified property; if it is the case of restriction, then there is a set of descriptions on which role is restricted (we denote it **restrole.**), what are the concreted fillers for the restrole, and how these are reflected in the subconcepts; if the first element of the partial ordering of a role newly appears in *pcon*, then, the new role-filler pair is described. We denote it **newrole**.

On the other hand, in the case of an individual concept, there are no subconcepts of it. Let us consider a set of individual concepts derived from the same superconcept. These concepts have at least one role with the same sort. And the difference among them is the filler values related to the role. This shows a restriction. From the standpoint of an individual concept, each superconcept connected to it has the same structure. Hence, the unit consists of: 1) the links to its superconcepts and 2) (different sort (corresponding superconcepts) of) restricted filler descriptions and a pointer to the identifier which denote the individual object.

We shall call the unit for a generic concept a **generic conceptual functional unit** (abbreviated G-CFU), one for an individual concept an **individual conceptual functional unit**, I-CFU, and a conceptual functional unit in general simply CFU.

4. Resolution Method of Descriptive Names

4.1 Interpretation of the Relation between Descriptive Names and SINet

4.1.1 Role Filler Pair in SINet and Pivot Concepts

Let us consider the relation between the role-filler pairs in the partial ordering for role **i** and related concepts in SINet.

If a concept C in the trunk of role **i** has a role-filler pair (r_i, f_{ijk}), then this role-filler pair shows the most generalized notions among the $(r_i, f_{ij+mk+s})$ (where m and s are integers) pairs which occur equal to or after it in the partial ordering. From this viewpoint, the concept C having the role-filler pair (r_i, f_{ijk}) can be regarded as the representative of the concepts in the trunk which

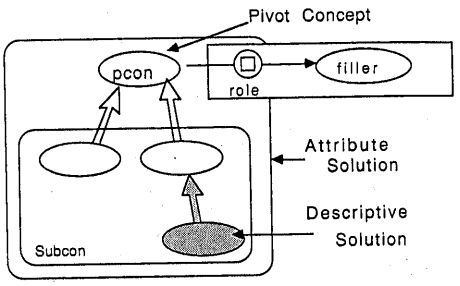

Fig. 2 Pivot Concept, Attribute Solution Set and Descriptive Solution Set.

have role-filler pairs occurring equal to or after $(r_i, f_{ijk}$ in the partial ordering for the role **i**. We shall call thi concept C a **pivot concept**. If the same role-filler pai continues in the several points in the partial ordering then several concepts in the trunk have the same role filler pair. In this case, the pivot concept is selected as concept where direct subconcepts of it are forked i several in the partial ordering. Thus, a pivot concept i always determined corresponding to each role-filler pa in the partial ordering.

Let the pivot concept having (r_i, f_{ijk}) be P. We can cor sider a set of (pivot) concepts which have the attribut tuple (r_i, f_{ijk}), or which have more specific ones tha (r_i, f_{ijk}) in the partial ordering. We shall call it an a **tribute solution set** and denote it by sol (P) (See Fig. 2

4.1.2 Partial Ordering and Pivot Concepts

From the definition in 4.1.1, a pivot concept is a co cept of which a role-filler pair is either at the beginnir of a partial ordering, at the place of occurring of branch by restriction, or in the last elements in the pa tial ordering.

Let us consider the branch case. Assume that its pa tial ordering is represented by role-filler pairs in t following:
$(r_i, f_{ijk}) < (r_i, f_{ij+1k+s})$, where $(1 \leq s \leq m)$ and m is branch number. The pivot concept for (r_i, f_{ijk}) is the la concept which has this attribute tuple from the defir tion. After it a branch occurs and the following co cepts have m different attribute tuples $(r_i, f_{ij+1k+}$ $(1 \leq s \leq m)$.

Since a concept can possess different role-filler pairs can be a common pivot concept for each of them. I dividual concepts in SINet are always pivot concep because their role-filler pairs are the last elements in t partial ordering.

A CFU in 3.2 is to be a pivot concept when it h either *newrole* or *restrole* or both. If there is no *newr* nor *restrole* in a CFU, then it cannot be a pivot concep

4.1.3 The Relation between Descriptive Names a the Set of Concepts in SINet

A tuple of an attribute type and value, as an eleme of a given descriptive name, is mapped to the pair o role-filler in SINet from 2.5. This mapping determine

pivot concept P for the attribute tuple in SINet and at the same time, so does the attribute solution set sol(P). Each pivot concept is determined in the above way. There remains the problem how to determine the pivot concept for all the attributes and their relations of a given descriptive name. As described in 2.3, our definition of a form of a descriptive name is composed of a set of attribute tuples and the relation among them. A sequence of attribute-tuples which are separated by commas shows an "and" relation. From these facts, the pivot concept for satisfying the given descriptive name will be derived from considering their relations of "and", inclusion and dependency. An attribute solution set is determined from a pivot concept. Therefore, the main issue of a resolution method of descriptive names is how to obtain pivot concepts.

If the pivot concept for the given descriptive name is determined, the last attribute solution set is obtained from it. Especially, since individual concepts are included in the last attribute solution set, their identifiers are regarded as showing objects in the sense of 3.1. Therefore we consider these individual concepts as resolved objects for a given descriptive name. So, we shall call this set a **descriptive solution set** and denote it by dsol (P), where P is the last derived pivot concept.

From the above, the process of name resolution for a given descriptive name is summarized as follows: to get attribute tuples from the given descriptive name, to map them to role-filler pairs in SINet, to determine the pivot concept for all the attribute tuples, and the corresponding attribute-solution set, to obtain the descriptive solution set from the attribute solution set and to take out the resolved objects.

4.1.4 Descriptive Names and Pivot Concepts

We investigate the relations among pivot concepts P_n ($n=1, \ldots, i, \ldots, j, \ldots, m$) in this subsection. We use the same notations during the following theorems.

Theorem 1 *The relation between the two pivot concepts P_i and P_j is one of the followings:*

a) *P_i and P_j are the same,*
b) *P_i and P_j are linked. They are super/sub concepts each other,*
c) *P_i and P_j have the same subconcept C_k, or*
d) *P_i and P_j have no common subconcepts.*

Proof

Let (r_i, f_{ikl}) and (r_j, f_{jmn}) be the role-filler pairs for the pivot concepts P_i and P_j, respectively. And let us denote the partial ordering for them by L_1 and L_2, respectively.

case a): If L_1 and L_2 are for the same role (that is, $r_i=r_j$), then P_i and P_j are in the same partial ordering of a trunk and they have ordering relation each other. This occurs when a given descriptive name has two different attribute values for an attribute type.

case b): If L_1 is a different partial ordering from L_2 and if they have an intersection at a concept C_1, then P_i and P_j have ordering relations by means of C_1. This case occurs when there is multiple inheritance for the role, or when the partial ordering of r_i and that of r_j are on the same link. We shall denote the ordering relation by $\ll$ or $==$, where $P_g \ll P_s$ denotes that generic concept P_g comes before specific one P_s. Then, the ordering relation between P_i and C_1 and between P_j and C_1 are represented as follows. Since P_i and P_j are symmetric, we may assume that P_i does not come after P_j, if there is an ordering relation between them.

Group A): $P_i==P_j \ll C_1$, or $P_i==P_j==C_1$, or $C_1 \ll P_i==P_j$
Group B): $P_i==C_1 \ll P_j$, or $P_i \ll C_1==P_j$, or $P_i \ll C_1 \ll P_j$, or $C_1 \ll P_i \ll P_j$, or $P_i \ll P_j \ll C_1$
Group C): $P_i \ll C_1$ and $P_j \ll C_1$ and there is no relation between P_i and P_j.
Group D): $C_1 \ll P_i$ and $C_1 \ll P_j$

When the focus of attention is restricted to P_i and P_j, group A) is $P_i==P_j$, group B) shows $P_i \ll P_j$, group C) indicates that there exists some concept which follows both P_i and P_j. P_i and P_j have no relation each other. Group D) is interpreted as having no common concepts between them.

case c): If L_1 and L_2 are different partial orderings and if there are no intersections between them, then there is no relation between P_i and P_j.

There is no other relation than the above between the partial ordering L_1 and L_2. This leads to the result that the relation between P_i and P_j is one of the cases a) through d). □

Theorem 2 *Let us consider the common subconcepts of P_i and P_j. Then the first pivot concept (denoted by P_k and called* **the latter pivot concept***) among them by the relation $\ll$ is as follows:*

a) *If P_i and P_j are the same concept, then the latter pivot concept is the same one. That is, $P_i==P_j==P_k$ holds.*
b) *If P_i is linked to P_j, then the latter pivot concept is P_j. That is, $P_i \ll P_j==P_k$ holds.*
c) *If there is no relation between P_i and P_j and if both are connected to the common subconcepts, then the latter pivot concept is the first concept P_k which satisfies the condition of pivot concepts and which is the subconcept of both P_i and P_j. Note that P_k is not necessarily the pivot concept from the given descriptive name, but it is regarded as a pivot concept of the descriptive name. We shall call it the* **derived pivot concept**.
d) *If P_i and P_j have no common subconcept, then the latter pivot concept cannot be determined.*

Proof

a), b) and d) directly follow from theorem 1.

Case c): If C_1 from theorem 1 satisfies the condition of the pivot concept, then $P_k==C_1$ holds. Else if C_1 is not a pivot concept, then because C_1 has neither *newrole* nor *restrole* there exists only a single concept for its subconcepts. Let it be C_2. If C_2 satisfies the condition of the pivot concept, then $P_k==C_2$ holds. Otherwise, repeat this comparison. Since individual concepts are pivot

concepts, this process certainly finishes. In this way, the first latter pivot concept is always determined. □

Theorem 3 *The intersection of two attribute solution sets for the two pivot concepts, P_i and P_j, is the same as the attribute solution set for the latter pivot concept of both of them.*

Proof

Let the attribute solution set for P_i be sol(P_i). From the definition of the attribute solution set, $C_a \in \text{sol}(P_i) \Leftrightarrow P_i \ll C_a$ or $P_i = = C_a (*)$ holds.

case a) and **case b)** If P_i and P_j is the same concept, or of $P_i \ll P_j$ holds, then each attribute solution set has its attributes and an element concept of the intersection of both sets has both attributes. That is, $\text{sol}(P_i) \cap \text{sol}(P_j) \subset \text{sol}(P_k)$ holds. On the other hand, P_k has both attributes so that any element concept has either attributes. That is, $\text{sol}(P_k) \subset \text{sol}(P_i)$ and $\text{sol}(P_k) \subset \text{sol}(P_j)$ hold. Hence, $\text{sol}(P_k) \subset \text{sol}(P_i) \cap \text{sol}(P_j)$ holds.

case c) It is clear that $\text{sol}(P_k) \subset \text{sol}(P_i) \cap \text{sol}(P_j)$ holds so, we show here the converse inclusion relation. If we take an element P_u from $\text{sol}(P_i) \cap \text{sol}(P_j)$, then from (*), $P_i \ll P_u$ and $P_j \ll P_u$ hold. Assume that $P_u \ll P_k$ holds. Then this contradicts the fact that P_k is the first common latter pivot concept for both P_i and P_j. Hence $P_u \in \text{sol}(P_k)$ holds. That is, $P_k \ll P_u$ and $\text{sol}(P_i) \cap \text{sol}(P_j) \subset \text{sol}(P_k)$ holds. Therefore, $\text{sol}(P_i) \cap \text{sol}(P_j) = \text{sol}(P_k)$ holds.

case d) If there is no common subconcept between P_i and P_j, then there is no common set for their attribute sets. $\text{sol}(P_i) \cap \text{sol}(P_j) = \phi$ holds. □

Theorem 4 *A necessary and sufficient condition that a given descriptive name indicates some object(s) is that the relation among pivot concepts corresponding to the attributes in the given descriptive name is one of the case a), b) or c) in theorem 1.*

Proof

If the relation to the case d) in theorem 1 occurs, then there is no common attribute solution set among the pivot concepts from the case d) in theorem 3.

From the case a), b) and c) in theorem 2, there is always a latter pivot concept between different two pivot concepts. By the repetitions of this process among the pivot concepts, one can reach the last derived pivot concept (let us denote it P_e). Since P_e is the last derived concept, the relation is obtained that for all i such that $P_i \ll P_e$ holds (**). □

Theorem 5 *If a given descriptive name satisfies the condition described in theorem 4, then the last derived pivot concept for it is determined uniquely. Moreover, the comparison order among its intermediate pivot concepts does not affect determination of P_{end}.*

Proof

There always exists a last derived pivot concept from (**) in theorem 4. Let us call it P_{end} and call the resolving comparison $C1$.

If another comparison, say $C2$, results in the last derived pivot concept different from P_{end} (let it be denoted by P_{kon}), then P_{end} cannot be the last derived pivot concept in this comparison $C2$, that is, $P_{end} \ll P_{kon}$ holds. On the other hand, if the comparison $C1$ is applied to P_{kon}, then $P_{kon} \ll P_{end}$ holds. From these facts, $P_{end} = = P_{kon}$ holds. Therefore, the last derived pivot concept is determined uniquely and is independent of the comparison of pivot concepts during a resolving process.

The desired object(s) are given from the set dsol(P_{end}). □

Theorem 6 *The condition that the different descriptive names indicate the same object(s) is to have the same last derived pivot concepts between them.*

Proof

Let the expression to represent the two different given descriptive names be A and B, respectively. Let the last derived pivot concepts be P_{ae} and P_{be} corresponding to A and B, respectively.

⇒ Assume that A and B show the same objects. Let us consider the relation between P_{ae} and P_{be}.

If the relation between P_{ae} and P_{be} occurs as in case b) in theorem 1, say $P_{ae} \ll P_{be}$, then $\text{sol}(P_{be}) \subset \text{sol}(P_{ae})$ holds. That is, A and B may show different object(s).

If the relation between P_{ae} and P_{be} occurs as in case c) in theorem 1, then from theorem 2, there exists a common latter pivot concepts so that each expression can show different objects.

If the relation between P_{ae} and P_{be} occurs as in case d) in theorem 1, then A shows the different object(s) from B.

From theorem 1, the relation between P_{ae} and P_{be} can be only as in case a), that is, $P_{ae} = = P_{be}$ must hold.

⇐ If A and B have the same last derived pivot concept, say P_e, then dsol(P_e) shows the objects. □

4.2 Standpoint and Method of Descriptive Name-Resolution

The resolution method for a set of descriptive names is to investigate the place of a pivot concept in SINet corresponding to each attribute tuple in a given descriptive name, to compare with pivot concepts for these attributes by means of theorems in the previous subsection and to determine the last derived pivot concept on the relation $\ll$.

It is expected that one may look for a pivot concept by mapping separately each attribute tuple in a descriptive name to a pivot concept in SINet. This takes time because several attributes may map to a concept.

Hence, considering the fact that the comparison order among the intermediate pivot concepts does not effect the last derived pivot concept from theorem 5, and considering that a concept usually has several different role-filler pairs, we take the standpoint whether each pair of role and filler in a concept is equal to some attribute tuple in a given descriptive name; first selecting a concept, then by exhibiting the pairs of role and filler which a concept has, comparing them with the attribute tuple which a given descriptive name has, and finally determine whether it is a pivot concept or not.

In the processing, two tables are used: one is called

CAN_List, which is used as a holder of candidate concepts; the other is called SI_Tab and used as a notebook to check the comparison has finished or not. CAN_List also checks double accesses and prevents loops in the search.

a) if a concept, say P_{at}, has a role-filler pair which is the same as some attribute tuple in the given descriptive name, then it becomes a pivot concept. Thus, this pivot concept P_{at} is compared with the latter pivot concept, which has already determined by the previous comparison. Its initial value is a pivot concept with an 'abstract_property.'

If the relation between them is as in case d) in theorem 1, then the comparison is finished because of the descriptive name having no objects.

Otherwise, the new latter pivot concept is determined according to theorem 2. It is the first pivot concept which appears after both of P_{at} and the latter pivot concept on the relation of $\ll$. The next candidate concepts to be compared are restricted to the subconcepts of the new latter pivot concept.

b) in the comparison, if only some type (let us denote it by i) from a descriptive name matches a role in a concept P_{at}, then the filler for this role is on the partial ordering of role i and both the superconcepts and the subconcepts of P_{at} are examined;

(b-1) the comparison search starts from P_{at} along with the partial ordering of the role i, goes upward to the concept link by comparing its fillers with the attribute values of a given descriptive name until appears a concept with the *newrole* for i. If some filler matches a value on the way, the concept with the filler is determined to be a pivot concept, and we continue the same process as in **a)**.

(b-2) otherwise, if no fillers are matched in the superconcept of P_{at}, we pick out the subconcepts of P_{at} and add to them in the candidate list.

c) if there is no pivot concept in a trunk, then the latter pivot concept is unchanged, that is, it indicates the initial value. In this case, the search moves to the different trunk.

From the viewpoint above, the resolution method for a descriptive name determines a new latter pivot concept from a given descriptive name, found by consulting two tables and by tracing the partial ordering in the SINet. This process is repeated until all the attribute tuples in the given descriptive name are checked (by consulting SI_Tab), or all the elements in CAN_List are exhausted, or the case 4) of theorem 1 is revealed. The first case indicates the last derived pivot concept, while in the latter two cases there is no indicated object corresponding to the given descriptive name.

5. Examination and Discussion

5.1 A Test of the Descriptive Name Resolution

5.1.1 Subjects of Examination

We examined the name-resolution algoithm proposed in 4.2, by comparing the number of concepts accessed in the resolution search and by tracing transition of the pivot concepts during the search. In both cases, we took into consideration the entries of the search and used several different forms of descriptive names, which all appeared to indicate the same object(s).

A set of proposed forms of descriptive names was considered to show the different levels of abstraction, different leaving_out of elements in the descriptive names, different sequence of elements or wrong description for the descriptive names.

We took two types of entries; one was the entry on the trunk where the desired objects resided (case I); the other was that the entry was on the trunk where the desired objects did not reside (case II). The former case was divided into: 1) the entry was before the beginning of some partial ordering, 2) the entry was just on the beginning of some partial ordering, 3) the entry was in the middle of some partial ordering, 4) the entry was at the end of some partial ordering, and 5) the entry was after some partial ordering.

Under these assumptions, we examined the following two cases by using the example in Fig. 1.

First case: we made several different forms of descriptive names which are to indicate the general concept 'program_file' and checked whether the sequence of elements in a descriptive name influence the result or not, and so did the ellipsis. To do this, we set the following four different descriptive names. Let us denote the descriptive name by UDN[i].

UDN[1]: there were fifteen different attribute tuples and there was no ellipsis of them. The element order was from concrete attributes to more general ones.

UDN[2]: there were fifteen different attribute tuples and there was no ellipsis of them. The element order was random.

UND[3]: two third of the description in UDN[1]. At least, one of the attributes in 'program_file' exists.

UDN[4]: two third of the description in UDN[2]. At least, one of the attributes in 'program_file' exists.

Second case: we checked the relation between the degree of abstraction in descriptive names and indicated objects. We used the target objects as the 'in_print' in Fig. 1 and made the following descriptive names.

UDN[5]: description of seven different attribute tuples.

UDN[6]: all the attribute types were the same as in UDN[5], but attribute values were more general than UDN[5]. It was expected to indicate the general concept 'functional_file' which was the superconcept of 'in_print.'

UDN[7]: more concrete description than UDN[5], and with the same attribute types in UDN[5].

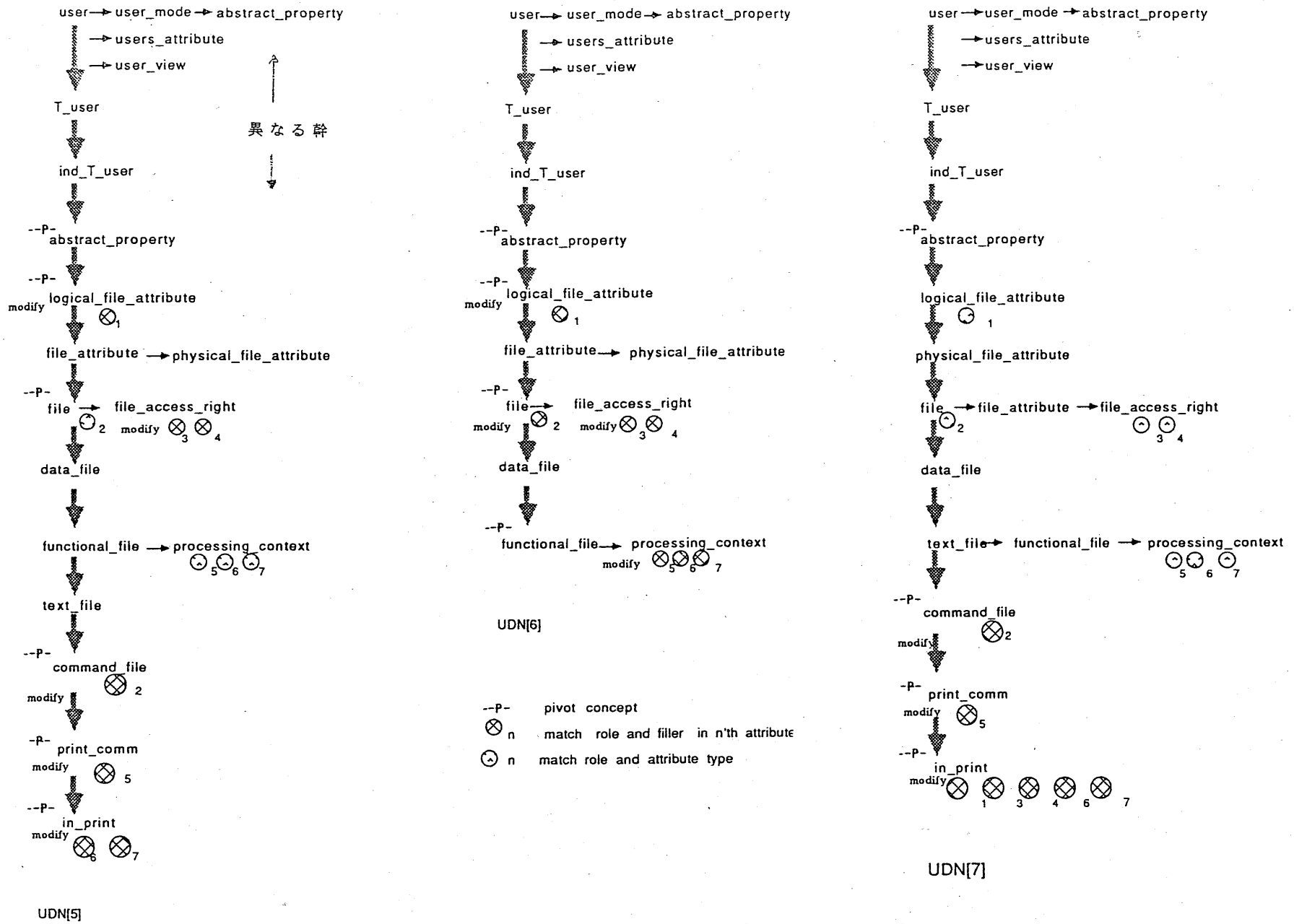

Fig. 3 Searching paths on the condition of different abstraction-level description. Each case of the entry point is incorrect trunk of the SINet.

5.1.2 Results

From the comparison of UDN[1] with UDN[2], or of UDN[3] with UDN[4] by means of the name-resolving method, we find that the order of elements does not affect the object indication. They all indicate the pivot concept 'program_file.'

If the entry of the search is on the trunk of case I in 5.1.1, then the number of concept accessed by the search for any UDN[i] ($1 \leq i \leq 4$) is the same: seventeen concepts in these cases (*).

The comparison of UDN[1] with UDN[3] or of UDN[2] with UDN[4] shows no difference from omitting of elements. In this case, the number of concepts accessed by any search is the same as the result in the previous comparison (*).

Figure 3 shows the tracing of paths in resolving process of UDN[5] through UDN[7]. It results in the desired object: UDN[5] and UDN[7] are resolved to the same concept 'in_print'; on the other hand, UDN[6] is resolved to 'functional_file.' It shows wider objects than the former. This is because UDN[6] has only generic attributes, so that its last pivot concept is 'functional_file,' which is the superconcept of 'in_print.'

The number of concepts accessed in the search is different according to their entries; if the entries are case I in 5.1.1, then the number is the same for any UDN[i] ($1 \leq i \leq 4$); if the entries are case II, then the number searched is increased by the number of accessed concepts on the search necessary to find the first pivot concept.

A restriction (in terms of 3.1) produces several different attribute values of the same type. These may be inherited to the same subconcept because of multiple inheritance. Thus, there may be several paths available in the course of a name resolution. For example, the concept 'program_file' inherits for the type 'file_type' two different attribute values, 'data' and 'procedure' from the concept 'data_file' and 'functional_file' respectively. It does not matter which series of path is used and this supports (3) d) in 3.1.

5.2 Discussion about Representation and Name Resolution of Descriptive Names

5.2.1 Equivalence of Descriptions: Flexibility and Restriction of Representation

Since descriptive names have flexibility of expression, we have focussed the problem of finding the representing rule among a set of descriptive names to denoting the same object(s) while they are expressed differently, and of organizing a way of name-management to be able to treat such expressions. Theorems in Chapter 4 lead us to the fact that different expressions of descriptive names can indicate the same object(s). Theorem 6 concludes the condition that a set of descriptive names indicates the same object(s); despite the different forms of descriptions, as with different order of elements, with different descriptive elements (=attributes) or with different generalization degree of attributes, these forms indicate the same object(s) if and only if they have the same last pivot concept.

For example, let us consider the case of ellipsis of description, such as the relation between UDN[1] and UDN[3]. because they both include the attribute tuple for 'program_file' they both indicate the concept 'program_file.' The next example is the case of description with different generalization degrees. In the resolution process, the transition of the latter pivot concepts for UDN[5] is different from that of UDN[7], as shown in Fig. 3, but the last pivot concept is the same for both UDN[5] and UDN[7]. This is because they both have attributes for 'in_print.'

5.2.2 Abstraction Degree and Indicated Objects

We observed that a set of indicated object(s) by a descriptive name was determined by the position of the last pivot concept in SI_Net. The degree of abstraction in a description is shown in the position of corresponding attribute tuples in the partial ordering. From this fact, the last pivot concept for an abstractive description stays upper part of SI_Net. This case indicates more objects than that of specific description. Because the specific attribute tuple is placed on the latter of the partial ordering, its last pivot concept is in the lower part of the SI_Net and its indicated objects are restricted.

For example, UDN[5] through UDN[7] in 5.1 shows this fact.

5.2.3 Resolution of Descriptive Names based on a Partial Ordering

From the algorithm a) through c) in 4.2, the searching path forms a reverse tree; this search is restricted to the related concepts in SINet; the search determines the latter pivot concept, which limits the range of the search; it examines only subconcepts of the latter pivot concept; the more the search goes down, the fewer subconcepts are related to the search.

This algorithm also considers the property of multiple inheritance in b-1) in 4.2. If the SINet in attention were tree-structured, this processing would not be necessary.

The entry for the search is arbitrary; if the entry is a point in some partial ordering, then the pivot concept is easily found. Otherwise if the entry is on the trunk where no related roles reside, then the latter pivot concept keeps the initial value (P_{abs}). This enables the search to move to another unsearched trunk.

We observe that this algorithm resolves uniformly; no distinction is made between the situations whether some pivot concept is found or not.

The algorithm checks all the role-filler pairs in a concept along with some partial ordering so that no matter where the entry is in the case I in 5.1.1, the number of searched concepts is the same.

5.3 Further Study for Descriptive Names

We take into consideration the descriptive names which explain proper attributes of objects. As described in 2.3, descriptive names can represent contingent attributes as in the descriptions of $\times .500$. The resolution of descriptions, which express contingent attributes for objects, is one for further study. Since KL-ONE has both a description part and an assertion part, it is expected for the resolution to use the assertion part of KL-ONE. These descriptions are considered as a marker transmission in an association memory [6] or as a wave in a Connection Machine. We would like to also consider these remarks as hints.

The notion of CFU in 3.2, is useful for resolution of descriptive names, because it shows whether it is a pivot concept or not directly. On the other hand, it is ambiguous whether the partition is useful for scalability of SINet. The evaluation of this remains for another study.

6. Conclusion

We have concentrated on constructing a resolution method of descriptive names to enable to use attributes of objects as identifiers.

First, we reviewed representational forms of attributes, their abstraction levels and ellipsis which are inevitable for defining the properties of computer resources.

Then, we have clarified the relation between various levels of semantic description and their indicated objects. These are detected by analyzing the set of partial orderings which each attribute class has, borrowing the notion of a semantic network, called KL-ONE, and theorems are proved about this analysis. Based on these theorems, we finally have introduced a new algorithm for resolving of descriptive names.

We think that descriptive names are key elements for constructing inevitable and effective information-resource dictionary systems, which are expected to be one of the important infrastructures in computer systems. This paper has developed a basic notion for the forms and a resolution algorithm of descriptive names as the first step to reach this goal. There remains the work of constructing a concrete system design.

Acknowledgements

We wish to thank Prof. J. A. Robinson of the University of Tokyo (He is now University Professor, at the University of Syracuse, Syracuse, NY.) for being so kind as to edit this text in September-October 1991.

References

1. BIRRELL, A. D., LEVIN, R., NEEDHAM, R. M. and SCHROEDER, M. D. Grapevine: An Experience in Distributed Computing, *Comm. ACM*, **25**, 4 (1982), 260–274.
2. BRACHMAN, R. J. An Overview of the KL-ONE Knowledge Representation System, *COGNITIVE SCIENCE*, **9**, 2 (1985), 171–216.
3. COMER, D. E. and PETERSON, L. L. A Model of Name Resolution in Distributed Systems, *The 6th International Conference on DISTRIBUTED COMPUTING SYSTEMS* (1986), 523–530.
4. DEBRAY, S. and PETERSON, L. L. Reasoning about Naming Systems, p. 20, TR 87–16, Department of Computer Science, University of Arizona (1987).
5. PROCEEDINGS OF THE 1981 KL-ONE WORKSHOP, Fairchild Technical Report No. 618, FLAIR Technical Report No. 4 (1982).
6. HIGUCHI, T., FURUYA, T., HANDA, K. et al. The Development of the Prototype of a Semantic Network Machine IXM (in Japanese) *IPSJ SIG Reports*, AI 61–4 (1988).
7. Final Text of DIS7498–3, Information Processing Systems—OSI Reference Model-Part 3: Naming and Addressing, ISO/IEC JTC 1/SC21 N 2872, 1988–7–20.
8. Information processing systems—Open Systems Interconnection—Directory—, ISO/IEC/DIS 9594 Part 1—Part 8 (1988).
9. OPPEN, D. C. and DALAL, Y. K. The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment, *ACM Trans. Office Information Systems*, **1**, (1983), 230–253.
10. PETERSON, L. L. A Yellow-Pages Service for a Local-Area Network, *Proc. the SIGCOMM '87 Workshop, ACM* (1987), 235–241.
11. PETERSON, L. L. The Profile Naming Service, *ACM Trans. Comput. Syst.*, **6**, 4 (1988), 341–364.
12. SHASTRI, L. Default Reasoning in Semantic Networks: A Formalization of Recognition and Inheritance, *Artif. Intell.*, **39**, 3 (1989), 283–355.
13. SOLLINS, K. R. Distributed Name Management, Doctor of Philosophy at Massachusetts Institute of Technology, p. 164 (1985).
14. TERRY, D. B. Structure-free Name Management for Evolving Distributed Environments, *The 6th International Conference on DISTRIBUTED COMPUTING SYSTEMS* (1986), 502–508.
15. TOURETZKY, D. S. The Mathematics of Inheritance Systems, Pitman, London, p. 220 (1986).
16. WATSON, R. W. Identifiers (naming) in distributed systems, Distributed Systems–Architecture and Implementation, *Lecture Notes in Computer Science, Springer-Verlag*, **105** (1981), 191–210.