1J−9

利用者向き記述名表現の字句解析とその名前解決法

古宇田フミ子 田中英彦

東京大学 工学部

1 Introduction

We propose a new syntax notation of descriptive names useful for users.

Currently we intend to construct a revolutional naming system including descriptive names for distributed environments. A conceptual view of our model is shown in figure 1. In this paper, we concentrate on the user interface part in the figure.

Descriptive name is defined in [IS] as a sentence that "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."

We consider descriptive names are important because there are at least following two reasons.

(a) Identifiers are used to distinguish objects. However, we can refer to an object by some identifiers only when we know them correctly. Thus, if we are aware of their identifiers ambiguously or do not know them, using identifiers cannot work for referring to objects. Another way is necessary. A solution to this is to use descriptive names.

(b) When we want to know whether an object with some property exists or not, we will inquire about it. How do we ask it? If a system supports a descriptive name scheme, then we can ask for it by descriptive names, else we have to, for example, phone a friend to ask it. The latter is inconvenient because if the friend does not know it, we can do nothing.

2 Background and Issues

What should we describe in each descriptive name? From the definition in [IS], properties of each object must be described. Then, what structure-elements are used for the descriptions? We have already discussed these structures in [TR] and have obtained that properties of each object are represented by attributes and their relations, and each attribute is composed of a type and its related value(s): a type shows a class and includes values which indicates various levels of abstraction. These values have relations, such as inclusion or dependency, each other; different types have relations, such as dependency on each other, independent of one another or uncompatible.

Syntax Notation of Descriptive Names for Users

Fumiko KOUDA and Hidehiko TANAKA
University of Tokyo

With the assumption of the structure above, that is, a descriptive name consists of attributes and their relations, we have already constructed a resolution method for such forms of descriptive names[TR].

As shown in figure 1, the input form of the name resolution is different from that of users. The form is appropriate for the method and not for users. Remains the problem to provide users with suitable forms of descriptive names. Therefore we consider that the main issue of this paper is to show the syntax notation which is convenient to users.

3 Requirements for the Syntax Expression

Since properties of objects are described by their attributes and relations with some structures in common, these features must be expressed by rules and users write descriptive names with them. These features are included:

(a) to be able to represent the correspondence between a type and a value.

(b) to enable to show the relation or dependency among types or that among values.

(c) to be able to choose level of generality for attribute as one wishes.

(d) to enable to leave out some elements which one considers unnecessary.

(e) to be able to put sequences of elements in free order.

(f) if a description shows incorrect objects, then the system answers for its incorrectness.

4 How to Represent the Syntax Notation

Step 1: Basic Notation

Basic attribute consists of a type and its corresponding single value. It is necessary to relate a type to its corresponding value. We shall denote this as a two-paired tuple covered with parenthesis:(**type-name, value**).

Generality level of a type-value pair can vary by selecting values so that the value determines generality of an attribute. If we choose a value with some generalization level, then we can show the attribute with such level of generality. This enables us to describe attributes with any level of abstraction.

Step 2: Considering Dependency
Usually, an object has more than a single attribute. Some attributes have dependency on the others, and some are not. If there is no dependency among attributes, then they are put sequentially in the arbitrary order. On the other hand, if there is some dependency between attributes, then its structure must be considered. From [TR] there are three kinds of dependency: between types, between values and among attributes.

case 1: Dependency of values mainly occurs in relation to a type, such as a possible scale of length of the type interval. Thus the notation of dependency on values is to relate to a type: it is denoted by (**type-name, value-range**) or more generally, (**type-name, ope(value,..,value**)).

case 2: Dependency between types should be denoted within the type position. If we denote dependency of types separately from their corresponding values, then the type-value relation, such as step 1, is destroyed. There is a need to include the dependency among different types in the parenthesis of type-value notation. On the other hand, if dependency of types is denoted in the left half of a parenthesis, then it may be difficult to put the values in the corresponding position of the types in the right half of the parenthesis. To overcome this difficulty, we propose the following notation: first put a type in the left position, and then follow a denotation of dependency on a set of types related to this type; after these type denotations, put the value corresponding to the leftmost type. This is denoted by (**type-name : dep(type,..,type), value**).

If we take this notation, any of type-value relation, dependency on different types and dependency among values for a type can be described in a uniform fashion.

case 3: Dependency between attributes can be described by the combination of attributes and its dependency functions.

Step 3: Leaving out or Sequence-Order From the users' point of view, it is important to be able to describe various levels of descriptions. To perform this, it is necessary to allow to leave out some attributes of an object which one considers unnecessary or not to compel the sequence order of attributes.

5 Resolved Expressions

Consideration in the previous section leads to the following syntax notations for users to describe descriptive names.

function = | *dep* | *ope* | *int* |
types = | type, type | type, types |
Type = | type | type : *dep*(types) |
values = | value, value | value, values |
Value = | value | values | *ope*(values) |
attribute = | (Type, Value) |
attributes = | attribute | attribute, attributes |
relation = | attributes | *int*(relation, relation) |
descriptive name = | relation |

6 Discussion

6.1 Requirements and Expressions

From the step 1 in section 4, requirement of (a) and (c) in section 3 is satisfied. Step 2 in section 4 shows (b) in section 3 and step 3, (d) and (e). On the other hand, the resolution method described in section 2 guarantees (f).

7 Conclusion

To provide users with convenient syntax for writing descriptive names, first, we have reviewed the structures of properties which objects have: attributes and relations stand for properties of objects. Then, we have checked the requirements for the users to describe these properties. Finally, by projecting the necessary structures of properties for users to construction of a syntax, we have obtained a new syntax notation to satisfy these requirements. It enables us to describe objects by their attributes and relations with some dependency and with no restriction of sequence order nor that of leaving out of some of their elements.

Conformation of the syntax notation and its prototype implementation are for further study.

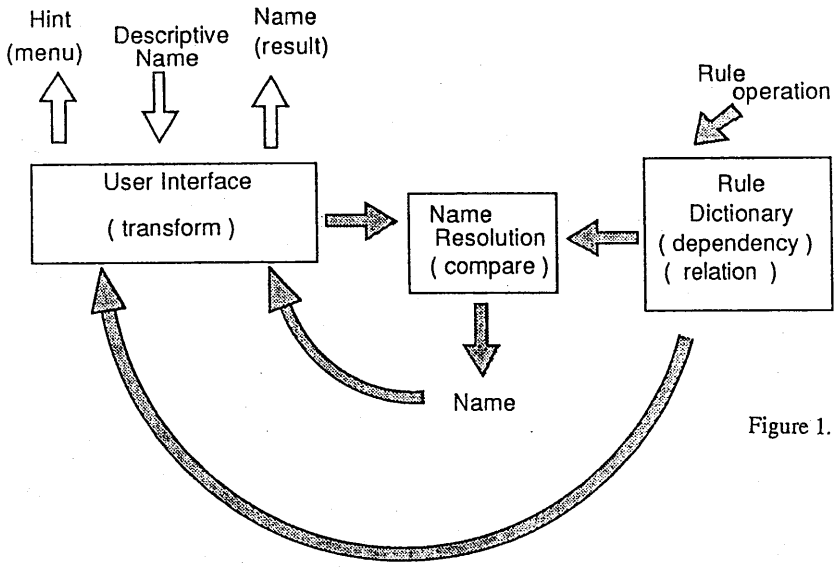

Figure 1. System Configuration

References

[TR]: F. Kouda and H. Tanaka: Analysis of Descriptive Name Resolution, TRIE-91-5, Technical Report of Information Engineering Course, University of Tokyo, June 1991

[IS]: IS 7498 Part 3: Naming and Addressing, 1988