Advances in Computer Methods for Systematic Biology

Artificial Intelligence, Databases, Computer Vision

EDITED BY

RENAUD FORTUNER

11

Object-centered Representation and Fish Identification in Antarctica

NICOLE GAUTIER, ALAIN PAVÉ, AND
FRANÇOIS RECHENMANN

Introduction

There is increasing concern about the deleterious effects of perturbations in the marine environment upon sustained marine harvests in Antarctica. To monitor the situation, fish populations in the Antarctic Ocean are assessed by regular research cruises (Kock et al. 1985) with multispecies sampling in well-delimited areas such as the Kerguelen Islands shelf. Identification of the many species of fish collected offers various levels of difficulty. Some species are recognizable at first glance whereas others are extremely difficult to differentiate. Our research attempts to provide a computational aid to nonexpert identifiers of species. The identification approach presented in this chapter utilizes artificial intelligence (AI) and expert systems.

In specimen identification, information almost always consists of qualitative (i.e., symbolic) morphological characters, even though it may include some quantitative observations. As a consequence, algorithmic computation is not suitable. Our approach to identification is based on traditional classification systems and on characteristics that are easy to observe and collect in the field.

For identification, an unknown *object* is compared to known ones in a reference network. The unknown object is said to be closest to the known object with which it has the fewest differences. If there are no differences at all, the two objects are said to be identical. Structural models based on an object-knowledge representation and hierarchical relations between these objects are widely available. For our application, we chose an object-centered

---

Advances in Computer Methods for Systematic Biology: Artificial Intelligence, Databases, Computer Vision, ed. Renaud Fortuner (Baltimore: Johns Hopkins University Press, 1993).

representation rather than object-oriented systems. While object orientation mixes knowledge and control and uses a procedural programming language, object-centered representation stores knowledge separately from its exploitation and it uses a declarative language (Nebel 1985).

Our identification system is based on SHIRKA, an object-centered knowledge base management system that was developed for the selection of mathematical models in biology (Pavé and Rechenmann 1986). It includes the functionality of an expert system, in particular an inference engine and a mechanism for providing explanations.

The relevance of the object-centered representation approach and of SHIRKA characteristics to identification problems was verified earlier for trees of tropical forests of the Ghat region in India (Gautier and Pavé 1990). The second example presented here uses identification of fish of the Kerguelen Islands.

SHIRKA

SHIRKA was developed by Rechenmann (Rechenmann 1985; Rechenmann et al. 1989). The knowledge model is based on schemes, a concept similar to frames (Fikes and Kehler 1985). In contrast to frames where the object is linked to the notion of prototype, there is a difference between class schemes and instance schemes in SHIRKA. SHIRKA includes the functionalities of an object-centered knowledge representation: slots inheritance, default values, and procedural attachment. In addition, it includes the following functions:

1. The knowledge model is totally uniform. Any scheme is an instance of an upper-level class scheme: a metascheme. In the same manner, any slot, facet, or value is an instance or a reference to an instance.
2. The slot type is mandatory, either basic (integer, real, character string, symbol, Boolean) or complex (pointing to another class by the name of its definition scheme).
3. The procedural attachment, which associates one or more procedures to a slot, uses external descriptions of procedures as schemes. The call procedure is thus completely controlled by SHIRKA.
4. The inference of unknown slots values uses pattern matching rather than rules. Pattern matching, as a crucial inference mechanism, is designed to be efficient; an original technique for compiling a knowledge base has been successfully implemented.
5. The network of classes and subclasses supports a identification mechanism to determine potential classes of an instance.
6. SHIRKA makes it possible to manage instances consistency through procedures attached to slots, which are activated when values are modified,

deleted, or added to the slot, a class scheme being an instance of a meta-scheme. Management of consistency is extended to the classes themselves. This makes knowledge base restructuration and the definition of knowledge acquisition mechanisms easier.

SHIRKA is written in Le_Lisp from I.N.R.I.A. (Institut National de Recherche en Informatique et Automatique) (Chailloux et al. 1986). Le_Lisp is a widely implemented Lisp dialect.

The Knowledge Model

Classes and Instances. A scheme describes a class of objects as well as a specific element of a class, an instance. Here, a class is an taxon (e.g., a species), and an instance is an individual fish. A class and the related instances are defined by slots, which are similar to characters. In turn, a slot is defined by a list of facets, which represent knowledge about these characters. A facet is defined by a list of values; a value may be a scheme or a reference to a scheme (Fig. 11.1), or it can be the range of possible states for the character. Species are described as class scheme and unknown specimens as instance scheme. In Figure 11.1, any two-dorsal-fin fish is specified by the dorsal fin and the first dorsal/second fin types to which it belongs. The slot **is-a** defines an instance of the *Two-dorsal-fish* class.

Classes are organized as an acyclic network of classes and subclasses. A subclass describes objects that are more specific than those described by its parent classes (Fig. 11.2). A subclass inherits the knowledge of the slots defined for its parent classes, and it contains specific knowledge stored in additional slots. Specific knowledge must be consistent with inherited knowledge.

To use SHIRKA, classes and subclasses are first defined (inclusion linkage is made by the slot **a-kind-of**). Then instances are created using the slot **is-a** to link the instance to its class). For example, a taxon is created as a subclass, and an unknown specimen is created as an instance. SHIRKA attempts to answer requests for the unknown slot values, using the knowledge attached to the class.

Creation of new classes and of new instances are two distinct tasks, with distinct complexity. The former is usually made by the designer of the knowledge base, the latter by the user.

Slots. The semantics of the knowledge model are specified to a great extent by the available facets. A slot type is defined by the facets *$one* or *$list-of*, according to the admissible number of potential values. The type is either basic (integer, real for numbers, character string, symbol, Boolean) or com-

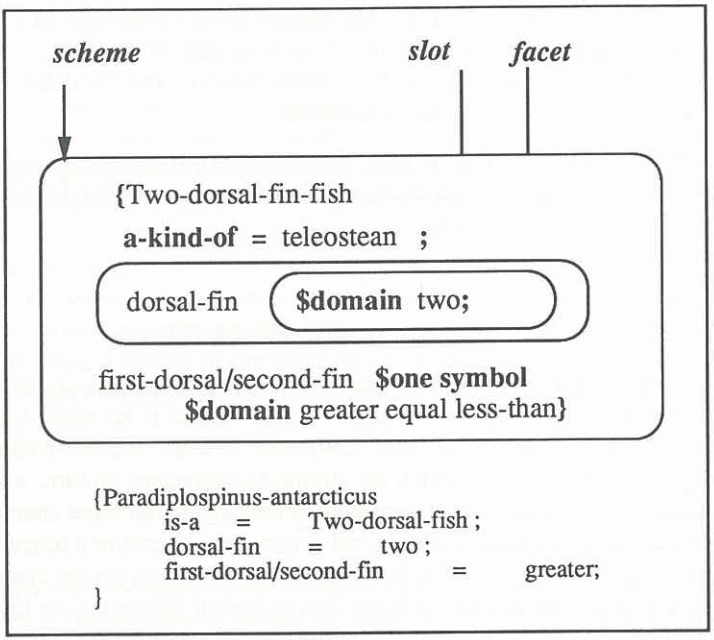

Figure 11.1. Structure of the scheme defining the class **Two-dorsal-fin-fish.**

plex (referencing another class by the name of its definition scheme); in this case, particular rules are applied for coherence maintenance.

Some facets are used to restrict the admissible values for a slot. *$domain* defines a complete list of admissible values, and *$range* a list of ranges of admissible values for a slot with type = ordered. *$card-min* and *$card-max* specify the minimum and maximum number of values for a multivalued slot. Complex predicates may be associated to a slot with the facet *$check*.

Other facets define how to infer the unknown value of an instance slot. *$value* introduces a class value, that is, a value assigned to a slot for every instance. *$default* is similar, but the relative value may be redefined in an instance.

Procedural Attachment

The procedural attachment in SHIRKA differs from that in classical frame systems. Each procedure is externally defined as a class, where the name of the Lisp function (its internal definition) follows the facet *$value* in the slot **fct-name**. The procedure call consists in instantiating the subclass scheme

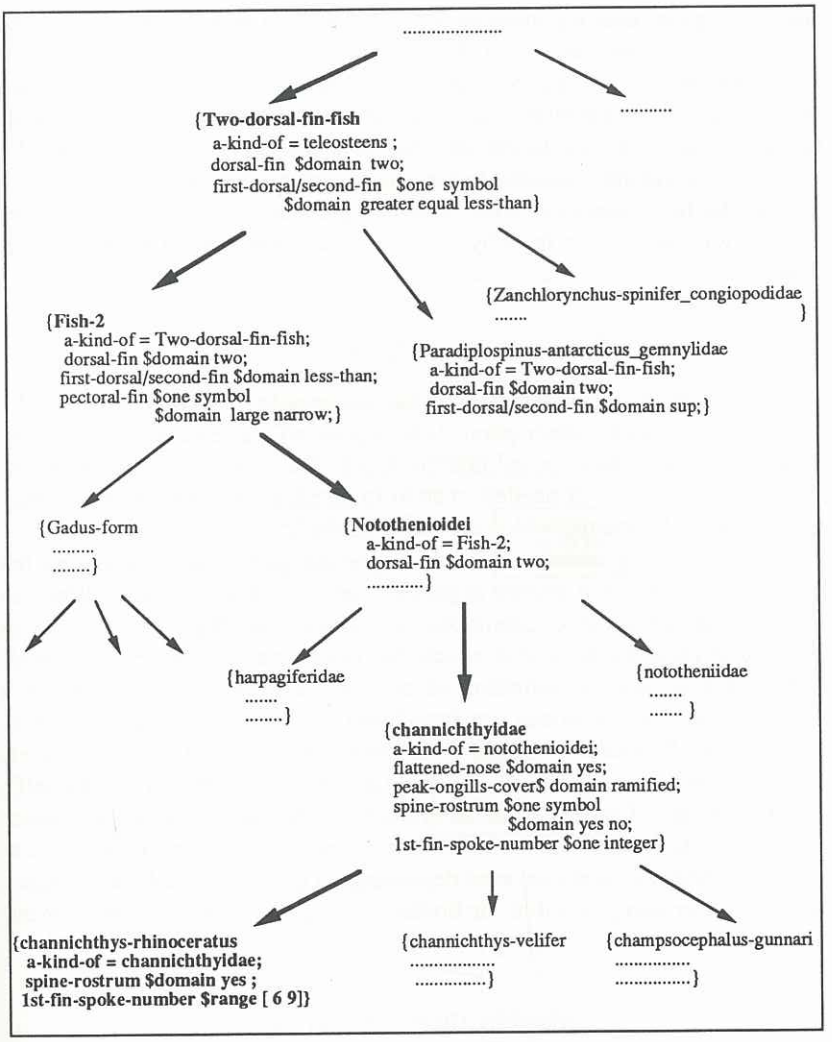

Figure 11.2. Knowledge organization in an object-centered representation: Schemes are linked following the hierarchy via the predefined slot **a-kind-of.**

and passing the resulting instance to the function. The function accesses the input parameters and carries out the necessary computation. When procedural attachment is used in a facet *$ifn-exec* (if needed, execute), the function provides the instance with the values of the slot attached to the output parameter. The value is passed to the slot that uses this procedural attachment by means of the variable attached to the output slot by the facet *$var->*. When used in the facet *$ifn-exec*, this controlled procedural attachment infers unknown slot values. It is fired by the inference engine and also by the facet *$check*.

Pattern Matching

Pattern matching is an inference mechanism specific of SHIRKA. First, the instances that match a description, (i.e., a pattern) are selected; then, among these instances, values for the unknown slots are selected via the variables. The facet *$ifn-match* (if needed, match) implements this mechanism; it may include several patterns, which are sequentially tried.

Pattern matching consists of instantiating the pattern and comparing the instances of specialized scheme in the base with the pattern instance. When an instance matches the set of conditions, the value of one of its slots becomes an admissible value for the slot to which the pattern-matching facet is attached.

SHIRKA allows the definition of constant and variable conditions in a pattern. Constant conditions are introduced by the facets *$value*, *$range*, *$excet*, and *$domain*. Variable conditions are described with the facets *$check*, *$var←* and *$var-list←*, *$var→* (attached to the predefined slot **self**), *$ifn-match*, and *$ifn-exec*. For identification, the classification mechanism uses the facets *$range*, *$domain*, *$except* to introduce constraints attached to slots. The first two facets are used depending on the type of each slot, integer, real, character string, symbol, or Boolean. The third facet, *$except*, is available in all cases.

Identification Mechanism

Since object classes are organized in a hierarchy (Fig. 11.2), in which the more general classes dominate classes that are more specific, a very natural reasoning mechanism consists of looking for the possible locations of a given object in this hierarchy. The only purpose of the root of the hierarchy, the *class object*, is to transmit to every class some bookkeeping slots that are required for proper operation of the system. It has no meaning regarding the domain which is modeled in the knowledge base. The knowledge base is made up of several hierarchies that are rooted in the subclasses immediately below the class object.

The object to be identified is initially known to belong to its instantiation

class in one of these hierarchies, that is, in the class that was used to create the instance. This class may be different from the root class of this hierarchy. It is supposed that the object satisfies all the constraints attached to this class, even when the object is incomplete (i.e., when some of its slot values have not been assigned during its instantiation). The identification algorithm is recursive and makes use of a *breadth-first* scheme. At the beginning, it attempts to match the object with every immediate subclass of the instantiation class (Fig. 11.3).

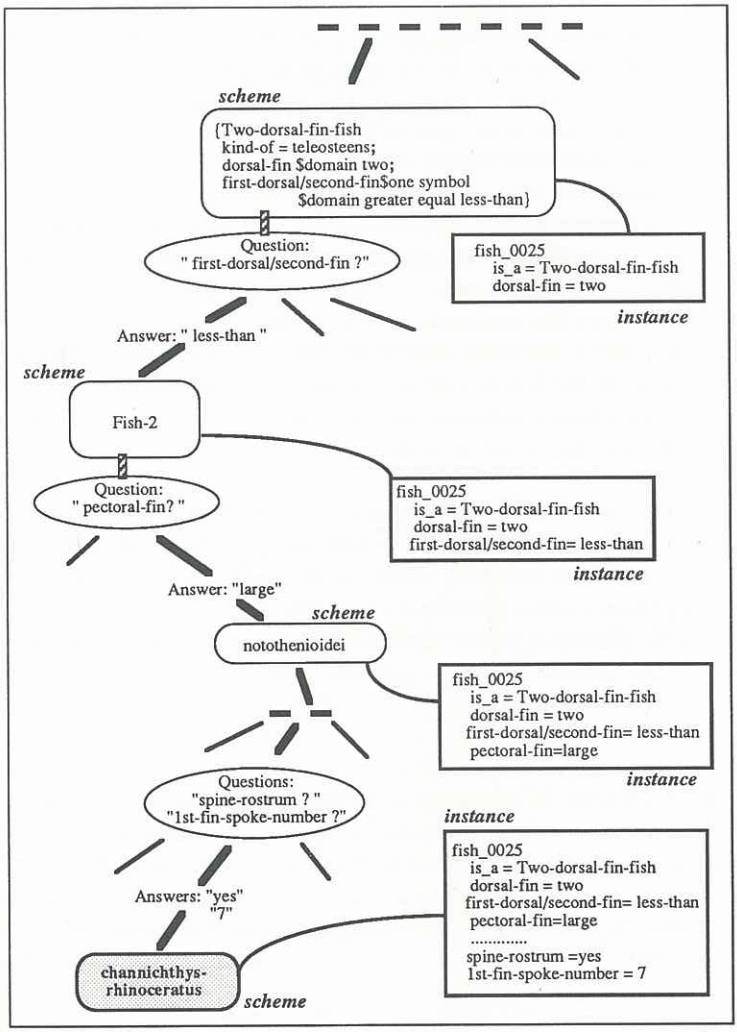

Figure 11.3. A simplified identification process. An instance (fish_0025) is created; then slot values are set by asking the user or by inference.

The constraints expressed in these subclasses and attached to their slots are checked against the slot values of the object instance. If all the constraints can be verified for a class, this class is said to be "certain" for the instance. As soon as one constraint in a class is not satisfied, this class is said to be "impossible" for the instance. When a slot that has not been inherited from the parent class appears in a subclass, it is necessary to get its value for the instance being matched with the subclass. The value cannot be obtained using knowledge from the parent class since it is a new slot. Moreover, procedures or default values attached to the slot in the current subclass should not be used, since the instance is being matched against this class in order to know whether it might belong to it. In our application, the user is asked to provide this value. If he is unable to do so, the slot value remains unknown, and the current subclass is said to be "possible" for the instance.

When all the immediate subclasses have been tested, the matching process is recursively applied to every certain or possible subclass, since these subclasses are not supposed to be exclusive. The process terminates when terminal classes are reached or when only impossible classes remain. The output of this identification process is a list of certain classes, a list of possible classes, and a list of impossible classes in the hierarchy.

To explain why a particular class is possible or impossible, SHIRKA provides the name of the first slot for which the value was unknown or the name of the first slot whose constraints were not satisfied. No explanation needs to be provided for a class given as certain since obviously all its constraints have been satisfied.

Our knowledge of an object increases through this identification process since the classes to which it could belong are more specialized than its instantiation class. This reasoning mode can be used in any diagnostic-oriented knowledge-based system (Puvilland et al. 1989).

A possible extension of the algorithm would consist in computing some measure of the distance between a potential class and the object under consideration. Checking a constraint would no longer be made in a binary (*yes/no*) manner, but by computing the distance. This mechanism is used in CLASSIC (Anonymous 1988b), a knowledge-based system development shell. Another problem to consider is the practical inability for an expert, or a group of experts, to define a unique hierarchy of classes and subclasses. Conflicting classifications systems can be used to define several hierarchies. These hierarchies certainly have several classes in common, but they differ in the respective locations of these classes and in the slots which are taken into account (Marino 1990).

Metaschemes

The consistency of a knowledge base must be maintained whether it uses production rules or object-centered representation. A Truth Maintenance Sys-

tem (TMS) solves this problem in rule-based systems for factual knowledge, either inferred or provided by the system (Doyle 1979). SHIRKA provides value consistency control mechanisms through the use of event-driven facets together with an adapted TMS when actions (e.g., addition, deletion or modification of values of a slot) are required (Euzenat and Rechenmann 1987).

The concept of metascheme (Fig. 11.4) makes it possible to consider a class scheme as an instance of the metaclass scheme or of one of its specializations. The class scheme has the following slots:

ako (a-kind-of), to link a class with its superclasses;
spec, to link a class with its specializations;
inst, to link a class with its instances;
slot-list, the list of the slots of a class.

This metadescription level provides a straightforward solution to the problem of class consistency. Specializations of scheme may be created in order to define specific event-driven methods such as adding, deleting, or modifying values of a slot, which is an item of the list **slot-list**. Such event-driven methods may also be defined in the class scheme itself in order to provide default event-driven behaviors, or in other slots (e.g., **ako** or **spec**) in order to deal with the modifications in the inheritance lattice.

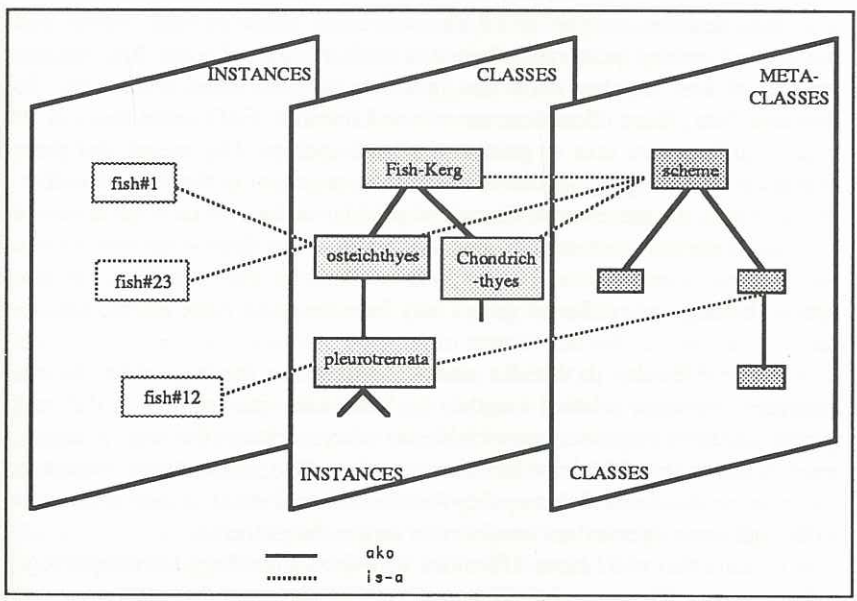

Figure 11.4. Metaclasses, classes, and instances. Any class is an instance or a specialization of the metaclass scheme.

Development of a Knowledge Base for Antarctic Fish

Capture of identification data in fishing boats must be simple, and it must rely mostly on conspicuous morphological characters. Identification should mimic the expert-reasoning process with a global approach using all the features simultaneously but also allowing instant recognition of specific forms.

To create a knowledge base for Antarctic fish we must find the most suitable class hierarchy and, for each scheme, we must find the slots that allow fine discrimination of objects. In addition, the hierarchy and the slots must allow updating, adding, deleting, or modifying the knowledge base without reprogramming the whole application. The traditional classification system (with orders, genera, species) seems to be a good model to build the knowledge base.

A first knowledge base has been built with the systematics characters used in the Food and Agriculture Organization (FAO) guide for the identification of Austral Ocean fish species (Fisher and Hureau 1985). These characters were used in a dichotomous key for thirty-four species (Sieffer 1988). Then, a second knowledge base was built with multiple criteria (i.e., multiple slots) at each node (i.e., each scheme).

Description of Slots

The slots describe morphological characteristics: shape of head, snout, and fins (dorsal, caudal, pectoral, pelvic, and anal), number of dorsal fins, relative lengths of fins, number of thorns, position of eyes, head scales, etc. To describe slots, those characters are selected from the FAO guide that can be used to differentiate taxa or groups of related species. This means that these characters are grouping species into families or genera in the various orders. For example, the presence or absence of prickles on fins would seem at first to be a valid identification criterion because it splits the thirty-four species into two groups of equal size. However, this character was discarded because species belonging to different orders may have the same value for this character.

Another difficulty is that the characters that are the easiest to observe (numbers, lengths, relative lengths, fin base) are often similar in different groups or, at the opposite, are variable between species in the same group or, even between individuals of the same species. Also, quantitative characters (size) were discarded because they depend of ecological or age conditions (although some species are consistently larger than others).

It appears that users have difficulties with the terminology for morphological characters. The term definitions are very precise, and there is a great risk of giving erroneous answers (not counting spelling mistakes!) if the user is not

quite familiar with the specialized vocabulary. This is particularly true if the correct character state must be picked up from a list of several closely resembling expressions. To avoid some of these ambiguities, the set of possible answers has been connected to an image base. The user can choose the slot value in a list of possible answers by selecting the corresponding picture (Piraud 1988).

Building a Hierarchy

The first version of our expert system was based on the traditional dichotomous principle, but this concept is outdated and virtually unmanageable.

The passage from a parent node to its children depended only on presence or absence of one character (e.g., presence or absence of chin-barbel). This does not take into account character variability (multivaluate slots), and it is much too crude. It results in the choice of common criteria that differentiate individuals without due consideration of their taxonomic relationships. For example, *Channichthys rhinoceratus* was identified on a spin-upon-premaxilla character. This does not take into account the fact that this species belongs to the "osteichthyes teleostean gadiform notothenioidei channischthyidae" taxonomic series, and it does not follow the taxonomic model chosen to develop the knowledge base. Also, the lack of taxonomic efficiency made it necessary to use too many intermediate nodes: one node for each value of a slot domain. Closely related taxa present very few obvious differences. Two fish species of same group might be distinguishable by only one character. Then, a second character differentiates a third species, and so on. Additional characters are used to distinguish additional species in the group. This complexity makes it difficult to choose the best hierarchy, and programming this *yes/no* process is almost impossible.

The SHIRKA language can solve these difficulties. The passage from one hierarchical level to the next is based on restrictions on the values of parent-class slots and on additional slots. Child classes inherit several slots, and they can have some values in common for some slots. In scheme writing, the number of slots and the number of slot-typing values are not limited. At the opposite, a *$except* facet can be used to restrict the admissible values for a slot. Combining these possibilities, a class scheme allows describing an object in fine details. The choice of a node on a lower level in the hierarchy scheme is made by analyzing the intersect between domains of the different slots of this node scheme. This mechanism is also available to solve ambiguity of value answers for an particular slot. We can explain this by the example in Table 11.1. Table 11.1 represents an object 0 and its five children, objects I to V. The child-object I inherits specifications a11 and a12 to slot 1, a21 to slot 2, etc. It is easy to see that objects II and IV are not distinguishable if the specimen to be identified has values a11 on slot 1, a22 on slot 2, and a31 on

Table 11.1. Example of Slot Values Domain for an Object and Its Five Children

		Slot Values Domain				
Object 0	Slot 1	a11	a12	a13	a14	a15
	Slot 2	a21	a22	a23		
	Slot 3	a31	a32	a33	a34	
Object I		a11	a12			
		a21				
		a31		a33	a34	
Object II		a11				
			a22			
		a31				
Object III				a13	a14	
				a23		
			a32			
Object IV		a11				a15
			a22			
		a31			a34	
Object V		a11	a12	a13	a14	
		a21	a22	a23		
		a31	a32	a33		

slot 3. In such case, it is necessary to consider an additional slot (slot 4), either at level 0 or at level 1 as shown in Figure 11.5.

It is better to attach slot 4 to the parent level (Fig. 11.5, strategy A) because this leaves the children at the same level. This fits quite well with the domain when object 0 is a genus and the child-objects are species belonging to this genus. It is obvious that the dichotomous approach that uses one character at a time is not suitable here, because the child-objects are differentiated by all the characters taken simultaneously (polytomous approach). The example shown in Table 11.2 represents the lower part of hierarchy schemes for Nototheniidae family in the Kerguelen fish knowledge base.

The SHIRKA Knowledge Base

During the inference process, the system goes down the hierarchy as far as possible using the values given to the slots. At each node, the choice depends on several answers linked to several slots. This allows one to choose among a great number of possible directions at each node. This mechanism mimics the

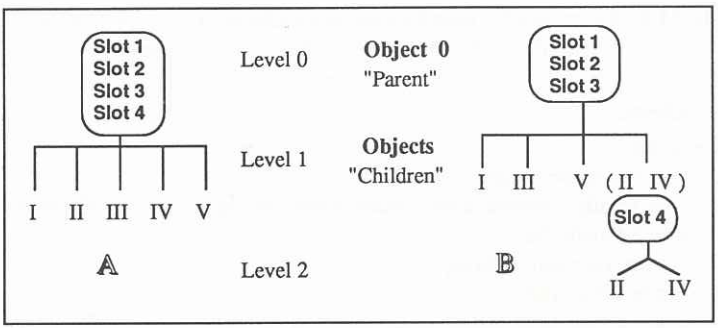

Figure 11.5. Comparison of two strategies for the placement of slot 4.

reasoning of a human expert who uses all relevant features simultaneously. The dichotomous approach can be seen as the most simplistic form of this process where a scheme has only one slot presenting a single alternative.

For the identification of individuals present in a well-delimited geographical area where the number of species is rather limited and where they belong to several taxonomic families, the best strategy for building the knowledge base is to select the criteria that best discriminate species into existing taxonomic units. The knowledge base is organized as follows:

1. The top of the hierarchy uses characters that allows the differentiation of large groups of species, for a clear separation into taxonomic groups. Each scheme needs only one or two slots.
2. At the intermediate level, these large groups of species are divided into smaller groups, each subgroup including closely related species.
3. Schemes at the terminal nodes of the hierarchy allow the identification of every species. For discrimination of related species, the selected characters (i.e., slots) must allow for the description of the smallest morphological differences. Scheme complexity increases, and it may be necessary to include multivaluate slots.

This organization facilitates updating the knowledge base and ensuring its coherence. New taxa can be added without major restructuring of the knowledge base. New species are introduced in the lower part of hierarchy with additional slots or modification of the values domains. A new family (or a new group of families) can be incorporated in the upper part of hierarchy, often by creating a new scheme.

Table 11.2. Example of Lower Part of Hierarchy Schemes for Nototheniidae Family in the Kerguelen Fish Knowledge Base

Parent-scheme

```
{nototheniidae
  a-kind-of = notothenioidei;
  self $com"family : nototheniidae, order perciform, suborder : notothenioidei";
  flat-noose $domain false;
  thorns-cover $domain without;
  scales-on-head$a symbol
    $domain all-the-head except-preorbital-space except-top-head without-scales;
  Pecto-fin-width>pelv-fin-width $a Boolean;
  ray.-pecto.-fin $a symbol
    $domain>23 22>>19 <19;
  %-interorbit/head-width $a symbol
    $domain %<15 15<%<25 25<% }
```

Children-schemes

```
{notothenia-squamifrons
  a-kind-of = nototheniidae;
  scales-on-head $domain all-the-head}
{nototheniops-mizops
  a-kind-of = nototheniidae;
  scales-on-head $domain except-preorbital-space;
  ray.-pecto.-fin $domain 22>>19;
  %-interorbit/head-width $domain %<15;
  pecto-fin-width>pelv-fin-width $domain false}
{notothenia-acuta
  a-kind-of = nototheniidae;
  scales-on-head $domain except-preorbital-space;
  ray.-pecto.-fin $domain 22>>19;
  %-interorbit/head-width $domain %<15;
  pecto-fin-width>pelv-fin-width $domain true}
{notothenia-coriiceps
  a-kind-of = nototheniidae;
  scales-on-head $domain except-top-head;
  ray.-pecto.-fin $domain <19;
  %-interorbit/head-width $domain 15<%<25;
  pecto-fin-width>pelv-fin-width $domain true}
{notothenia-rossii
  a-kind-of = nototheniidae;
  scales-on-head $domain except-top-head;
  ray.-pecto.-fin $domain <19;
    (i.e., $domain >23 22>>19)
  %-interorbit/head-width $domain 25<%;
  pecto-fin-width>pelv-fin-width $domain true}
```

(*continued*)

Table 11.2. (*Continued*)

```
{paranotothenia-magellanica
  a-kind-of = nototheniidae;
  scales-on-head $domain without-scales;
  %-interorbit/head-width $domain 25<%;
  ray.-pecto.-fin $domain <19;
  pecto-fin-width>pelv-fin-width $domain true}
{notothenia-cyanobrancha
  a-kind-of = nototheniidae;
  scales-on-head $domain without-scales;
  %-interorbit/head-width $domain 15<%<25;
  ray.-pecto.-fin $domain 22>>19;
  pecto-fin-width>pelv-fin-width $domain true}
```

Conclusion

SHIRKA, although still at the prototype stage, is a complete development tool for knowledge-based systems. It includes an inference engine, a scheme editor, a spreadsheet-like user interface allowing any operation on instances, an explanation module, and a truth maintenance module. However, the user interface is very primitive, using a *line per line* mode. Icon selection would be a useful aid, and other facilities must be developed. The succession of questions asked to the user must be optimized so that the questions follow each other logically. If the user suspects he made an error in character entry, backtracking is not easy with the current system. A single erroneous answer makes the inference process fail, or it downgrades the correct answer from "certain" to "possible." In case of missing or imprecise characters, the system does not provide the option to explore alternative paths as an expert would do. The system has some limitations in difficult cases where experts would use rules of thumb, backtracking, and alternative answers when their first intuition has proved to be false.

In its present version, SHIRKA offers the essential characteristics of an object-centered knowledge base management system for identification of species that are known in a well-defined ecological environment. Implementation of a knowledge base a few kilobytes large is not difficult. The Kerguelen fish knowledge base with 34 species and the knowledge base on tropical trees from the Ghat forest with about 300 species have been created on a Macintosh II.