An Object-Oriented Framework of Pattern Recognition Systems

Norihiko Yoshida and Kouji Hino

Department of Computer Science and Communication Engineering
Kyushu University
Hakozaki, Fukuoka 812, JAPAN

Abstract

In this paper, we describe a purely object-oriented framework of pattern recognition systems. Its aim is in dealing with knowledge representation issues in pattern recognition. In our approach, everything works in an entirely autonomous and decentralized manner. Even a search procedure for sample-concept matching is distributed onto every concept object itself by being implemented in what we introduced as the recursive agent-blackboard model. We developed an experimental prototype of character recognition systems in Smalltalk-80, which proved the ability of the object-oriented framework and the cooperative search procedure.

1. Introduction

Pattern recognition plays an important roll in human information processing on such as characters, images and speeches. It is a process of examining a set of sensed stimuli with abstract concepts so as to identify it.

Several systems which simulate the pattern recognition process have been researched and developed for various applications. In designing such systems, however, there still exist some issues as follows :

1) How to formalize and represent concepts ;
2) How to structure a sample with sensed stimuli ;
3) How to find a concept which matches a sample.

These issues are, in fact, common in designing every intelligent system.

The object-oriented principle could be exploited for knowledge representation which integrates declarative and procedural knowledges. Concepts could be represented as objects with inheritance and component relationships. We, therefore, apply this principle to pattern recognition systems.

In this paper, we describe an purely object-oriented framework of pattern recognition systems. We explain how we can deal with the issues mentioned above, and show a Smalltalk-80 prototype of character recognition systems as an example.

The eminent characteristic of our approach is that everything works in an entirely autonomous and decentralized manner. An object for a sample takes stimuli in by itself, extracts its own features, and structures itself. Even a search procedure for sample-concept matching is distributed onto every concept object itself.

After Chapter 2 reviews pattern recognition briefly, Chapter 3 introduces an object-oriented framework of pattern recognition systems, and chapter 4 explains a cooperative search procedure in the recursive agent-blackboard model. Chapter 5 shows an experimental prototype of character recognition systems. Chapter 6 contains concluding remarks.

2. Pattern Recognition

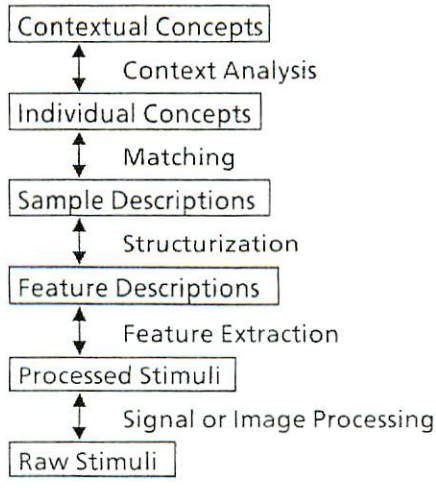


Figure 1. Outline of Human Information Processing.

When we understand what we see or hear, there occur several processing stages in our minds, as shown in Figure 1 [Lind77]. Among them especially, feature extraction, structurization and matching together are refered to as pattern recognition.

2.1 Data-Driven versus Concept-Driven Analyses

For the pattern recognition process, there are two types of analyses, data-driven (or bottom-up) one and concept-driven (or top-down) one [Shir87].

Data-driven analysis is initiated by sensed stimuli, and climbs up the recognition stages. Features and their relationships are extracted out of the stimuli, and a sample is assembled as a structure of the features†. Then a concept which matches the sample is searched for.

On the other hand, concept-driven analysis is initiated with expectation, and climbs down the recognition stages. The most general concept is expected, and it is gradually specialized until a concept which matches sensed stimuli is specified.

The data-driven analysis alone is not sufficient, since sensed stimuli usually contain redundant informations or lack necessary informations for recognition. The concept-driven analysis alone is not efficient, since it tends to cause a lot of trial-and-errors. These two should interact with and complement each other.

2.2 Formal Representation of Concepts

An appropriate representation of concepts is essential in understanding how we understand things. Several models for knowledge representation such as the frame model have been proposed [Mins75]. In them, a concept is represented as a package of relevant knowledges, which contains its own properties as values or as procedures. Concept packages are linked with some types of relationships such as a-kind-of (or is-a) ones and a-part-of (or has-a) ones.

An a-kind-of relationship is a property inheritance, and organizes a speciality hierarchy of concepts. A frame for some special concept is *a kind of* a frame for a general concept. On the other hand, an a-part-of relationship is a structure composition, and organizes a structure hierarchy of concepts. A frame for some component concept is *a part of* a frame for an assembled concept.

2.3 An Example

Here we show a primitive example to illustrate the pattern recognition process.

In recognition of figures, concepts would be of figures with their shape descriptions, while stimuli would be a set of points or a plane of pixels. The data-driven analysis would be as follows : ① extract lines, curves and acute angles with their connections from the stimuli, ② assemble segments with the lines and the curves, ③ assemble a sample with the segments, ④ examine the sample with the figure concepts. The concept-driven analysis would be as follows : ① expect the most general concept, Figure, ② specialize it gradually, regarding the stimuli, to such as Triangle, Quadrilateral, Circle or else, ③ specify a concept such as Square which matches the stimuli.

As implied in this example, an inheritance hierarchy of concepts is related to the concept-driven analysis, while a component hierarchy is related to the data-driven analysis.

3. An Object-Oriented Framework

Several systems which imitate pattern recognition have been researched and developed for industrial, military and office applications. They usually simulate the process reviewed in Chapter 2. In designing such systems, however, there still exist some issues on knowledge representation, which are related especially to the structurization and matching stages.

A formal representation of concepts is, in its essence, a package of relevant knowledges with a-kind-of and a-part-of relationships, while an object in the object-oriented principle is a package of relevant informations with inheritance and component relationships. Some attempts have, therefore, been made to apply this principle to knowledge representation [Bobr 82, Toko86]. They aim in dealing with general types of knowledges, and exploit this principle as only one basis of their frameworks, together with other principles such as access-oriented programming or concurrency.

We are examining the ability of a purely object-oriented framework applied to knowledge representation in pattern recognition. At an extreme of the principle, we claim that an object should be self-contained for all it concerns ; namely, everything should work in an entirely autonomous manner with no centralized managers or controllers. This could be similar to the autonomous decentralized systems, which are fit well

† A structure of features is often also called an *object* or an *instance*, but we will not use the names, since they both have partcular meanings in object-oriented programming.

for distributed environments and highly tolerant to faults [Ihar84].

Here we explain our object-oriented framework in an autonomous decentralized approach.

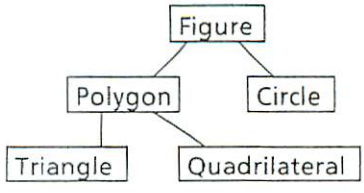
3.1 How to Formalize and Represent Concepts

Each concept is represented as an object respectively, as in other object-oriented knowledge representation forms. An object, or more precisely, a class for a concept contains some features which are to identify itself, and has a facility of comparing features of a given sample with its own for sample-concept matching. A concept class has no instance.

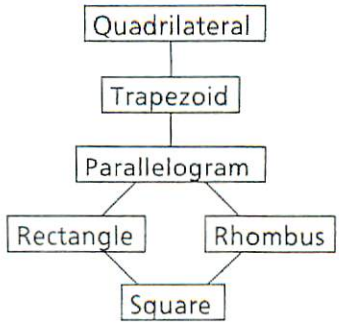
The classes are linked with inheritance relationships, and relevant concepts are placed onto an inheritance hierarchy. A class at the root (or the top) has the most general features, while classes at the leaves (or the bottoms) have the most special features for specific concepts. The exact form of this hierarchy depends on the nature of patterns to recognize. Figure 2 shows primitive examples of concept hierarchies. Classes at each level need not be uniform, and any sub-hierarchy could be of a different form, if its protocols are consistent.

3.2 How to Structure a Sample with Sensed Stimuli

A sample is represented as an object at the root of a component hierarchy. A sample instance has a set of segment instances, each of which has a set of sub-segment instances. Instances at the leaves have stimuli data. The exact form of this hierarchy depends on the nature of patterns to recognize. Instances at each level need not be uniform, and any sub-hierarchy could be of a different form, if its protocols are consistent.



(a) Single Inheritance.



(b) Multiple Inheritance.

Figure 2. Examples of Concept Hierarchies.

Structurization in the data-driven analysis is performed on the component hierarchy of a sample. A sample instance itself has a facility to take stimuli in and to give them to leaf instances. Instances at each level of the component hierarchy structure themselves with sub-level instances ; namely, they assemble their sets of sub-level instances according to extracted features. Then they give themselves to super-level instances. This proceeds up to structuring a sample. There is no centralized structuring engine in a system. Figure 3 shows a primitive example of structurization.

3.3 How to Find a Concept which Matches a Sample

When a sample instance is structured, a concept class which matches it is searched for in the inheritance hierarchy. A concept is nominated one by one, and it compares features of the sample with its own so as to find whether it matches the sample or not.

Specialization of expectation in the concept-driven analysis is performed as downward search in the inheritance hierarchy. The root concept, which has the most general features, is nominated first, and if it matches the sample, then each of the sub-level concepts are nominated. This proceeds down to specifying one of the leaf concepts, which have the most special features.

We distribute a search procedure onto each concept class itself ; namely each has the identical procedure,

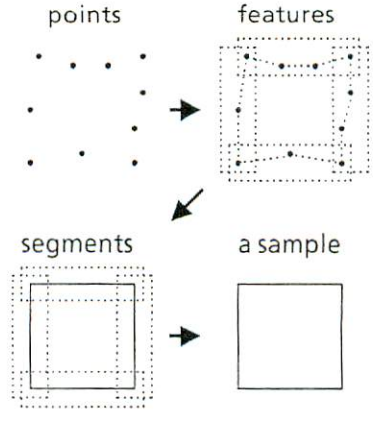


Figure 3. An Example of Structurization.

which searches in only its neighborhood, but cooperates with one another so as to search in the entire hierarchy. There is no centralized search engine in a system. Each procedure could be non-identical, and we could use a different one to any sub-hierarchy, if its protocols are consistent.

We explain the details of this cooperative search procedure in the next Chapter.

4. Search in a Concept Hierarchy

The agent-blackboard model is exploited to illustrate and simulate human information processing [Lind 77]. This model is composed of one blackboard and several agents around it. The agents are certain autonomous experts, and they watch and update bulletins on the blackboard. Typically, the blackboard announces a problem to its agents, then each of them solves its part of the problem by itself, and finally the blackboard collects their answers.

For example, in sample-concept matching implemented in this model, individual concepts would act as agents, and one of them would claim when a sample was announced on a blackboard. There would be no concept-driven analysis.

Here we introduce an augmentation of this model, and name it the recursive agent-blackboard model [Yosh87]. In it, agent-blackboard relationships are nested recursively, as illustrated in Figure 4. An agent itself acts as a blackboard against its inner agents, while a blackboard itself acts as an agent against its outer blackboard. An agent, getting a part of a problem from its outer blackboard, then, as a blackboard, announces its own part to its inner agents. This could be considered as a model of the divide-and-conquer strategy.

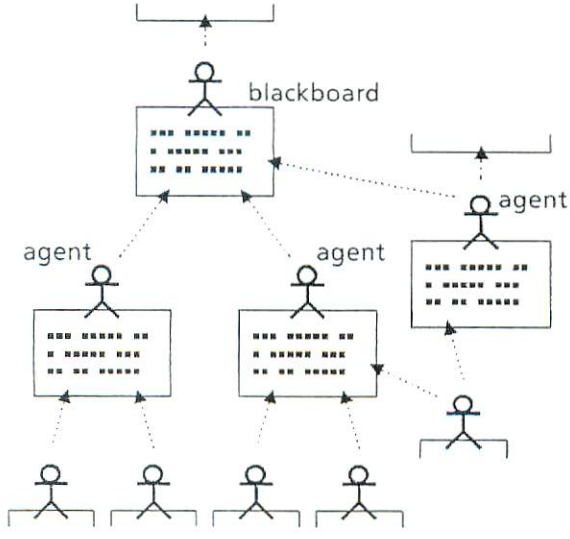


Figure 4. The Recursive Agent-Blackboard Model.

The recursive agent-blackboard model can correspond directly to an inheritance hierarchy. Each class in the hierarchy would act both as an agent and as a blackboard ; namely, in each inheritance relationship, a subclass would act as an agent, while a superclass would act as a blackboard. It is important to note that a class delegates its task to its subclasses in this model, while a class could delegate its task to its superclasses in ordinary object-oriented programs.

Our cooperative search procedure for sample-concept matching is implemented in this recursive agent-blackboard model. Every concept class in an inheritance hierarchy has an identical procedure using the method inheritance mechanism.

The procedure works as follows : ① when a sample is given to a concept, the concept (as an agent) checks the sample, namely compares the features of the sample with its own, ② if the check ends in success, the sample must be of the concept itself or of its descendants, ③ then the concept (as a blackboard) announces the sample to its sub-concepts, ④ each sub-concept performs the same procedure, ⑤ the concept (as a blackboard) collects answers which its sub-concepts reply, ⑥ the concept (as an agent) replies its own answer to its super-concept. This proceeds recursively from the root to the leaves.

4.1 The Case of Single Inheritance

In single inheritance, which is a restricted type, a hierarchy has a tree form. Each class has only one superclass, and this means that each agent watches only one blackboard. The procedure could be considered as an augmentation of the decision tree mechanism.

An agent can reply yes or no immediately after its blackboard inquires, namely announces a sample to examine. A blackboard has only to find whether one or none of its agents replies yes.

For example, if a sample of a square is given to the figure hierarchy shown in Figure 2 (a), the following sequence would occur : ① Figure inquires to Polygon, ② Polygon inquires to Triangle, ③ Triangle replies 'no' to Polygon, ④ Polygon inquires to Quadrilateral, ⑤ Quadrilateral replies 'yes' to Polygon, ⑥ Polygon replies 'yes' to Figure.

Figure 5 (a) shows an Algol-like description of an implementation of the search procedure, search, in the case of single inheritance. In it, %....% denotes a comment, and object.selector(parameters) denotes me-

thod application. Search replies self instead of 'yes' so that we identify which class replies yes.

4.2 The Case of Multiple Inheritance

In multiple inheritance, which is the general case, a hierarchy has a network form. Each class may have several superclasses, and this means that each agent may watch several blackboards.

An agent can reply yes or no to its all blackboards only if they all inquire to it. It must reply no if any of its blackboards does not inquire to it after all, and this can not be known until the entire hierarchy is searched in. For example, in a quadrilateral hierarchy, a class Square should be a subclass of both Rectangle and Rhombus by nature. A given sample could be found whether it is a square or not only if both the superclasses inquire to Square.

In order to solve this situation, we introduce a fractional answer, which is a tuple of a concept itself and a fraction value. A fraction expresses how many blackboards have inquired to the corresponding concept out of all so far. When collecting answers, the procedure sums up fractions of a concept, then divides every fraction by the number of the blackboards, and replies a set of fractional answers. After the entire hierarchy is searched in, a concept with its fraction 1 is the true answer, since this concept has replied partial yeses against inquiries of all of its blackboards.

For example, if a sample of square is given to the quadrilateral hierarchy shown in Figure 2 (b), the following sequence would occur : ① Parallelogram inquires to Rectangle, ② Rectangle inquires to Square, ③ Square replies {Square:1/2} to Rectangle, since it has two superclasses, ④ Rectangle replies {Square: 1/2, Rectangle:1} to Parallelogram, ⑤ Parallelogram inquires to Rhombus, ⑥ Rhombus inquires to Square, ⑦ Square replies {Square:1/2} to Rhombus, ⑧ Rhombus replies {Square:1/2, Rhombus:1} to Parallelogram, ⑨ Parallelogram replies {Square:1, Rectangle:

```
class Concept has
  ......
class method subclasses
  % returns a set of subclasses % ;
class method check(sample)
  % should be defined in each subclass % ;
class method search(sample) is begin
  var subclass ;
  if self.check(sample) is not 'no' then begin
    for every subclass in self.subclasses do
      % if empty, do nothing %
      if subclass.search(sample) is not 'no' then
        return subclass.search(sample) ;
    return self end
  else
    return 'no' end ;
end  %Concept%.
```

(a) Search for Single Inheritance.

```
class Concept has
property numberOfSuperclasses ;
  ......
class method subclasses
  % returns a set of subclasses % ;
class method check(sample)
  % should be defined in each subclass % ;
class method search(sample) is begin
  var subclass, answerSet ;
  assign AnswerSet.new to answerSet ;
  if self.check(sample) is not 'no' then begin
    for every subclass in self.subclasses do
      answerSet.collect(subclass.search(sample)) ;
    answerSet.addLast((self:1)) ;
    answerSet.divideAllBy(numberOfSuperclasses)
    end ;
  return answerSet end ;
end  %Concept%.

class AnswerSet has
  superclass OrderedCollection ;
instance method collect(answerSet) is begin
  var class, fraction, oldFraction ;
  for every (class:fraction) in answerSet do
    if (class:oldFraction) is in self then
      replace (class:oldFraction)
      with (class:(oldFraction + fraction))
    else
      self.addLast((class:fraction)) end ;
instance method divideAllBy(number) is begin
  var class, oldFraction ;
  for every (class:oldFraction) in self do
    replace (class:oldFraction) with
    with (class:(oldFraction / number)) end ;
instance method getTrueAnswer is begin
  var class, fraction ;
  for every (class:fraction) in self do
    if fraction is 1 then
      return class end ;
end  %AnswerSet%.
```

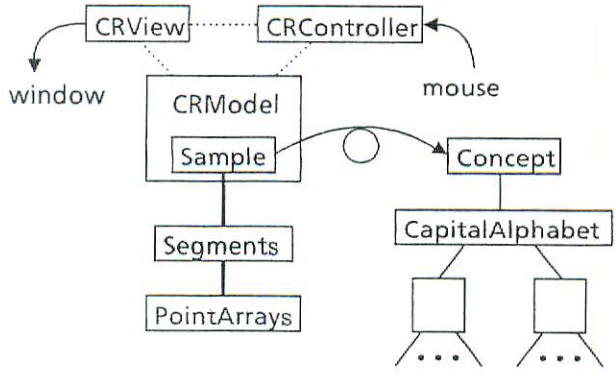
(b) Search for Multiple Inheritance.

Figure 5. The Cooperative Search Procedure.

1, Rhombus:1}. The first concept with the fraction 1 in the set is Square, which is the true answer.

Figure 5 (b) shows an Algol-like description of an implementation of the search procedure in the case of multiple inheritance. In it, (c:f) denotes a tuple. The class AnswerSet is for a set of fractional answers.

This procedure is somewhat primitive and inefficient, since it searches more than once in a sub-hierarchy below a class with several superclasses. But we could improve it by introducing a reference counter in each class.



(a) The Overall Configuration.

5. An Experimental Prototype

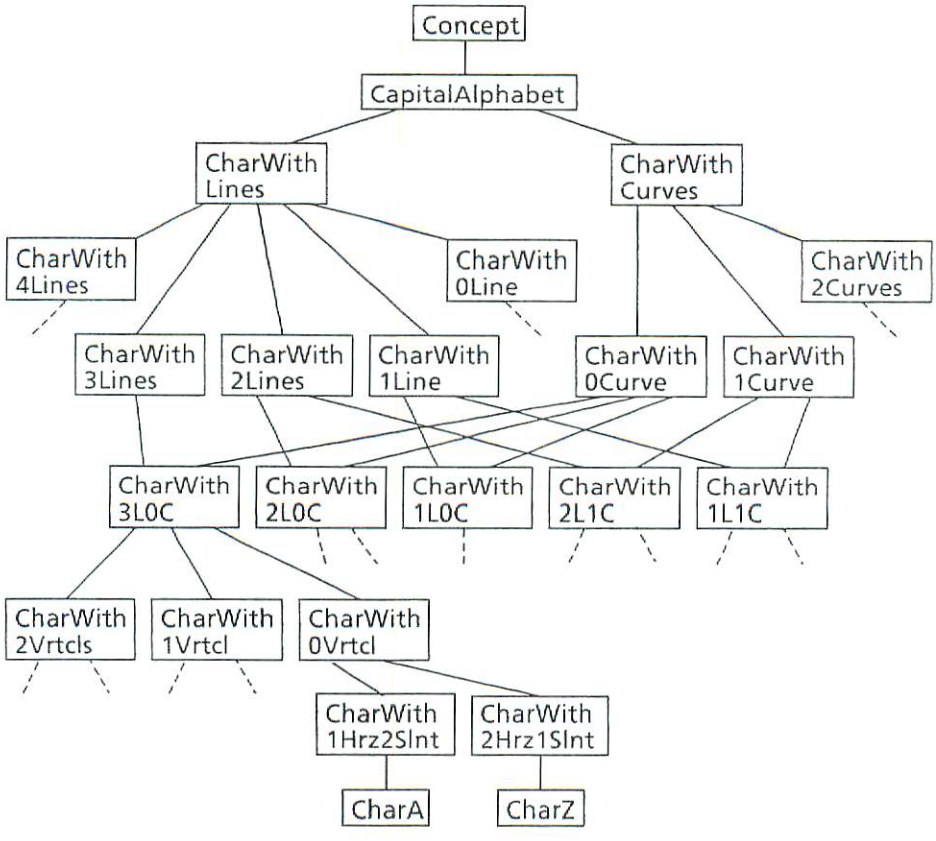
We have developed an experimental prototype of pattern recognition systems in Smalltalk-80 [Gold83]. Its aim is in proving the ability of the object-oriented framework and especially the cooperative search procedure described in Chapter 3 and 4 respectively.

The specifications of the prototype system are as follows :

- It is to recognize a capital alphabet ;
- It is to read an on-line drawing with a mouse and a window.

When we draw a capital alphabet on a window using a mouse, the system reads it as a series of coordinates, then recognizes it, and shows us the answer. This is a primitive example of pattern recognition, since only trivial image pre-processing is required so as to normalize the position and the size of a drawing.

Figure 6 (a) shows the configuration of the prototype system. There could be several instances of the system on a machine simultaneously, and they could share the concept hierarchy.



(b) The Concept Hierarchy.

Figure 6. The Configuration of the Prototype System.

CRModel, CRView and CRController (CR stands for "character recognition") together manage user interface using the Model-View-Controller mechanism of Smalltalk-80. They take mouse tracks into PointArray instances, and show us the answer in a text form.

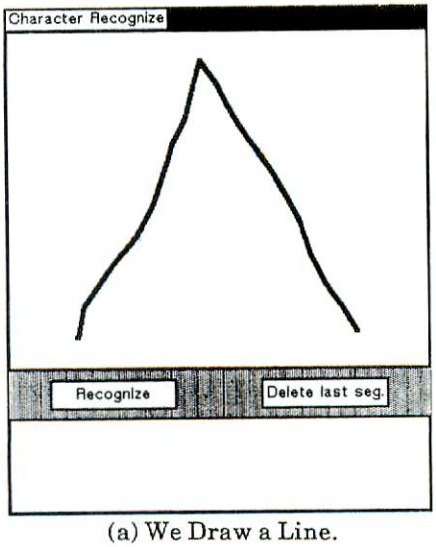
CRModel : a subclass of Sample with a facility for interacting with CRView and CRController.
CRView : a subclass of a pre-defined class View for managing an interface window.
CRController : a subclass of a pre-defined class MouseMenuController for managing mouse inputs and menu selections.

Sample, Segment and PointArray organize a component hierarchy of a sample. After structurization, A Sample instance gives itself to the Concept class.

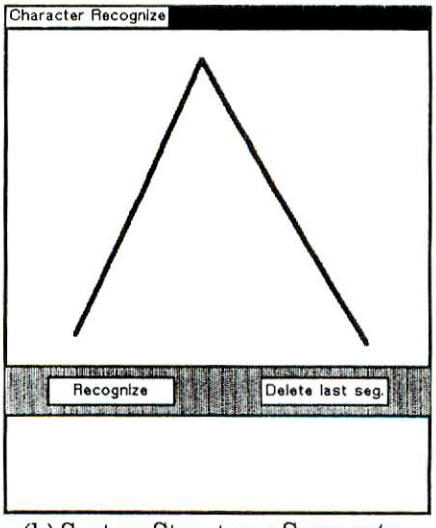
Sample : a structure of several segments with a facility for checking its own features such as the number of its segments and their connections.
Segment : a structure of several points with a facility for checking its own features such as its type of line or curve.
PointArray : a series of raw input coordinates with facilities such as for dividing itself into two at an acute angle.

Concept, CapitalAlphabet and its descendant classes, which are several tens, organize a inheritance hierarchy of concepts. None of them has an instance.

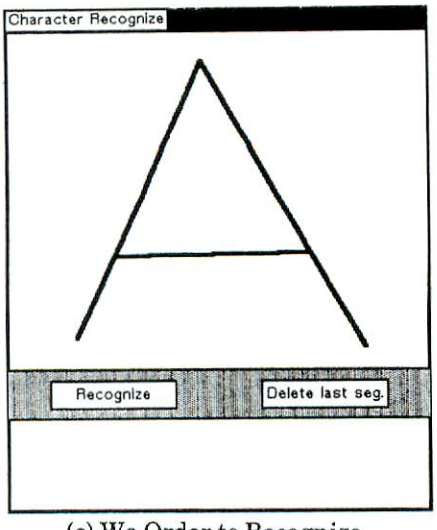
Concept : the template of concepts in the hierarchy defining the methods introduced in Chapter 4, which all concept classes inherit.
CapitalAlphabet : the root class of the concept hier-



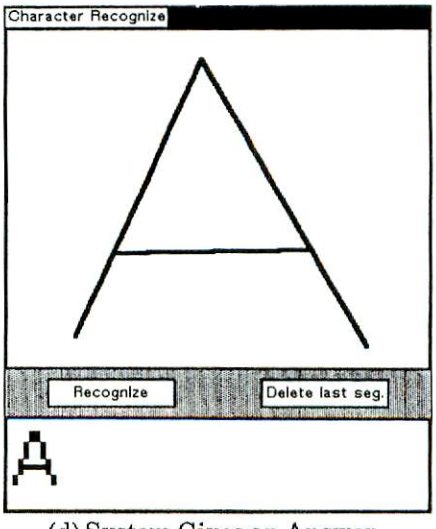
(a) We Draw a Line.



(b) System Structures Segments.



(c) We Order to Recognize.



(d) System Gives an Answer.

Figure 7. An Example Behavior of the Prototype System.

archy.
AnswerSet : a set of fractional answers.

Figure 6 (b) shows a part of the concept hierarchy. Its configuration was designed in a somewhat heuristic manner.

Figure 7 shows an example behavior of the prototype system. It proceeds as follows :

(a) We create an window, and draw something on it using a mouse ;
(b) The system divides the drawing into two, structures two line segments, and displays them ;
(c) When we finish the drawing, we select the menu to recognize ;
(d) Then the system examines the drawing to find which capital alphabet it is, and shows the answer on an auxiliary window.

6. Conclusions

In this paper, we described a purely object-oriented framework of pattern recognition systems. Concepts and a sample are represented as objects with inheritance and component relationships. In our approach, everything works in an entirely autonomous and decentralized manner. A sample object takes stimuli in by itself, extracts its own features, and structures itself. Each concept object compares a given sample with itself, and even has a search procedure in itself.

The cooperative search procedure for sample-concept matching is our major contribution. Each procedure searches in only its neighborhood, but cooperates with one another so as to search in the entire hierarchy. It is implemented in what we introduced as the recursive agent-blackboard model. In this model, agent-blackboard relationships are nested recursively. This could be considered as a model of the divide-and-conquer strategy. This procedure can work well on a multiple inheritance hierarchy.

We developed an experimental prototype of character recognition systems in Smalltalk-80. This prototype system proved the ability of the object-oriented framework and especially the cooperative search procedure.

We now have an attempt to design and implement a concurrent object-oriented programming language which directly supports the recursive agent-blackboard model.

Acknowledgment

The authors would like to thank Professor Kazuo Ushijima of Kyushu University for his valuable support and suggestions.

References

[Lind77] Lindsay,P.H. and Norman,D.A., "Pattern Recognition and Attention", *Human Information Processing* Chap.7, Academic Press (1977).

[Shir87] Shirai,Y ed., *Pattern Understanding* (in Japanese), Ohm Publishing, JAPAN (1987).

[Mins75] Minsky,M., "A Framework for Representing Knowledge", *The Psychology of Computer Vision* (Winston,P.H. ed.), McGraw-Hill (1975).

[Bobr82] Bobrow,D.G. and Stefik,M., "The LOOPS Manual — A Data and Object Oriented Programming System for Interlisp", Memo KB-VLSI-81-13, Xerox PARC (1982).

[Toko86] Tokoro,M. and Ishikawa,Y., "Concurrent Programming in Orient84/K : An Object-Oriented Knowledge Representation Language", ACM SigPlan Notice 21:10 (1986) 39-48.

[Ihar84] Ihara,H. and Mori,K., "Autonomous Decentralized Computer Control Systems", IEEE Comp. 17:8 (1984) 57-66.

[Yosh87] Yoshida,N., "Nested Objects" (in Japanese), Workshop on Object-Oriented Computation, Japan Soc. Soft. Sci. and Tech. (1987).

[Gold83] Goldberg,A. and Robson,D., *Smalltalk-80 — The Language and Its Implementation*, Addison-Wesley (1983).