Reprint

from

Proceedings of the Second International Symposium on Autonomous Decentralized Systems

Phoenix, Arizona

April 25 - 27, 1995

The Object-Field Model for Managing a Group Activity

Toyohide Watanabe, Fumihiko Nishio

Washington, DC ◆ Los Alamitos ◆ Brussels ◆ Tokyo

PUBLICATIONS OFFICE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264 USA

The Object-Field Model for Managing A Group Activity

Toyohide WATANABE and Fumihiko NISHIO

Department of Information Engineering,
Faculty of Engineering, Nagoya University
Furo-cho, Chikusa-ku, 464-01 Nagoya, JAPAN
(phone) +81-52-789-4409 (fax) +81-52-789-3816
(E-mail) watanabe@nuie.nagoya-u.ac.jp

Abstract

The object-oriented model is very applicable to represent various phenomena in the real world from an entity-interaction point of view. The successful modeling methods have been developed on many current topics. However, it is not sufficient to model a group of interrelated objects and dynamic actions of individual objects effectively under this paradigm. This model is in short of specifying environments as the activity ranges among objects and representing the interactions between objects and environments.

In this paper, we introduce the concept of field in addition to the notion of object with respect to constructing the cooperative environment for objects, and then discuss our modeling method based on the object and field. Our modeling method is successful to construct a group of objects through the field, and to organize a hierarchical structure among objects by looking upon the field as an abstract object conceptually. Also, we explain a property adaptation mechanism of objects in fields with respect to the group activity of objects.

1 Introduction

The object-oriented paradigm is effective to model various phenomena in the real world and gave an impact to the design and implementation of information systems [1, 2, 3]. It is not only important to look upon objects as concurrently controlled entities, but also necessary to represent the interactive relationships among objects. In the object-oriented paradigm, all objects are self-identified entities, and can solve their sharing problem by means of communication among them. However, in modeling various kinds of entities in the real world we observe that they are not always independent existences, and that they organize several groups together with interrelated objects and act/react cooperatively in the organized groups. Thus, we have many problems if we specify such group activities of objects in the traditional object-oriented model. This is partly because the object-oriented model focuses on the characteristics of object itself but does not pay attentions to the group activities and interrelations among objects, and partly because it can not support the concept of grouping. Traditionally, in the object-oriented model the constructive relationship is derived from the class hierarchy, and the generalization and specialization are the most general relationships. These relationships are a classification on the basis of characteristics of individual objects, but do not correspond to the object grouping with respect to the behaviors of objects. If we intend to model our complex phenomena so that an object group can compose another group as a composite element stepwisely, it is necessary to deal with parts of constructive relationships.

In this paper, we propose a framework for representing/implementing the cooperative interaction among objects on the basis of object-oriented model. The concept of field is introduced to define the cooperative environment among interactive objects [4, 5]: the field is a passive entity. Moreover, we look upon the field as an active entity in order to model the group activity of objects. Namely, our approach supports the modeling method from two perspective points of view: one is to regard a group of objects as a passive entity from the inside of field; and another is to look upon the group as an active entity from the outside of field. Of course, we need a definition mechanism which characterizes the entity from different modeling views: the overlay mechanism is proposed with respect to this requirement. Moreover, we deal with a property adaptation mechanism of objects for field. This provides objects with supports to the roles to be assigned tentatively in the field. These mechanisms make it successful to specify/control the group activity of objects effectively.

2 Object-Field Model

When we observe the activities of entities in the real world, an entity organizes a group with other entities with a view to attaining to their common goal: companies and societies are typical examples. We attack to the following issues in order to model the cooperative interactions among objects.

(1) Communication among objects: Objects in a group communicate mutually with a view to attaining to their common goal. An object does not only communicate with another object directly, but also must exchange messages with many unspecified objects.

(2) Abstraction of groups: Mutually related objects organize a common group and the group itself interacts with other groups or objects which do not belong to the group cooperatively. It is successful to look upon groups as abstract objects with a view to modeling various organizations.

(3) Dynamic alteration of object property: Individual objects belong to appropriate groups and interact with members of the groups. Of course, objects can move among groups owing to their own intentions.

0-8186-7087-8/95 $4.00

When objects belong to a group, they take peculiar roles. Thus, the new properties, which were not specified originally in the object definition, should be assigned so that each object plays a peculiar role in the group activity.

As for the issue (1), the traditional object-oriented model supplies only objects as modeling elements, but does not support the concept of environment or group. It is difficult to specify the indirect interaction among unspecified objects though the direct communication of object-to-object is basically established. Therefore, the modeling methods, which support new constructive elements in addition to ordinary objects as identifiable entities, have been reported: the field in Kamui-88 [6] and the environment in organizational computation model [7]. However, these methods do not enable to represent a group of objects, corresponding to the issue (2). As for the issue (2), E.Blake et al. [8], N.Craske [9] and R.Helm et al. [10] proposed methods which manipulate the interrelations among groups of objects on the basis of the "part-of" relationship. However, these methods must predefine individual objects, which construct a group, as members statically and these preassigned relationships can not be changed dynamically. Additionally, these methods can not deal with the cooperative interactions among members because they focus on only the hierarchical "part-of" representations among objects. As for the issue (3), the property alteration mechanism of objects was not addressed in the traditional object-oriented model partly because even the concept of environment is not introduced as the modeling element and partly because the properties of objects are specified absolutely in the definition. In the object-oriented model, the multi-inheritance mechanism may make it possible that objects have various properties at once. However, this mechanism is applicable to only objects attended with ISA relationships in the class hierarchy, but does not make the dynamic alteration of object property possible.

We adopt the following approach, concerning these issues. First, we introduce the field to represent a group of objects. The field is an environment to restrict the object activity and is regarded as a passive entity, while the object is an active entity. Second, the field is modeled as an abstract object, if necessary, in order to specify our observable phenomena from a viewpoint of two modeling ways: inside and outside of fields. Finally, we supply a property adaptation mechanism of objects so that objects should take individual roles of group activities. These features are arranged as follows:

1) The field supports the indirect communication way practically. We assign the broadcasting function to each field so that an object can exchange messages with other unspecified objects, which coexist in the same field;
2) We provide a mechanism to redefine the field as an abstract object. In our observable phenomena, the outside object, which is allocated to the outside of the field, may look upon the field as an abstract object because the field is a distinguished entity for the outside object, while the inside object, which is located in the inside of the field, can regard the field as an environment because it is not necessary for the inside object to identify the field. This is smartly attained by using an overlay mechanism, which overlaps new properties of object on the predefined field. In accordance with this mechanism, we can represent hierarchical structures among entities;
3) Objects can change the properties dynamically according to the activity. When an object joins to the field, the object may acquire new properties, which are different from the properties specified originally in the object definition, in order to play the peculiar role.

3 Concepts of Object and Field

Objects with a common goal organize a group and are coordinated practically in the group. The field is a constructive component to define a group of objects and is an environment in which objects can interact cooperatively.

3.1 Field as Communication Media

The object belongs to an appropriate field to interact autonomously with other objects and coordinates its own activity. If objects belong to an appropriate field, they can communicate with other objects through one-to-many interaction way. The message which was sent to the field from an inside object is propagated to every object accommodated in the same field. The broadcasting-based communication way is effectual because messages can be exchanged without the source and destination addresses.

The field is different from the object as a modeling element, and can not interact with objects directly. Of course, it is not impossible to compose objects, which take a role of our field, in the framework of the traditional object-oriented model. However, such an approach can not model various phenomena clearly on the basis of the interactions among entities and the transition of states. In our model we distinguish these two entities explicitly: the object is an active entity; and the field is a passive entity. Fig.1 shows the relationships between objects and fields, concerning the activities of objects. It is desirable that objects can move among fields by themselves in order to manage new tasks. Introducing the concept of field, we can represent a group of objects smartly, and also construct the state in which some objects can communicate with many unspecified objects in the group.

3.2 Field as A Group of Objects

Our framework based on the passive field and active object does not always model various kinds of complex phenomena sufficiently. As discussed previously, the group which is composed of individual objects can be looked upon as an identifiable entity logically when the group is observed from outside objects. The distinction between objects and fields is dependent on the modeling view: inside of field and outside of field. For example, consider a relationship between a train and persons. Persons in the train feel that the train is an environment because all information exchanges or propagations are enclosed in the train. On the other hand, the persons outside the train feel that the train is an object as the distinguished entity. Therefore, we must not only define a group of interacted objects as an environment for object activity, but also necessarily deal

with an abstract object to distinguish the activity of individual groups.

The relationships between objects and fields are illustrated hierarchically in Fig.2 by analyzing both inside and outside objects for each field recursively. In Fig.2, fields "A" and "B" construct a nested structure: "A" is enclosed by "B". In the previous example of a train and persons, "B" corresponds to a universe of discourse. The train is the object "e" in "B", and also is the field "A". The objects "a", "b", "c" and "d" are persons in this example. Of course, "c" and "d" may be other trains as the identifiable objects. In "B", "e" as well as "c" and "d" must be distinguished clearly as an active entity, and also is looked upon as a group of related objects "a" and "b". "a" and "b" do not recognize the properties of "e", but are influenced passively from the characteristics of "A". Therefore, our framework supplies the concept "grouping" naturally in addition to the inherent concepts "generalization" and "specialization" in the object-oriented model. In particular, our grouping concept has the dynamic feature in comparison with the ordinarily addressed relationship "grouping".

Moreover, we discuss the interactions among objects in the hierarchical structure. In our framework, the one-to-one direct communication way among objects is basic as well as the traditional object-oriented model. While, the one-to-many indirect communication way is useful for inside objects through the field: messages sent from an outside object to the field can be controlled by looking upon the field as an abstract object. In Fig.2, the message sent from the object "d"(or "c") to the field "A" is interpreted as a message sent directly to an abstract object "e"(and may also be manipulated as the broadcasting message in "A").

3.3 Adaptation of Object for Field

Each object belongs to an appropriate field in accordance with the activity. Also, objects can move from one field to another field according to their own intentions. In this case, objects may not only act with their own predefined properties, but also are characterized supplementally from the newly existing field. In the traditional object-oriented model, the multi-inheritance mechanism may be applicable to implement alteration procedures for properties of objects. Consider a driver problem [1]. In order to model this problem, a class "US-DRIVER" which specifies the properties of drivers in USA is first defined. Similarly, a class "FRANCE-DRIVER" for drivers in France is done. In order to define a person who wants to drive in USA and France, the class "FRANCE-US-DRIVER", which is inherited from existing classes "US-DRIVER" and "FRANCE-DRIVER", should be defined. Fig.3 shows such a class hierarchy. In this approach all properties of objects must be specified statically in the class definition.

It is desirable that objects could acquire new properties selectively in terms of the activities and tasks. Such a property acquisition mechanism must be more flexible and dynamic than the approach based on the multi-inheritance. The new properties must be assigned to the object with respect to the role-relationships of individual objects in the field, field by field. For example, consider the previous driver problem, again. In our model, France and USA are defined as fields. The object "aDriver" can move freely among these two fields. Fig.4 shows such a mechanism. In the class "DRIVER", the basic properties as drivers are defined. The properties for USA drivers or France drivers are defined in the properties "FRANCE-DRIVER" and "US-DRIVER" beyond the class definition. "US-DRIVER" and "FRANCE-DRIVER" are assigned to instances of "DRIVER" in entering into individual fields "US" and "FRANCE", respectively. Objects can act with the newly assigned properties in the current field through this mechanism even if they were not specified in the creation time. This mechanism makes the behaviors of objects autonomous. Even if the collision of property names encountered between the originally defined properties and newly assigned properties, objects can act appropriately so as to be consistent to the underlying field though the approach based on the multi-inheritance can not solve easily.

4 Specification of Object and Field

Here, we address the functional structures of object and field, and describe the definition forms. The syntax is based on Smalltalk.

4.1 Specification of Object

The objects in our model are autonomous entities, and execute their own tasks concurrently. The objects themselves must be characterized by the following requirements so that they organize appropriate groups and interact with each other in the groups cooperatively.

1) The asynchronous communication must be supported as well as the synchronous communication. Objects must accept and evaluate messages sent asynchronously from other objects;
2) Objects must determine the reactions for accepted messages by themselves. For example, an object "finite buffer" must suspend the action until some messages are produced, if the message "get" is received when the buffer is empty.

Although objects in the traditional object-oriented model are ordinarily composed of state variables and methods, message buffer and message selector are added [4, 5] in our object structure.

The object does not execute accepted messages instantly, but stores into its own message buffer temporarily. This mechanism supports an asynchronous communication facility in which asynchronously sent messages are successively accepted and then executed according to the evaluation conditions. The message selector retrieves messages from the buffer in terms of the object state, selects an executable message, and then fires the corresponding method. The object in the object-oriented model executes the method instantly after having received the message. Namely, the action of object is dependent on the message, and its mechanism is heteronomous. While, in our model the object can choose the message by itself. Our object is not a simple reactor for messages.

The object is defined by the specification of state variables, methods and message selector. The approximate syntax form is as follows:

```
ex.)
  Entity subclass: #ObjectA
    instanceVariableNames: 'i1 i2'
    classVariableNames: 'c1 c2'
```

```
poolDictionaries: 'p1 p2'!
  !ObjectA class methods!
    <definition of class methods>!!
  !ObjectA methods!
    <definition of instance methods>!!
  !ObjectA selector: #SelectorA!
    <definition of message selector>!!
  !ObjectA selector: #SelectorB!
    <definition of message selector>!!
  !ObjectA property: #PropertyA!
    <definition of property>!!
  !ObjectA property: #PropertyB!
    <definition of property>!!
```

An object is defined as a subclass of the class "Entity". In this description, "ObjectA" is defined. The message selector selects and executes an appropriate method on the basis of accepted message and object state, and this description contains a pair of method and condition to be satisfied for the execution.

```
ex.)
  !<Object class> selector: #<Selector name>!
    <Method name> ^<Condition>!
  !!
```

The message selector checks up the acceptable condition for the corresponding method. Moreover, as many selectors may be defined for an object class, we can create different objects with appropriate selectors by specifying the selector name and class name at once in the object creation. For example, we create objects "a" and "b" as follows:

```
ex.)
  a:=ObjectA create: SelectorA.
  b:=ObjectA create: SelectorB.
```

In this case, "a" is different from "b" because they have different guards "SelectorA" and "SelectorB" though they are instances of the same "ObjectA". Thus, in our model individual objects are characterized as autonomous entities because they can associate with different selectors even if they are created from the same class. We consider the description of selector in the finite buffer object as a simple example. This finite buffer has methods "put" and "get" to manipulate data. These methods are not executed instantly if the buffer condition is not satisfied. This condition for selector is specified as follows:

```
ex.)
  !FiniteBuf selector: #BufSelector!
    put
      ^n<N!
    get
      ^n>0!!
```

4.2 Specification of Field

The field is an environment to make objects active. The entity characterized as a field is different from ordinary objects. This distinction is one of important aspects in introducing the field. The field has the following roles:

- The field broadcasts the message, sent from the outside object, to all inside objects;
- The field manages the object list, which records inside objects, and updates the list for the entrance and exit of objects;
- The field holds state variables to represent an environment state. The state variables are referred to or updated by objects.

Objects can communicate indirectly to other inside objects by sending messages to the field. The field is defined by the description of state variables and procedures, which refer to the variables. The description form is similar to the object definition.

```
ex.)
  Field subclass: #FieldA
    instanceVariableNames: 'i1 i2'
    classVariableNames: 'c1 c2'
    poolDictionaries: 'P1 P2'!
      !FieldA class methods!
        <definition of class methods>!!
      !FieldA methods!
        <definition of methods to access
          instance variables>!!
      !FieldA adaptation!
        <definition of object adaptation>!!
```

"!FieldA adaptation!" and the follows are the description about the property adaptation of objects for the field. As the broadcasting function is a common feature for every field, it is unnecessary to specify it. Objects can be created by the command "create" and deleted by "destruct". Also, objects can participate to the existing field by a command "enter", and retire from the field by "exit".

```
ex.)
  f:=FieldA create.
  f enter: anObject.
  f exit: anObject.
  f destruct.
```

4.3 Overlay of Object and Field

The field provides a common interactive environment for objects, and represents a group of objects. However, we must look upon it as an active entity as well as the object in some cases. In order to assign such an active feature to the field, we adopt a mechanism that overlays newly declared properties on the existing field and redefines the field as an abstract object. Fig.5 shows such a mechanism conceptually. The object "c" exists in the field "A", and the object "b" in the outside of "A" looks upon the entity as an object "a", but not as a field "A". Such two different modeling views are supported through our overlay mechanism of the object "a" for the field "A". Although methods based on the grouping concept have already been proposed until today, these methods focused on only the grouping facility, but did not concentrate on the exclusive modeling view such as our overlay mechanism of object and field.

The description is to redefine the properties of the abstract object in addition to the properties of the existing field, and is similar to the object description.

```
ex.)
  Entity subclass: #ObjectA
    overlay: FieldA
    instanceVariableNames: 'i1 i2'
    classVariableNames: 'c1 c2'
    poolDictionaries: 'p1 p2'!
```

In this definition, overlay: "FieldA" represents that "ObjectA" is overwritten to the predefined "FieldA". The object overlayed in the field is created as follows:

```
ex.)
  a1:=FieldA create.
  a2:=ObectA createOn: a1.
```

This mechanism makes it possible to describe a program as two program fragments separately in accordance with external and internal characteristics of entity. Thus, our description makes it easy to specify the functionalities of entities because a group of objects can be regarded not only as an operative environment for objects, but also as an abstract object.

4.4 Property Adaptation of Object

In our model, objects can get new properties dynamically so as to be adaptable to the group activity in the field. The properties, which an object can acquire newly from the field, are methods and state variables. Such newly attached methods and state variables are manipulated as well as the originally defined methods and state variables. Fig.6 shows such a property adaptation mechanism. In order to implement this mechanism, we first define the adaptable properties as independent constructs. Of course, newly attached properties are selected superiorly in the case that their names conflicted with names of originally defined properties.

The general syntax form is as follows:

```
ex.)
  !ObjectA property: #PropertyA withVariables:
    'v1 v2'!
    <method name>
    <method body>!
   !!
```

In this description, "PropertyA" is a subclass and is defined as the property to be attached to the class "ObjectA"'. The state variables are followed by "withVariables:". Of course, if state variables are unnecessary, they may be abbreviated. The methods are specified with method names and method bodies in the same form as the method definition of the object. In this case, the method body can refer to state variables in the object definition in addition to state variables in the property definition. When the properties are attached to objects, the field needs to know which properties should be assigned. Namely, we must describe the interrelations between objects and properties. An example of the description form is:

```
ex.)
  !FieldA adaptation!
    ObjectA acquire PropertyA!
    ObjectB as #roleA acquire PropertyB!
    ObjectB acquire PropertyC!!
```

This description illustrates which properties are assigned to objects in the field "FieldA": when instances of the class "ObjectA" participate, the property "PropertyA" has to be attached; and when instances of "ObjectB" do, "PropertyC" has to do so. In the case that instances of "ObjectB" enter into "FieldA" with a particular task "roleA", "PropertyB" is assigned.

We consider the driver problem again. "Driver" is defined as an object, and "US" and "France" are as fields. "US-Driver" and "France-Driver" are defined as the properties to be attached to "Driver", respectively. Individual properties have a method "payYearlyFee" to calculate the tax, which is dependent on the duty rules of individual nations.

```
ex.)
  !Driver property: #USDriver!
    payYearlyFee
      <calculation of tax in US duty rule>!
    !!
    !Driver property: #FranceDriver!
      payYearlyFee
     <calculation of tax in France duty rule>!
    !!
```

Next, we define fields for properties to be attached in entering into "US" and "France".

```
ex.)
  !US adaptation!
  Driver acquire USDriver!!
  !France adaptation!
  Driver acquire FranceDriver!!
```

When instances of "Driver" participate to the field, "USDriver" in S" and "FranceDriver" in "France" are attached, respectively. After this adaptation, when the message "payYearlyFee" is sent to "Driver", the method "payYearlyFee" of "USDriver" is executed in the case that the object is located to "US", and "payYearlyFee" of "FranceDriver" is done similarly in "France". As a result, the taxes in individual nations are correctly calculated.

5 Conclusion

In this paper, we proposed an object-field model by introducing the concept of field newly in order to represent a group of objects, though the traditional object-oriented model is appropriate to model individually distinguished entities and specify the interaction. The features in our object-field model are as follows:

- to support an indirect communication way of broadcasting type by field;
- to look upon a group of objects as an abstract object though a group is modeled as the field of a passive entity;
- to adapt the property of objects to the field under the interrelationship among objects.

Since it is necessary that objects should communicate with many unspecified objects with a view to representing the cooperative interaction among objects, we introduced the field as a passive modeling element and characterized it as a communication media. Also, we addressed an overlay mechanism of object for field in

order to model a group of objects from a different point of view. The concept of field in our model is different from the concept of field(or environment) in Kamui-88 and organizational calculation model with respect to looking upon a group of objects as an abstract object of the active entity in addition to the field of the passive entity. This is very successful to classify objects into different groups by the functionality and model the problem in the hierarchical representation style in the case that we must specify a very large-scale problem. Moreover, objects must act with peculiar properties in addition to the originally defined properties in the case that objects should interact with other objects cooperatively in the field. As for this requirement, we introduced the property adaptation mechanism so as to attach to the roles of objects adaptable to the field. Our modeling approach based on the object and field supports every activity of entities effectively. Now, we have developed a prototype system based on our object-field model, using Digitalk's Smalltalk/V on Macintosh. In order to implement our composite elements such as objects, fields, properties, message communication control mechanism and so on, we constructed several classes on the Smalltalk's class hierarchy, as illustrated in Fig.7.

In our future work, we must investigate the indirect influence mechanism from object to field. Although we could provide the indirect influence mechanism from field to object as a property adaptation mechanism, the inverse mechanism is required with a view to modeling our various phenomena. In introducing self-organization mechanism, we can support an effectual modeling method to design and implement various kinds of cooperative integration systems.

Acknowledgements

We are grateful to Prof.T.FUKUMURA of Chukyo University, Prof.Y.INAGAKI, and Prof.J.TORIWAKI of Nagoya University, and Prof.N.SUGIE of Meijyo University for their perspective remarks, and also wish to thank our research members for their many discussions and cooperations.

References

[1] B.Meyer: *Object-oriented Software Construction*, Prentice Hall (1988).

[2] J.Rumbaugh et al.: *Object-Oriented Modeling and Design*, Prentice Hall (1991).

[3] B.J.Cox and A.J.Novobilski: *Object-Oriented Programming: An Evolutionary Approach*, 2nd ed., Addison-Wesley (1991).

[4] F.Nishio, T.Watanabe and N.Sugie: "Design and Implementation of a Programming Language Based on Objects and fields", *Proc.of TENCON'92*, pp.623-627 (1992).

[5] F.Nishio, T.Watanabe and N.Sugie: "A Programming Language Based on the Concepts of Objects and Fields", *Proc.of OOPSLA'92: Addendum*, pp.201-203.

[6] S.Watanabe, Y.Harada, K.Mitani and E.Miyamoto: "Kamui-88: A Parallel Computation Model with Fields and Events", *Computer Software of Japan*, Vol.6, No.1, pp.41-55 (1989) [in Japanese].

[7] T.Maruichi, M.Ichikawa and M.Tokoro: "An Organizational Model of Computation and Its Application", *trans.on Information Processing of Japan*, Vol.31, No.12, pp.1765-1779 (1990) [in Japanese].

[8] E.Blake and S.Cook: "On Including Part Hierarchies in Object-Oriented Languages, with an Implementation in Smalltalk", *Proc.of ECOOP'87*, pp.41-50 (1987).

[9] N.Craske: "SNOOPS: An Object-Oriented Language Enhancement Supporting Dynamic Program Reconfiguration", *ACM SIGPLAN Notices*, Vol.26, No.10, pp.53-62 (1991).

[10] R.Helm, I.M.Holland and D.Gangopadhyay: "Contracts: Specifying Behavioral Compositions in Object-Oriented Systems", *Proc.of ECOOP/OOPSLA'90*, pp.169-180 (1990).

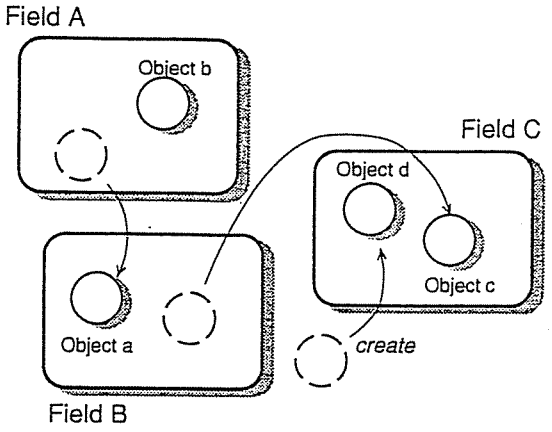

Fig.1 Relationships between objects and fields

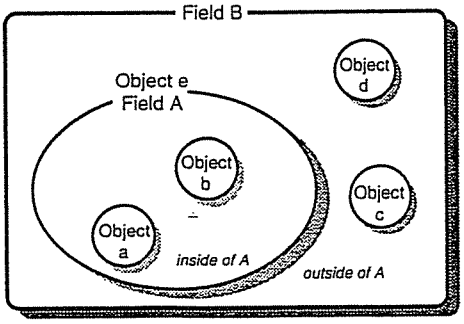

Fig.2 Hierarchical structure between objects and fields

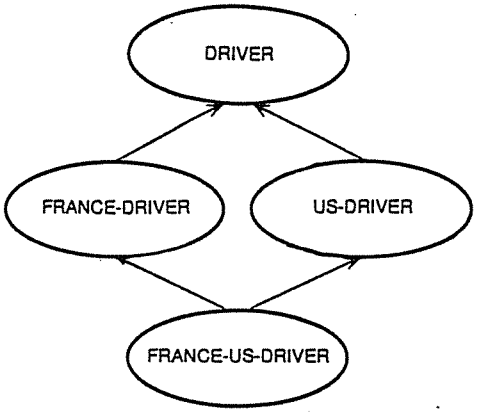

Fig.3 Modeling of driver problem by multi-inheritance

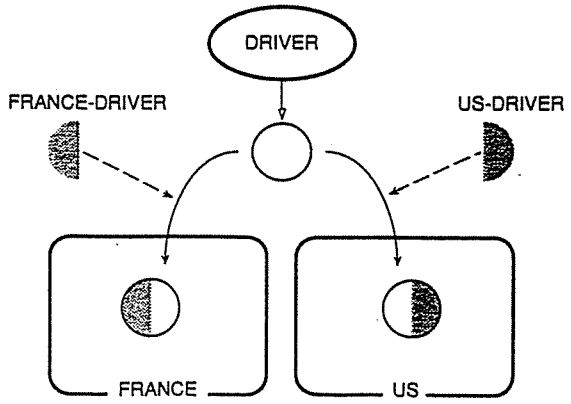

Fig.4 Modeling of driver problem by our object-field model

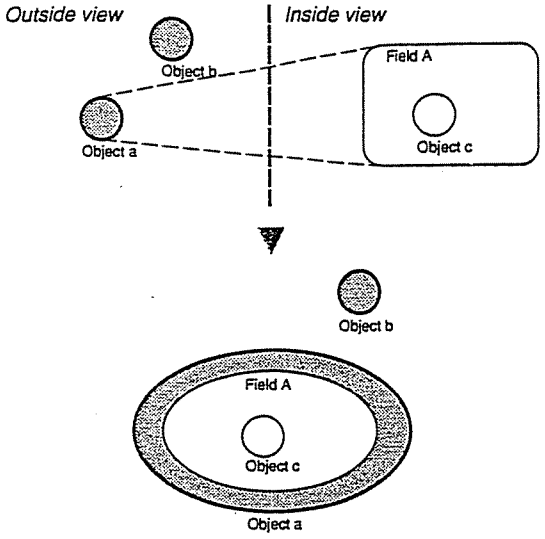

Fig.5 Overlay mechanism between object and field

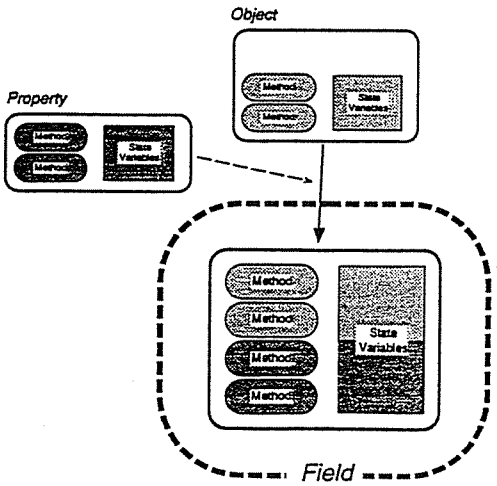

Fig.6 Property adaptation mechanism

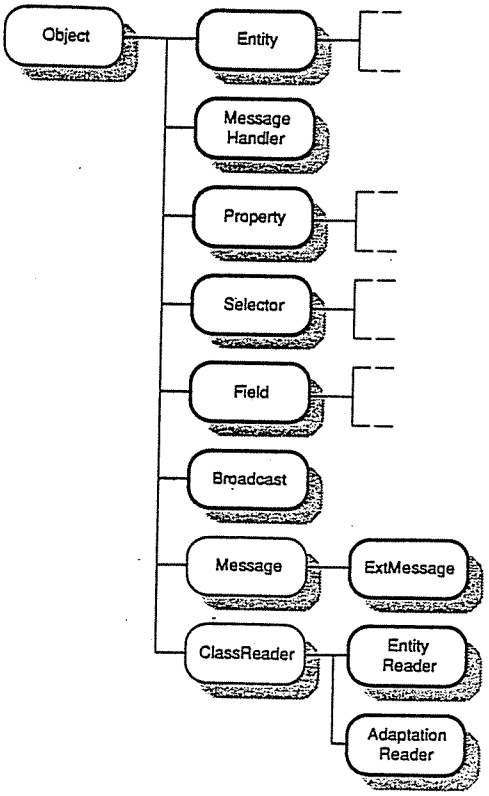

Fig.7 Class hierarchy on implementation