EDITING MODEL BASED ON THE OBJECT-ORIENTED APPROACH

Toyohide Watanabe
Yuuji Yoshida
Teruo Fukumura

Reprinted from PROCEEDINGS OF THE TWELFTH ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, Chicago, IL, October 5-7, 1988

Editing model based on the object-oriented approach

Toyohide WATANABE, Yuuji YOSHIDA and Teruo FUKUMURA

Department of Information Engineering
Faculty of Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464, JAPAN

Chukyo University
101 Tokotate, Kaizu-cho,
Toyota-shi 470-03, JAPAN

***** abstract *****

The editing facility is one of the most basic software products and constitutes direct user interfaces in programming environments. Although various types of editing facilities have been developed or improved until today, they do not always provide the powerful abilities to manipulate different kinds of data uniformly, to share them effectually with the other processing facilities and to construct the functions adaptable to their characteristics. This is partly because the data structures managed by the editing facilities depend on the particular editing data, and partly because the editing facilities are specified on the basis of their application-specific requirements.

Our approach proposes a fundamental framework for the editing facility suitable to the uniform manipulation for various kinds of data, the data sharing among different facilities and the functionality for application-specific processing. The basic principle is to separate the object schema and the data instance from the editing data extractively. This separation method makes it possible to specify various kinds of data under the explicitly structured forms and then manipulate them with the uniform access paths. In this paper, we address mainly our design concept of the editing mechanism.

***** keywords *****

editing facility, editing object, editing model, object schema, data instance, user interface, multi-media/form, integration, functionality, data definition language, relational data model, object-oriented approach

1. INTRODUCTION

The effective management for various kinds of data is regarded as one of a number of very important issues, now that computers are necessarily adaptable to a wide range of applications. Especially, as the basic facility of the computer usage in the office working, the successful functions for multi-media/form data have been strongly required under the subject of the office automation.[5] For example, in the office environment document preparation is one of the fundamental tasks. This task must be not only effective to input, edit and output the original data, but also successful to compose various kinds of documents. The computer-aided tools used to support such a task must provide end-users with excellent man-machine interfaces for effectual office environments.[6]

Various types of editing facilities have been developed today, corresponding to application-specific requirements under individual computer systems. However, they provide almost similar functions for string manipulation though their command syntaxes, their user interactive interfaces and their application-specific features are more or less different. The difference is mainly derived from the fact that the system configurations for the editing mechanism depend too strongly on individual applications because many editing facilities have been designed with the application-specific requirements. If the system architecture of the editing mechanism could be designed uniformly so as to be independent of individual application-specific editing structures, we can develop a generic system with the editing mechanism adaptable to various kinds of editing data. Such an editing mechanism may be implemented systematically if the editing mechanism could be managed so not as to include individual application-specific data forms internally.

In this paper, we investigate an architectural framework for the editing mechanism, and as a result propose an experimental editing model. Our editing model has an object-oriented feature on the basis of the framework of the relational data model.[9] The description form is based on the data definition language and extended with the procedure-specification abilities with respect to their peculiar functions attended to individual editing objects.

2. EDITING MECHANISM AND EDITING OBJECT

A number of editors, which are available today in a wide range, have their own application-specific features. The editing objects which the editing facility must desirably manipulate as a composite module in the information system are classified from simple data sequences such as numerical data, to complex structures such as documents, or to syntactic constructions such as sentences and programs. Unfortunately, our current environment does not allow us to manipulate these different editing objects uniformly by using the same facility, but forces us to select appropriate tools owing to their editing characteristics.

For the purpose of making the computer utilization environment functional, the editing facility must to be matchable to many user requirements: these requirements may be satisfied, in many cases, by the concepts of multi-media/form data manipulation, cooperative integration and adaptable functionality, in addition to effectual operativeness which has been commonly discussed as the most important concept for the user interface.[3,4] Much recent research pays attention mainly to the design or control of the operational windows from a viewpoint of the subject of the man-machine interface: the typical functions are the multi-

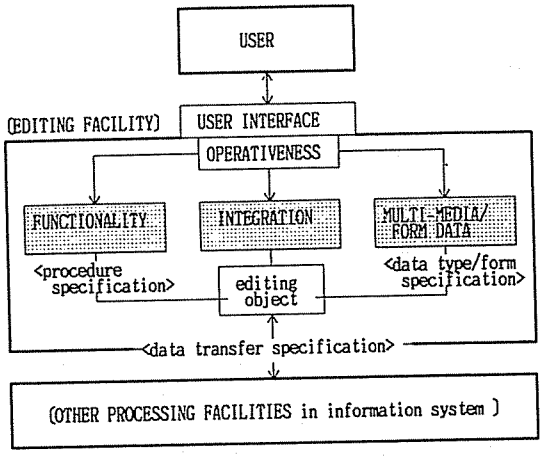

Fig.1 Editing environment

window control interface, and the icon picking-up or mouse pointing operation.[7] However, an approach based on the operativeness can not always build cooperative man-machine interfaces. The excellent man-machine interface will be constructed on the basis of the design method combined with the multi-media/form, the integration and the functionality. These concepts and the compositive design have been disregarded in the traditional approach because the editing facility does not have its own system architecture, but has been implemented as an application-specific system or preprocessing tool for some application system.

Concerning the issue of multi-media/form manipulation, it is required that the editing facility can accept every editing object without being dependent on individual data of sorts. For the purpose of the realization of this concept, the editing mechanism necessarily controls conceptual editing objects by excluding application-specific structures attended inherently to individual editing data. For the issue of cooperative integration, it is required that the editing objects are shared among the processing facilities in the information system. This requirement is to make the efficiency for application processing high and make the procedural operation simple. Finally, as for the issue of adaptable functionality, the editing facility is desirable to perform data-specific functions owing to the characteristics of individual editing objects: that is, the sorting function for some ordered data or the syntax checker for some programs. This ability must be specified by the properties of each editing object under the uniform framework.

This conceptual view in the editing facility is shown in Fig.1. It is not so difficult to introduce these concepts into the architectural design. This mechanism will be solved by looking on the characteristics of editing objects such as data structures, media types and syntactic forms, as the logical specification schema: generally we can consider that the object characteristics are the object schemata.

3. EDITING MODEL

From an architectural point of view, our framework of the editing facility is based on the object-oriented

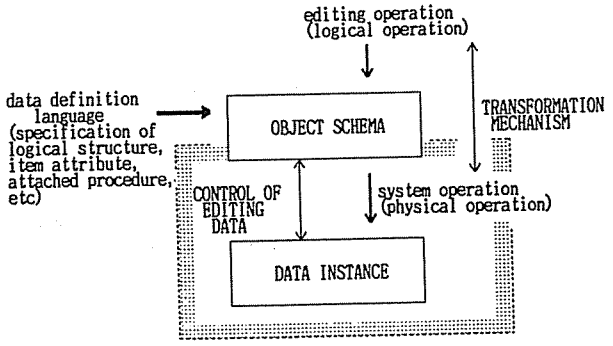

Fig.2 Data manipulation through object schema

approach as the basic control mechanism of editing objects, and is also derived from the data definition feature in the relational data model as the specification method. As the class-instance relationship in the object-oriented model,[9] editing objects are composed of the object schemata, which specify the logical structure class of editing objects, and the data instances, which are unstructured real data to be adapted to each logical structure class. In comparison with the object-oriented programming paradigm, our editing objects are characterized by more static properties because they mainly consist of data fractions manipulated by editing operations. Namely, editing objects in themselves are not so autonomous as the objects in the object-oriented programming in point that each object does not exchange messages. Of course, it is not necessary for editing objectsto do so. At least, the traditional editors manipulate only the data fractions, and the peculiar properties attended to the editing data are performed directly by the editing operations.

While, concerning the description of the object schema, our data definition language is similar to that in the relational data model. Each data item is specified by the column descriptors with some attribute parameters. In addition, our definition language can accompany some functions as the attached procedure-like forms. The description for such an object schema corresponds to the table description in the relational data model. Our editing mechanism based on both the relational database framework and the object-oriented approach is called the editing model, here. This editing model is shown in Fig.2. The object-oriented feature is mainly useful to data composition and editing processes, and the relational database framework is fundamentally adaptable to data access and editing processes.

Individual editing objects are not only controlled independently or flexibly, but also can construct the other editing objects as composite elements. The independence among editing objects is naturally derived from the characteristics of the relational data model, on which our editing model is based: a database may be composed of one or more tables. The flexibility for editing objects depends on the fact that the descriptive schemata for the logical structures are operational objects by themselves.

As for the flexibility, several data views for one editing object can be supported as well as the schema-subschema

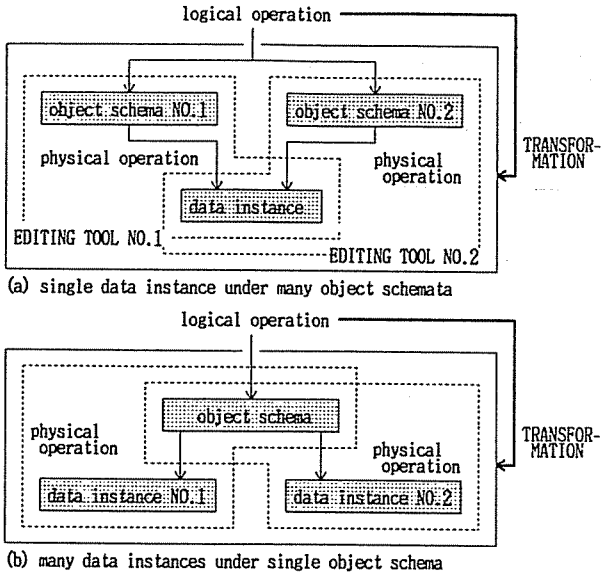

Fig.3 Flexibility in editing model

relationship in the relational data model. Of course, our framework is more powerful than the schema-subschema relationship. Such relationships are illustrated in Fig.3: one is to manipulate a data instance through many suitable object schemata; and another is to manipulate many data instances through only one object schema. This mechanism makes it possible to look upon one editing structure as another structure if another object schema which is different from the original schema could apply suitably to a data instance. In the schema-subschema relationship, every subschema must be always derived from the underlying schema, while in our framework such a constraint is never imposed. This flexibility is a solution for the issue of multi-media/ form data manipulation.

On the other hand, with respect to the independence our framework can provide a very successful solution for the issue of cooperative integration. It is better to exclude the physical information for the control or management of the data organizations from the editing structures of individual applications when various kinds of data must be shared among several processing facilities cooperatively. The data sharing method, which utilizes only the logical information: object schema, is suitable to many different applications uniformly. In our model, the data sharing mechanism can be implemented easily by means of assigning an appropriate mapping function to the mutual application facilities. For example, we can consider a relationship between the document preparation facility and the editing facility. Fig.4 shows 2 types of relationships: the type (a) is a traditional approach constructed between the editor and the formatter; and the type (b) is our approach based on the correspondence of individual object schemata. In the editor and formatter, the formatter makes up documents with text', generated from the editor. The formatter depends on the editor because the formatter can interpret only the data manipulated by the editor: the source data must always contain the form control information. On the other hand, in our approach the relationship between the editing facility and the document preparation facility is mutually equivalent. The correspondence of their logical information(e.g. names in our approach) is completely assured.

4. DESCRIPTION OF EDITING OBJECT

Our editing objects are structure-independent, composite and operational entities. Every editing object does not associate with its own particular structure, but is changeable to arbitrarily structured data. Namely, editing objects without any particular forms can be composed as structured data specified by the data definition language.

In our framework, a uniform manipulation of editing objects is an important issue even if the individual editing components could associate with different types of attributes. Our editing components are divided into a text object and an image object with respect to the attribute class. Furthermore, the text objects of byte-

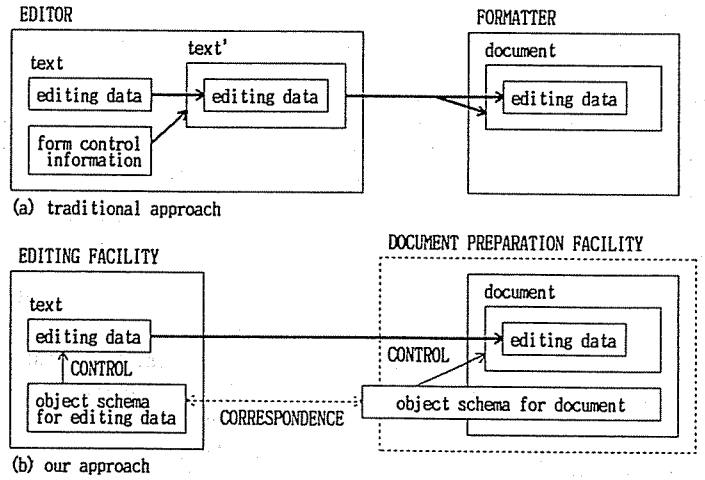

Fig.4 Relationships between editing facility and document preparation facility

```
TITLE: Data Integration in Distributed Databases
AUTHOR: S.M.Deen, R.R.Amin & M.C.Tayler
AFFILIATION: PRECI Project, Department of Computing Science, University of Aberdeen.
ABSTRACT: Data integration in a distributed database refers to the production of uni-
          ..............
ARTICLE_STATEMENT: 1. Introduction
      Data integration refers to the creation of an integrated view over apparently
          ...........
```

(a) article data

DOC.NO.	12936
TITLE	Natural language interfaces to computer system: an experimental
AUTHOR	Guida,G.
AUTHOR AT	Politecnico di Milano, Istituto di Elettrotecnica ed Elettronica
TAKEN-FROM	Alta Freq.(Italy)
CODEN	Alfraj
VOL.NO.	Vol.47, No.9
PAGE	668-74
PUB.DATE	Sept. 1978
PUBLISHED BY	Springer-Verlag
PUB.DATE	1978

(b) catalog data

```
      INTEGER FUNCTION VALUE(M)
      M0=M
      M1=0
      L=1
   10 IF (M0.EQ.0) GOTO 20
      M1=M1+MOD(M0,2)*L
      M0=M0/2
       ..........
      IF (M0.NE.0) GOTO 30
      VALUE=M1
      RETURN
      END
```

(c) program data

NO	NAME	AMOUNT	ADDRESS	
			CITY	COUNTRY
S1	Smith	20	London	U.K.
S2	Jones	10	Paris	France
..	.....	...	.....	.....
Sn	Adams	30	San Francisco	U.S.A.

(d) table data

Fig.5 Examples of individual editing objects

oriented data consist of several sub-objects: an article object, a catalog object and a program object. While, the image objects of bit-oriented data contain sub-objects such as a table object, a simple-graph object, a complex-graph object and a pixel object, according to the functions associated to the object generation process.

Such a classfication is introduced in order to make the manipulation of each entity easy. For example, in the image objects their own functions become available in the object generation and so on by means of the class attributes: image-readers are usable to pixel objects; and business graphs such as circles, histograms, etc, must be created easily for the simple-graph objects. Moreover, the syntax-checker as an attached procedure, which we will mention later, must operate meaningful programs. Of course, this descrimination is not necessarily essential for our processing if the advanced techniques were developed in the future. Today, in the case of performing our data manipulation effectually, such a classification is successful.

4.1 EDITING OBJECT

Now, we specify the object schemata for individual editing objects concretely by using the data definition language. The descriptions for image objects, except for the table object, are only to declare the size, and the functions such as the graph depiction, the image input, the tabular arrangement, etc, may be specified as optional procedures of types. The descriptions for a simple-graph object, a complex-graph object and a pixel object are as follows:

(ex.1)

```
    structure IMAGE1: simple;
      term IMA: bit(1000,1000);
     end;
or
    structure IMAGE2: complex;
      term IMA: bit(1000,500);
     end;
or
    structure IMAGE3: pixel;
      term IMA: bit(256,512);
     end;.
```

These descriptions are distinguished by the optional indicators "simple", "complex" and "pixel", and these indicators make it possible to manipulate different objects functionally in the generation process.

Next, we explain the descriptions for an article object, a catalog object, a program object and a table object. The typical examples for these objects are shown in Fig.5.

(a) article object: This object consists of several data

items: for example, a paper is composed of a title, authors, an affiliation, an abstract, keywords and an article statement, as illustrated in Fig.5(a). The description is as follows:

(ex.2)

```
structure ARTICLE: article;
  term TI: char(50)     'TITLE:';
  term AU: char(100)    'AUTHOR:';
  term AF: char(200)    'AFFILIATION:';
  term AB: char(1000)   'ABSTRACT:';
  term KW: char(50)     'KEYWORDS:';
  term AS: char(50000)  'ARTICLE STATEMENT:';
 end;.
```

The indicator "article" may be abbreviated as the default value of the description.

(b) catalog object: This object is similar to the article object, except for the manipulation of multiple records. The catalog object shown in Fig.5(b) is specified by the next description:

(ex.3)

```
structure CATALOG: catalog(100);
  term DN    : integer   'DOC.NO.';
  term TI    : char(50)  'TITLE';
  term AU(5) : char(20)  'AUTHOR';
  term AF    : char(100) 'AUTHOR AT';
  term TF    : char(100) 'TAKEN-FROM';
  term CD    : char(20)  'CODEN';
  term VO    : char(20)  'VOL.NO.';
  term PG    : char(5)   'NO.OF PAGE';
  term PA    : char(10)  'PAGE';
  term PB    : char(20)  'PUBLISHED BY';
  term PD    : char(15)  'PUB.DATE';
 end;.
```

In comparison with the article object, we can not observe the difference besides the indicators "article" and "catalog(100)". The parameter "100" in "catalog" represents that the maximum number of entries is 100. If this parameter is abbreviated, infinite entries are assumed. Moreover, "catalog(1)" is equal to "article" in point of the number of entries.

c) program object: This consists of only one data item in many cases. For example, the description for the FORTRAN program shown in Fig.5(c) is as follows:

(ex.4)

```
structure PROGRAM: program;
  term PB(100): char(80);
 end;
```

or

```
structure PROGRAM: program;
  term PB: char(8000);
 end;.
```

Moreover, we can assign to this description more information in order to manage the program object effectively. The next description is a typical example:

(ex.5)

```
structure PROGRAM: program(FORTRAN);
  procedure syntax-check;
  term 'FORTRAN PROGRAM';
  term PB(100): char(80);
 end;.
```

In this example, 3 descriptive points are newly introduced: the parameter "FORTRAN" in the indicator "program", the descriptor "procedure syntax-check;" and the descriptor "term 'FORTRAN PROGRAM';". "FORTRAN" indicates that this description is adaptable to the FORTRAN program. "procedure syntax-check;" declares that the procedure "syntax-check" must be used to handle strings in the data item PB. Therefore, the attached procedure "syntax-check" is successful for this FORTRAN program. "term 'FORTRAN PROGRAM';" points out that this term displays the message "FORTRAN PROGRAM". The syntax of "term" must be interpreted so that the general form is

(ex.6)

```
term <data item name> : <data type/length> <message>;.
```

The first and second parameters <data item name> and <data type/length> are abbreviated. Namely, the syntax is

(ex.7)

```
term <message>;.
```

(d) table object: This is basically similar to the catalog object in point of the definition. The basic form in the table shown in Fig.5(d) is defined as follows:

(ex.8)

```
structure TABLE: table(5);
  term NO: char(2)  'NO';
  term NA: char(20) 'NAME';
  term AT: integer  'AMOUNT';
  term AD: record   'ADDRESS';
    term CT: char(20) 'CITY';
    term CN: char(10) 'COUNTRY';
   end;
 end;.
```

In this description, lines around each value are not explicitly defined. Usually, lines are automatically specified by the indicator "table" in making up the table form practically. Moreover, we can observe that a hierarchical structure is defined between the data items AD and CT/CN. In composing tables, various arithmetic processing is often required: percentage of some columns, total amounts, etc. Such requirements are satisfied with the next modified description:

(ex.9)

```
structure TABLE: table(5);
  term NO : char(2)  'NO';
  term NA : char(20) 'NAME';
  term AT : integer  'AMOUNT' -SUM;
  term ATR: def AT            -VALUE(AT/SUM(AT)*100);
  term AD : record   'ADDRESS';
    term CT: char(20) 'CITY';
    term CN: char(10) 'COUNTRY';
   end;
 end;.
```

In the data item AT, "-SUM" indicates that the total value in this column is calculated and inserted into the last added entry. While, the data item ATR is introduced in order to store the percentage value of AT: calculating the percentage values is defined; and storing the values into AT is specified by "def AT".

4.2 EDITING OBJECT FOR MULTI-MEDIA/FORM

The main descriptors to define the object schemata for various kinds of editing objects were outlined. Of course, many supplementary descriptors are assumed, furthermore. Our editing model has a fundamental framework based on the relational data model. Therefore, various types of data

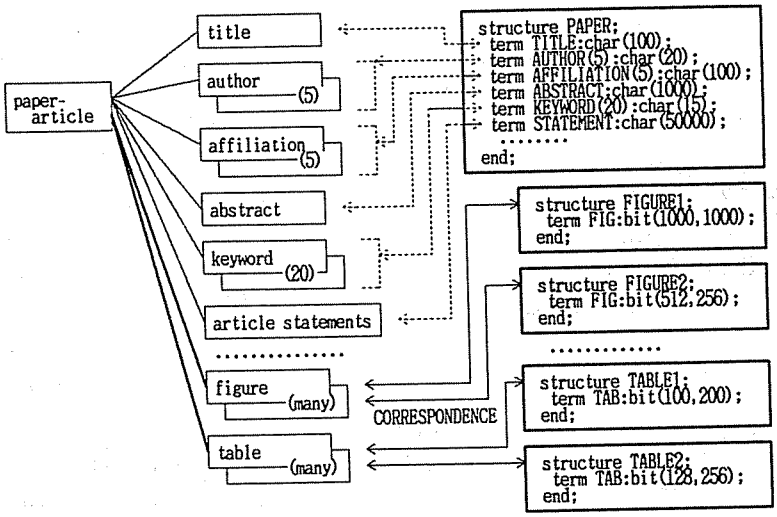

Fig.6 Relationships between object schemata and editing components

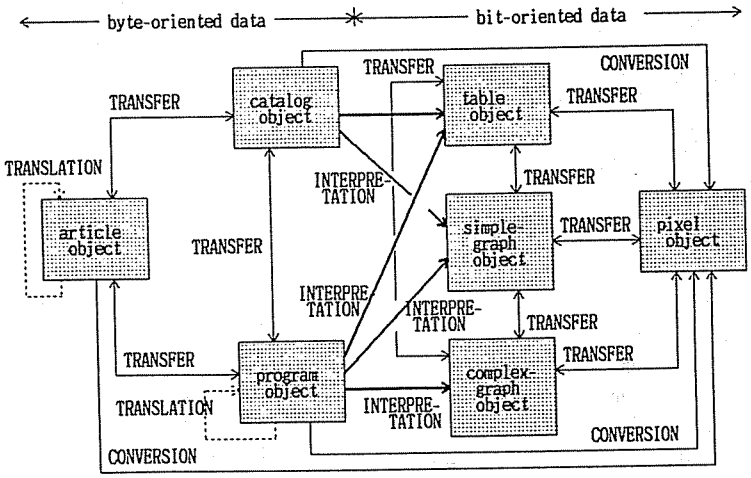

Fig.7 Attribute transition among editing objects

such as the image, the character and so on can be represented as the column attributes in the composite elements. The data attributes attended to column properties specify various types of data. While, various kinds of editing structures can also be constructed easily by the data definition language. We consider a paper-article as an example. This editing object consists of several editing components of different data attributes. The description is illustrated in Fig.6. Our data definition language does not only specify the data structure like the traditional data definition language(e.g. in databases), as we have already understood in the above example, but also provides the ability to define particular functions attended to each editing object. Thus, we can consider that our editing model is based on the object-oriented approach. However, in our model the object schema is not always constrained so as to attend inherently with the particular editing object, but the editing object can be conveniently applied by some object schema. If necessary and possible, different data instances can be composed by applying another object schema. This feature makes it possible to manipulate various kinds of data forms, or different types of data attributes.

Next, we investigate the attribute transition among editing objects. Fig.7 shows the transition graph of object attributes. 3 types of the attribute transitions are really meaningful: "transfer"; "conversion"; and "interpretation". The transfer is the attribute transition in the same class of editing objects: the text object and the image object. The conversion and interpretation are the attribute transitions from the text object to the image object. In the conversion, the transition is performed simply without helps of any procedures. While, in the interpretation for the program object, the transition must be performed interpretively under the control of the suitable translators(or interpreters). The transition from the catalog object to the table object is a class exchange, and makes up a tabular form with the addition of lines. Additionally, the closed transition("translation" in Fig.7)

within the article objects or the program objects does not change the object types, but are the exchanges of the values. These processes will be executed by more advanced processing abilities(e.g. machine translation, program conversion).

5. MANIPULATION OF EDITING OBJECT

Our approach based on the object schema and the data instance is very applicable to the issue of integration and functionality. Namely, the editing objects in our frame-work are not controlled by application-specific constraints, and are also independent of application-specific editing structures. Therefore, different facilities in the information system can share their processing data mutually if the processing protocol for their object schemata were systematically established, because the object schemata do do not contain the physically controlled information.

5.1 INTERFACE OF EDITING FACILITY

To make the issue of integration and functionality clear we will research firstly the practical characteristics of the facility-facility interfaces, derived from the current systems. As for cooperative relationships(integration) between the editing facility and the other facilities:

(1) program interpretation: interpreter, compiler;
(2) machine translation : machine translation system;
(3) document preparation : formatter.

As for functional relationships(functionality) associated together with the editing facility:

(4) program-syntax generation : syntax-oriented editor, LISP structured editor;
(5) KANA-KANJI/ROMAN-KANJI conversion: Japanese word processor.

Furthermore, application-specific editing functions, whose abilities compose an application system as one of the functions, are observed in many cases. Many application systems adopt the calling means to attach to the existing editing tools on the basis of the linkage protocol. We can classify relationships between the editing facility and the other facilities into 4 types, as illustrated in Fig.8.

The type (a) corresponds to the above (1) and (2), the type (b) does to (3) and the type (c) does to (4) and (5), respectively. The type (d) is observed in many application systems. In (d), 2 facilities keep an equivalent relationship mutually, concerning their program constructions though the calling sequence is controlled under a master-slave relationship with respect to their actual execution. Therefore, (d) can be in advance transformed into (a) from a viewpoint of the facility-facility interface. In (c), the actual relationship is successful only in very constrained areas: data input process. For example, the KANA-KANJI conversion function is useful for only an input operation with the exclusion of the other operations. This is a very powerful ability in Japanese processing, in which it is difficult to put in KANJI characters directly from keyboards. Moreover, we can find out the data entry system as a construction of (c). This cooperative relationship has already been designed in our framework: for example, see (ex.7). At least, (c) provides critical basements so that

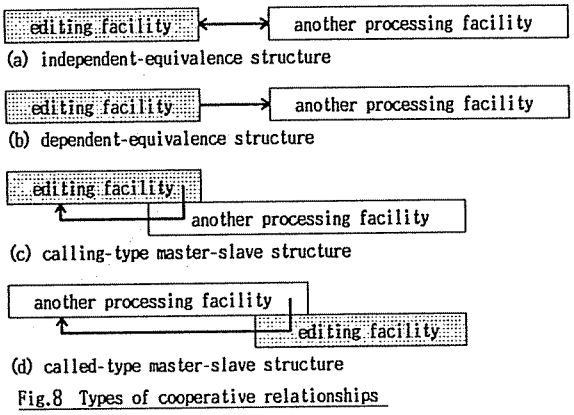

Fig.8 Types of cooperative relationships

the editing facility is adaptable to various kinds of data, or various types of applications. While, in (b) another processing facility can accept only the editing data, which the particular editing facility generates, though they are designed so as to provide individual functions. Namely, they always keep a master-slave relationship in spite of being their own independent and characteristic functions. Such a tightly-coupled organization is neither effective nor flexible with respect to cooperative relationships among various types of facilities.

5.2 INTEGRATION AND FUNCTIONALITY

From the previous discussion, we must have an adaptable solution to construct 2 different types of relationships uniformly:

(1) independent-equivalence structure;
(2) calling-type master-slave structure.

It is better that 2 types of relationships are approached as different issues. The former is an issue about an external interface and is related to the integration concept. While, the latter is an issue about an internal organization and is related to the functionality concept. If the editing facility did not introduce its own particular data structure, it is not difficult to facilitate our issues. Our framework satisfies at least such a requirement.

(1) INTEGRATION INTERFACE:

The former issue can be solved with respect to the data compatibility among several facilities. For example, the cooperative relationships among the processing facilities such as database managements, document editings, document preparations and so on(as typical facilities in the information system) can be established if the correspond-ence of the individual object schemata, associated to the corresponding facilities, is kept consistently.[1,2] Fig.9 shows relationships about their object schemata. Although the facilities for document preparation, database management, window management and so on are not yet discussed in this paper, it is so clear that they can associate their own object schemata. Of course, it is necessary to match for their basic attributes in each column specification in order to make the mapping functions successful. In the database management, the framework is almost similar to the mechanism in our editing model, and it is easy for us to understand

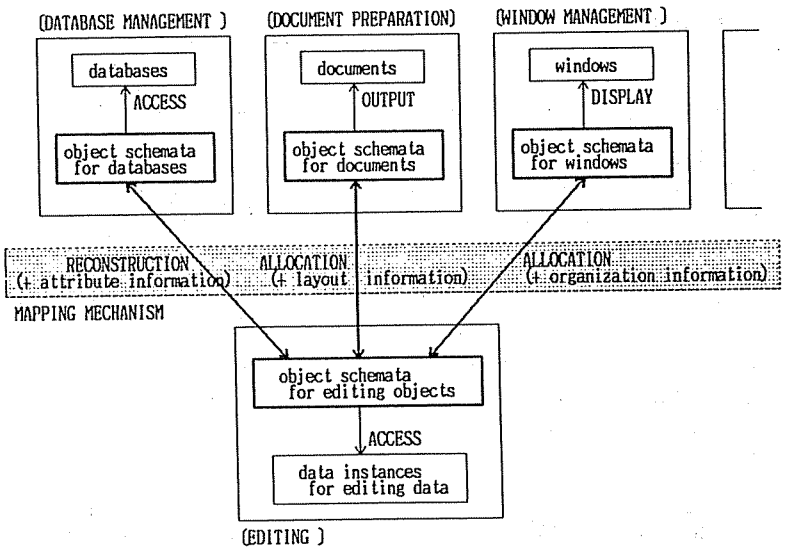

Fig.9 Mapping mechanism among object schemata

object schema: the object schema is the database schema in itself though it must include the characteristic information for the database organization, in addition to the object schema of the editing object. While, the object schemata for documents and windows contain the layout or positioning information for the data indicated by the corresponding names in order to design the physical configurations.

(2) FUNCTIONALITY INTERFACE:

The latter issue is not always a difficult problem to be solved in our framework. In this case, the attached procedure mechanism is successful though a special case has been already shown in (ex.7). When some attached procedure is available to perform particular processing at once, besides the ordinary editing, it is possible to assign the procedure to one or more data items as the parameters. The procedure to be attached is the built-in system routine or the user-defined procedure, which must be written by a tiny system-provided programming tool, and then be registered into the editing facility in advance.

6. CONCLUSION

It is desirable for the editing facility to manipulate various kinds of data uniformly and share the editing data cooperatively among the processing facilities.[1-4] In current information system environments, the issues about multi-media/form data manipulation, cooperative integration and adaptable functionality are important in addition to the concept of easy operativeness. We think that the advanced user interface in the information processing environment will be systematically developed by means of effective mechanisms,[8] based on these concepts.

Our approach is one of architectural frameworks for such candidate information systems, and assesses the most fundamental issue from a viewpoint of the design principle. Our framework of the editing facility, discussed in this paper, may not always provide users with easy operations directly, but can address the effectiveness and powerfulness for user interfaces. The description for the object schema may be not easy for end-users, but the improvement of menu-window mechanism or the addition of parameterized-interaction function makes the operation simple because the mappig mechanism into the interactive display window from the object schema is conceptually refined(e.g. through the window management in Fig.9). The successful mechanism for multi-media/form, integration and functionality is the fundamental requirement for the advanced man-machine interface, and the sufficient method for operativeness will be designed with respect to these 3 concepts.

Acknowledgements --- We are grateful to Prof.Y.INAGAKI and Prof.J.TORIWAKI, Nagoya Univ., Prof.M.NAGAO, Prof.H.HAGIWARA and Prof.S.HOSHINO, Kyoto Univ., for their respective remarks, and wish to thank Mr.T.OGASAWARA for his eager cooperation. Also, we would like to thank the referees for their constructive review.

References

1) T.WATANABE & I.OKETANI:"Functional Design of Cooperatively Integrated Information System", P.36, Technical Report of Data Processing Center in Kyoto Univ., A-16(1986).
2) T.WATANABE:"Architecture of Integrated Office Information System: a cooperative integration method for various data processing facilities", Proc.of the 6th PCCC, pp.320-327 (1987).
3) L.BOLC & M.JARKE(ed.):"Cooperative Interfaces to Information Systems", on Topics in Information Systems, P.328, Springer-Verlag(1986).
4) P.DEGANO & E.SANDEWALL(ed.): "Integrated Interactive Computing Systems", P.374, North-Holland, Amsterdam(1983).
5) D.TSICHRITZIS(ed.): "Office Automation", on Topics in Information Systems, P.441, Springer-Verlag(1985).
6) M.M.ZLOOF: "Office-By-Example: A Business Language that Unifies Data and Word Processing and Electronic Mail", IBM system journal, Vol.21, No.3, pp.272-304(1982).
7) W.TEITELMAN: "A Tour through Ceder", IEEE trans.on Software Engineering, Vol.SE-11, No.3, pp.285-302(1985).
8) S.W.DRAPER & D.A.NORMAN: "Software Engineering for User Interfaces", IEEE trans.on Software Engineering, Vol.SE-11, No.3, pp.252-258(1985).
9) B.J.COX:"Object-Oriented Programming", Addison-Wesley(1986).