数理解析研究所講究録
第655巻 1988年 209-223

Editing mechanism for the uniform manipulation of various kinds of data

渡辺豊英、小笠原達男、吉田雄二、福村晃夫

Toyohide WATANABE, Tatsuo OGASAWARA, Yuuji YOSHIDA and Teruo FUKUMURA

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

***** Abstract *****

The editing facility is one of the most fundamental computer software tools, and constructs direct user interfaces in the programming environments. Especially, the functional roles of the editing facility against the other processing facilities are very important with a view to realizing the effective user interfaces, now that it is strongly required that the issues about the software integration and the manipulation of multi-media/form data must be investigated successfully. In this paper, we address a conceptual framework of the editing mechanism adaptable to the manipulation of multi-media /form editing data in cooperatively integrated information systems. Our discussion characterizes the basic architecture of the editing facility in the more progressive information systems.

***** Keywords *****

Editing facility, integration, logical structure, multi-media, multi-form, definition language, relational database, editing object, object attribute, user interface, operativeness-

1. INTRODUCTION

Today, the effective management of various kinds of data has been a very important issue because of extensive usages of computer abilities in a wide range of fields. Especially, the computer utilization technology in the office working must provide the means to perform the effective information processing/communication under the subject of the office automation.[5, 6] For example, in the office environments, the document preparation is one of the fundamental tasks. The task is composed of several procedures such as updating, processing, filing, retrieving and so on, and also these individual procedures must be systematized mutually and cooperatively. Therefore, the computer-aided tool to make up documents must be always implemented so as to satisfy the requirements from end-users.[8]

Editing facility is the most basic computer-aided tool and supplies usable functions for every computer processing.[7] Various kinds of editing facilities have been developed now, but they only provide almost similar functions for the string manipulation though their command syntaxes and their application-oriented features are more or less different. The difference is mainly derived from the fact that the editing structures depend too strongly on the individual applications because many editing facilities have been developed with the application-specific requirements. If the editing structure could be designed abstractly so as to distinguish the application-oriented structure in the editing facility, we can implement practically a generic editing mechanism adaptable to the manipulation of various kinds of editing data. From a viewpoint of this discussion, our approach is primarily based on a separation method which extracts abstractly the logical structure and the physical structure from every editing structure.

In this paper, we propose our experimental method to manipulate various kinds of editing data on the basis of descriptions for individual logical structures. We address that a uniform data manipulation becomes possible by means of such a method. Our model, which is composed of the logical structure and the physical structure, is conceptually similar to the data model in the database management systems as the storage-oriented representation structure. However, our model, which is designed as the manipulation-oriented representation structure, is different from the database model from viewpoints of the data representation, the data processing and the data sharing.

2. MODEL OF EDITING MECHANISM

The conventional editing tools introduce more or less their application-specific editing structures, and provide individual characteristic functions in point of the command names, the command syntaxes, the command parameters, the response messages and so on. However, the functional abilities are almost similar without regard to individual application-specific

editing functions. If the editing structure could be designed as an application-independent form, we can construct a more powerful editing tool adaptable to the manipulation of various kinds of editing data.

Our framework is based on the architectural concept with respect to the logical structure and the physical structure, which are distinguihed from the ordinary editing structure. The logical structure is an abstracted editing structure, and is independent of particular data organizations in individual editing tools. On the other hand, the physical structure is a facility-dependent data structure. The former is specified conceptually by users, while the latter is practically managed by the system functions. In Fig.1, our editing model, based on the separation mechanism between the logical structure and the physical structure, is illustrated conceptually. User's operations are transformed into the system functions through the data definition information.

The similar approach is observed in the framework of the traditional database management systems. Our specification method for the logical structure is almost similar to the schema description form in the relational database model, except for several editing-specific features. The data definition informations, associated with individual editing data, take part in the control of the data manipulation as well as the table handling in the relational database. The framework, in which a database may be composed of one or more tables, is very appropriate to our editing data organization.

Individual editing data are not only controlled independently or flexibly, but also can construct the other editing data as compositive elements. The independence among editing data is naturally derived from the characteristics of the relational database model, on which our editing model is based. The flexibility of editing data depends on the concept that the descriptive procedure for the logical structure can become an operational object by itself.

Concerned to the flexibility, we can get user views for editing structures, corresponding to the schema-subschema relationship in the relational database model. Of course, our framework is more powerful than the schema-subschema relationship. For example, we can illustrate such relationships in Fig.2. In Fig.2, 2 cases are shown: one is to manipulate an editing data through many suitable definition informations; and another is to manipulate many editing data through only one definition information. This mechanism makes it possible to look upon one editing structure as another structure if another logical structure which is different from the original structure could apply suitably to the editing data. In the schema-subschema relationship, every subschema must be always derived from the underlying schema, while in our framework such a constraint is not necessary.

While, with respect to the independence our framework can provide a very successful solution for the issue of the software integration. It is better to exclude the physical

information for the control or management of the data organization from the data 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, is applicable to many different processings uniformly.[1, 2] If the relational database model were replaced by the hirearchical database model(or the network database model) as our basic model, it is difficult to control each compositive editing data successfully. The hierarchical model is poor in case of sharing data among the other processing facilities though it may be appropriate to the representation and management of the editing data, associated inherently with the hierarchical property. 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 and the editing. Fig.3 shows 2 types of relationships: the type (a) is a traditional approach between the editor and the formatter; and the type (b) is our approach based on the correspondence of individual logical structures. In the editor and the formatter, the formatter makes up the documents with text', generated from the editor, as a source data. 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 equivalent mutually as whole. The correspondence of their logical structures(e.g. names in our approach) is completely assured.

3. EDITING OBJECT

Our editing data are structure-independent, compositive and operational objects. Every editing data does not associate with its own particular structure, but is changeable to arbitrarily structured data. Namely, the editing data without any particular forms can be composed as the formed data described by the data definition language, and then manipulated in relation to the user's specified structure.

In our framework, a uniform manipulation of editing data is an important issue even if the individual editing components associated with different types of attributes. Our editing components are divided into the text object and the image object with respect to the attribute class. Furthermore, the text objects of byte-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 associated functions in the object generation process. These objects are shown in Fig.4. Each object has the following characteristics, respectively.

1) article object : This is alphanumeric data which conventional text editors manipulate. This object such as a paper-article, a report and a letter is usually composed of only 1 occurrence of each data item.

2) catalog object : This is a set of records such as the library catalog.

3) program object : This is almost similar to the article object in the data organization except that the data sequence is restrained syntactically so as to be easily interpreted by some program translators like compilers. If natural language processings could have researched successfully at all, the difference between article objects and program objects is not necessary.

4) table object : This is 2 dimensional data associated with a tabular form. This object is usually not distinguished from the other image objects with each other, because this type is useful only in the object generation process, but not effectual in the patchedwork process.

5) simple-graph object : This is a business graph whose elements are circles, lines, histograms and so on. This object is usually not distinguished from the other image objects with one another, too.

6) complex-graph object: This is composed functionally by the graphic subroutines. This object is also not distinguished from the other image objects mutually.

7) pixel object : This is 2 dimensional image data. The other image objects are also manipulated as the pixel object since each image object is not distinct in our editing facility explicitly.

4. DESCRIPTION OF EDITING OBJECT

Now, we specify the logical structures of individual editing objects concretely by the data definition language. The data definitions 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 kinds of optional procedures. For example, the data 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 indicator "simple", "complex" or "pixel", and these indicators make it possible to manipulate different objects functionally in the generation process.

Next, we explain the data definitions 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 such as 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 data definition.

(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" concerned to 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" of the indicator "program", the descriptor "procedure syntax-check;" and the descriptor "term 'FORTRAN PROGRAM';". "FORTRAN" represents 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 the FORTRAN program. "term 'FORTRAN PROGRAM';" points out that this term displays the message "FORTRAN PROGRAM". The syntax of "term" must be interpreted so that in the general form

(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. For example, 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 specifies by the indicator "table" in making up the table form practically. Moreover, we observe that the hierarchical structure is defined between the data items AD and CT/CN. In composing tables various arithmetic processings are often required: percentage of some columns, total amounts, etc. Such requirements are satisfied with the next modified description:

(ex.9)

```
structure TABLE: table;
  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: using the value of AT and calculating the percentage value are defined and then storing the value into the same column AT is specified by "def AT".

The main descriptors to define the logical structures for various kinds of editing data were outlined. Of course, many supplementary descriptors are assumed furthermore. 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 examples, but also provides the abilities to define particular functions attended to each data object. Thus, we can consider that our model might be based on the object-oriented approach.[9] However, in our model the logical structure is not always constrained so as to attend inherently to the particular editing object, but editing data can be conveniently applied by some logical structures. If necessary and possible, users can manipulate the stored editing data(as unstructured data) through another logical structure. This feature makes it possible to perform the attribute transition among editing objects.

5. MANIPULATION OF EDITING OBJECT

In our framework, the data manipulations are always performed through the logical structures adaptable to individual editing data. Basically, these logical operations must work uniformly without depending on the characteristics of individual editing objects. The logical structure, which accommodates the control/management information about the data structure, the compositive editing data, the associated or attached procedures and so on concerned to the editing object, controls the operational units under the same access method. Of course, since our data organization is based on the relational databases the operational units are defined so as to manipulate the indicated strings according to the specified data form. The operational units are arranged in Fig.6. These operational units are selected, corresponding to the kinds of editing objects because the levels are different by editing objects.

Each editing object is commonly operated under the same commands: command names, parameter sequences and so on. The difference occurred by each editing object can be implicitly interpreted with relation to the processing status. For example, we consider a command REPLACE to update the existing data by new data. The syntax of this command is as follows:

(ex.10)

REPLACE <source> <destination>.

This means generally that <source> modified a part or all data indicated by <destination>. <source> and <destination> may be the names(e.g. editing data, term, etc), the line numbers, the range of editing data and so on. These parameters work selective in relation to kinds of editing data or the processing sequence. Of course, the content-dependent positioning

is powerful in addition to the above form-dependent indication. For example, the explicit manipulation of sentences or paragraphes in article objects is convenient in point of editing. This manipulation ability can be defined syntactically by means of assigning such meanings to special symbols. Such a mechanism can be supported by the descriptor "assign" as well as the descriptor "procedure" in (ex.5). In the image objects of bit-string data, the same famework is useful though some parameter selection constraints are at least imposed.

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 mainly available: "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 the interpretation are the attribute transitions from the text object to the image object. In the conversion, the alteration is performed simply without helps of any procedures. While, in the interpretation for the program objects, 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. The closed translations within the article objects or the program objects do 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).

6. CONCLUSION

It is desirable for an editing facility to manipulate various kinds of data uniformly, and share the editing data cooperatively among the processing facilities. In current information system environments, the issues about the uniform manipulation of the multi-media/form data and the software integration are necessarily important in order to construct advanced user interfaces.[1-4] Our approach proposed a fundamental framework as one candidate of such information systems architecturally, and addressed the effectiveness and the powerfulness for systematic user interfaces in place of researching directly the operativeness.

From an advanced user interface point of view, the concepts of the operativeness, the integration and the multi-media/form must be investigated in cooperation with their mutual relationships though individual design concepts are always important. In this paper, the uniform manipulation of the multi-media/form editing data is mainly focused under the logical view. The integration, and the mutual relationships among these concepts are the other issues in our future work. Moreover, we will have always to consider the concept of the intelligence on the basis of these concepts. The research based on these concepts of the intelligence, the multi-media/form, the integration and the operativeness will be an excellent

solution for user interfaces in the information systems.

Acknowledgements --- The authors are grateful to Prof.Y.INAGAKI and Prof. J.TORIWAKI, Faculty of Engineering in Nagoya University, for their perspective remarks, and also wish to thank Prof.M.NAGAO, Prof.H.HAGIWARA and Prof.S.HOSHINO, Kyoto University, for their useful advices.

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 annual international IEEE conference on computers and communications, pp.320-327(1987).

3) L.BOLC & M.JARKE(ed.): "Cooperative Interfaces to Information Systems", on Topics in Information Systems, P.328, Springer-Verlag, Berlin-Heiderberg(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, Berlin-Heidelberg(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).

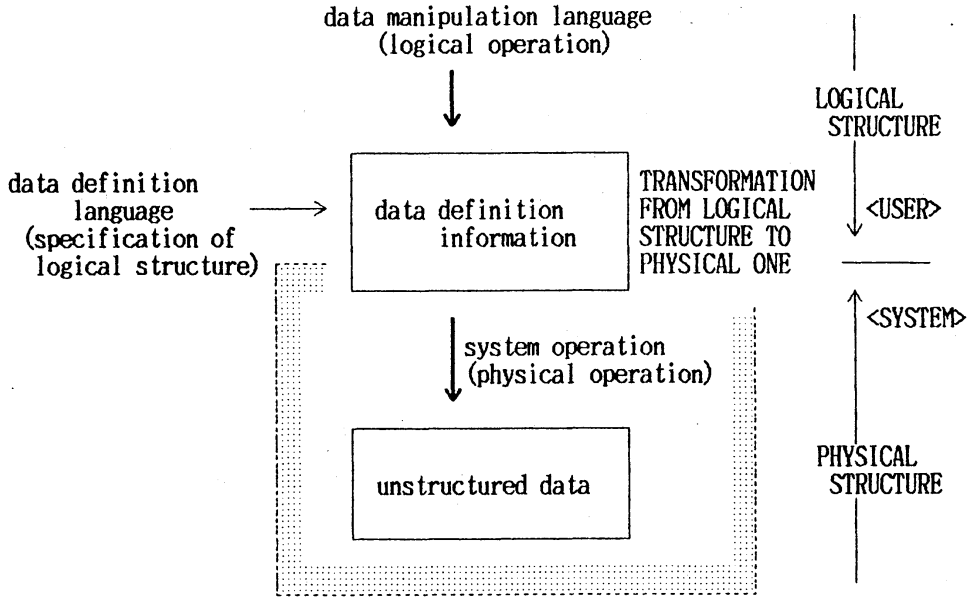
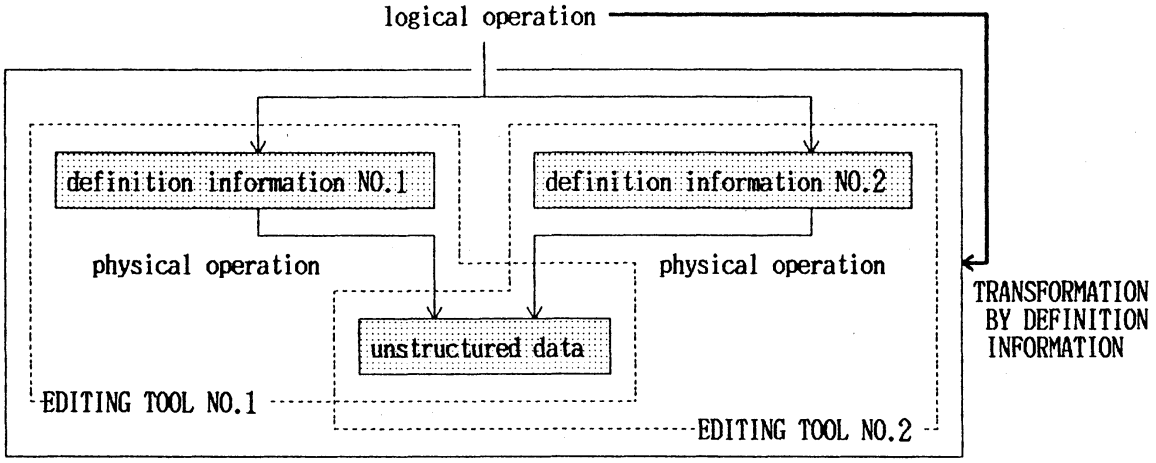
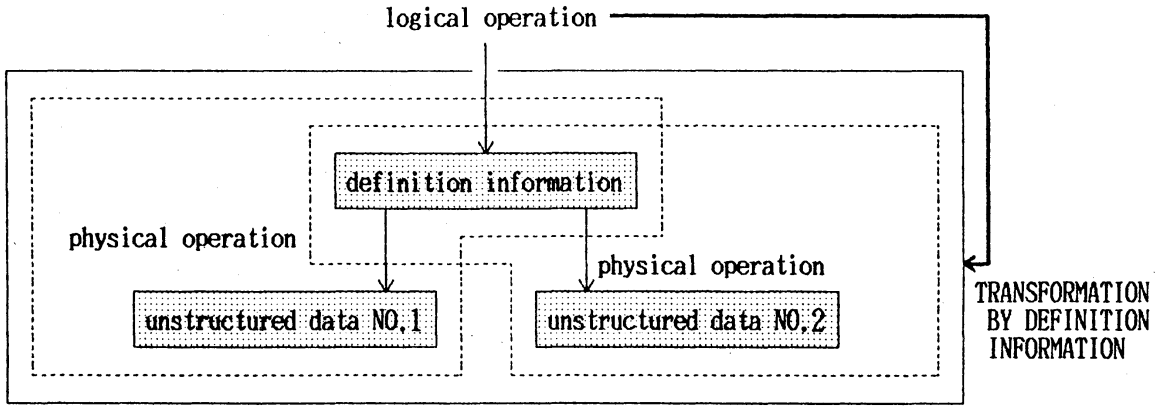


Fig.1 Data manipulation through logical structure

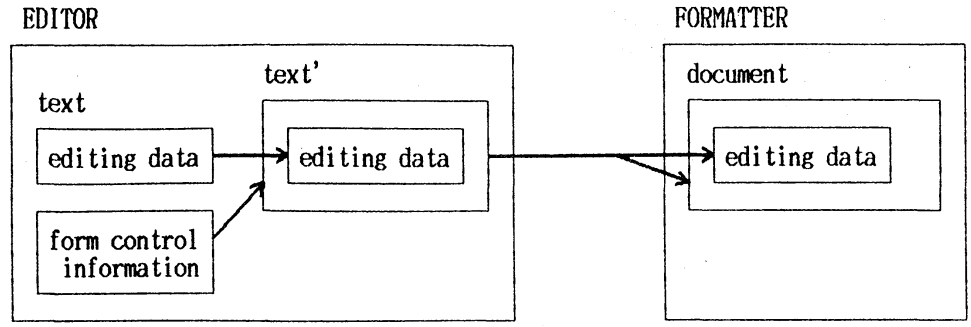


(a) single unstructured data under many definition informations

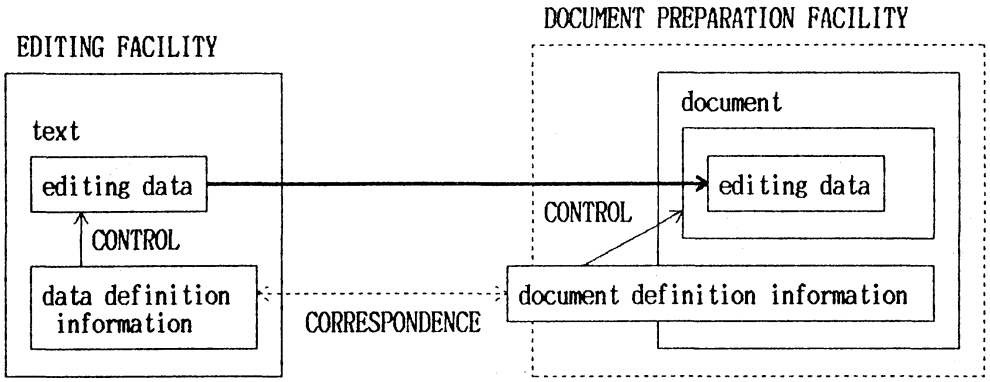


(b) many unstructured data under single definition information

Fig.2 Flexibility in editing facility



(a) traditional approach



(b) our approach

Fig.3 Relationships between editing facility and document preparation facility

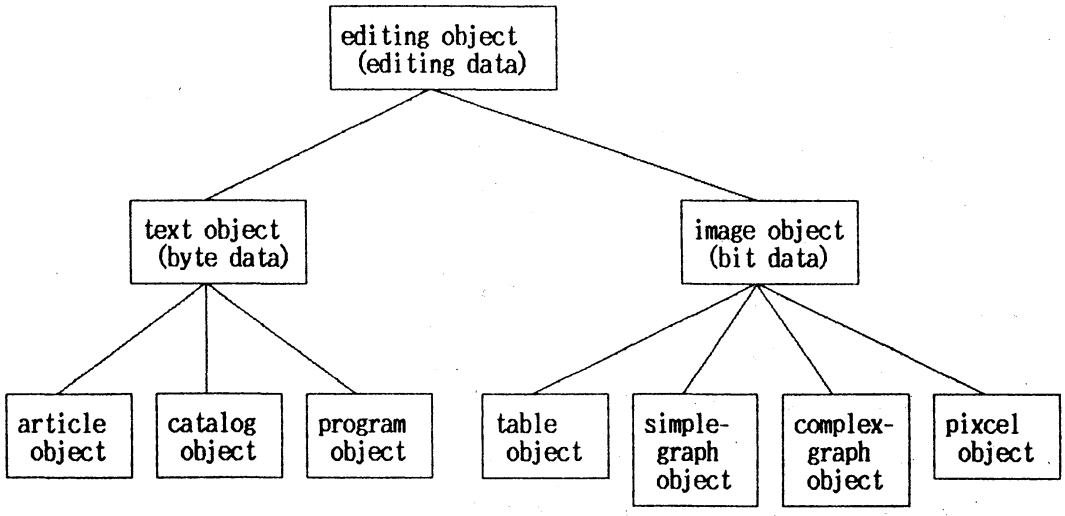


Fig.4 Types of editing objects

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

(a) article data

DOC NO	12936
TITLE	Natural language interfaces to computer system: an experimen....
AUTHOR	Guida,G.
AUTHOR AT	Politecnico di Milano, Istituto di Elettrotecnica ed .....
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

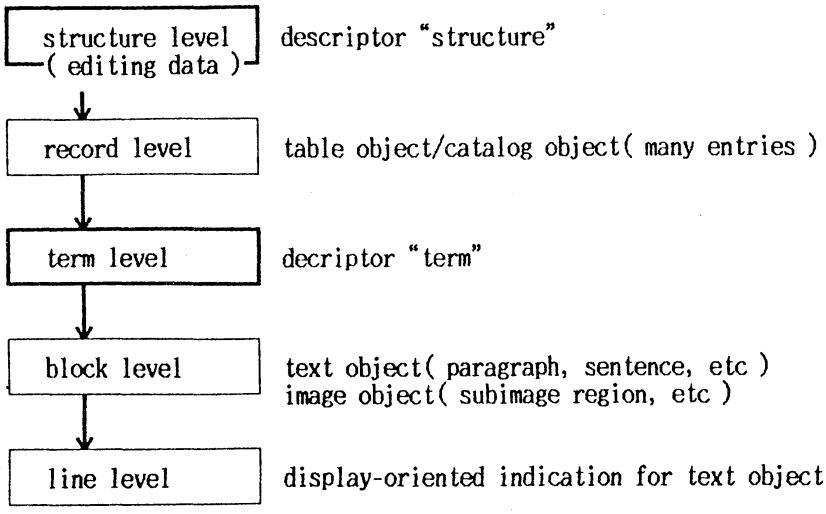


Fig.6 Operational unit

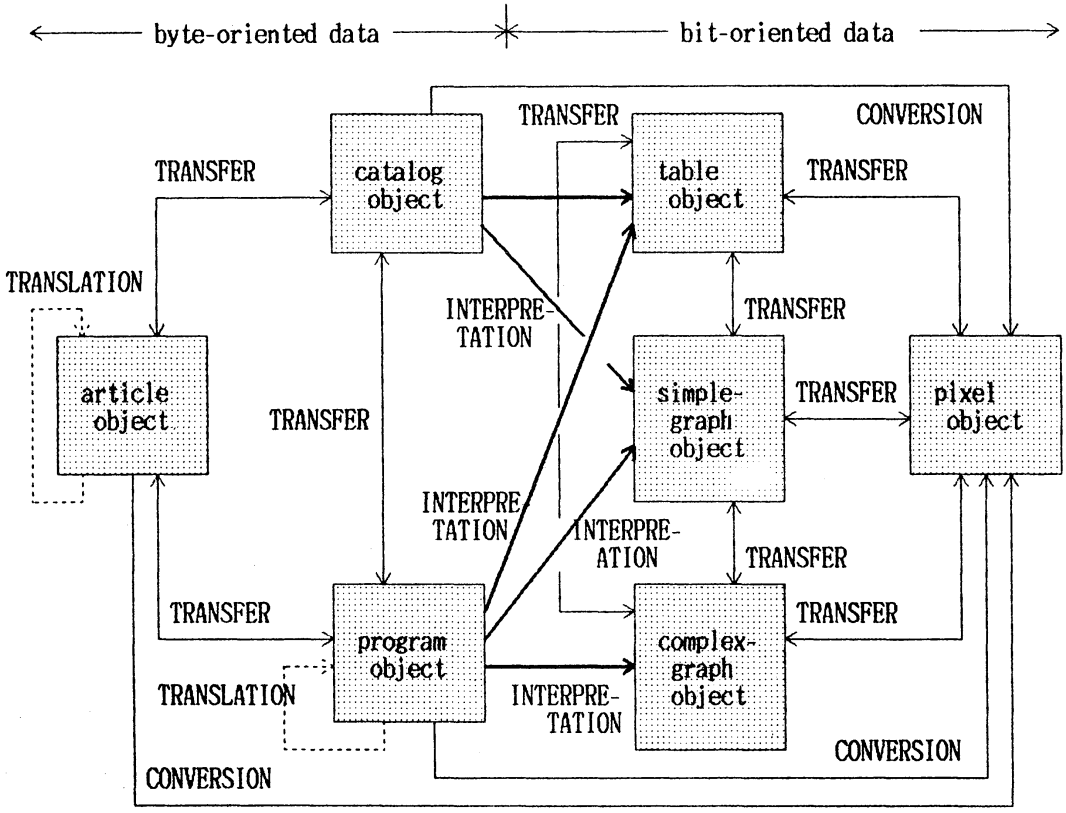


Fig.7 Attribute transition among editing objects