THE TYPE CONCEPT IN OFFICE DOCUMENT RETRIEVAL +

Barbic, F. * and Rabitti, F. **

* Dipartimento di Elettronica, Politecnico di Milano
P.zza L. da Vinci, 32 20133 Milano, Italy
** IEI - National Research Council
Via S. Maria, 46 56100 Pisa, Italy

Abstract

The problem of the retrieval by content of office documents is addressed here. However, the retrieval by content is greatly enhanced if the semantic role of document objects can be described. For this reason we introduce a conceptual level of modeling resulting in the definition of conceptual structures of documents.
Type definition is essential for the retrieval, but since office document structures tend to greatly differ from instance to instance, we introduce the concept of weak type, allowing the definition of types at different levels of detail (type hierarchies).
In this paper a modeling approach based on these ideas is presented. Particular emphasis is put on the type definition and the use of types in query formulation and processing.

1. Introduction

In office environment a very large amount of information is manipulated in form of documents.

+ The ideas exposed in this paper were developed by the authors during the ESPRIT pilot project 28 entitled "Development of a Mixed-mode Message Filing System (MMFS)", however the views expressed here are those of the authors and not necessarily those of the EEC's Information Technology Task Force.

This information can be either in a formatted form (as attributes in office forms) or in a unformatted form (i.e. text, image, and voice, or combinations of them). It is important to study tools for representing in a formal manner office documents, in order to allow a better handling of these documents by automatic procedures implemented in office information systems. In particular, filing, distribution and retrieval of office documents are basic operations of all office procedures [1].

Office documents are structurally more complex than objects usually managed in form processing systems [2] or information retrieval systems [3]. Increasing efforts in the office automation are giving more and more importance to the new concept of electronic office documents [4]. Electronic documents will be created and manipulated at user workstations, transmitted over communication networks, archived in high-capacity file servers, etc. The task of retrieving electronic documents in this dynamic and distributed environment becomes rather difficult.

Retrieval by name is possible only for a limited number of well known items (as on the personal workstation). Retrieval by location requires a careful organization of the overall system and is not always possible. In this framework retrieval by content (i.e. giving some specification of the presumed content of the wanted documents) assumes an increasing importance.

An electronic document has components which may contain different types of data and may be further structured into other components (such as the body of a paper that is composed of sections and paragraphs and contains images and data attributes embedded in text).
In the following we will

following we will refer to office documents assuming for them the electronic documents of the coming office information systems.

The problem of modeling such documents is receiving increasing attention in the research. Implicit document definition models can be recognized in several different proposals about systems managing office documents [5]. Among the systems oriented to filing and retrieval, the more common approach is to extend the functions of database management systems, adding the capability to deal with text, image etc. in addition to the original formatted data. Thus, the associated models are usually extensions of well known database models. Particular attention has been given to the relational model [6] and the entity-relationship models [7].

In the BIG project [8] office documents can be described in a conceptual schema defined using a database-like model, closely derived from the entity-relationship model. An extension is the introduction of text units and picture units as attributes of entity classes. These units do not present the atomicity property of data attributes. In fact the system allows operations on their content. For example, operations on text units, such as searching for key-words, inserting and deleting strings of characters, are defined in the system. Moreover, the system enforces principles such as data non-redundancy, integrity etc. which are more typical of database systems than of document systems.

The TIGRE project [9] aims at the implementation of a DBMS with capabilities for handling generalized data. In this case texts and images are big sized objects while office documents are considered as representative of complex multi-media objects. In TIGRE a data model has been defined as a typed extension of the entity-relationship model which includes the document formalism as a type constructor. Two categories of abstractions are supported: generalization and aggregation. They are similar to those defined in other semantic data models [10] [11]. In the formalism to represent structured documents, each document is represented using a standard form which takes into account its logical structure and its presentation and semantic attributes.

The type concept is a key factor to understand the fundamental difference between the document modeling approach in editing/formatting systems and filing/retrieval systems. A document type for editor/formatter models is a skeleton (specified either by its syntax or by formatting commands) which can be useful for creating new document instances of that type without having to start from scratch every time [12]. The document type in systems oriented to filing and retrieval is the specification of the structure and components common to all the document instances belonging to the same class. Since all the documents belong to some already defined type, it is possible to implement better storage and access methods. Moreover the retrieval can be enhanced because the system can interpret the content of any component of the document in a class (according to the definition of the corresponding type).

This paper has the following structure: in Section 2 we discuss the criteria assumed in the definition of the proposed model. In Section 3 we describe the formalism for document structure definition and the different levels of document modeling. In Section 4 we present the adopted concept of type, outlining the resulting advantages in document retrieval. In Section 5 we briefly discuss other operations in the model and the future work planned.

2. Issues in document modeling

A suitable document model should integrate the filing/retrieval and editing/formatting typing approaches, trying to satisfy two conflicting criteria:

1. To be as flexible as possible, because the structure of the office documents is impossible to predetermine.
2. To provide as much knowledge as possible about the structure of a given document, in order to assist in the creation, filing and retrieval of these documents.

The first criterion would lead to a model without types (in the database meaning), letting each document instance have a structure defined for its own, like in usual document editors and formatters, while the second criterium would lead to a strongly typed document model, like the ones for filing and

retrieval systems. This allows to efficiently store and handle the corresponding system objects (such as records, forms or tuples), since the system could take advantage of the objects regularity and generate storage structure on a per-type rather than per-instance basis. Unfortunately, this is not the case with more general objects as office documents, since similar documents can have different structures. However the type concept is still useful as it can aid in formulating queries and creating and modifying documents.

An office document model should aim at a type concept that is a good compromise between the two above criteria trying to obtain the advantages of both database-oriented models (for filing and retrieval operations) and editor-oriented models (for composition, editing and presentation operations). A type should be considered as being the definition of all common properties of a set of documents, letting a document instance have a much more complex and detailed structure.

In many existing document models the internal structuring of documents has been optimized according to different requirements for different operations, such as transmission efficiency for document interchange or access efficiency for filing and retrieval. In order to avoid several conversions of the internal structure of a document during its lifetime, it is apparent the need of a unified model including all the structuring aspects necessary for the different operations. However, in order to be useful, this model should be widely accepted and understood. In fact, if a document is generated in an office system workstation, an internal structure is associated to its visible data elements. The internal structure is transmitted together with the document. If the receiving workstations or systems do not know about that document model, they could not interpret the internal structure of the document and no further processing could be performed on it. The interchange of documents is the operation where the need of standardization is more apparent. In fact, when a document with its internal structure is encoded in some format at one site, the same format should be known at the other side of the transmission in order to decode and reconstruct the document and the internal structure.

A document may go through successive editing operations after its creation. Besides the document final form [13], it is necessary to store its revisable form which can be further modified. A part of the internal structure should include its syntactic components and is called logical structure.

Documents must be presented on physical output devices by mapping the logical structure into the external representation. This is called the rendition process [13]. The added information necessary in this process should be coupled to the logical structure and should be included in the internal structure (for instance, see the template concept in [14]). This information is called layout structure since it shows how the data elements should fit the layout of the document at presentation time.

Two modes of document retrieval should be possible: retrieval by location and retrieval by content. The first mode of retrieval mimics the organization of conventional offices, where documents are retrieved picking them from the location where they were stored. In the second mode of retrieval the user specifies a query by creating a partially complete specification of the internal structure and specifying some conditions on the document content [15] [16]. In retrieval by content the desired document must be addressed by some of its characteristics. Among them, a crucial role is played by the semantic characteristics which describe the meaning of the document composing objects. This information constitutes the conceptual structure that is defined giving names to the semantic components of a document. The conceptual structure is added to the layout and logical structures and complete the document internal structure.

3. The office document model

As resulting from the previous discussion, in a office document we can distinguish different levels of structure. At a more general level, we can see the document as composed of semantic components. They reflect the common understanding by the users of a class of similar documents. These semantic components describe the conceptual structure of the particular document, which is also common to several documents with the same function

in the office organization (e.g. the "meeting subject" in a "meeting announcement letter", the "topics of interest" in a "conference call for papers").

At a more superficial level, the syntactic structure of the document is apparent. In fact, what is seen at the surface is the composition of multimedia data values in the document. These syntactic components constitute the logical structure of the document (e.g. section, pie chart, table, etc.). This structure can sensibly vary even among documents with the same semantic structure while can be similar for documents with different semantic content. A clear distinction between the two levels of conceptual structure and logical structure is that the former reflects the document content with respect to an application environment (and then it cannot be subject to standardization) while the latter reflects the content of any office document in any application environment (and then it can be subject to standardization).

In order to guide the presentation of a multimedia document, a layout structure should be strictly associated to the logical structure. The layout structure shows how and where the logical elements should be displayed in the physical document. Logical structure and layout structure should be defined according to the international standards under development [13].

For the definition of all these internal structures we will follow the same approach of syntax-directed editors [17]. Syntax-directed editors are very flexible in defining document structures [18]. They can be exploited in the definition of the logical, layout and conceptual structures. However they usually allow to operate on a per instance basis, that is in the definition of each document instance structure. Instead we intend to explore this approach as a formalism for a complete definition of an office document model in all its three components (logical, layout and conceptual). This requires to operate not only at the instance level, but also at the type level.

In the following we introduce a formalism based on context-free grammars for the definition of the model. In fact, different grammars will be used for the definition of the three different levels of modeling. Since each document structure is defined by the formalism of the corresponding modeling level, these different grammars will allow to define the three different structures of the documents. We will present the formalism in abstract terms, implying that it will be applied to the three different cases.

Let us assume to adopt a context-free grammar G for the specification of a document structure (either logical, layout or conceptual);

G=(N,T,R,P)

where:
N is the set of non-terminal symbols,
T is the set of terminal symbols,
R in N is the root symbol
P is the set of grammar productions.

Productions have the form:

A -> "alpha"

where A is a non-terminal (in N) and "alpha" is a string of terminals (in T) and non-terminals (in N). A is the left hand side (LHS) of the production, "alpha" is the right hand side (RHS) of the production. Only two forms of productions are allowed: in the first the RHS is a string of non-terminals, in the second the RHS is one terminal. The first is called a non-terminal production while the second a terminal production. These restrictions on the format of productions do not restrict the power of the context-free grammar.

For the specification of the productions we use an extended BNF representation, where each production has the format:

<A> -> <B.1> [+]...<B.N>[+]

where <B.i> are non-terminal symbols which can be optionally tagged by a "+" if they are repeating.

The document structure (either logical, layout and conceptual) is called Structure Tree, which is conceptually equivalent to the parse-tree containing all the productions of G applied from the root symbol until the complete document is obtained.

The Structure Tree is composed of branches. A branch is a subtree of only one level in depth (this means that an upper node is connected by an edge to several lower nodes), shaped according to the productions in P of G, and having the nodes labelled with unique identifiers IDs.

The branch B is defined as instantiation of the production PN iff:

- Its upper node is the instantiation of the non-terminal symbol in the LHS of the production PN.
- Its lower nodes are the instantiations of the non-terminal symbols (in the same order, established by the connecting edge) in the RHS of the production PN.

The node corresponding to a repeating symbol may or may not be repeating. In the former case it is tagged by a "+", i.e. ID+. Notice that for the branches instantiating terminal productions the lower level contains only one node which is the instantiation of a terminal symbol. We assume in this case that ID is the actual value (or pointer to) of the data item represented by the terminal symbol.

A Structure Tree is composed by set of branches which correspond to the productions exploited in the generation of a document or type. The root of the tree is the ID instantiating the root symbol R of G. Then the branches can be incrementally added to the structure tree according to the grammar productions. We can notice that in the Structure Tree only one edge can depart from a node instantiating a non-repeating symbol, while more edges (each corresponding to a branch) can depart from a node instantiating a repeating symbol.

We define a leaf node as a node with no exiting edges. A terminal node is a leaf node instantiation of a terminal symbol. A live node is a node which can generate a new branch according to the productions of the grammar. A live node is essentially a spring for new subtrees in the Structure Tree. Hence, a non-terminal leaf node is alive (that is, it can be a spring for new subtrees), while any live node that is not leaf must be repeating. A partial example of a grammar and a derivated Structure Tree is shown in Fig.1.a and Fig.1.b.

Restrictions are dynamic limitations for the introduction of new branches in the Structure Tree. Restrictions are expressed in form of special statements attached to some nodes in the Structure Tree. Restriction statements are useful in the instance creation (or in the specification of more detailed types) starting from types, since they can impose restrictions in this process. This can be a guidance for less experienced users who cannot manage the full power of the grammar productions.

GRAMMAR PRODUCTIONS

.....			P5 :	B	b
P1	A	BC+D	P6 :	D	d
P2	C	FG+	P7 :	H	h
P3	C	H+JK+	P8 :	I	i
P4 :	G	IL	P9 :	L	l

Fig. 1.a

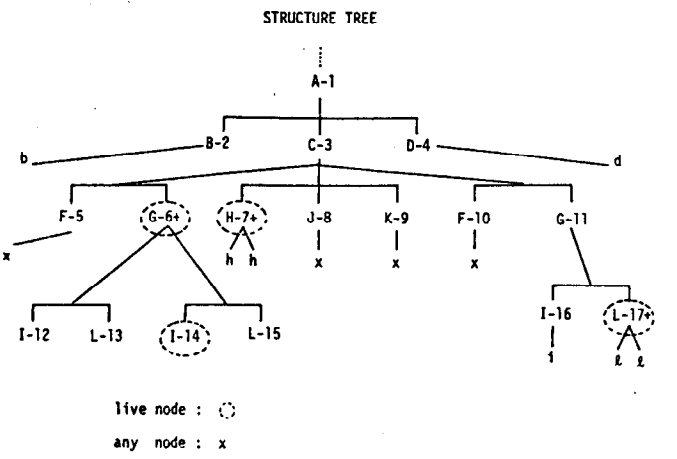


Fig. 1.b

For this reason, restriction statements are meaningful only as part of type structure trees. Different types of restrictions are introduced.

Quality restriction on nodes

We can restrict the set of productions in P of G which can be used to instantiate a new branch starting from a live node. A special statement is attached to the node saying that a new branch in the Structure Tree must be the instantiation of one of the productions enlisted in the restriction statement (see Fig.2.a).

Quality restriction on subtrees

We can pre-define different subtrees (composed of several levels of branches) starting from a live node. In this case, this restriction statement says that if a subtree is to be generated from this live node, its structure is bound to be like one of the pre-defined subtrees (see Fig.2.b).

Quantity restriction on nodes

We can restrict the number of branches which can be generated from a repeating node in the

Structure Tree. A special restriction statement is introduced saying that at most MAX branches departing from this repeating node can be present at any time in the Structure Tree (see Fig.2.c).

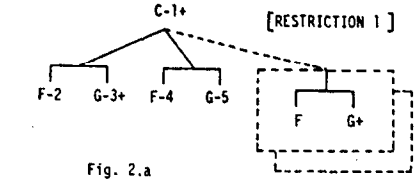


Fig. 2.a

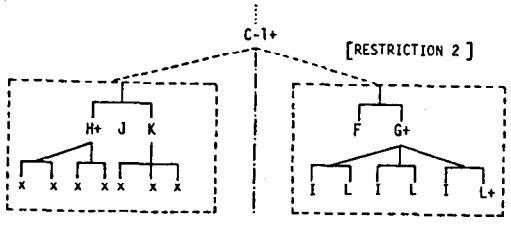


Fig. 2.b

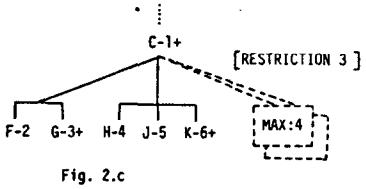


Fig. 2.c

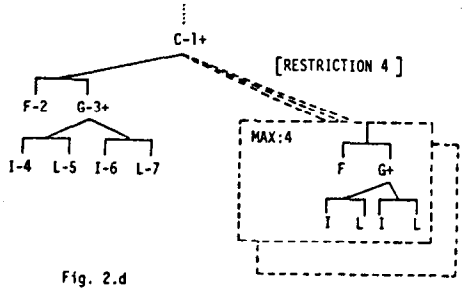


Fig. 2.d

Quantity restriction on subtrees

We can predefine a subtree (composed of several levels of branches) starting from a live node. In this case, this restriction statement says that in the specialization/instantiation process either this live node will generate a null element (that is, applying the terminal production <X>->[NULL]), or will generate a subtree whose structure is bound to be like the predefined subtree. If the live node is repeating, the quantity restriction on subtrees can specify the maximum number of replications of the pre-defined subtree which can depart from this live node (see Fig.2.d).

Restrictions are very useful in the type definition. They allow to define type characteristics inside structure trees, so resulting in a common representation of types and instances via Structure Trees.

3.1 Type and instance concepts

Having formally presented the adopted model, it is possible to define the concept of document type and instance in the new meaning of this document modeling approach.

A Structure Tree corresponds to a pure instance iff there are no live nodes in it. Since terminal nodes are leaf nodes which are not alive, in a pure instance Structure Tree all leaf nodes are terminals. Intuitively, any document instance has a "complete" structure tree, that is, a structure tree where all the possible top-to-bottom paths starting from the root node are completed up to a leaf node whose ID is a multimedia data value, instantiation of a terminal symbol.

A Structure Tree corresponds to a type iff there is at least one live node in it. This type concept is more general than the usual data modeling concept [19] of type for formatted data. In fact it is comprehensive of the type concept of both strongly typed and weakly typed data models. The instantiation or specification process on a type is adding new branches in the Structure Tree as instantiations of grammar productions and with the restriction statements already in the Structure Tree. We define a strong type as a type which can only be instantiated by adding terminal nodes. We define a weak type as a type which can be instantiated by adding any node (also non-terminal).

This concept of strong type is equivalent to the type concept at the schema level of data base models. In fact, instantiating a strong type can only consist in adding multimedia data values of specific types, which is equivalent to adding terminal nodes to the Structure Tree. Instantiation of a strong type is a DML operation in the data base terminology.

The concept of weak type is more general. Since non-terminal nodes can be added to the Structure Tree, composed objects can be added during the instantiation process. These document

components correspond to new subtrees of any complexity. (Restrictions on subtrees can impose limits to this freedom.) Thus, the instantiation of a weak type may correspond to a phase of type definition, at schema level, in data base models. This is a DDL operation in the data base terminology. It can be combined to a DML operation, if values are also defined. It is clear that in this approach, using the partial specification process on the structure trees, there can be complex hierarchies of inclusion among weak types. They may be, in the general case, non-tree-like hierarchies.

The flexibility resulting from the weak type concept is necessary because of characteristics of the objects (multimedia documents in the office) to model. The flexibility required, as for adding new complex components (i.e. a new section with tables and graphics), cannot be given by strong types. Moreover the system can exploit the complex hierarchies of weak types by keeping catalogs of system enforced types. These system types can be useful for the document instantiation, giving the user flexible document skeletons; for the query definition, giving the user flexible query-by-example skeletons; for query processing, if the document search is restricted within a certain type (the most specific type applicable for the query).

The flexibility of this typing approach allows types for the different classes of documents in the office environment to be defined. Strong types are suitable for all form-like documents (i.e. those with a very stable structure). Weak types are more suitable for less structured objects, such as letters, memos, reports, brochures and other office documents containing tables, graphics, images, voice comments and conversations. In this case, the possibility of establishing hierarchies on weak types is very useful.

Since structure trees can represent the internal structures of both types and instances, it is possible to query types as well as instances. In fact a query specification contains a partial structure tree as well as some conditions on data values (appearing as terminals in the partial structure tree). Query resolution consists in the matching the partial structure and satisfying the terminal conditions. This same process can be applied on structure trees of both instances and types. Querying types can be very useful in this environment since there is no well formed, consistent, complete schema as in data base systems, but a "sparse" set of interrelated types. Consequently the user may have to choose between many different types when performing an operation.

Naming is a crucial problem for a system adopting this approach. Different names can be defined for types and type components which are structurally and conceptually very similar. A system may support support certain types keeping catalogs of type and component names, with classes of synonyms.

3.2. Levels in document modeling

We have already said that there are three levels in document modeling: conceptual, logical and layout. They are described with the same grammar-based formalism but they are essentially different. Also different are the three resulting structure trees. However these trees are interconnected since, at the end, they aim at describing the same objects: the office documents.

The conceptual structure describes the semantic components of the documents, giving names to these components. Names are useful in defining the type-level part of the document structure and, being meta-level information (as they do not appear on the document as values) they correspond to names that are assigned in data base systems at the DDL level. Name catalogs should be maintained by the system in order to facilitate the users in naming choices. Since the form of document conceptual structure depends on the semantic of the document, the syntax for defining it must be very flexible. A meta-grammar should allow any semantic component's hierarchical decomposition, giving names (i.e. meta-names) to these hierarchical semantic categories. The meta-grammar basic production, defining semantic components, should be like:

<semantic_component> ->
<component_name> <semantic_component>+

We have already stressed the importance of standards for the logical modeling level as well as for the strictly related layout modeling level, which is necessary for the presentation of the logical structure. In fact, an Office Document Architecture (ODA) [13] is in the process to be standardized by the

European Computer Manifacturers Association (ECMA), the International Standards Organization (ISO) and the CCITT (the international organization for communication). The ODA will give a formal description of the document composition (in the logical structure) and a formal device-independent description of the document presentation/rendition (through the layout structure). In this manner, it will achieve the goals of standard document interchange and document presentation on different devices. The ODA will also include the standard formats for the multimedia data elements contained in the office documents, such as characters elements, geometric elements, photographic elements, etc. The grammar definition for the logical level and layout level of this model will be defined according to the specification language of the ODA. A simple example of conceptual, logical and layout structures is given in Fig.3.

3.2.1. The ODA sub-model

In the ODA sub-model, the nodes in the logical structure tree and the layout structure tree are called logical objects and layout objects. Terminal nodes are called basic objects, while non-terminal nodes are called composite objects. Only basic objects have portions of the document content (i.e. multimedia data elements) directly associated. Therefore the document content is divided into content portions which correspond, from the logical view-point, to basic logical objects, and from the layout view-point, to basic layout objects. The content of a basic object is always of a single multimedia data type. The basic objects can have a further internal structuring, called sub-architecture, which also will be standardized. However, since sub-architectures have no relationship with their exterior (they only concern editors, printers, scanners), they are not part of the model.

Pure tree structures are not sufficient to express all office document structures (e.g. tables with nested elements) at logical and layout modeling level. Therefore, in addition to the hierarchical order it is necessary to specify the organizational order among the constituents of the composite objects. A constructor defines how a composite object is built by its constituents and which selectors can be used to access the constituents [13].

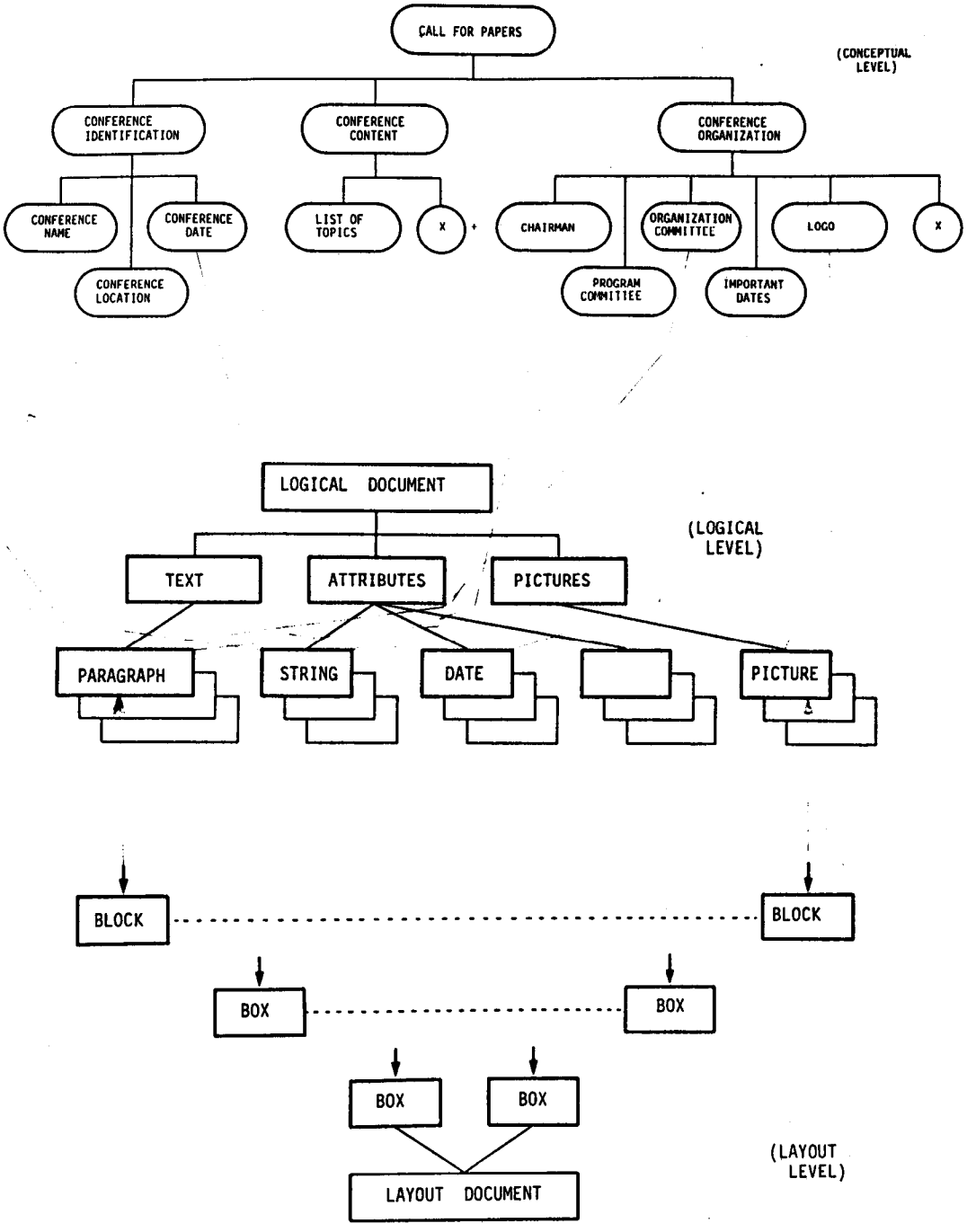
Since in the model formalism only the hierarchical tree structure can be specified using the grammar productions, the constructors in the logical and layout structure tree cannot be directly expressed in the structure itself. A solution is to represent a one-level subtree with a particular constructor as a two-level subtree in which the first level is a branch whose RHS node specifies the constructor, and the second level is a branch whose RHS nodes specify the constituent nodes. There are three types of constructors:

1. The SEQUENCE constructor specifies a sequential order for the constituents of an object. The constituents are of the same type and are sequentially accessible.
2. The ARRAY(n) constructor specifies a n-dimensional orthogonal matrix-like order for the constituents. They can be of the same type or different types. They are directly accessible: the selectors are n-tuples of indices.
3. The AGGREGATE constructor specifies no particular order of the constituents, which can be of the same type or different types. They are directly accessibles by constituents names.

Any object is also characterized by its properties. A property (e.g. the number of a section) is of a certain property type and has a certain property value. Property types and value ranges are standardized. Property types must be described in the grammar of the logical and layout levels of the model as special symbols associated to the object symbol in the RHS of a production. In a logical or layout structure tree, property values will be the RHS of a simple branch whose LHS is the property type node. This node is linked to the object node as RHS of the same branch.

Most layout objects are two-dimensional ares of rectangular form, called boxes. Composite boxes are composed of basic boxes and/or other composite boxes. Basic boxes are called blocks. All boxes have properties ot type size, position, character font, background color, etc.

Another important component of a document structure, according to ODA, is the document profile. It contain information for handling tha documents as a whole and consist of a set of attributes. The values of these attributes may or may not appear as document component (i.e. basic logical



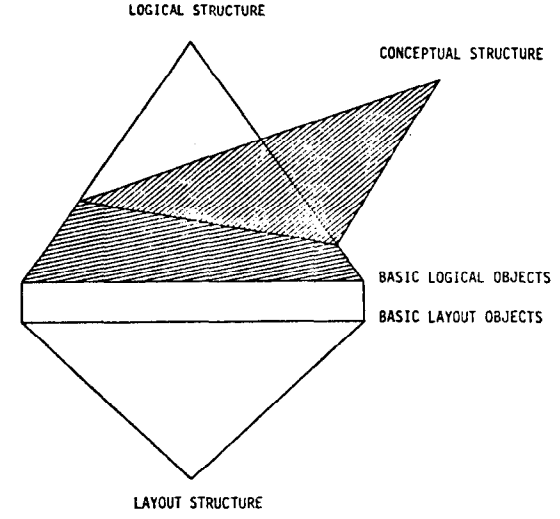
Conceptual, Logical, Layout Structure Trees

Fig. 3

objects). This means that attributes such as document name, author, date may also appear in some part of the physical document, but may also be internal attributes useful for other purpose such as document indexing, filing and retrieval, interchange, etc.

4. Using types for document retrieval

As seen in Section 3, a weak concept of type is necessary in the proposed model because of the need of flexibility in the office document definition, manipulation and retrieval. While a document instance is composed by the data values and internal structures of the physical document, a document type in our model is a subset of the internal structures common to a class of document instances. A document type is still represented by its internal structures, where live nodes show where and how (see also the restrictions) components and data values can be added to obtain instances. When types are enforced by the system in order to support the retrieval-by-content process, the conceptual structure plays an important role in the type definition. When types are mainly exploited for document creation/modification, the logical and layout structures are more relevant (see the concepts of "style" and "galley" in JANUS [20]).



Relationships among internal structures

Fig. 4

The relations among the three different structures are shown in Fig.4. Layout and logical structures are orthogonal, while conceptual structure intersects at a certain level the logical structure. It is important to highlight some properties of conceptual and logical structures. For each document internal structure, the leaves of the conceptual tree represent the semantic objects at the lowest level that is relevant in the application context. These objects are also described in the logical structure, where their syntactical composition is specified. Hence, it is possible to link the conceptual leaves to the corresponding logical nodes. Four categories of links can be identified:

1. 1-1 Link
   A conceptual leaf is linked to one and only one logical node. In this case a logical (composite or basic object) plays the role described by the conceptual leaf. For instance, a piece of text plays the role of introduction in a business letter.
2. 1-N Link
   A conceptual leaf is linked to several logical objects. This case happens when the conceptual leaf is a complex object and its semantics is embedded in different logical objects. For instance, the description of some characteristic in a business report may span over several sections or over several paragraphs of one section.
3. N-1 Link
   Several conceptual leaves are linked to one logical object. In this case the logical object plays different semantic roles within the document. For instance, in a hardware product announcement a picture can be considered as the histogram of costs of a video display and an example of its resolution capabilities.
4. N-N Link
   This case is the most complex and includes both the previous cases 2) and 3).

The type specification is mainly conceptual, but can contain some logical aspects and even some actual values of the physical document (the lowest level of the logical structure). For instance, let us consider that the type "Corporate business letter" is enforced by the system. Its structure mainly describes the conceptual objects that are common to all the instances of these letters. However, it is also possible to specify some logical characteristics, as the presence of the logo, and even some values such as the sender address that

is common to all the instances. Moreover, if the logo and the address always appear in the same portion of the physical page, it is also possible to think to include the corresponding layout information in the type definition. Another possibility is to include in the type definition some attributes of the document profile. This can be accomplished linking the leave elements of the conceptual structure to selected attributes contained in the document profile according to the ODA standard. In Fig.4 the dashed part represents the possible extension of the type specification.

In the following we will consider first the way in which types can be defined explicitly by the Office System Administrator or semi-automatically by the system. Then we will briefly describe the advantages of the weak typing approach in query specification and query processing.

4.1. Explicit type definition

The type definition as an explicit operation is essentially an a-priori operation, as in database environment. The Office System Administrator, who is an expert of the application environment, will design the types specifying their structures, and give names to types. Type structures will have live nodes. Restrictions associated to live nodes will constrain the user when sub-structures or data values are to be added during document creation or query formulation for that type. These types will be inserted in a system catalog, and so they will be made available to the office users for all the document operations.

The flexibility of the typing approach will allow to define types for the different classes of documents in the office environment. Strong types are suitable for all form-like documents, with very stable structure. Live nodes, in this case, can only generate data values (using terminal grammar productions). Weak types are more suitable for less structured objects, as all kinds of letters, reports, brochures etc. which may contain tables, graphics, images, voice comments and conversations. Live nodes, in this case, can generate various forms of new structures (using any grammar production and obeying to the associated constraints).

4.2. Implicit type definition

Since it is always difficult to define a-priori all and only the document types that are actually useful in an office environment, given its dynamic operational characteristics, it is useful to allow also the type definition as an a-posteriori operation. This second mode of type definition, which is made possible by this modeling approach, can be partially supported by the system. During office lifetime, the conceptual and logical structures of all the instances can be compared and the regularities of these structures can be synthesized in new type structures. These structures are actually proposals of conceptual and/or logical structures that can fit a number of instances greater than some threshold value. This can be accomplished at regular intervals by some background process, which then reports to the Office System Administrator for confirmation and assistance (e.g. defining names of new types, names of internal conceptual components, exact positioning of live nodes and associated restrictions, etc.).

Implicit type definition includes also a-posteriori type assignment and reorganization. In fact, the internal structures of the document instances can be compared with already defined types, other than those they where created from. In this case, document instances can be re-assigned to catalogued types which are more specialized than their original types.

Proposals of type definition can also be derived from the analysis of the documents coming from outside the office via electronic mail and even in facsimile form. In the last case, the common characteristics will be only logical or layout such as the presence of a table or a picture on several coming instances. The table in Fig.5 summarizes these situations.

4.3. Hierachies of types

It is not reasonable to imagine the type catalog as a single level, static and completely defined schema as in database systems. A more probable situation is one in which complex hierarchies of types and subtypes exist and are connected by relations similar to the "is_a" relationship of database models [19].

INPUT MODE	PROCEDURE	OUTPUT
TYPE EDITING	FROM TYPE CATALOG	NEW TYPE
INSTANCE	SELECT A TYPE, THEN REFINE AUTONOMOUS DEFINITION	NEW INSTANCE
NO CONCEPTUAL STRUCTURE ELECTRONIC MAIL	RECOGNITION OF TYPE	NEW INSTANCE/TYPE + LINK TO TYPE
WITH CONCEPTUAL STRUCTURE	COMPARISON OF INTERNAL STRUCTURES	NEW INSTANCE + TYPE PROPOSAL
FACSIMILE	SEGMENTATION BLOCK RECOGNITION CHARACTER RECOGNITION COMPARISON OF LOGICAL AND LAYOUT STRUCTURES	NEW INSTANCE + TYPE PROPOSAL

Instances and types in document input

Fig. 5

The more general types are at a higher level of the hierarchy and several conceptual or logical subtypes are origined from them by refining some objects. For instance, a possible type can be "Product announcement letter" and two possible subtypes are "Hardware products" and "Software products". Thus, in document creation and query specification, the most specific type (or subtype) is chosen and then its internal structure is refined until the basic objects and the actual values of document data.

Each instance is linked to the most specific type that is present in the type catalog and that fit its structure. If an instance is structurally modified, it may happen that its structure does not fit any more the type specification. In that case, the link is shifted towards the higher levels in the type hierarchy until the consistence between instance and type is realized. In Fig.6.a the type hierarchy and the links with the set of instances are shown.

4.4. Query specification

The query specification in our approach is very similar to the document instance definition. In fact, in a query specification it is possible to have conditions on the internal structures (logical, layout and conceptual) as well as conditions on data items of the wanted documents (see Fig.7).

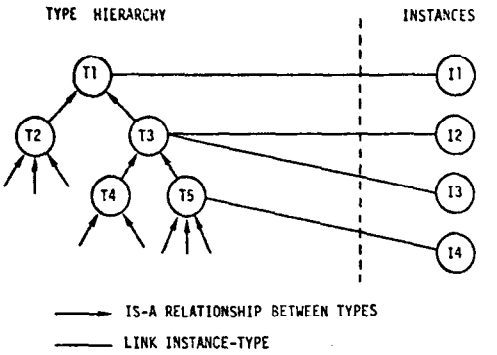
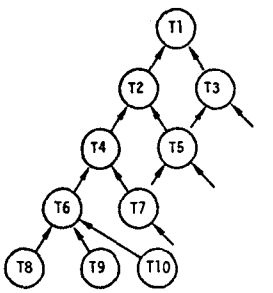
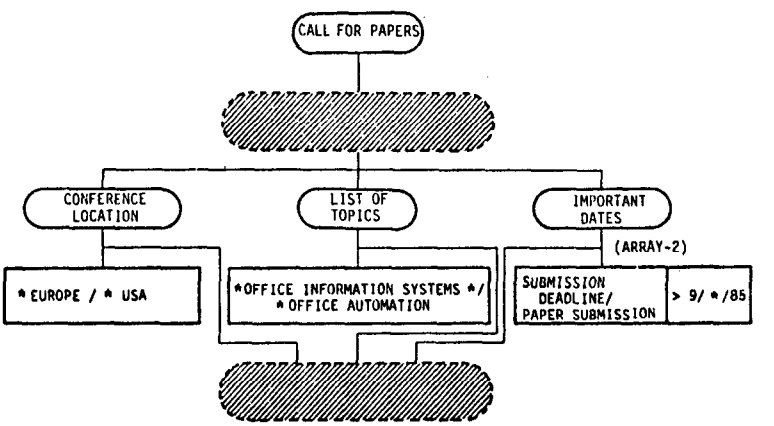


Fig. 6.a



Example of type hierarchy in query processing

Fig. 6.b



Query specification

Fig. 7

The system is then asked to select the matching document instances. In this process we implement the retrieval by content on documents.

Moreover, the user can query the type catalog. In this case, the query conditions refer to types. The selected

type is the most specific in the type catalog containing the desired characteristics. A further step is then possible, i.e. the user can refine and extend the selected type specification by using the model tools for the definition of the internal structures of the document instances. The result is a type specification that in general is not present in the type catalog, but can be useful in query processing for restricting the class of matching instances. Some undefined areas are allowed in the type specification. These undefined areas can match with any structure portions of the stored documents, while the defined areas are required to match exactly (both for structures and data value conditions).

4.5. Query processing

The query processing problem is similar, in principle, to the partial subtree matching problem found in semantic network interfaces to database systems. During query processing the established relations between instances and types can be used for narrowing the number of instances that must be checked for matching. First, the most specific type matching the conditions is identified by comparing the corresponding structures, then all the instances directly linked to the type are matched. At this point, the system goes down in the type hierarchy and, for each (sub)type, checks first the type definition and possibly the linked instances. In this way the instances that are linked to a non-matching (sub)type are quickly disregarded. Let us imagine the situation of Fig.6.b in which a type hierarchy is shown. We can select T2 as the starting type specification in the query specification and enrich its structure according to T4 and not to T5. During query processing it is sufficient to start from the instances directly linked to T4. Moreover, if the structure of T6 does not match the query, all the instances linked to T6 and to its subtypes can be disregarded. Sometimes query processing can be performed even more efficiently if the storage files containing the specified data item values are searched first and then the structures are compared.

When the specified condition is complex, it is worthwhile to start different parallel processes for matching it. Thus, one process can search the storage files containing the document data, while another process can select the matching structures and the correspondent instances. When a document is retrieved, it is immediately displayed at the workstation and, while the user is examining it, the system can proceed in the query processing.

Moreover, in this approach the user can decide to dynamically change the filter or to choose a retrieved document type as a new filter (i.e. substituting data values with conditions on them and leaving undetermined other components). This is a very flexible environment for dynamic query definition by the user.

5. Conclusions and future work

Office documents, as will be used in future office information systems, can have very complex structures. Moreover, these structures tend to differ from instance to instance, so making impossible a strict type definition for classes of documents. In the modeling approach for office documents presented here, we have tried to be as flexible as possible in order to allow a suitable representation for the document structure and content, as well as for operations such as editing/formatting, interchanging, presentation, and especially filing and retrieval.

In this paper we have focused on the office document retrieval. However, other operations necessary in office environment can be supported by this model. For example, document modification means to operate on the document internal structures, according to grammar rules and restrictions, as well as on multimedia data values. Document rendition, on all possible display/printing devices, means to present the document data values, which are basic logical objects, according to the specifications contained in the layout structure. Document input may be a critical operation in our model. In fact the system must understand and internally reproduce the structures of a document when it is entered. If the document is entered through an interactive document editor the user can take advantage of the already defined document types and can be helped by the interactive system that knows about admissible actions at every step of the process. If the document is presented to the system in facsimile format, it is necessary to scan and parse the document as a compiler would parse a program.

Completely automatic scanning and parsing is in most cases unfeasible unless a type is explicitly associated to the facsimile documents by the operator. If the document is sent by electronic mail from another system, we must suppose that it complies to the ODA standard. Therefore, logical and layout structure should be included in the trasmission and should be understood by the system. The conceptual structure would be present only if the sending system knows about our model.

Future studies will deal with formal specifications of operations on office documents (both those concerned with standards and those concerned only with retrieval functions). An important topic will be the investigation of methods for fast access to document internal structures for an effective implementation of retrieval by content. For this purpose, also efficient representations in storage of these modeling structures (i.e. using bit patterns) will be studied. The proposed model will be further studied and evaluated in the context of a project in the area of "Office Systems" entitled "Development of a Moxed-mode Message Filing System (MMFS)", which is part of the European Strategic Programme for Research in Information Technology (ESPRIT).

REFERENCES

[1] Tsichritzis, D. and Christodoulakis, S., "Message files", ACM Trans. Office Information Syst. Vol. 1(1), pp.88-98 (Jan. 1983).

[2] Gehani, N., "The potential of forms in office automation", IEEE Trans. on Commun. Vol. Com-30(1), pp.120-125 (Jan. 1982).

[3] Croft, W.B., "Applications of information retrieval techniques for the office", Proc. 6th ACM-SIGIR Conference on Research and Development in Information Retrieval, pp. 18-23 (1983).

[4] Limb, J.O., "Integration of media for office services", Office automation conference digest, pp.353-355 (March 1981).

[5] Bracchi, G. and Pernici, B., "The design requirements of office systems", ACM Trans. Office Information Syst. Vol. 2(2), pp.151-170 (April 1984).

[6] Schek, H.-J., "Nested transactions in a combined IRS-DBMS Architecture", Proc. 3rd Joint ACM-BCS Symposium on Research and Developement in Information Retrieval (1984).

[7] Adiba, M., Nguyen, G., "Information processing for CAD/VLSI on a generalized data management system", Proc. Tenth Int. Conf. on Very Large Data Bases (1984).

[8] Crampes, J.B., Chrismet, C.Y., Zurfluh, G., "The BIG project", Proc. 2nd International Conf. on Databases (Sept. 1983)

[9] Lopez, M., Velez, F., "Modeling and handling generalized data in the TIGRE project", Working Paper, IMAG, St. Martin d'Heres, France (1084).

[10] Smith, J.M. and Smith, D.C.P., "Database abstractions: Aggregation and generalization", ACM Trans. Database Syst. Vol. 2(2), pp.105-133 (June 1977).

[11] Brodie, M.L., "Data abstraction, databases and conceptual modelling", Proc. Sixth Int. Conf. on Very Large Data Bases, pp.105-108 (1980).

[12] Furuta, R., Scofield, J., and Shaw, A., "Document formatting systems: Survey, concepts and issues", Comput. Surv. Vol. 14(3), pp.417-472 (Sept. 1982).

[13] Horak, W. and Kronert, G., "An object-oriented office document architecture model for processing and interchange of documents", Proc. 2nd ACM SIGOA Conf. (1984).

[14] Gibbs, S. and Tsichritzis, D., "A data modelling approach for office information systems", ACM Trans. Office Information Syst. Vol. 1(3), pp.299-319 (1983).

[15] Tsichritzis, D., Christodoulakis, S., Economopoulos, P., Faloutsos, C., Lee, A., Lee, D., Vandenbroek, J., and Woo, C., "A multimedia office filing system", Proc. Ninth Int. Conf. on Very Large Data Bases (1983).

[16] Tsichritzis, D., Thanos, C., Rabitti, F., Christodoulakis, S., Gibbs, S., Bertino, E., Fedeli, A., Faloutsos, C., Economopoulos, P., "Design isuues of a file server for multimedia documents", Proc. 1st ESPRIT Technical Week (Sept. 84).

[17] Meyrowitz, N. and van Dam, A., "Interactive editing systems: Part I and Part II", Comput. Surv. Vol. 14(3), pp.321-415 (Sept. 1982).

[18] Fraser, C.W., "Syntax-directed editing of general data structures", Proc. ACM Symposium on Text Manipulation, pp.17-21 (June 1981).

[19] Tsichritzis, D. and Lochovsky, F., "Data Models", Prentice-Hall, Englewood Cliffs, N.J. (1982).

[20] Chamberlin, D.D., King, J.C., Slutz, D.R., Todd, S.J.P, and Wade, B.W., "JANUS: An interactive system for document composition", Proc. ACM Symposium on Text Manipulation, pp.82-91 (June 1981).