Fermi

ESPRIT BRA Project N. 8134

Technical Report Series Fermi 4/96

A model for multimedia information retrieval

Yves Chiaramella,
Philippe Mulhem and
Franck Fourel

Department of Computing Science
University of Glasgow, Glasgow G12 8QQ
Scotland

Other Fermi participants:

Istituto di Elaborazione dell'Informazione del CNR - Pisa - Italy
LGI IMAG, Université J. Fourier - Grenoble - France
Informatik 6, Universität Dortmund - Dortmund - Germany

FERMI Technical Report
Fermi 4/96
July 1996

A model for multimedia information retrieval

Yves Chiaramella,
Philippe Mulhem and
Franck Fourel

Available from:
Fabio Crestani (Ed.)
Dept. of Computing Science
University of Glasgow
Glasgow G12 8QQ - Scotland

Phone: +44 141 330 4582
Fax: +44 141 330 4913
E - mail: fermi-reports@dcs.gla.ac.uk

A Model for Multimedia Information Retrieval

Yves Chiaramella, Philippe Mulhem, Franck Fourel
CLIPS
IMAG-Campus - BP 53
F-38041 Grenoble Cedex - France
E-mail: {Yves.Chiaramella,Philippe.Mulhem,Franck.Fourel}@imag.fr

July 4, 1996

1 Introduction

Representations of digitalized documents have considerably evolved with the integration of multimedia technology. Compared to textual documents, the content as well as the layout of multimedia documents have been extended to manage images, graphics, schemata, and more generally any multimedia information. Simultaneously, the way of consulting such documents evolved towards non-linear browsing based on hypertext and hypermedia links (which we will refer further as navigation links).

The model described here integrates two fundamentals aspects of this evolution of multimedia documents:

- the structure of documents,
- the management of multimedia data.

However it is important to notice that we do not claim here to provide a complete model of structured multimedia documents; we will instead focus on the aspects of such documents that have at least a potential interest from an Information Retrieval (IR) point of view.

1.1 The notion of structure and its impact on IR

The description of structured documents is the goal of existing norms like ODA [Hor85] and SGML [Bur94]. The origin of these two norms comes from similar approaches, but have led to different results. They provide a framework for the representation of structured documents allowing the management and the exchange of such information. Two types of document structures have been pointed out :

- the logical structure (present in SGML and ODA),

- the layout structure (present on ODA).

The former structure expresses the way a document is logically organized and reflects the discourse structure of the author(s). For instance a document contains a title, then a chapter having a title and several sub-chapters and so on, each of these parts containing an element of the discourse and having its own internal organization. The layout structure of a document defines the way it is presented to the user in terms of page presentation, font sizes, and so on. Though they refer to the same data, the two above structures are well separated : the layout structure provides an external view of the document, and the logical structure describes the internal structure of the document. Following the evolution of present standards already in use for storing and managing multimedia information, a primary goal of any model in the domain is to focus on structure. In the context of FERMI we have of course to investigate the impact of this notion when retrieving multimedia information:

- The impact of multimedia information. In the context of multimedia documents, documents combine several classes of media, each actual instance of them corresponding to a specific class of information. To this heterogeneity of multimedia information corresponds specific requirements about indexing and retrieving this data. This has been extensively studied in the former tasks of this work package for text, image and graphics [Mec95b, Par95, Meg95, Mec95a]. From this point of view alone, one cannot ignore that this heterogeneity of multimedia data is reflected within document structures, which has then to be integrated within retrieval models. Languages like SGML [Bur94] or Hytime [Erf94] provide a syntax for describe the logical structure of multimedia documents. The example of Fig. 1 shows a multimedia document comprising a title, two annotated images and a text. A logical structure usually corresponds to a tree whose nodes are the components of the document and whose edges implement the composition relationship. The root node of such a tree represents the whole document, and the leaf nodes correspond to atomic data defining the raw content of the document. One have to notice here that this notion of atomic data associated to leaves of the logical structure is related to this particular view of documents.

- The impact on semantic content. The notion of semantic contents of documents (i.e., their aboutness [BH94]) is of central importance in IR. In the framework of FERMI, models of the semantic content have been designed by M. Mechkour [Mec95b, Mec95c] and C. Meghini [Meg95] for 2D images, by F. Paradis for textual documents [Par95]and M. Mechkour for graphics [Mec95a].The structuration of a document as defined by the author(s) plays a central role when considering its semantic content because it corresponds to a discourse structure (i.e. an organization of the underlying knowledge that constitutes the document content). As indicated before, this important feature of documents is commonly expressed by the notion of *logicalstructure*. Based on this, we will be able to define the content of multimedia documents as a composition of information entities.

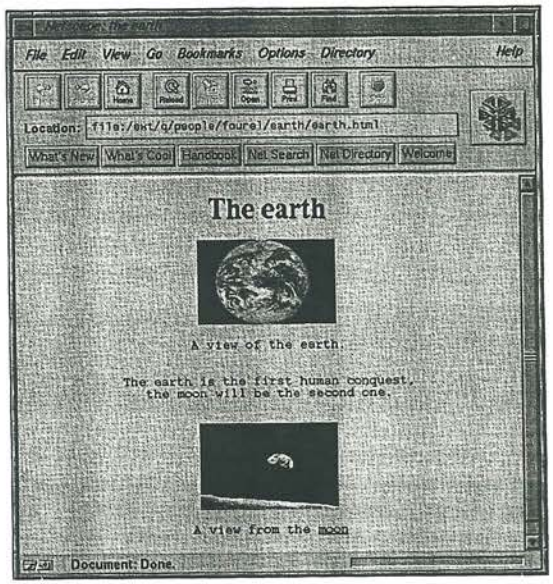

Figure 1: A multimedia structured document

- The impact on the corpus. In standard Information Retrieval, any document is usually considered as an atomic entity that can be indexed and retrieved as a whole by the system, and presented to the user as a query result. Said in other words, documents constitute the information units on which IR systems are based either when indexing or when retrieving. In the context of structured documents, index units, and consequently retrievable units, are document components instead of documents. In this context, the classical notion of corpus is extended to the set of all index units (i.e. potentially the set of all document components).

Finally, one usually associate to the notion of structured document the notion of *attribute*. Attributes are predefined, formatted data attached to documents or document elements belonging to a given class. They apply to the whole document, or document element they are related to, and are for instance names of authors, publication dates, etc. They have a rather simple structure and allow access to documents based more on "contextual information than based on their content. Despite this they are of course useful in the process of retrieving information; for example, one may use attributes to retrieve documents about "relevance feedback written by G. Salton.
Aside these classical attributes there exists an other class of attributes that may be

used to describe properties of discourse elements themselves, like for example the language of a word or of a sentence when it differs from the rest of the document. These attributes act as annotations and may be defined by the author(s) and/or by other persons. They may be used for retrieval (e.g. retrieve latin citations in a given corpus) and also to facilitate some processes (i.e. a classical text indexing program designed for english language would most probably be disoriented when processing english documents containing latin citations).

It should then be clear from this discussion that there is no longer a need to talk about "structured multimedia documents" in a context where the notion of structure cannot be ignored; the simpler terms of "multimedia document", or even "document", will from now refer to this notion.

1.2 Querying and Browsing Multimedia Documents

The notion of browsing has demonstrated the utility of hypermedia systems for organizing, storing and retrieving information. Users of such systems can browse across several pre-defined paths (links) to access information (within databases) that is organized into units of storage (nodes). Hypermedia systems are also user-friendly systems that provide nice interfaces and necessitate no particular system expertise from the user. However, experience has demonstrated that extensive browsing has its own limitations : in the general context of large, complex hyperbases this approach for retrieving information supposes a lot of time-consuming search by try-and-error, and users often have to face the well-known problem of orientation [Hal88, WB90]. After a while they may be lost in the network and need to know where they are, where to go to resume an effective browsing, and how to be properly relocated in the hyperbase.
Information Retrieval systems on their side provide powerful and effective access-by-content processes, but need more expertise from the users who have to master the indexing language which describes the stored information and the query language of the system. In this sense, IR systems often provide only poor user interfaces. More fundamentally, they are based on querying for retrieving information, and this is a strong limitation in terms of man-machine interaction : each query provides as its answer a set of documents, and the user cannot "see" anything else than this set of documents except by issuing a new query. But then, how to formulate it to improve the answer in the needed way ? Depending on the users expertise this often results in a long interaction process at the end of which he rarely knows wether he has found all the relevant information or not. On the other hand there is nothing like orientation problems in an IR session because the user is permanently asked to express his information need (i.e to make it explicit via a query).
Hence we believe with others [CT89, ACG91, LDH92, DR93] that querying and browsing are two complementary approaches that must be integrated to provide a more efficient and effective environment for accessing and retrieving multimedia information. As a consequence we think that we cannot ignore the existence of

browsing using navigation links in the design of a model for retrieving multimedia data

Navigation links are related to non-linear access to document components (nodes). Links relate a starting point, called **anchor**, to one or more **targets**. Although both anchors and targets are parts of documents, it is important to notice here that document components related to anchors and targets may or may not correspond to elements of the logical structure. As an example, an anchor may be associated to a word (which is then necessarily part of a leave in the logical structure) and the target may be a sentence located in an other leave of the logical structure. In the example of Fig. 1, the underlined word *moon* is the anchor of a navigation link related to target information about the moon. A navigation link also includes a process that implements the access from the anchor to the target(s). The nature of the information accessed via links depends on the semantics of the links which is only denoted by the anchors (e.g. the underlined word *moon*in fig. 1). But nothing prevents errors such as assigning a picture of Mars to the anchor *moon*. On the other hand, even if the target information is consistent with the anchor definition, the user usually cannot predict the kind of information that constitute the hidden target. It could be a picture of the moon, a survey covering everything known about the moon, a bibliography about the moon etc. A typology of navigation links (e.g. *illustration, explanation, documentation*types of link) could help the user while deciding to activate or ignore and anchor and thus would be helpful to prevent useless browsing and disorientation. The HTML language [BLC, BLC93], for instance, does not provide a typology for navigation links, and the underlying hypothesis is that human beings are able to properly use these links based only on anchors and on their context of occurrence within the document. In our opinion this hypothesis is too weak and significantly contributes to problems like disorientation and cognitive overload that often affect hypermedia systems. Models for multimedia documents should thus take into account the potential richness of typed links, either for browsing or for querying.

1.3 Rationale

The integration of all the aspects developed above is our prime goal when designing a model for retrieving multimedia data. Our main guidelines when addressing these problems will be the following:

- Combining Querying and Browsing
  At the present time, the available models provide either a pure database approach [AC95, Boh95] of multimedia documents, or an hypermedia approach for structured documents [AMC95]. The first approach, usually based on an object-oriented framework, concentrates on structure, but lacks a suitable representation of the semantic content of the documents. This implies poor retrieval capabilities. The hypermedia approach manages structured documents

and navigation links. Like the database approach however, the hypermedia approach often does not provide content-based querying (sometimes it does not at all provide such facilities, like the World Wide Web for example, where the numerous links starting from HTML pages can easily disorient the users). The approach proposed here is based on a solution proposed by Kheirbek and Chiaramella in [KC95] which integrates both approaches (i.e. querying and browsing). The basic principle that underlies this integrated model is a new definition of *domain knowledge* of an application encompassing *structural knowledge* and *content knowledge*. This will allow to manage the logical structure of documents, the navigation links, the semantic contents and attributes of document components. An important and original aspect of this model is also the integration of the previous results of the FERMI project to provide a complete and consistent model for multimedia documents.

- Definition and Use of Links
  The definition and the use of navigation links in hypermedia and IR systems may look like an egg-and-chicken problem: while the hypermedia community tries to use IR techniques to help building navigation links, the IR community takes them as given and tries to use them for improving retrieval performances. There is no longer a paradox when observing that these approaches in fact do not consider the same classes of links: hypermedia investigates how IR could help building similar-content links (i.e. classes of linked documents having similar contents), while IR investigates how reference links, or aggregative links of the logical structure, could help improving retrieval. This is precisely what we will do in this study.

- Representation and Retrieval of Multimedia Documents
  We consider that documents cannot be considered as atomic entities any longer; a query language has to allow retrieval of document components as well as of documents as a whole, and this process has to offer selection criteria ranging from attribute selection to content selection. We propose an approach inspired from MULTOS [Tha90]. The MULTOS document model considers two kinds of components: the active ones and the passive ones. Active components can be retrieved using their semantic content, while passive components can be queried only based on their existence. Consider, in the document in Fig. 1, that the first image is a passive component, and that the second image is an active component. The second image (active) may be retrieved using queries like : "Retrieve documents containing an image of the earth viewed from the moon". On the other hand, the first image (passive) may be retrieved from queries like : "Retrieve documents containing an image". The only property checked here is the existence of an image object in the retrieved documents. Active components can also be used as passive when needed. We want to provide a model that encompasses these two classes of components in a uniform way. To achieve this goal, we integrate the models of Mechkour [Mec95b, MBC95] and Paradis [Par95].

- Semantic Content and Structured Documents
  If we consider a document A aggregating two components B and C, a "natural" approach is to consider that the semantic content of A depends on those of B and C. This hypothesis is similar to a cartesian approach used in programming: a program composed of several modules processes its task as a composition of sub-tasks, and the whole program does what its parts do. In fact, this description has to be refined; one could observe that the whole program does not necessary use all the exported functionalities of its modules, and thus that the program does not exactly sum up the potentialities of its components. Similarly, considering the whole content of a multimedia document as an aggregation of the content of its components is an approximation: some components may include for example references to other documents corresponding to adding implicit (i.e. not directly available) content to the referencing document (e.g. a reference to an other document containing the definition of a concept). As an acceptable approximation of this difficult problem (related in fact to the problem of indexing which is out of the scope of FERMI) we will (recursively) consider that the explicit content of a composite document is an aggregation of the explicit content of its components, and that the explicit content of an atomic element is strictly limited to its local data (i.e. ignores the possible external references). We will have however to define the nature and properties of this aggregation to be able to design an algorithm that allows the retrieval of document components in a consistent way.

The proposed model aims at integrating these different aspects, while keeping in mind that the composite nature of a multimedia document induces the necessity of being also able to individually retrieve elements of each class of information that are aggregated in multimedia documents. In the framework of FERMI our goal then is not to design a new, complete framework for the representation of multimedia documents, but instead to concentrate on the aspects discussed above and which have a major impact in the retrieving of multimedia documents. This will in turn lead us to some developments that are not part of the existing representation norms (in particular the view mechanism); this means that this non-sandard information will have to be produced, after the initial loading of documents in the database, by specific processes related to the indexing of documents.

In the next section we describe the proposed model for multimedia document (part 2, page 7). Section 3 (page 26) is dedicated to the indexing of multimedia documents.

2 A Model for Retrieving Multimedia Documents

2.1 Introduction

The notion of structured, multimedia document includes many aspects which, when considered as a whole, make the modelling of such information a harduous and diffi-

cult task. This is the main reason why we have chosen to study the proposed model considering three complementary classes of information that we consider of prime interest for defining and retrieving multimedia documents. This approach is close to the one used within Workparts WP3.1, WP3.2 and WP3.3 of FERMI for modelling single-media data by M. Mechkour [Mec95b, Mec95c] and C. Meghini [Meg95] for 2D images, by F. Paradis for textual documents [Par95], and M. Mechkour for graphics [Mec95a]. For each media these views correspond to specific features each of them contributing in a specific way to the overall notion of "document content".

A first problem now is about how to integrate these specialized models in the framework of a multimedia document ("multimedia" refers here to any combination of these three single medias). The basic principle about this integration is to consider that each single-media model generates specific attributes called *Content Attributes* assigned to leaf-nodes of the logical structure of documents, and then to define the needed properties of these attributes for retrieving *document components*. This leads us to view *logical structure* and *attributes* as two classes of information to be investigated. An other important hypothesis here is that one cannot consider multimedia information retrieval as a process based only on querying; in our opinion, browsing using *navigation links* is a complementary approach that has to be considered and integrated in the model. The access structure then constitutes the third class of information underlying the proposed model.

Let us first introduce informally these three complementary classes of information, next section (see 2.2) gives their formal definition and the global constraints ensuring their mutual consistency (and thus the overall coherence of the model).

The Logical Structure: The logical structure of documents is viewed in a much classical way as a hierarchy of *structural objects* whose leaves correspond to instances of single-media data models. This structure plays a central role in our approach because: i) it implements an explicit organization of the discourse, and consequently of the document's semantic content, and ii) all the other classes of information are related to the logical structure when defining basic properties needed for indexing and retrieving multimedia information. As an example of these dependencies, we will see later that the semantic content of any document component is defined as a composition of its logical structure and of the index expressions assigned to its component elements. The logical structure includes the definition of a standard order of the discourse, as designed by the author(s) for a linear, complete, consultation of the document. For instance, this allows to state that a part of document is "before" another one with respect to a standard consulting order. In the process of retrieval, this information can help in discriminating relevant answers (e.g. retrieving an image located "after" a given section, where "after" refers to the standard consultation order).

The Attributes: Attributes are viewed here from a more general point of view than usual.They refer of course to classical attributes such as authors, dates etc., but also to more complex information such as *index expressions* describing the semantic content of document components. We consider then that attribute values are in general not atomic, and are *expressions* of a given language, while *attribute domains* correspond to these languages. As an example, the above-mentioned "index expressions" constitute values of an attribute named *Symbolic* and belong to an index language LI (a set of index expressions) which is the domain of attribute *Symbolic*. The model includes the case of multi-valued attributes in which case attribute values are sets of expressions. Attributes are then used for defining any kind of information attached to document components (i.e. elements of the logical structure), and then to define all properties related to the notion of document content. An important aspect of the model is the consideration of properties induced by attributes on constituants of the logical structure; we refer to this as: *Attribute Propagation*. These properties are the basis of further retrieval mechanisms for accessing document components.

The Navigation Structure: Access is considered here in a general way: it integrates all aspects in the *representation* of documents that are intended to help users accessing document components. This excludes relations that remain purely symbolic, and are usable only manually (i.e. without standard specific support by the system). An example of this distinction is provided by bibliographic references: they may remain purely symbolic (i.e. for accessing a full reference from its citation, a user has to manually browse the bibliography), or they may be part of the access structure, in which case a direct access from the citation to the actual reference is provided by the system (via a navigation link or any other access method). The navigation structure inludes all navigation links that may exist between documents and document components and which usually correspond to classifications and/or non-linear browsing structures within the corpus.

2.2 Formal Definition of the Model

While presenting the formal definition of each of the individual information classes introduced above, we will use the following notations:

- upper case identifiers represent sets,
- lower case identifiers represent function names,
- notation 2^N refers to the powerset of N,
- notation 2^{N+} denotes the powerset of N, not including the empty set.
- whenever a relation R corresponds to an order, we note it $\preceq_r$.

2.2.1 The Logical Structure

The logical structure represents document components in the form of typed *structural objects*, and defines the aggregative relation combining them. Structural objects correspond to basic entities when considering physical and semantic contents of documents. The logical structure describes a hierarchical aggregation of structural objects:

$$\mathbf{LS} = (OS,\ \preceq_{str}, \prec_{seq},\ TYPE_{ST},\ \preceq_{tst},\ type_{st},\ TYPE_M,\ type_m)$$

where:

OS : is the finite set of document structural objects, noted os_i.

$\preceq_{str}$: is an aggregative relation between structural objects that defines their hierarchical composition:

$$\preceq_{str} \subset OS \times OS$$

where the first element of each tuple corresponds to the *aggregating object*, and the second element corresponds to the *aggregated (i.e. component) object.* This relation defines a hierarchical partial order on OS:

- $\preceq_{str}$ is partial order on OS and thus is reflexive, asymmetric and transitive.
- the order is hierarchical:
$$\forall os_i, os_j, os_k \in OS,$$
$$os_i \preceq_{str} os_k \text{ and } os_j \preceq_{str} os_k \Rightarrow (os_i \preceq_{str} os_j \text{ or } os_j \preceq_{str} os_i)$$
- or is the unique minimal element of the hierarchical order:
$$\forall os_i \in OS,\ os_i \preceq_{str} or \Rightarrow os_i = or$$

$\prec_{seq}$: this relation defines a linear sequence on OS, and corresponds to the standard, linear order to access components already mentioned in the introduction. When considering logical structures as hierarchies, this relation might for example correspond to a prefixed access-sequence of structural nodes. In the general case however, the only basic constraint is that $\prec_{seq}$ corresponds to a total, strict order on structural components:

$$\prec_{seq} \subseteq OS \times OS$$

- $\prec_{seq}$ is irreflexive, asymmetric and transitive.
- For any tuple (os_i, os_j) of this relation, os_i is the *origin* of the tuple, while os_j is the *extremity* of the tuple.

- we note os_{first} and os_{last} the unique minimal and maximal elements of this order.

$TYPE_{ST}$: is the set of types of structural objects. These types can be derived from SGML descriptions of documents [Bur94]. For instance, for a description of books, $TYPE_{ST}$ can be $\{Document, Chapter, Section, Sub-Section, Paragraph, Figure\}$. Types of structural objects usually correspond to various *abstraction levels* which are used for organizing the logical structure of documents, and ease their understanding and manipulation. These abstraction levels are defined by a *partial order* on the structural object types (see $\preceq_{tst}$ below).

$\preceq_{tst}$: is a relation on structural object types that defines the hierarchy of abstraction levels used for a given document base; this order behaves like a hierarchy of classes:

$$\preceq_{tst} \subseteq TYPE_{ST} \times TYPE_{ST}$$

The properties are similar to those of $\preceq_{str}$, except that the order allows the existence of several minimal elements corresponding to possible different document types (e.g. letters, books etc.):

- $\preceq_{tst}$ is partial order on $TYPE_{ST}$ and thus is reflexive, asymmetric and transitive.
- the order is hierarchical:
$$\forall ts_i, ts_j, ts_k \in TYPE_{ST}$$
$$, ts_i \preceq_{tst} ts_k \text{ and } ts_j \preceq_{tst} ts_k \Rightarrow (ts_i \preceq_{tst} ts_j \text{ or } ts_j \preceq_{tst} ts_i)$$

$type_{st}$: is a total, surjective function assigning to each structural object a structural type of $TYPE_{ST}$:

$$type_{st} : OS \rightarrow TYPE_{ST}$$

$TYPE_M$: is a set of media types, with $TYPE_M = \{text, image, graphic, multimedia\}$. These types define, for any structural object, the type of media associated to its physical content (see "Constraints" below).

$type_m$: is a total function assigning to each structural object its media type in $TYPE_M$:

$$type_m : OS \rightarrow TYPE_M$$

To simplify further definitions and notations we introduce here some useful functions based on the orders $\preceq_{str}$ and $\prec_{seq}$ introduced before:

1. We note $Desc_{str}(os_i)$ the set of all component objects of $os_i \in OS$:

$$Desc_{str} : OS \rightarrow 2^{OS}$$

with:

$$\forall os_i \in OS, Desc_{str}(os_i) = \{os_j \in OS \mid os_i \neq os_j \ \& \ os_i \preceq_{str} os_j\}$$

2. We note $LOS \subseteq OS$ the subset of structural objects that are maximal elements considering the partial order $\preceq_{str}$ (i.e. leaves of the logical structure):

$$LOS = \{os_i \in OS \mid Desc_{str}(os_i) = \emptyset\} \subseteq OS$$

3. we note $NextDesc_{str}(os_i)$ a restriction of $Desc_{str}(os_i)$ to the closest elements os_j of OS satisfying $os_i \preceq os_j$:

$$NextDesc_{str} : OS \rightarrow OS$$

with:

$$\forall os_i, os_k \in OS, NextDesc_{str}(os_i) =$$
$$\{os_j \in OS \mid os_i \preceq_{str} os_k \preceq_{str} os_j \Rightarrow (os_k = os_i \ or \ os_k = os_j)\}$$

Note that for all $os_i \in LOS$, $NextDesc_{str}(os_i) = \emptyset$.

4. we note $Next_{seq}(os_i)$ a function giving the closest element to os_i according to the order $\prec_{seq}$, os_{last} being the maximal element of this order, we have then:

$$Next_{seq} : OS - \{os_{last}\} \rightarrow OS - \{os_{first}\}$$

with:

$$\forall os_i \in OS - os_{last}, os_j \in OS, Next_{seq}(os_i) = os_j \Leftrightarrow$$
$$os_i \prec_{seq} os_j \ \& \ \nexists os_k \mid os_i \prec_{seq} os_k \ \& \ os_k \prec_{seq} os_j$$

CONSTRAINTS: we define here the additional constraints needed for ensuring the consistency of structural components:

1. Consistency between aggregation of structural objects and abstraction levels: when two structural objects are related by the aggregation relation $\preceq_{str}$, their types must conform to relation $\preceq_{tst}$ on types :

$$\forall os_i, os_j \in OS, os_i \preceq_{str} os_j \Rightarrow type_{st}(os_i) \preceq_{tst} type_{st}(os_j)$$

Note that since function $type_{st}$ is surjective, the above constraint has as a corollary that the type of the root structural element is necessarily a minimal type of $TYPE_{ST}$ according to $\preceq_{tst}$ (i.e. a type corresponding to a maximal abstraction level).

2. Consistency between aggregation of structural objects and media types: the **multimedia** type applies only to structural objects whose components are at least of two different media types. As a consequence type **multimedia** does not apply for leaves of the logical structure which are considered as single-media elements; this is of course a convention in the model which does not prevent its extension to other medias (see comment below). On the contrary, types **text**, **image** and **graphic** apply only to structural objects whose components are all of the same media type:

$\forall os_i \in OS,$
$(\exists os_j, os_k \in Desc_{str}(os_i) \ \& \ type_m(os_j) \neq type_m(os_k)) \Leftrightarrow type_m(os_i) = \textbf{multimedia}$
$\forall os_i \in OS, (\forall os_j \in Desc_{str}(os_i), type_m(os_i) = \textbf{text}) \Leftrightarrow type_m(os_j) = \textbf{text}$
$\forall os_i \in OS, (\forall os_j \in Desc_{str}(os_i), type_m(os_i) = \textbf{image}) \Leftrightarrow type_m(os_j) = \textbf{image}$
$\forall os_i \in OS, (\forall os_j \in Desc_{str}(os_i), type_m(os_i) = \textbf{graphic}) \Leftrightarrow type_m(os_j) = \textbf{graphic}$

Would the model be extended to other medias like **video-sequence**, this means that **video-sequence** would be viewed as a single media type, even if composite in its nature; the video information and the sound information associated to this type would be modelled as distinct views of a model named $\mathcal{M}_{video-sequence}$. Note also that these definitions allow the modelling of structured, single-media documents as for example structured textual documents (in which case all their component units are of type *text*).

2.2.2 Attributes

We consider here attributes in a uniform and general way; the model associates to attribute names attribute values that have to conform to the domains defined for attribute names. As introduced before, each attribute domain is a set of *expressions* of a given language having its syntax and its semantics; this allows to consider in a very general way the great variety of attributes that may be encountered in actual applications. As will be seen later, this also allows the definition of *attribute classes* which constitute a powerful property when considering multimedia information retrieval. This class of information is modelled as:

$$\mathcal{A} = (OS, NAME_A, VALUE_A, name_a, domain_a, value_a, SM)$$

where:

OS : is the set of structural objects in the document, noted os_i.

NAME$_A$: is the set of attributes names.

VALUE$_A$: is the set of all possible attribute values. This set is the union of all the domain languages of all attributes (see below).

name_a : is a partial function that associates to structural objects a non-empty set of attribute names:

$$name_a : OS \rightarrow 2^{NAME_A+}$$

domain_a : is a total function defining the domain of any attribute name (i.e. all the expressions of its associated language):

$$domain_a : NAME_A \rightarrow 2^{VALUE_A+}$$

value_a : is a partial function assigning to structural objects the value of a related attribute name; the definition allows multi-valued attributes:

$$value_a : OS \times NAME_A \rightarrow 2^{VALUE_A+}$$

Singletons : to simplify notations when considering singletons of attribute values, we will use function *elem* that applies to the power set of any set A:

$$elem : 2^A \rightarrow A$$

with:

$$\forall e \in 2^A, elem(e) = e$$

SM : a set of *single-media models* defining the various abstractions of single-media data. We develop this notion in the next section (see 2.2.3)

CONSTRAINTS: the only constraint for ensuring the consistency of attributes is given below, where α is a metasymbol for any attribute name:

- For every structural object, the attribute value for a given associated attribute name belongs to the domain of this attribute:

$$\forall os_i \in OS,\ \forall \alpha \in name_a(os_i),\ value_a(os_i,\ \alpha) \subseteq domain_a(\alpha)$$

Given these definitions we may now consider some additional features of attributes which are important in the context of multimedia information retrieval.

2.2.3 Integration of Single-Media Models

As said in the introduction, this integration is done in considering views of the single-media models as attributes of leaves of the logical structure (i.e. single-media nodes, as defined before). To ensure this integration we have first to introduce in a formal way how we view these single-media models at the abstraction level of multimedia documents.

2.2.3.1 Single-Media Models

We refer from now to these models as a set $SM = \{\mathcal{M}_{text}, \mathcal{M}_{image}, \mathcal{M}_{graphic}\}$, and we will note $\mathcal{M}_\mu$ any element of SM. We call *instances* of these models any abstraction obtained while applying one of them to actual (single-media) data. The process that produces model instances is the *index process*, and one may then intuitively consider that each model is associated to a function defined on single-media objects and having complex values corresponding to the various abstractions it generates. As an example, from an image i the proposed model $\mathcal{M}_{image}$ (see [Mec95b, Mec95c]) gives a tuple aggregating the five views that are part of the standard image model: $\mathcal{M}_{image}(i) = (Ph_i, St_i, Sp_i, Sy_i, Pe_i)$, where Ph_i is the *physical view* of image i, St_i is its *structural view*, Sp_i its *spatial view*, Sy_i its *symbolic view*, and Pe_i its perceptive view.

The correspondance between single-media models and their associated set of views is defined in considering each model $\mathcal{M}_\mu$ as a set of elementary functions $\mathcal{M}_\mu^\alpha$ defined on leaves of the logical structure (i.e. LOS), and producing a particular instance of view α for the considered media μ. According to the convention mentioned before, these instances are all elements of particular languages noted $\mathcal{L}_\mu^\alpha$:

$$\mathcal{M}_\mu = \{\mathcal{M}_\mu^\alpha\}$$

and:

$$\mathcal{M}_\mu^\alpha : LOS \rightarrow \mathcal{L}_\mu^\alpha$$

We may then instanciate these generic definitions to the actual models designed for the three medias considered in the framework of FERMI:

1. *Model for Texts:*

$$\mathcal{M}_{text} = \{\mathcal{M}_{text}^{physical}, \mathcal{M}_{text}^{structural}, \mathcal{M}_{text}^{symbolic}\}$$

with:

$$\mathcal{M}_{text}^{physical} : LOS \rightarrow \mathcal{L}_{text}^{physical}$$
$$\mathcal{M}_{text}^{structural} : LOS \rightarrow \mathcal{L}_{text}^{structural}$$
$$\mathcal{M}_{text}^{symbolic} : LOS \rightarrow \mathcal{L}_{text}^{symbolic}$$

2. *Model for Images:*

$$\mathcal{M}_{image} = \{\mathcal{M}_{image}^{physical}, \mathcal{M}_{image}^{structural}, \mathcal{M}_{image}^{symbolic}, \mathcal{M}_{image}^{spatial}, \mathcal{M}_{image}^{perceptive}\}$$

with:

$$\mathcal{M}_{image}^{physical} : LOS \rightarrow \mathcal{L}_{image}^{physical}$$
$$\mathcal{M}_{image}^{structural} : LOS \rightarrow \mathcal{L}_{image}^{structural}$$
$$\mathcal{M}_{image}^{symbolic} : LOS \rightarrow \mathcal{L}_{image}^{symbolic}$$
$$\mathcal{M}_{image}^{spatial} : LOS \rightarrow \mathcal{L}_{image}^{spatial}$$
$$\mathcal{M}_{image}^{perceptive} : LOS \rightarrow \mathcal{L}_{image}^{perceptive}$$

3. *Model for Graphics:*

$$\mathcal{M}_{graphic} = \{\mathcal{M}^{physical}_{graphic}, \mathcal{M}^{structural}_{graphic}, \mathcal{M}^{symbolic}_{graphic}, \mathcal{M}^{perceptive}_{graphic}\}$$

with:

$$\mathcal{M}^{physical}_{graphic} : LOS \rightarrow \mathcal{L}^{physical}_{graphic}$$
$$\mathcal{M}^{structural}_{graphic} : LOS \rightarrow \mathcal{L}^{structural}_{graphic}$$
$$\mathcal{M}^{symbolic}_{graphic} : LOS \rightarrow \mathcal{L}^{symbolic}_{graphic}$$
$$\mathcal{M}^{spatial}_{graphic} : LOS \rightarrow \mathcal{L}^{spatial}_{graphic}$$
$$\mathcal{M}^{perceptive}_{graphic} : LOS \rightarrow \mathcal{L}^{perceptive}_{graphic}$$

An important point in the context of logic-based information retrieval, is that all languages $\mathcal{L}^{\alpha}_{\mu}$ correspond to *logical expressions, or closed sentences of a logic*: their syntax and semantics are at the moment those defined for every media. An important aim of WP4 is then to integrate these preliminary models in the single formalism of a Multimedia Information Retrieval Logic; MIRTL (see [Seb94a, Seb94b]) is of course a good candidate for this, though various aspects can also be experimented based on Four-Valued Datalog (see [RF96, TR96]) and Conceptual Graphs (see [Mec95c]).

2.2.3.2 Content Attributes

Considered as a whole, the three single-media models $\mathcal{M}_{\mu}$ involve five types of views:

- *the physical view*, which is used in the three models,
- *the structural view*, which is used in the three models,
- *the symbolic view*, which is used in the three models,
- *the spatial view*, which is used only in the image and graphic models,
- *the perceptive view*, which is used only in the image and graphic models,

To these five views we may associate five *standard attribute names*, called *Content Attributes*, that are elements of $NAME_A$:

- **physical**, for the physical view,
- **structural**, for the structural view,
- **symbolic**, for the symbolic view,
- **spatial**, for the spatial view,
- **perceptive**, for the perceptive view.

Values of these attributes are those of view-functions $\mathcal{M}^{\alpha}_{\mu}$ described before, and which are assigned to leaves of the logical structure. We may then associate sets of Content Attributes noted $\mathcal{A}_{\mu}$ to the various single-media models $\mathcal{M}_{\mu}$:

- $\mathcal{A}_{text} = \{physical, structural, symbolic\}$, for the text model,
- $\mathcal{A}_{image} = \{physical, structural, symbolic, spatial, perceptive\}$, for the image model,
- $\mathcal{A}_{graphic} = \{physical, structural, symbolic, spatial, perceptive\}$, for the graphic model.

The overall consistency between the definitions of logical structure, attributes and single-media models is then defined by the following generic constraint where μ stands for any single media in $\{text, image, graphic\}$, and β stands for any Content Attribute of $\{physical, structural, symbolic, spatial, perceptive\}$:

$$\forall os_i \in LOS, type_m(os_i) = \mu \Rightarrow$$
$$\forall \beta \in \mathcal{A}_{\mu}, \beta \in name_a(os_i) \;\&\; value_a(os_i, \beta) = \mathcal{M}^{\beta}_{\mu}(os_i)$$

As an example, one may instanciate this constraint for textual information (where $\mathcal{A}_{text} = \{physical, structural, symbolic\}$); the same principle applies for the two other medias:

$$\forall os_i \in LOS, type_m(os_i) = \textbf{text} \Rightarrow$$
$$\textbf{physical} \in name_a(os_i) \;\&\; value_a(os_i, \textbf{physical}) = \mathcal{M}^{physical}_{text}(os_i)$$
and
$$\textbf{structural} \in name_a(os_i) \;\&\; value_a(os_i, \textbf{structural}) = \mathcal{M}^{structural}_{text}(os_i)$$
and
$$\textbf{symbolic} \in name_a(os_i) \;\&\; value_a(os_i, \textbf{symbolic}) = \mathcal{M}^{symbolic}_{text}(os_i)$$

Finally, it may be inferred from the previous definitions that *domains* of Content Attributes are composite languages defined as unions of the sub-languages that are the value sets of related *view functions* (i.e. in the general case, distinct sub-languages):

$$domain_a(\textbf{physical}) = \mathcal{L}^{physical} = \mathcal{L}^{physical}_{text} \cup \mathcal{L}^{physical}_{image} \cup \mathcal{L}^{physical}_{graphic}$$
$$domain_a(\textbf{structural}) = \mathcal{L}^{structural} = \mathcal{L}^{structural}_{text} \cup \mathcal{L}^{structural}_{image} \cup \mathcal{L}^{structural}_{graphic}$$
$$domain_a(\textbf{symbolic}) = \mathcal{L}^{symbolic} = \mathcal{L}^{symbolic}_{text} \cup \mathcal{L}^{symbolic}_{image} \cup \mathcal{L}^{symbolic}_{graphic}$$
$$domain_a(\textbf{spatial}) = \mathcal{L}^{spatial} = \mathcal{L}^{spatial}_{image} \cup \mathcal{L}^{spatial}_{graphic}$$
$$domain_a(\textbf{perceptive}) = \mathcal{L}^{perceptive} = \mathcal{L}^{perceptive}_{image} \cup \mathcal{L}^{perceptive}_{graphic}$$

If this multiplicity of languages and sub-languages does not simplify the model, one can foresee that its major impact will be when defining the query language and when implementing the model. From what has been done in the previous Work Parts about single-medias one may assert that the only real problem is about $\mathcal{L}^{physical}$: in this case there is no possible unifying point of view since all these medias correspond to different standards imposing their own syntax and semantics. All other languages correspond in fact to expressions of specific abstractions (views) of specific classes of physical data; considering this, it is then possible to define $\mathcal{L}^{structural}$, $\mathcal{L}^{symbolic}$, $\mathcal{L}^{spatial}$ and $\mathcal{L}^{perceptive}$ in a much uniform way, whatever the considered media. This means that the union of sub-languages related to the same type of view may be attempted in merging their syntax and their semantics, instead of simply considering the union of expressions produced by distinct sub-languages. This is in fact what is attempted for $\mathcal{L}^{symbolic}$ where all kinds of content concepts from any view may be expressed in a unique language. For $\mathcal{L}^{structural}$ this is obviously possible because for all medias, structural views are based on the same hierarchical scheme of aggregated components. $\mathcal{L}^{spatial}$ and $\mathcal{L}^{perceptive}$ apply only for images and graphics; in both case they express much similar notions and it should then be also relatively easy to merge each of their sub-languages into single ones.

2.2.3.3 Attribute Classes

Attributes usually correspond to properties assigned to elements of the logical structure (e.g. structural object os_i has author **Smith**). When considering structured documents, one have to address the problem of possible *propagation* of these properties among related structural objects of the logical structure; said in other words, one have to define what *inferences* may be defined on these attributes. If we consider for example to the classical case of attribute *Author*, and given an assigned value of this attribute to a structural object os_i, an intuitive asumption is that this property applies to all component objects of os_i (if any):

$$\forall os_i \in OS, \forall os_j \in Desc_{str}(os_i), value_a(os_i, \textbf{author}) = value_a(os_j, \textbf{author})$$

Of course the reality is more complex: there are multi-author documents (e.g. conference proceedings, encyclopediae etc.), and the inheritance mechanism of attribute *Author* in the logical structure is in fact not so obvious: the definition given above holds to some extent, but does not provide any notion of co-authoring. In our opinion it is important to integrate such properties in the model to allow proper retrieval of document components based on attribute values. To address this problem we have chosen to define *classes of attributes*, these classes being defined by a common behaviour considering inheritance of *attribute values* (which has not to be confused with the inheritance of *attribute names*). Considering the extreme variety of attributes that can be used in actual applications, it is of course difficult to foresee a complete classification based on propagation of attribute values; we will then limitate ourselves to three broad classes which, in our opinion, encompass most of the cases. Of course, this classification also applies to the *Content Attributes* introduced

above. In the following discussion we use again α as a metasymbol for any attribute name.

1. **Dynamic Attributes:** these attributes propagate their values in the logical structure. This means that if some attibute α of this class has a defined value $\mathbf{v}$ for a given structural object os_i (i.e. $value_a(os_i, \alpha) = \mathbf{v}$), one may infer the values of the same attibute for some other structural objects os_j related to os_i in the logical structure (i.e. $value_a(os_j, \alpha) = f(\mathbf{v})$, where $f(\mathbf{v})$ symbolizes this dependancy). Modelling this class of attributes then implies the definition of a *propagation condition* (i.e. *in which condition* attribute values may propagate), and an *assignation operation* (i.e. *how* propagated values apply to related structural objects). We consider here two subclasses of Dynamic attributes: *Descending Dynamic Attributes* and *Ascending Dynamic Attributes*.

   - **a. Descending Dynamic Attributes (DDA):** Considering the order of the logical structure, these attributes propagate values from top to bottom; the assignation mechanism is here an operator which computes the attribute values of component objects of $os_i \in OS$; we note it $\otimes_\alpha$. In a more formal way, the following defnition applies to every instance of operators $\otimes_\alpha$:

     $$\otimes_\alpha : \mathcal{L}^\alpha \rightarrow \mathcal{L}^\alpha$$

     where $\mathcal{L}^\alpha$ is the domain of attribute α. The descending propagation of attribute values is then defined as:

     $$\forall os_i \in OS, \forall os_j \in Desc_{str}(os_i), value_a(os_j, \alpha) = \otimes_\alpha(value_a(os_i, \alpha))$$

     An example of attribute belonging to this class is publication date (noted *Pub-Date*), the value of which applies to every component of a document; in this case, the operator $\otimes_{pub-date}$ is simply *Copy*:

     $$\forall os_i \in OS, \forall os_j \in Desc_{str}(os_i), \\ value_a(os_j, Pub-Date) = Copy(value_a(os_i, Pub-Date))$$

     There are more complex examples of DDAs, like for example the numbering of document components where each component number depends on the number of its embedding component.

   - **b. Ascending Dynamic Attributes:** Considering the order of the logical structure, these attributes propagate values from bottom to top;

in this case, the assignation affects the attribute value of common parent-components. The assignation operator $\oplus_\alpha$ corresponds here to some *aggregation* whose definition depends on the considered attribute α. We may then define this class as the set of all $\alpha \in NAME_A$ such as:

$$\forall os_i \in OS, \forall os_j \in D_i = Desc_{str}(os_i), value_a(os_i, \alpha) = \bigoplus_{os_j \in D_i}^{n}{}_{\alpha} value_a(os_j, \alpha)$$

An example of such attributes is *Author*: if two distinct components os_{11} and os_{12} of a single structural object os_1 are assigned different author names n_1 and n_2, then the infered value of *Author* for os_1 is some aggregation of the two author names that models the notion of co-authoring (for example a set: $\{n_1, n_2\}$).
When assigning any attribute α to this class, one have obviouly to define the properties of the corresponding operator $\oplus_\alpha$, and particularly its associativity (as suggested by the generalized notation used in the definition above). We will discuss later (see 3) an important example of such a definition for Content Attribute *symbolic*.

2. **Static Attributes:** static attributes do not propagate their values in the logical structure; they correspond to properties that remain purely local to the structural object they are assigned to. An example of such attributes is *Title* which applies to a structural object, but neither to its possible components nor to its possible parents in the logical structure. Note that this not prevent several structural components to share the same title, even if these elements are related in the logical structure; this may occur only if these components have been *assigned* the same title for some reason, but is not due to *inheritance* of attribute values. This simply means that, for any attribute of this class, there is no propagation condition and no affectation operator.

2.2.3.4 Classification of Content Attributes

To achieve the integration of these models in the multimedia model we have now to define their properties about propagation of attribute values in the logical structure; this is of course related to the notion of *inference* which underlies the retrieval of structural components. From this point of view, it is important to remind here that content attributes are until now assigned only to leaves of the logical structure by functions $\mathcal{M}_\mu^\alpha$; without any additional properties, it is then impossible to compute any inference involving these attributes. We have chosen to define these needed additional properties of content attributes simply by classifying them in the three attribute classes defined above; the choice here is generic:

- All content attributes are ascending, dynamic (ADA).

This means that we have to define for each of them its corresponding *aggregation operator* $\oplus_\mu$ and its properties considering inference; we will develop this point in a

specific section dedicated to all querying aspects, and in particular to the various inferences involved in this process. An important example of such specification will also be found before in section 3 where we consider the specific case of operator $\oplus_{symbolic}$ used for aggregating index expressions.

In the context of logic-based information retrieval, we have already observed that all domain values of attributes (including of course Content Attributes) must be viewed as logical expressions belonging to given languages. This refers to the particular syntax and semantics of the logic developped within FERMI; operators $\oplus_\mu$ are then considered as *internal operators* of this language, matching the following definition:

$$\oplus_\mu : \mathcal{L}^\mu \times \mathcal{L}^\mu \rightarrow \mathcal{L}^\mu$$

2.2.4 Document Model, Document Base and Hyperbase

We may for now complete our model with

2.2.4.1 Document Model

The *model for multimedia document* may now be defined as a structure combining the two classes of information presented above:

$$\mathcal{D} = (\mathcal{LS}, \mathcal{A})$$

A *multimedia document* is then an instance of this model; such an instance is any assignation of actual objects, sets, relations and functions as defined in $\mathcal{D}$. We note document instances $d_i = (LS_i, A_i)$, and $\chi_\mathcal{D}(d_i)$ a predicate evaluating its conformity with the document model (i.e. checking all the integrity constraints defined for $\mathcal{LS}$ and $\mathcal{A}$). For simplicity, when referring to document instances we will use the same notations as those already used for defining $\mathcal{D}$; in particular, we note OS_i the set of instances of structural objects of document d_i.

2.2.4.2 The Document Base

A document base $\mathbf{B}$ is a set of document instances satisfying the following database integrity constraints:

- (a) Every document of the document database conforms to the document model: $\forall d_i \in \mathbf{B}, \chi_\mathcal{D}(d_i)$.
- (b) Independance of document components: $\forall d_i, d_j \in \mathbf{B}, OS_i \cap OS_j = \emptyset$, where OS_i and OS_j are respectively the sets of structural components of document instances d_i and d_j.

We may then define a document base as a structure:

$$\mathcal{B} = (\mathbf{B}, \mathcal{D}, \chi_\mathcal{D})$$

The relationship between this model of the document database and document instances (related to the document model) is simple and we will not develop this aspect in much details; it is sufficient to say that:

- we note $OS_{\mathcal{B}}$ the set of all instances of structural components in the database; given the integrity constraint (b) above, we have:

$$OS_{\mathcal{B}} = \bigcup_{i \in \mathbf{B}} OS_i$$

  where sets OS_i are *disjoint.*

- all sets defined in $\mathcal{D}$ apply to the whole database (i.e. $TYPE_{ST}$, $TYPE_M$, $NAME_A$, $VALUE_A$ were not related to any *specific* document). As a consequence, relations and functions involving these sets also apply to the database (i.e. $\preceq_{tst}$, $domain_a$)

- relations on structural objects (i.e. $\preceq_{str}$ and $\prec_{seq}$) are simply extended to the database by considering the union of instance relations related to document instances. Again, constraint (b) ensures that these relations are all distinct, and that this union preserves their definition.

- the same principle applies for functions (i.e. $type_{st}$, $type_m$, $value_a$); since they have all distinct application domains OS_i, they may be extended to $OS_{\mathcal{B}}$ in a straightforward way.

2.2.4.3 The Hyperbase

As it is defined, a document database does not allow browsing based on navigation links, and one may remember that the only access function defined until now is $Next_{seq}$, the standard consultation order defined on logical structures of documents. As said in the introduction, *browsing* (which is in the scope of hypertext and hypermedia) constitute, aside *querying* (which is in the scope of classical IR), a much needed complementary way for retrieving complex multimedia information. On the other hand, we have not yet defined how information within the document database may be retreived using queries. As a consequence, we extend the classical notion of hyperbase by introducing the notion of *index objects* and *navigation links* as the third class of information intervening in the model. Index units are structural components of logical structures that may be retrieved using content queries, while links are *typed* objects relating any couple of structural objects os_{ik}, os_{jl} of documents d_i and d_j.

1. Index Objects

   Index objects are defined from a functional point of view as *retrievable units* for content-based queries. The complete presentation of this sub-model needs an extended discussion involving, in particular, a presentation of the relationship

between indexing and querying. Due to the particular importance of this aspects, and to avoid a long digression while giving here the overall presentation of the model, we have chosen to give this detailed presentation in a separate section (see 3). It is sufficient for now to provide the following information about this sub-model:

(a) *Index objects* correspond to a subset $OI \subseteq OS$ of structural objects, defined according to a subset $TYPE_I$ of $TYPE_{ST}$, the set of structural object types introduced before. This distinction is introduced to cope with general situations where all structural objects may not be considered as *relevant* response objects to queries, either because they are estimated as too specialized or, on the contratry, too general from an average point of view.

(b) Due to the integrity constraint that binds $\preceq_{str}$ and $TYPE_{ST}$, the restriction of index objects to a subset of OS induces on $\preceq_{str}$ a sub-order noted $\preceq_{ind}$ on index objects.

(c) The index process assigns to index objects *index expressions* that are a formal expression of their semantic content. Index expressions all belong to the same language named $\mathcal{L}^{symbolic}$ discussed before. When considering related index objects (i.e. comparable in $\preceq_{ind}$), their index expressions bound by a property named *index dependancy* which may be stated as:

$$\forall oi_i, oi_j \in OI,$$
$$oi_i \preceq_{ind} oi_j \Rightarrow (value_a(oi_i, \mathbf{symbolic}) \rightarrow_{symbolic} value_a(oi_j, \mathbf{symbolic})).$$

   where $\rightarrow_{symbolic}$ is an implication defined on index expressions. This constraint has been introduced to improve the retrieval process when considering large amounts of structured objects: the goal here is to retrieve, from any given query, not *all* structural units satisfying this query, but instead the most *specific* index objects that satisfy this query (*specific* is understood here as *the greatest according to* $\preceq_{ind}$; see section 3 for a complete presentation).

The submodel related to index objects may be then defined as the structure:

$$\mathcal{I} = (OI, TYPE_I, \preceq_{ind})$$

As said above, this sub-model is presented in full details in section 3.

2. Navigation Links

   What follows in this section is an attempt to incorporate browsing in the model presented before; since this problem of integrating the two approaches is a research topic in itself, we limitate ourselves here to some aspects that we believe of main interest considering this integrated view.

$$\mathcal{N} = (ON, RNAV, TYPE_L, cross, type_l)$$

where:

ON : is the set of node objects (abbreviated as *nodes* in the following) of the hyperbase, noted n_i. Nodes may correspond to any objects considered as individually accessible; in the model, they are all structural objects: $ON \subseteq OS_{\mathcal{B}}$.

RNAV : this relation defines navigation links on the set of nodes; they may of course relate *any* couple of structural components, wether belonging to the same document instance or not. This allows the definition of *intra-document link* (i.e. within instances of logical structures) and of *inter-document links* (i.e. between elements of distinct instances of logical structures). We focus only on features of navigation links that are of interest in the context of IR (in particular we do not consider here the various position and layout features of navigation links).

$$RNAV \subset ON \times ON$$

$\mathbf{TYPE_L}$: a set of link types. Link types are useful to disambiguate existing links with regard to the types of information they actually relate. Examples of link types are *class links*, which connect access objects sharing a given property (for example having the same author; such links would be labelled "Same Author"), or "Similar topic" which could relate access objects having similar values of attribute "symbolic" (the similarity being evaluated with respect to a given similarity function on $\mathcal{L}^{symbolic}$).

cross : the standard access function related to navigation links:

$$cross : RNAV \rightarrow ON$$

$$\forall (n_i, n_j) \in RNAV,\ cross(n_i, n_j) = n_j$$

Here n_i is the *anchor* of link (n_i, n_j), while n_j is its *target*. Note that in this simplified view of navigation links, nodes correspond to structural units; this is a restriction compared to the classical definition of anchors and targets which may in fact correspond to more elementary objects (e.g. a word or a sentence). This means that from the IR point of view, we are only interested by the fact that a given anchor or target is within this or this structural unit which we are able to *index* and to *retrieve* (i.e. using queries); *effective* use of the link for *browsing* may be done using anchors that are visualized within any retrieved structural component by a process which is out of our scope here (the visualisation process, using the layout structure). The same principle applies for accessing the target: this document component may not correspond to a standard structural unit. In this case we consider that it is sufficient here to know the target's embedding structural component; the visualization of the target is then processed separately. Said in other words, anchors and targets *may* correspond to units that are sub-atomic compared to the standard logical structure, and we consider that a visualization process is able to present these units to the user when displaying their embedding structural component.

$\mathbf{type_r}$: a total function assigning to every link of $RNAV$ a type of $TYPE_L$:

$$type_r : RNAV \rightarrow TYPE_L$$

CONSTRAINTS: definition of the constraints ensuring the consistency of the navigation structure.

(a) relations $\prec_{seq}$ and $RNAV$ are not necessarily disjoint; one can superimpose on any subset of $\prec_{seq}$ a corresponding set of navigation links of $RNAV$; in this case there are sequences of structural components that are accessible either using the standard access order or using navigation links.

(b) Relation $RNAV$ has very weak general properties; this is mostly related to the necessity of allowing the definition of navigation links in a very unconstrained way. This of course does not prevent to enforce these constraints in particular situations. There may also exist integrity constraints related to the management of certain classes of links; a simple example is when navigation links implement a formal property on classes of nodes such as "having the same author").

- $RNAV$ is not reflexive; there is no use of reflexive navigation links.
- $RNAV$ is in general not symmetric; the existence of a navigation link between two relational objects does not imply the existence of the reverse link. Note that the standard *backtracking function* commonly associated to links is different fom the notion of link, and thus does not contradict this non-symmetry.
- $RNAV$ is in general not transitive; the existence of a sequence of navigation links does not imply the existence of "shortcut" links within this sequence.

Given this definition of the third class of information of the model, which corresponds to *access information* as given by $\mathcal{N}$ and $\mathcal{I}$, we may now introduce the notion of hyperbase $\mathcal{H}$ which allows both querying and browsing:

$$\mathcal{H} = (\mathcal{B}, \mathcal{N}, \mathcal{I})$$

2.3 Conclusion

The model defined in this section integrates main features of multimedia, structured, documents as this has been delimitated in the scope of the FERMI project. The model is centered on the notion of logical structure (a main information about the discourse structure of any document) and two related, complementary classes of information (attributes and access structure) that complete the definition of what we think a good basis for multimedia information retrieval. In particular, we have shown how the single-media models (here viewed as sub-models) may be integrated in the context of the broader notion of multimedia documents. The central notion of (hierarchical) logical structure might be estimated somewhat limitative; in our opinion however, this kind of structure is still intensively used in many applications, in particular those involving hyperbases built from various sources of structured documents loaded from SGML or HTML formats, and then interrelated by navigation links. One can also notice that in the context of hypermedia, most modelling approaches tend to preserve, if not enforce, the notion of *abstraction level* which clearly helps users in the understanding and manipulation of what would be otherwise a gigantic network of *unorganized* information. These abstraction levels are also designed in a hierarchical way, and usually preserve and embed the original logical structure of documents. We have also put a certain emphasis on the necessity to integrate querying and browsing as complementary aspects of information retrieval, a feature we think even more needed in the particular context of retrieving within large, heterogeneous and highly structured repositories of multimedia information. This finally led us to the notion of hyperbase that constitutes, in our opinion, the right modelling level for multimedia information retrieval.

3 Indexing multimedia documents

3.1 Introduction

Classical IR systems usually provide an *index relation* which gives access, from any term of the index language, to the set of all documents indexed by this term (inverted files are implementations of such index relation). This is due to the necessity of ensuring acceptable response times while managing vast amounts of information (thousands of index terms, tens of thousands documents, and often more); the index relation is then no more than a tabular form of an access function of the form $is - about(t)$ which could in principle (at high computing cost) be evaluated in real time. From the previous discussions about the extension of the notion of corpus to the set of all structural components, it seems mandatory to follow a similar approach, and to consider that each indexed structural object is assigned a *symbolic* attribute whose value expresses its semantic content: though these values might be dynamically computed, this would be unacceptable in terms of efficiency. The index process described here explains how such attribute values are pre-computed and assigned in the context of the model presented before. Of course the index

process is not semantically independent of the retrieval model, since the later is based on information produced by the former, and it is then difficult to present them separately. Without anticipating too much on the next section dedicated to retrieval, and for clarity, we have to present here the basic retrieval principle that underlies the whole indexing strategy detailed below. We have said in the introduction that an important problem to solve when retrieving structured information is the possible structural dependency of retrieved objects; if not processed consistently, this could lead to system responses containing for example a section s and one or several of its components. We consider such responses as inconsistent for three major reasons: 1) they are redundant, 2) they are misleading for the user, and 3) they increase the cognitive load of the user who has to find himself which of the related retrieved units are the most specific to his information need. One might object that these problems may be solved using navigation links which the user could browse for examining related response units; in our opinion, this is simply shifting the problem from querying to browsing, and does not solve problem 3) above in an efficient way. We think that the retrieval process, in the context of large amounts of structured information, has to focus on the *smallest units* (i.e. of the lowest level in the logical structure), that *fulfill the query* (i.e. in the context of logic-based retrieval, that logically imply the query).

Retrieving structural components while satisfying this constraint requires that index expressions of related units are *logically bound*: stated informally *the index expression assigned to any parent structural-object has to logically imply the index expressions assigned to each of its possible component objects in the logical structure.* This property of index expressions will be exploited by the retrieval algorithm to filter, among all the structural components implying the query, those which are the most $specific$ to the query. One will find a complete explanation about how this property is used to fulfill this goal in section 4, page 37 (see also [CK96]).

We describe in this section the index process that assigns index expressions verifying this implication property to indexed structural-components. We define first the notion of *index object* that corresponds to structural objects that are retrievable using content-based queries (i.e. that are assigned a value of attribute *symbolic*, see 2. We consider that, in the general case, only a subset of structural units are indexed and thus retrievable (see discussion below); this distinction is based on specific types of structural units $TYPE_I \subseteq TYPE_{ST}$. The index process then operates on each structural object of each document, and assigns index expressions (i.e. values of attribute *symbolic*) only to structural objects of type in $TYPE_I$ (i.e. index objects). Then, we propose a definition of the operator $\oplus_{symbolic}$ introduced in the model when considering propagation of attribute values (remember that *symbolic* is a standard, ascending dynamic, content attribute), and which is defined on expressions of the index language $\mathcal{L}^{symbolic}$. This operator produces index expressions satisfying the *index dependency constraint* introduced in the previous section, and which is noted '$\rightarrow_I$'. We do not distinguish the different medias of the document components because, as we already stated in part 2, all index expressions belong to the same language $\mathcal{L}^{symbolic} = domain_a(symbolic)$ whatever the considered media.

Since there is no ambiguity about the considered Content Attribute *symbolic* which is the unique one referred in all this section, and for improving conciseness of notations, we will abbreviate this symbol by σ.

3.2 The Symbolic attribute

3.2.1 Characteristics

As presented in the document model, attribute *symbolic* is one of the five standard Content Attributes and is specifically dedicated to the representation of *semantic* content of documents. When considering the retrieval of structured objects (i.e. structural objects in our model) four problems arise: 1) the definition of *index objects* among the set of structural objects, 2) the definition of the index language itself, 3) the possible dependence among related structural objects considering their content, and 4) the incidence of this content dependency on retrieval.

1. We name *index objects* structural objects that are indexed (i.e. that are assigned an explicit representation of their semantic content using the *symbolic* attribute) and consequently units that are individually retrievable from queries that include *semantic content* requirements. Defining index objects in this way reflects the specific importance given to the logical structure of documents: this is a direct consequence of our preliminary assumption that the logical structure reflects the structure of the discourse, and hence that structural objects are considered as consistent units considering their semantic content.
   The choice of the proper subset of units of this kind within the set of structural objects is thus related to the notion of *informative units*, or said in other words, units that bear self-explaining information from the users' point of view. Lets consider an example: a leaf object of the logical structure corresponding to a graphic can be self-explaining for example if it consists in a histogram showing the annual gross benefit of IBM between years 1970 and 1995 (provided of course that the graphic contains the proper textual captions and title). A graphic representing a curve with G as y coordinates, f as x coordinates, and entitled Variations of Gain with Frequency for solution 3 is probably not self-explaining, because one do not know what gain, what frequency it is about, and what is solution 3. Clearly this object, if presented to the user, has to be displayed jointly with some textual node that *explains* these notions. Thus while the first example may correspond to an index object, the second will most probably not. Such choices are then directed by application requirements about the types of units to be managed and the typology of the users who are querying the hyperbase (i.e. are they, at least in average, knowledgeable enough about the application domain to properly interpret any informative object?). From the modelling point of view this means that one have to define, among the various abstraction levels on structural objects (i.e. types of $TYPE_{ST}$), the ones which correspond to index objects in a particular application; objects belonging to instances of these types will be retrievable from queries; the user will be able, if needed, to browse from them in the document hyperbase using links which eventually start from these nodes.
   As introduced in section 2, we name *index objects* all structural objects that are assigned a *symbolic* attribute (i.e. that are indexed), and we note them $oi_i \in OI \subseteq OS$.

2. The definition of the *index language* (i.e., the domain of attribute *symbolic*) has to fulfill two major requirements: 1) due to the explosion of the corpus size (in terms of retrievable objects, the corpus is now the set of index objects which will have a much larger cardinality than the set of documents) it should allow the definition of precise concepts to improve precision (i.e. to avoid ambiguities in the expression of information needs, and to improve the discrimination power of index expressions), and 2) it has to allow inferences to match the basic requirements of logic-based retrieval.
   In IR, and whatever the underlying retrieval model, there always exists an associated index language that can be defined either in extension or (most often) in intention using a formal grammar of the form $G = (S, V, \Sigma)$ where S is the starting non-terminal symbol, V is the vocabulary (both terminal and not terminal), and Σ is the set of rules. To avoid confusion with *index terms* which often refer to elementary entities like keywords, we will refer to the elements of the index language as *index expressions*. As an example, the Boolean retrieval model offers index terms that are keywords (i.e. elements of V) and index expressions that are sets of keywords. Here Σ is a (simple) set of rules that allows, given a set of keywords $V_t \subset V$, a set of non-terminal symbols $V_{nt} \subset V$, to derive expressions that belong to the language of Boolean conjunctions like $t_1 \wedge t_3 \wedge t_{12}$, where the $t_i \in V_t$. Index expressions are produced from raw data by specific *index processes*. Considering multimedia documents, we will assume here that there is a unique index language used for describing the semantic content of any kind of document component (i.e. of any media). This does not mean that a given media has no incidence on the nature (classes) of concepts that can be used to properly describe the semantic content of corresponding data; we in fact assume here that all these possible media-dependent facets about content description are integrated within a unique language (as described in part 2).
   At this step of the presentation we will consider the index language $\mathcal{L}^\sigma$ as a set of index expressions which are *sentences of a given logic*; our main concern in this section will be to specify the needed properties of such expressions to allow indexing and retrieval of structured information.

3. If we suppose that the previous problem is solved, then have already observed that the semantic content of different structural objects may not be independent: for example, what is the incidence of the content of a sub-section on the content of its embedding section? This kind of problem has never been extensively studied in its own, though several researches are more or less related to this problem (e.g. one can mention here the numerous investigations aimed

to evaluate the impact of citations and bibliographical references on retrieval performances). Whenever two index objects are related by $\preceq_{str}$ (see the *logical structure* in section 2), we will consider that there is a dependency between the index objects, which is modelled by relation $\preceq_{ind}$. This relation states how index objects are aggregated *and* that they are logically bound as introduced before.

4. When index objects are structured, the whole strategy of searching is changed in a drastic way. Let us consider a simple example: suppose that given a query Q, and given a proper indexation of sections, paragraphs of the documents, the system can retrieve a paragraph p and a section s that are considered relevant for Q. Then what happens if p is a component of s (i.e if $s \preceq_{str} p$)? If the system answer contains p and s then it will be most probably estimated redundant (and maybe misleading) by the users. Solving this particular problem has to be considered in the framework of logic-based retrieval. The main requirement here is that an index object oi_s is relevant for a query Q iff its index expression $ie_s \in \mathcal{L}^\sigma$ logically implies Q; this is noted in the following as: $ie_s \rightarrow_\sigma Q$. Going back to the problem of structured index objects, our proposition for a proper retrieval of such information (i.e. for retrieval avoiding redundancy) is to use the reverse implication $Q \rightarrow_\sigma ie_s$ as a filter among index objects satisfying the direct (classical) implication. This principle is described in section 4; what is needed for now is to understand that such a filtering is possible only if the index expressions assigned to structured index objects oi_i and oi_j satisfy the *index dependency* property:

$$oi_i \preceq_{ind} oi_j \Rightarrow (ie_i \rightarrow_\sigma ie_j)$$

where ie_i and ie_j are respectively the index expressions assigned to oi_i and oi_j.

3.2.2 Properties of Content Attribute symbolic

To sum up the points above, all index objects, and only them, are instances of $TYPE_I$, a subset of the logical structure types $TYPE_{ST}$. When considering index objects of types in $TYPE_I$, one can also derive from $\preceq_{tst}$ the order $\preceq_{ind}$ that relates structured index objects. One important thing to notice is that index objects being structural objects, all the properties of structural objects apply also to index objects. The *index model* of a hyperbase, as introduced in section 2, is then defined as:

$$\mathcal{I} = (OI, TYPE_I, \preceq_{ind})$$

where:

OI : the set of indexed structural objects of the corpus, named *index objects* and noted oi_i. Because all types of components may not be potential responses to users' queries (see discussion above), we have $OI \subseteq OS$.

$\mathbf{TYPE_I}$: the set of index object types. This subset of $TYPE_{ST}$ determines the types of structural objects that may correspond to index objects (i.e that may be retrievable objects using content-based queries).

$$TYPE_I \subseteq TYPE_{ST}$$

$\preceq_{\mathbf{ind}}$: a relation representing the structural dependency between index objects, induced by relation $\preceq_{tst}$ on types:

$$\preceq_{ind} \subset OI \times OI$$

with:

$$\forall oi_i, oi_j \in OI, oi_i \preceq_{ind} oi_j \Rightarrow type_{st}(oi_i) \preceq_{tst} type_{st}(oi_j).$$

One may easily see that $\preceq_{ind}$ is a sub-order of $\preceq_{str}$, induced by the restriction of $TYPE_{ST}$ to $TYPE_I$ and to the integrity constraint relating $\preceq_{str}$ and $\preceq_{tst}$ (see "Logical Structure" in the previous section).

For more concision and clarity, we introduce here the following notation:

index : a total function returning the index expression of $\mathcal{L}^\sigma$ associated to any index object oi_i. We consider here that attribute *symbolic* is *mono − valued*:

$$index : OI \rightarrow \mathcal{L}^\sigma$$
$$\forall oi_i \in OI, index(oi_i) = element(value_a(oi_i, \sigma))$$

where functions $value_a$ and *element* have been defined previously in the document model. The use of function *element* refers to the mono-valuation of attribute *symbolic* (remember that function $value_a$ has its values in 2^{VALUE_A+}).

CONSTRAINTS: the constraints that ensure the consistency of indexes are:

1. Any structural object of a type in $TYPE_I$ is an index object :

$$OI = \{oi \in OS \mid type_{st}(oi) \in TYPE_I\}$$

2. The relation $\preceq_{ind}$ is a sub-order of $\preceq_{str}$:

$$\forall oi_i, oi_j \in OI, oi_i \preceq_{ind} oi_j \Rightarrow oi_i \preceq_{str} oi_j$$

3. Semantic expressions indexing related index objects are logically bound; this property, named *index dependency*, is needed for proper retrieval of structured information:

$$\forall oi_i, oi_j \in OI, oi_i \preceq_{ind} oi_j \Rightarrow (index(oi_i) \rightarrow_\sigma index(oi_j)).$$

where the symbol $\rightarrow_\sigma$ denotes the *logical implication* between index expressions of $\mathcal{L}^\sigma$.

3.2.3 One example of indexing structure

The example described here shows an instantiation of symbolic attributes. We see in Fig. 2 the partial order $\preceq_{tst}$ in the context of documents composed of chapters, sections, subsections, ... The set $TYPE_I$ is represented on the right of the figure as a subset of $TYPE_{ST}$. In this figure, the partial order $\preceq_{tst}$ is shown from "top to bottom", which means that $Chapter \preceq_{tst} Section \preceq_{tst} Subsection$ etc.

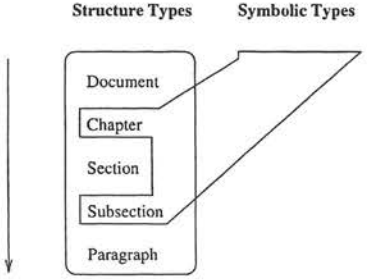

Figure 2: An example of structure and index hierarchy types

According to figure 2 the only document components that will be potentially retrieved are of types *Chapter* or *Subsection*. In this context, we express that neither whole documents nor paragraphs correspond to index objects. Minimal components of document that can be retrieved are *Subsections*, and not *Paragraphs*; this means that, for this type of documents, paragraphs are not considered as enough informative for users, and the system is entailed to provide at least subsections. Said in other words, this means that in this application *paragraphs* may be retrieved only in the context of their embedding *subsections*.

Let us now consider an instance of *Document* containing two *Chapters*, with *Chapters* containing *Sections* etc., as shown in Fig. 3. In this figure, we name $os_i, 1 \leq i \leq 19$ the structural objects that constitute the Document. The *index objects* appear in bold and are of type *Chapter* or *Subsection* and correspond to retrievable structured components noted Ui in Fig. 3. The figure also shows (circled by dotted lines) the structure of each index object (based on the $\preceq_{str}$ partial order).

3.3 Properties operator $\oplus_\sigma$

Operator $\oplus_\sigma$ has as arguments index expressions that are elements of $\mathcal{L}^\sigma$ and produces values of the same type; its properties (see the following) are then constrained because 1) index expressions are *logical expressions* and the operator must then have the properties of an *internal operator* on logical expressions, and 2) as already introduced, the result of the operation is logically bound to its arguments.

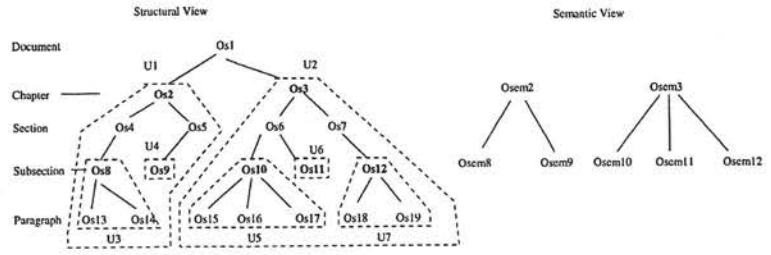

Figure 3: Parts of the structural and semantic views of a document

In what follows, we have to make a distinction between the syntactical and semantic levels of language $\mathcal{L}^\sigma$ which corresponds to the classical distinction between syntax and semantics in logic. The symbol "$\rightarrow_\sigma$" denotes a reflexive relation between two elements of $\mathcal{L}^\sigma$: $ie_1 \rightarrow_\sigma ie_2$ expresses that ie_1 implies ie_2 in the particular logic of $\mathcal{L}^\sigma$. We also consider a reflexive binary operator "$\equiv_\sigma$", denoting that two expressions of $\mathcal{L}^\sigma$ are logically equivalent. At the syntactical level, we introduce one specific element of $\mathcal{L}^\sigma$, namely ε_σ, denoting the empty index expression. This element is needed when some structural unit may be not indexed (i.e. belonging to $OS - OI$). A logical property needed for ε_σ is:

$$\forall ie \in \mathcal{L}^\sigma, ie \rightarrow_\sigma \varepsilon_\sigma$$

As an example, if $\mathcal{L}^\sigma$ were the language of first order logic, *true* would be a satisfying denotation for ε_σ.

Given these considerations, and remembering that attribute *symbolic* is an Ascending Dynamic Attribute, we can now postulate needed properties of operator $\oplus_\sigma$ which is dedicated to compute index expressions of parent index objects from index expressions of component index objects. We consider at first the properties relating the operator to the implication $\rightarrow_\sigma$:

- $\oplus_\sigma$ is a binary, internal operator on $\mathcal{L}^\sigma$:

$$\oplus_\sigma : \mathcal{L}^\sigma \times \mathcal{L}^\sigma \rightarrow \mathcal{L}^\sigma$$

- Implication property:

$$\forall ie_i, ie_j \in \mathcal{L}^\sigma \quad ie_i \oplus_\sigma ie_j \rightarrow_\sigma ie_i \quad \text{//} \quad \rightarrow_\sigma ie_j$$

This property specifies how *index dependency* is ensured while aggregating index expressions. One may notice that this property in fact expresses a notion of "information conservation" when combining index expressions: nothing is lost from the argument expressions since the result expression remains a possible response to any query that would match with any of the argument expressions.

- $\oplus_\sigma$ is **commutative**. This property expresses that the semantic content of a parent index object is independent of the order in which we consider the aggregation of its components' index expression:

$$\forall ie_i, ie_j \in \mathcal{L}^\sigma, ie_i \oplus_\sigma ie_j \equiv_\sigma ie_j \oplus_\sigma ie_i$$

Note that this commutativity does not contradict the index dependency property.

- by definition $\rightarrow_\sigma$ is reflexive; using this property, one may infer that operator $\oplus_\sigma$ is **idempotent**. This property is consistent when considering the notion of "conservation of information" mentioned before:

$$\forall ie \in \mathcal{L}^\sigma, ie \oplus_\sigma ie \equiv_\sigma ie$$

Again, this reflexivity does not contradict the index dependency property.

- $\oplus_\sigma$ is **associative**. This property is used when recursively combining index expressions:

$$\forall ie_i, ie_j, ie_k \in \mathcal{L}^\sigma, (ie_i \oplus_\sigma ie_j) \oplus_\sigma ie_k \equiv_\sigma ie_i \oplus_\sigma (ie_j \oplus_\sigma ie_k)$$

Note that associativity does not contradict the implication property:

$$\begin{array}{lllll} \forall ie_i, ie_j, ie_k \in \mathcal{L}^\sigma, & (ie_i \oplus_\sigma ie_j) \oplus_\sigma ie_k & \rightarrow_\sigma & ie_i \oplus_\sigma ie_j & \\ & \quad // & & & \rightarrow_\sigma \quad ie_i \\ & \quad // & & & \rightarrow_\sigma \quad ie_j \\ & \quad // & \rightarrow_\sigma & ie_k & \end{array}$$

We obtain the same result when considering $ie_i \oplus_\sigma (ie_j \oplus_\sigma ie_k)$

- ε_σ is a neutral element for $\oplus_\sigma$:

$$\forall ie \in \mathcal{L}^\sigma, ie \oplus_\sigma \varepsilon_\sigma \equiv_\sigma ie$$

3.4 The index process

We describe now the index process that computes and assigns index expressions to index objects using operator $\oplus_\sigma$. It is important to notice that computation and assignation of index expressions are distinct operations that do not necessarily affect the same sets of structural units. This is because index objects (to which index expressions are assigned) are in general subsets of structural objects, and because, despite of this distinction, the computation of index expressions may involve all structural objects:

- the *computation* of index expressions is performed from bottom to top of the logical structure on every structural component, up to the structural units corresponding to the maximal level of index objects (i.e. minimal value of $TYPE_I$ for each indexed document).

- the *assignation* of the computed index expressions concerns only structural objects that are *index objects* (i.e. of type in $TYPE_I$). This corresponds to the assignation of a *symbolic* attribute to each index object.

3.4.1 Computation of Index Expressions

We define the index process as a recursive function named δ, defined on structural components of documents, which computes index expressions in a bottom-up way. The function of course involves the binary operator $\oplus_\sigma$ that combines expressions of $\mathcal{L}^\sigma$. The definition of this recursive function supposes that atomic (monomedia) structural objects have been already indexed (i.e. have been assigned Content Attributes as defined in section 2). We have to remind here that these indexes are computed by specific functions named $\mathcal{M}^\sigma$, and assigned only to leaves of the logical structure (set LOS) viewed as atomic monomedia objects. So, for any os_i representing a document or a document component, the definition of δ is:

$$\delta : O_S \rightarrow \mathcal{L}^\sigma$$
$$\delta(os_i) = \begin{cases} os_i \in LOS & : \mathcal{M}^\sigma(os_i), \\ os_i \notin LOS & : \delta(os_{j1}) \oplus_\sigma \cdots \oplus_\sigma \delta(os_{jn}),\ \forall\, os_{jx} \in NextDesc_{str}(os_i) \end{cases}$$

where function $NextDesc(os_i)$ gives the immediate descendants of os_i in the logical structure (see "Logical Structure" in 2).

3.4.2 Assignation of Index Expressions

A procedure named Δ assigns index expressions computed by function δ to index objects having a structural type in $TYPE_I$:

$$\Delta : O_S \rightarrow O_S \times \mathcal{L}^\sigma$$
$$\forall os_i \in OS \begin{cases} os_i \in O_I & : value_a(os_i, \sigma) = \delta(os_i), \\ os_i \notin O_I & : value_a(os_i, \sigma) = \varepsilon_\sigma \end{cases}$$

As an example, if we consider figure 3, function *delta* is computed for each structural objects $os_1, \ldots, os_{19}$, but the assignation concerns only index objects os_2, os_3, os_8, os_9, os_{10}, os_{11}, os_{12}.

In part 3.3, we have defined the property of *index dependency* that logically binds index expressions assigned to related index objects:

$$\forall os_i, os_j \in OS_I, os_i \preceq_{ind} os_j \Rightarrow (index(os_i) \rightarrow_\sigma index(os_j))$$

Given the recursive definition of Δ and the properties defined for operator $\oplus_\sigma$, it is easy to verify (in a recurrent way) that the index expressions assigned by this function to index objects satisfy this property.

3.4.2.1 Example

Figure 4 shows a simple case of logical structure used here to illustrate how function Δ ensures the index dependency. For simplification we consider in this example that $TYPE_I = TYPE_{ST}$. This simplification conforms with the definition of $TYPE_I$ (see definition 3.2.2). Considering a non-atomic structural object of this structure, for instance o_{s6}, we have:

$$NextDesc_{str}(os_6) = \{os_{10},\ os_{11}\}$$
$$NextDesc_{str}(os_{10}) = \{os_{15},\ os_{16},\ os_{17}\}$$
$$NextDesc_{str}(os_{11}) = \emptyset$$
$$NextDesc_{str}(os_{15}) = \emptyset$$
$$NextDesc_{str}(os_{16}) = \emptyset$$
$$NextDesc_{str}(os_{17}) = \emptyset$$

The value of $\delta(os_6)$ is then computed as follows:

$$\begin{array}{rll} os_6 \notin LOS & \Rightarrow \delta(os_6) & = \delta(os_{10}) \oplus_\sigma \delta(os_{11}) \\ os_{10} \notin LOS, os_{11} \in LOS & \Rightarrow \delta(os_6) & = (\delta(os_{15}) \oplus_\sigma \delta(os_{16}) \oplus_\sigma \delta(os_{17}) \oplus_\sigma (ie_{11}) \\ os_{15}, os_{16}, os_{17} \in LOS & \Rightarrow \delta(os_6) & = (ie_{15} \oplus_\sigma ie_{16} \oplus_\sigma ie_{17}) \oplus_\sigma (ie_{11}) \end{array}$$

where $ie_{11} = \mathcal{M}^\sigma(os_{11}), \ldots, ie_{17} = \mathcal{M}^\sigma(os_{17})$ are the index expressions associated to the symbolic views of single-media objects $os_{11}, \ldots, os_{17}$.

According to the property of index dependency we have then:

$$\begin{array}{ccc} index(os_{10}) & \rightarrow_\sigma & index(os_{10}) = ie_{15} \oplus_\sigma ie_{16} \oplus_\sigma ie_{17} \\ \vdots & \vdots & \\ index(os_{10}) & \rightarrow_\sigma & index(os_{17}) = ie_{17} \end{array}$$

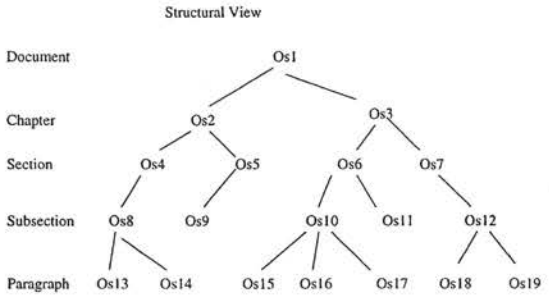

Figure 4: A document with all queryable parts

Having $TYPE_I = TYPE_{ST}$ implies here that procedure Δ assigns to os_6 and to all its descendants Content Attributes *symbol* with values corresponding to the

index expressions described above for each of these objects. Would we go back to a definition of $TYPE_I = \{chapter, subsection\}$, then $\delta(os_6)$ would have made exactly the same calculus, but Δ would have done the assignation only for objects os_{10} and os_{11} which are the only component objects of os_6 with types in $TYPE_I$.

3.5 Conclusion

We have defined in this section an *IndexModel* that applies to structured documents. This model introduced first the notion of *index objects*, defined as classes of structural objects which, depending on application requirements, correspond to retrievable objects using content-based queries. We have then defined how values of Content Attribute *symbolic*, which describe the semantic content of document components according to an *index language*, are computed and selectively assigned to *index objects*; we have also introduced the notion of *index dependency*, a property that index expressions have to verify to allow retrieval of minimal, relevant objects. This later aspect, which we think important in the context of large, complex corpus of information, is developed in the next section.

4 Retrieving multimedia structured documents

4.1 Introduction

All the studies presented in the previous sections are dedicated to the data definition level of the model. Here we address the problem of how to query this data to retrieve structured multimedia documents. One important statement here is that the query language proposed here combines classical database capabilities and typical IR capabilities based on attribute selection or consultation for example. The second case typically produces answers that implicitly contain only document components, while the first produces answers that main relate to any fact from the database, as for example the author(s) of a document. One may notice that this later case could include the former, providing that the database model offers proper operations to deal with content and uncertainty (which is not currently the case). On the other hand, one may also notice that in the complex process of information retrieval, users might need at some moment classical database queries: for example, one might ask what are the authors of cited documents without accessing them explicitly. In our opinion, these two ways of querying should then be integrated in the model, though they might correspond to separate querying tools at practical level. This is the reason why we propose a query language that does not implicitly retrieve documents only, but rather provides access to any kind of information stored in the document database. The proposed query language is presented in four steps. First we introduce the notion of predicate that implement the main features of the document model presented in section 2, and we consider a new definition of the Document Base, derived from this notion of predicate. The query language itself is then presented in both its syntactical and semantic levels. Finally we consider the requirement stated

before about the *optimality* of system responses, which stated that when answering queries the system should avoid multiple presentations of the same documents in responses. We show in this context how the classification of attributes proposed in the definition of the data model (basically the *index dependency property*) may be used to fulfill this requirement.

4.2 Basic Predicates

We define here a set of predicates that are used to describe and to query the document database. The essence of these predicates is to capture the basic notions and properties introduced in the data model in a logic-based way.

Notations

- In the following, we note constants using italic lower case characters (like *smith*), the variables using normal lower case characters (like author), and all predicates names have an uppercase initial letter and are written in italic (like *CompStr*).

- The set of constants C on which all predicates are defined is globally referred as:

$$C = OS \cup TYPE_{ST} \cup TYPE_M \cup NAME_A \cup VALUE_A \cup RNAV \cup TYPE_L$$

- The set of variable names VAR is defined such that $VAR \cap C = \emptyset$.

- Given any predicate $P(\lambda_1, \ldots, \lambda_n)$, the ordered sequence $\Sigma_p = \lambda_1, \ldots, \lambda_n$ defines the *sort* of P. Every symbol λ_i refers to a *place* and to one set D_i of constant symbols among the ones listed in C. At syntactical level we note γ_p a predicate checking the conformity of any notation $P(s_1, \ldots, s_n)$ involving P, where the s_i are actual symbols respectively assigned to places λ_i. These symbols can be either variable symbols, or constant symbols that conform to (i.e. are elements of) the reference set D_i assigned to position λ_i:

$$\gamma_p(s_1, \ldots, s_n) = \begin{cases} true & : \text{iff } \forall s_i \in s_1, \ldots, s_n, (s_i \in VAR \text{ or } s_i \in D_i) \\ false & : otherwise \end{cases}$$

We organize the presentation of these basic predicates according to the three classes of information used when defining the data model: logical structure, attributes, and access structure.

4.2.1 Logical Structure

CompStr(os_1, os_2) : this predicate is based on order $\preceq_{str}$ to verify that a document component os_2 is a structural component of an other document component os_1.

$$CompStr(os_1, os_2) =_{def} os_1, os_2 \in OS \;\&\; os_2 \in Desc_{str}(os_1)$$

InfTst(t_1, t_2) : This predicate is related to the partial order $\preceq_{tst}$ on types of structural objects $TYPE_{ST}$:

$$InfTst(t_1, t_2) =_{def} t_1, t_2 \in TYPE_{ST} \;\&\; t_1 \preceq_{tst} t_2$$

TypeSt(os, t) : This predicate is true if document os is of structural type t, and false otherwise:

$$TypeSt(os, t) =_{def} os \in OS \;\&\; t \in TYPE_{ST} \;\&\; t1 \neq t2 \;\&\; t = type_{st}(os)$$

TypeM(os, t) : is true if a document os has t for media type:

$$TypeM(os, t) =_{def} os \in OS \;\&\; m \in TYPE_M \;\&\; t = type_m(os)$$

Precede(os_1, os_2) : this predicate is based on relation $\prec_{seq}$ which defines the standard linear consultation order for documents. Predicate *precede* is true wherever the structural objects os_1 and os_2 are related by relation $\prec_{seq}$:

$$Precede(os_1, os_2) =_{def} os_1, os_2 \in OA,\ os_1 \prec_{seq} os_2$$

4.2.2 Attributes

We focus now on basic predicates related to attributes; one have to remind that attributes, as defined in the data model, are generally multi-valued. This property causes representation problems in the fact base since no symbol or variable may represent complex objects like sets. Thus, when *attribute* is multi-valued, this is modelled in FB by a list of predicates involving the same structural object os and the same attribute name *attribute* : $HasValue(os, attribute, value_1), \ldots, HasValue(os, attribute, value_n)$ is the representation of $value_a(os, attribute) = \{value_1, \ldots, value_n\}$ in the fact base.

EqualValue(c_1, c_2) : this predicate evaluates the equality of constants c_1 and c_2 belonging to C.

HasAtt(os, attribute) : this predicate is true iff a document component os has an attribute named *attribute* (i.e., $attribute \in name_a(os)$).

$$HasAtt(os, attribute) =_{def} os \in OS \;\&\; attribute \in NAME_A \;\&\; attribute \in name_a(os)$$

HasValue(os, attribute, value) : this predicate is true iff a document component *os* has *value* among its values for *attribute*.

$$HasValue(os, attribute, value) =_{def} \quad os \in OS \;\&\; attribute \in NAME_A \;\&\; attribute \in name_a(os) \;\&\; value \in value_a(os, attribute)$$

Aside these basic predicates, we consider predicates associated to Content Attributes. Of particular importance in the context of IR is attribute *symbolic*, which is related to the semantic content of document components. A predicate related to attribute *symbolic* implements the notion of implication between index expressions $ie \in \mathcal{L}^\sigma$:

MatchSymbolic($\mathbf{ie_1}$, $\mathbf{ie_2}$) : is a predicate that indicates if the value of index expressions ie_1 matches index expression $ie_2 \in \mathcal{L}^\sigma$ of the index language:

$$MatchSymbolic(ie_1, ie_2) =_{def} \; ie_1, ie_2 \in \mathcal{L}^\sigma \;\&\; ie_1 \rightarrow_\sigma ie_2$$

This type of predicate is of slightly different nature compared to those presented before; the difference comes here from ie_i which is a special case of constant. Being an element of a language, ie_i generally does not correspond to a self-defining, atomic value as constants usually do; instead, ie_i (as already discussed in section 2) has a more complex syntax and semantic, and in fact has properties that have to be interpreted in a given sub-logic (here the one corresponding to the semantic of $\mathcal{L}^\sigma$). This is the meaning of the condition $value_a(os, symbolic) \rightarrow_\sigma ie$ mentioned in the definition above. There is thus a double interpretation of ie_i: one at the level of our model, where it behaves exactly like a constant, and an other one as a *closed formula* in the logic underlying $\mathcal{L}^\sigma$. Although this difference should be fully integrated in a formal way within the model, we think that not doing this at this step does not prevent the understanding of what follows (i.e. how queries are solved).
The same discussion applies for the four other Content Attributes *physical, structural, spatial, perceptive*; as discussed in section 2, they all have their associated domain languages and corresponding sub-logics providing specific definitions of operator $\rightarrow_\alpha$, where α stands for any of these attributes. We will then consider the availability of predicates *MatchPhysical, MatchStructural, MatchSpatial, MatchPerceptive* whose definition is a direct adaptation of the definition of *MatchSymbolic* to these attributes.

4.2.3 Access Structure

Predicates related to the access structure of documents are:

Navigation($\mathbf{a_1}$, $\mathbf{a_2}$) : this predicate is true iff there exists a navigation link between the two argument access-objects (remember that all access objects are structural objects: $OA \subseteq OS$):

$$Navigation(a_1, a_2) =_{def} \; a_1,\; a_2 \in OA \;\&\; a_1 = target(a_1)$$

TypeNav($\mathbf{a_1}$, $\mathbf{a_2}$, t) : this predicate is true iff the navigation link between two access objects is of the given type t:

$$TypeNav(a_1, a_2, t) = def \quad a_1, a_2 \in OA \;\&\; t \in TYPE_L \;\&\; a_2 = target(a_1) \;\&\; t = type_L(a_1, a_2)$$

To cope with application requirements, the above list of basic predicates may be extended by predicates which express other specific properties of sets of constant symbols such as, for example, the existence of orders (like the total ordering of integers for instance). Since there is nothing specific here to our model we will not give here a complete list of these complementary basic predicates.

4.3 The Fact Base

Using the predicates defined above, it is then possible to describe a Fact Base (named FB) that constitutes a logic-based redefinition of the Document Base described in section 2. Two approaches may be used to achieve this; the first is alike the one proposed by *deductive databases*, where FB would be in fact a *knowledge base* containing *facts* and *rules*. Facts would correspond in this case to a minimal set of closed predicates (i.e. predicates without variables) of the sorts described in the previous section, and rules would be deduction rules allowing the inference of facts not explicitly represented in the knowledge base. As an example, considering order $\preceq_{str}$ and its associated predicate *CompStr*, the minimal set of facts would correspond to closed predicates describing each elementary arc of each tree-like logical structure (i.e. arcs linking nodes to their immediate descendents), and associated rules would complete the definition of predicate *CompStr* to include all properties of order $\preceq_{str}$:

- Reflexivity:

$$CompStr(x, x) : -;$$

- Transitivity:

$$CompStr(x, z) : -CompStr(x, y) \;\&\; CompStr(y, z);$$

The second approach considers that *all* explicit and *all* implicit facts that can be derived using *all* deduction rules are explicitly represented in the knowledge base (hence its name *fact base*) using closed predicates. These two solutions are logically equivalent considering the set of facts represented and retrievable in the knowledge base; the difference between the two approaches is related to implementation and knowledge management issues. Since this choice does not deter the generality of the data model, or the generality of the query language as described below we will assume in the following that we are in this second case, and thus that FB is the finite set of *all* the closed predicates that correspond to every explicit or implicit fact deductible from the document base. We consider also that the fact base FB is a *consistent* set of predicates (i.e. containing no contradictory fact).

4.4 The Query Language

The query language is defined in a much classical way and integrates, as discussed in the introduction, database queries and IR queries.

4.4.1 Syntax

A query belongs to the set of well formed formulas of first order logic:

- Every predicates P is of one of the sorts described in section 4.2. As defined also in this section, each of them is syntactically defined by its name and its sort, namely a finite, ordered list of arguments noted $\Sigma_p = \lambda_1, \ldots, \lambda_n$, where each λ_i represents a place associated to a finite set of constants. As an example, predicate $CompStr$ is of sort λ_1, λ_2, where λ_1 and and λ_2 are associated to OS. Places in predicates may be affected either to variable names or to constant symbols (see consistency constraint below).
- every consistent predicate notation P is a formula.
- if F and G are formulae, then $\neg F$, $F \wedge B$, $(\exists x)F$ are formulae.

Before detailing in the next section how queries are evaluated, and what are their responses it is necessary to introduce the following notations:

- A consistent predicate notation of predicate P is any expression $P(s_1, \ldots, s_n)$ where $\gamma_p(s_1, \ldots, s_n)$ holds (that is, s_i are either variables symbols, or constant symbols that conform to D_i). We note σ_p any such consistent, ordered sequence of (variable and/or constant) symbols assigned as arguments of P.
- Given a predicate P and $\sigma_p = s_1, \ldots, s_n$ a consistent sequence of symbols, we note $\overline{\sigma}_p = c_1, \ldots, c_n$ any ordered sequence of *constant symbols* obtained by substituting in σ_p *all* existing variables symbols s_i by a constant symbol c_i of D_i. Formally this is defined as:

- let V_p be the (possibly empty) subset of symbols in σ_p that are variables,
- let C_p be the (possibly empty) subset of symbols in σ_p that are constant,
- then $\overline{\sigma}_p$ is obtained from σ_p by substituting every variable symbol $s_i \in V_p$ by a constant $c_i \in D_i$. The sub-sequence C_p of σ_p is unchanged when deriving any sequence $\overline{\sigma}_p$.

From this definition it is then clear that:

1. $\gamma_p(\sigma_p) \Rightarrow \gamma_p(\overline{\sigma}_p)$.
2. given σ_p, if C_p is not an empty list the corresponding ordered sequence of constant symbols is part of *any* consistent derived sequence $\overline{\sigma}_p$.

4.4.2 Evaluation of Queries

A query belongs to the set of well formed formulas of the first order logic and may have any number of free variables. The predicates that constitute the most elementary query level are the basic predicates described above. The free variables of the query define a structure of the response which is generally a set of ordered lists of constant symbols corresponding to these free variables. An extreme case is when a query has no free variable, in which case its response will be either *true* or *false*.
Here are some simple examples giving an informal illustration of the way queries are evaluated:

- to find all the documents of type *Book*, we write:

$$TypeSt(\text{os}, book)$$

This case is the simplest, because it involves only one predicate; the response will be the set of constants os_i such that $TypeSt(os_i, book)$ is *true*.

- to find if a document of type *Book* exists:

$$\exists \text{os}\ TypeSt(\text{os}, book)$$

The answer here will be either *true* or *false*, depending on the existence of at least a constant os_i satisfying $TypeSt(os_i, book)$.

- to find the books having "Smith" as one of its authors we write:

$$TypeSt(\text{os}, book) \wedge HasValue(\text{os}, author, smith)$$

This example combines two attributes related by a conjunction, the two predicates sharing the free variable os in the first position. The response will here the set of constants os_i satisfying the conjunction $TypeSt(os_i, book) \wedge HasValue(os_i, author, smith)$

- when looking for books illustrated by at least one image, a query is:

$$TypeSt(\mathrm{os}, book) \wedge (\exists \mathrm{os_i})(CompStr(\mathrm{os}, \mathrm{os_i}) \wedge TypeM(\mathrm{os_i}, image))$$

This example is more complicated, because we use a quantified variable that denotes any component of the document corresponding to the free variable os. The response will be the set of constants os_i satisfying the formula $TypeSt(os_i, book) \wedge CompStr(os_i, os_j) \wedge TypeM(os_j, image)$, for any os_j.

Of course this intuitive way of defining query responses has to be defined in a more formal way. We will base this definition on the notion of *query denotation* using a function $\mathfrak{F}$ that gives *all* the constant sets that match query F.

4.4.3 Denotation of queries

The function $\mathfrak{F}$ provides a denotation of queries which are, in general, open formulae. The queries considered here conform to the syntax presented before and may be either single predicates or formulae combining several of them. In a query, the free variables correspond to the elements of the result: we name the *result variables*. As will be seen later, result variables are any *non-quantified variables*.
We use in the definition of the denotation of queries the usual notation $A \models B$ where A is a set of closed predicates and B a formula to indicate that A satisfies B (i.e. all models of A are models of B). We also use the notation " $a \mid C(a)$ " as a definition of a set composed of elements a such that the condition "$C(a)$" is satisfied. We will then introduce the definition of function $\mathfrak{F}$ in considering gradually more complex queries. All the definitions below are based on the notations introduced in section 4.4 about *predicate sorts* noted Σ_p, *predicate argument lists* noted σ_p, and *predicate value lists* noted $\overline{\sigma}_p$.

4.4.3.1 Single-Predicate Queries

Consider a single-predicate query $P(\sigma_p)$ with $\gamma_p(\sigma_p)$ asserting the conformity of argument list σ_p to the sort of P. We define α_p as the (possibly empty) subsequence of σ_p that corresponds to variables. Then, all variables of α_p are considered as free variables of the query. Given σ_p and any (possibly empty) sequence α_p as defined before, then any valid sequence $\overline{\sigma}_p$ contains a sequence of constants noted $\overline{\alpha}_p$ which corresponds to constants assigned to places occupied by variables of α_p in σ_p. Moreover, we can state that any constant (possibly) existing in σ_p still exists (at the same place) in $\overline{\sigma}_p$.

We note $P(\sigma_p)[\alpha_p]$ for telling that the predicate P has σ_p as argument list, and that this list contains a (possibly empty) list of free variables α_p. We consider then two cases in the evaluation of $\mathfrak{F}$:

- When $\alpha_p \neq \emptyset$, we have :

$$\mathfrak{F}(P(\sigma_p)[\alpha_p]) = \{\overline{\alpha}_p \mid FB \models P(\overline{\sigma}_p)[\overline{\alpha}_p]\}$$

The response then contains all ordered constant sequences $\overline{\alpha}_p$ that satisfy P together with all others constants of σ_p

- When α_p is empty, all the places in σ_p are constants (and then σ_p is equal to $\overline{\sigma_p}$). We consider then that the query is closed, and that the result is either *true* or *false*:

$$\mathfrak{F}(P(\sigma_p)[\emptyset]) = \begin{cases} true & \text{iff } FB \models P(\overline{\sigma}_p)[\emptyset] \\ false & \text{iff } FB \not\models P(\overline{\sigma}_p)[\emptyset] \end{cases}$$

For example, consider the case of predicate $TypeSt$:

- $\mathfrak{F}(TypeSt(\mathrm{os}, \mathrm{t})[\mathrm{os}, \mathrm{t}])$ corresponds to the set of couples (structural object, structural type) so that the structural object has the associated type.
- $\mathfrak{F}(TypeSt(\mathrm{os}, chapter)[\mathrm{os}])$ corresponds to the set of constants (structural objects) that have the structural type *chapter*.
- $\mathfrak{F}(TypeSt(os_{34}, chapter)[\emptyset])$ is *true* if structural object o_{34} is a *chapter*, and false otherwise.

These examples already illustrate how the proposed query language may combine database-style queries and IR queries (in which case predicates would always have os as unique result variable).

4.4.3.2 Complex queries

We refer to complex queries as queries combining formulae (as described above). To describe how query responses are defined we introduce now some notations that are direct extensions of the ones presented before for predicate queries. We define Σ_f, the sort of formula F, as a concatenation of the Σ_p of the predicates combined in formula F. The order for this concatenation has to be fixed in some way; at this level in the model this choice has no particular importance, provided only that there is one (for example the lexical order of predicates involved in the expression of F). Similarly, we note σ_f any argument list derived from Σ_f for the different places of predicates involved in F. Then, $\overline{\sigma}_f$ is the result of any valid substitution of variables of σ_f by constants of the related domains defined by Σ_p. Note that in this process the *conformity function* γ_f for formula F is simply defined by:

$$\gamma_f(F) \equiv \gamma_p(P) \wedge \gamma_q(Q), \ldots, \wedge \gamma_r(R)$$

where $P, Q, \ldots, R$ are all the predicates involved in the definition of F.

We note then α_f, a (possibly empty) sequence of variable symbols of σ_f that are free variables of F. Quantified variables of F, when they exist, are not considered here as free variables, because they do not participate to the definition of query results.
Like in the case of predicates, when a query is a sentence (i.e. it reduces to a formula with only constants and/or quantified variables), the result is either *true* or $false$:

$$\mathfrak{F}(F(\sigma_f)[\emptyset]) = \begin{cases} true & \text{iff } \{\overline{\sigma}_f | FB \models F(\overline{\sigma}_f)[\emptyset]\} \neq \emptyset \\ false & \text{iff } \{\overline{\sigma}_f | FB \not\models F(\overline{\sigma}_f)[\emptyset]\} = \emptyset \end{cases}$$

We notice that this definition is consistent with the simpler case of single-predicate queries given before.
When the query is a negation of a formula F having free variables, we have:

$$\mathfrak{F}(\neg F(\sigma_f)[\alpha_f]) = \{\overline{\alpha}_f \mid FB \not\models F(\overline{\sigma}_f)[\overline{\alpha}_f]\}$$

We consider now the case where some of the variables of the formula are quantified. In this case, we need to represent formula $F(\sigma_f)$ as $F(\sigma_f)[\alpha_f, \beta_f]$ where:

- α_f is the (possibly empty) list of free variables in $F(\sigma_f)$,
- β_f is the list of quantified variables.

Then, such a query gives as its answer any ordered list $\overline{\alpha}_f$, sub-sequence of $\overline{\sigma}_f$ that satisfies F. In the following definition, $\exists\beta_f$ is an abbreviation for $(\exists\beta_1)(\exists\beta_2)\ldots(\exists\beta_n)$, where $beta_i$ are all quantified variables of β_f:

$$\mathfrak{F}((\exists\beta_f)\ F(\sigma_f)[\alpha_f, \beta_f]) = \{\overline{\alpha_f} \mid FB \models F(\overline{\sigma}_f)[\overline{\alpha}_f, \overline{\beta}_f]\}$$

When β_f is empty, we are in the case presented before with no quantified variable. When α_f is empty, the results is either *true* or $false$, since there is no result variable:

$$\mathfrak{F}((\exists\beta_f)\ F(\sigma_f)[\emptyset, \beta_f]) = \begin{cases} true & \text{iff } \{\overline{\sigma}_f | FB \models F(\overline{\sigma}_f)[\emptyset, \overline{\beta}_f]\} \neq \emptyset \\ false & \text{iff } \{\overline{\sigma}_f | FB \models F(\overline{\sigma}_f)[\emptyset, \overline{\beta}_f]\} = \emptyset \end{cases}$$

Of course the same result applies when both α_f and β_f are empty lists; in this case we have again $\sigma_f = \overline{\sigma}_f$ and the query is a sentence.
We notice of course that these three cases above are generalizations of the similar ones about single predicates.

Let us consider now the problem of formulae that are combinations of two sub-formulae. In our case, a query is a conjunction or a disjunction of well formed formulas. We note then σ_{f_1} and σ_{f_2} the argument sequences of F_1 and F_2 defined exactly as described before for each of these formulae. Since F_1 and F_2 may share variables we have then to consider new sub-sequences of variables:

- we note α_{f_1} and α_{f_2} the sequences of result variables that are *exclusive* to F_1 and to F_2 respectively.
- we note α_c the sequence of result variables that are *common* to F_1 and F_2.

So, for a query F combining (either with a conjunction or a disjunction) two subqueries F_1 and F_2, we have to describe the evaluation of F_1 and F_2 using more elaborate notations which are: $F_1(\sigma_{f_1})[\alpha_{f_1}, \alpha_c]$ and $F_2(\sigma_{f_2})[\alpha_{f_2}, \alpha_c]$.
We may consider now the two different cases that can occur with such queries. We consider below the case of a *conjunction*, and then the case of a *disjunction*:

- **Conjunction:**If one sequence α_{f_1}, α_{f_2} and α_c is not empty, we have :

$$\begin{aligned} &\mathfrak{F}(F_1(\sigma_{f_1})[\alpha_{f_1}, \alpha_c] \wedge \\ &\quad F_2(\sigma_{f_2})[\alpha_{f_2}, \alpha_c]) = \{\overline{\alpha}_{f_1}, \overline{\alpha}_{f_2}, \overline{\alpha}_c \mid FB \models F_1(\overline{\sigma}_{f_1})[\overline{\alpha}_{f_1}, \overline{\alpha}_c] \\ &\qquad\qquad \wedge F_2(\overline{\sigma}_{f_2})[\overline{\alpha}_{f_2}, \overline{\alpha}_c]\} \end{aligned}$$

  The introduction of sequence α_c defines that there is no repetition of corresponding constant sequences $\overline{\alpha}_c$ in the response.

- **Disjunction:**We consider below the case of a *disjunction*; this definition applies whenever at least one of the sequences α_{f_1}, α_{f_2} and α_c is not empty:

$$\begin{aligned} &\mathfrak{F}(F_1(\sigma_{f_1})[\alpha_{f_1}, \alpha_c] \vee \\ &\quad F_2(\sigma_{f_2})[\alpha_{f_2}, \alpha_c]) = \{\overline{\alpha}_{f_1}, \overline{\alpha}_{f_2}, \overline{\alpha}_c \mid \\ &\qquad\qquad FB \models F_1(\overline{\sigma}_{f_1})[\overline{\alpha}_{f_1}, \overline{\alpha}_c] \vee F_2(\overline{\sigma}_{f_2})[\overline{\alpha}_{f_2}, \overline{\alpha}_c]\} \end{aligned}$$

- In both cases of conjunction and disjunction, sub-sequences if α_{f_1}, α_{f_2} and α_c are all empty, then the query is again a sentence and the answer is either *true* or $false$, as in the similar cases described before.

Let us consider now examples of such queries:

- Attribute values can be used to find documents "written by Smith in 1995". In this case, the query is composed of two sub-formulae that are predicates:

$$HasValue(\text{os}, pub-date, 1995) \wedge HasValue(\text{os}, author, smith)$$

  If we name F this query formula and F_1 and F_2 its two sub-queries, we have $F_1(\text{os}, pub-date, 1995)[\emptyset, \text{os}]$ and $F_2(\text{os}, author, smith)[\emptyset, \text{os}]$ according to our description above. The result is then a set of constants (structural objects) that match the query.

- The logical structure can be used to find documents containing a structured section followed by one image:

$$(\exists os_2,\ os_3,\ os_4)\quad CompStr(os_1, os_2) \wedge CompStr(os_1, os_3) \wedge TypeSt(os_2, section) \wedge CompStr(os_2, os_4) \wedge TypeM(os_3, image) \wedge Precede(os_2, os_3)$$

  Here, the only result variable is os_1 and the query result is a set of structural objects.

- Queries combining content-based and structural criteria may be issued, like for example when searching books about relational databases. We suppose then here that topic "relational database" is expressed by the index expression $relational - database$ of $\mathcal{L}^{\sigma}$ (a constant):

$$(\exists ie)\quad (TypeSt(os, book) \wedge (HasValue(os, symbolic, ie) \wedge MatchSymbolic(ie, relational - database))$$

  The only result variable here is os and the query result is a set of structural objects (in this case index objects, since only them have attached *symbolic* attributes).

- Queries can also refer to the access structure. We may for example refine the previous query in requiring books about relational databases and citing author "Smith":

$$(\exists ie, os_1, os_2)\quad TypeSt(os, book) \wedge HasValue(os, symbolic, ie) \wedge MatchSymbolic(ie, relational - database) \wedge CompStr(os, os_1) \wedge TypeNav(os_1, os_2, citation) \wedge HasValue(os_2, author, smith)$$

  Note that the notion of citation has been expressed here as the existence of at least one component object os_1 of os linked to a structural object os_2 having author "Smith" by a link of type "citation". Note also that there is no type constraint on os_2 (the structural object of author "Smith"); this means that any kind of document component is considered here as a possible source of a citation, provided that it has the required author. Again, the only result variable here is os and the query result is a set of structural objects (index objects in this case too).

4.5 Retrieving structured documents

When we consider the above definition of the evaluation of query results, we do not consider at all the fact that specific *classes* have been associated to attributes (see section 2. Three classes for attributes have been proposed: the Dynamic Ascending Attributes (DAAs), the Dynamic Descending Attributes (DDAs), and the Static Attributes (SAs).

These classes have been introduced with the specific goal to optimize query responses in terms of *minimality*, which means here avoiding redundant responses. This notion of redundancy is related only to structural objects which constitute the basic retrievable objects in this context (index units): redundancy occurs whenever a response $R = \{os_1, os_2, \ldots, os_n\}$ is such that there exists couples (os_i, os_j) that are structurally related : $os_i \preceq_{str} os_j$ or $os_j \preceq_{str} os_i$. To manage such structural properties of the documents, we limit then the application of this optimization to queries that have one result variable related to structural objects. In this context we can avoid the retrieval of redundant document components; the proposed approach is based on *attribute classes* and on a notion of *query type*.

To be more generic, we consider that given a query F satisfying the property given above (i.e., one result variable on the domain of structural objects), its optimal response $\mathfrak{F}$ is defined as:

$$\mathfrak{F}(F(\sigma_f)[\alpha_f]) = Opt_{T_f}(\{\overline{\alpha_f} \mid FB \models F(\overline{\sigma}_f)[\overline{\alpha}_f, \overline{\beta}_f]\ \})$$

Where function $Opt_{T_f}(\mathfrak{R}_f)$ handles the simplification of the result $\mathfrak{R}_f$ obtained in the classical way from F, according to the type of the query noted T_f. We consider first three different basic query types corresponding to the three attribute classes.

4.5.1 DAAs Optimization

With a dynamic ascending attribute A, a propagation rule defines the value of this attribute for a document component based on the attribute values of its components. In this case, we consider that when two *related* structural units are answers to a query involving a Dynamic Ascending Attribute, the response should contain only the greatest according to the order $\preceq_{str}$ (i.e. the deepest component in the logical structure). This choice corresponds to the selection of the most *specific* components of the document that satisfy the query. Since we are in an hyperbase environment, one have to remind then that from these smallest document components as entry points (obtained using queries), the user can then decide to browse in the structure if he needs so.

We name Opt_{asc} the optimization function associated to ascending attributes:

$$Opt_{asc}:\ 2^{OS} \rightarrow 2^{OS}$$
$$\forall S \in 2^{OS}, Opt_{asc}(S) = \{os_i \in S \mid \nexists os_j \in S\ \&\ os_i \preceq_{str} os_j\} \subseteq S$$

This optimization may be implemented using both *direct* and *reverse* implications; this will be illustrated in the next section while considering the important case of attribute *symbolic* (which, as all content attributes, is a DAA).

4.5.2 DDAs Optimization

We consider now the case of Dynamic Descending Attributes to determine the best document chosen when a response contains several related (always relatively to $\preceq_{str}$) document components.

With such attributes, values affected to document components depend on the value of their parent components. The consequence is then that attribute values of parent components logically imply those of their descendants in the logical structure. So, when a structural component of a document satisfies a query involving a DDA, all its components also satisfy the query. We consider then that in this case, the optimal response is the parent component, because it constitutes an aggregation of components that all satisfy the query. For instance, in the context of the DDA *Pub – date*, if a *Book* has a publication date that matches the user's need, we consider that the book is the optimal answer for the query (as opposite to giving as response all its components).

The optimization function Opt_{desc} for Dynamic Descending Attributes queries, is then defined as:

$$Opt_{desc}: \ 2^{OS} \rightarrow 2^{OS}$$
$$\forall S \in 2^{OS}, Opt_{desc}(S) = \ \{os_i \in S | \nexists os_j \in S \ \& \ os_j \preceq_{str} os_i\} \subseteq S$$

4.5.3 SAs Optimization

Because the definition itself of a static attribute does explicitly indicates that there is no dependency of any kind between parts of documents containing these attributes, no *a priori* decision can be made to simplify the results of queries dealing with static attributes. In fact, the documents structure can be used in the same way as for DAAs when related components match the query; this may occur only when for some reason the same attribute value has been assigned at different levels of the same logical structure. We consider then that the best response are the most specific units (i.e. the lowest in the logical structure). The definition the Opt_{stat} is then identical to Opt_{asc}:

$$Opt_{stat}: \ 2^{OS} \rightarrow 2^{OS}$$
$$\forall S \in 2^{OS}, Opt_{stat}(S) = \ \{os_i \in S | \nexists os_j \in S \ \& \ os_i \preceq_{str} os_j\} \subseteq S$$

4.5.4 Mixed Optimization

A query may combine attribute predicates of various classes; there is then a conflictual situation to be solved since, for example, Opt_{asc} and Opt_{desc} have opposite optimization strategies. The only way to solve this problem is to define a *priority order* among conflictual solutions. Since our overall strategy is to maintain a balance between the notions of *exhaustivity* and of *specificity* of a response, we tend to favor the most specific responses that satisfy the query. As a consequence, the proposed priority order for optimizing responses in conflictual situations is first *asc*, the *desc* and finally *stat*:

- when a query is of type *asc*, *desc* or *stat*, the corresponding optimization function is applied (no conflict).
- when a conflictual situation occurs, and if it involves at least an ascending attribute then function Opt_{asc} is used,

- when a conflictual situation without any DAA occurs, and if it involves at least a descending attribute then function Opt_{desc} is used,

4.6 The Fetch and Browse approach for DAAs

We describe here a possible solution for implementing function Opt_{asc} described before. The characteristics of dynamic ascending attributes have been described in section 3 dedicated to the indexing of structured documents. We consider here again the example of attribute *symbolic* which belongs to this class. We describe also the properties of operator $\oplus_\sigma$ on index expressions of $\mathcal{L}^\sigma$ in a way to ensure that they satisfy the *dependency constraint* presented in section 3. In fact, the dependency constraint is related to the logical approach to IR introduced by Van Rijsbergen [Rij86], and which is based on the following principle: "given a query Q and a document D, D is relevant to Q if D logically implies Q". We do not focus here on one or another definition of this implication; because we still need to remain general, but we consider here that the implication involved underlying the retrieval process is assimilated to $\rightarrow_\sigma$, the implication between index expressions. We consider however an interesting extension of Van Rijsbergen's proposition due to Nie [Nie90]. This extension proposes to evaluate document relevance based on two distinct criteria: the direct implication it D implies Qand its reciprocal it Q implies D. This approach is formulated as follows: "given a query Q and a document D, the matching R between D and Q is determined by a function F of the exhaustivity of the document about the query (measured by it D implies Q) and the specificity of the document about the query (measured by it Q implies D):

$$R(D,Q) = F[P_K(D \rightarrow Q), P'_K(Q \rightarrow D)]$$

where P, P' are two functions measuring the implications' uncertainties, F is a function that combines these two measures, and K expresses that these implications are evaluated according to a knowledge base K including domain knowledge and knowledge about the users. We use in the following these two implication in a way to find the optimal document components that are responses to Q.
Returning to our context where we base the evaluation of document relevance on implication $\rightarrow_\sigma$, we may then translate Nie's proposition as follows:

$$R(D,Q) = F[P_K(D \rightarrow_\sigma Q), P'_K(Q \rightarrow_\sigma D)]$$

Since we are mainly interested here in some kind of *structural optimization* of responses, we ignore at the moment the definitions of uncertainty evaluations P, P' and F, and we will consider that a document D is relevant to a query Q simply based on the fact that implications holds.
An important point here is to remind about the optimization strategy: as usual, many of them could be defined. Considering the impact of structure, our choice for dynamic ascending attributes is to select the greatest document components (according to $\preceq_{str}$, they correspond to deepest components in the logical structure)

that satisfy the exhaustivity criterion. This means that the exhaustivity criterion has a higher priority rank than the specificity criterion; whenever there is no ideal solution that satisfy both of them (which will be the most frequent case) we will then prefer those satisfying exhaustivity. In the following we propose a way to implement this approach (see also [CK96]), based on the direct use of direct and reverse implications.

We illustrate this process using the example of the document shown in figure 3 of section 3. This figure shows in its left side the logical structure of a document D, and only the *Chapters* and *Subsections* of this document correspond to index components (i.e. potential results of queries). We complete this representation in fig 5 by assigning to index units values of attribute *symbolic*. We also exhibit in this figure the dependency constraint $\rightarrow_\sigma$ that exist between index expressions assigned to *Chapters* and *Subsection*.

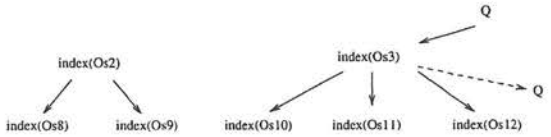

Figure 5: The $\rightarrow_\sigma$ relation between indexes of semantic objects

We describe now the expressions of the indexes (only the final forms of the values of the index are given):

$$
\begin{array}{lll}
index(os_2) & =_\sigma & \delta(os_{13}) \oplus \delta(os_{14}) \oplus \delta(os_9) \\
index(os_8) & =_\sigma & \delta(os_{13}) \oplus \delta(os_{14}) \\
index(os_9) & =_\sigma & I(os_9) \\
index(os_3) & =_\sigma & \delta(os_{15}) \oplus \delta(os_{16}) \oplus \delta(os_{17}) \oplus \delta(os_{11}) \oplus \delta(os_{18}) \oplus \delta(os_{19}) \\
index(os_{10}) & =_\sigma & \delta(os_{15}) \oplus \delta(os_{16}) \oplus \delta(os_{17}) \\
index(os_{11}) & =_\sigma & I(os_{11}) \\
index(os_{12}) & =_\sigma & \delta(os_{18}) \oplus \delta(os_{19})
\end{array}
$$

As we described in section 3 dedicated to indexing, we have then the following dependency constraints:

$$
\begin{array}{lll}
index(os_2) & \rightarrow_\sigma & index(os_8) \\
index(os_2) & \rightarrow_\sigma & index(os_9) \\
index(os_3) & \rightarrow_\sigma & index(os_{10}) \\
index(os_3) & \rightarrow_\sigma & index(os_{11}) \\
index(os_3) & \rightarrow_\sigma & index(os_{12})
\end{array}
$$

Let us consider the index expression ie_3 assigned to os_3 and the index expression ie_q involved in Q. Is this object an optimal answer for Q? To ease the explanation, we assimilate D to its index expression ie_3 and its components D_i to the index

expressions ie_{10}, ie_{11} and ie_{12} respectively assigned to os_{10}, os_{11} and os_{12}. The following situations may occur when matching D_i and Q:

1. $D \equiv_\sigma Q$: the index expressions associated to D and Q are logically equivalent. This is the ideal case, and component D is an optimal response to Q (i.e., it satisfies both implications). Though logically possible, this case is of course certainly rare in actual situations; much frequent are the following cases.

2. *not*$(D \rightarrow_\sigma Q)$ *and not*$(Q \rightarrow_\sigma D)$, which means that D does not satisfy both exhaustivity and specificity criteria. The document is not relevant for the query, and so are each of its components (if any). We may assert this due to the dependency constraint: since $\forall ei_i, ei \rightarrow_\sigma ei_i$, having any $ie_i \rightarrow_\sigma ie_q$ would mean, by transitivity of $\rightarrow_\sigma$, that $ie \rightarrow_\sigma ie_q$, equivalent to $D \rightarrow_\sigma Q$ which contraries the hypothesis. The same reasoning applies to the reverse implication.

3. $D \rightarrow_\sigma Q$ *and not*$(Q \rightarrow_\sigma D)$: this case is illustrated by the dotted arrow in figure 5. The document is relevant to the query considering the criteria of exhaustivity. This means that the document represented by os_3 as a whole matches the query. Here we have to notice that it might also exist in its structure a component D_i that also matches the exhaustivity criterion (i.e., such that $D_i \rightarrow_\sigma Q$); in this case D_i would be a better response than D considering the specificity criterion . Of course if we had also $Q \rightarrow_\sigma D_i$ we would match the ideal case described above, but again, this situation will not occur most of the time. More often what happens when considering a sub-unit D_i falls into one of these two possibilities:

   - $(D_i \rightarrow_\sigma Q)$ *and not*$(Q \rightarrow_\sigma D_i)$: it is still possible to find a better match a level down in the indexing structure. This correspond to a *browsing*situation.
   - *not*$(D_i \rightarrow_\sigma Q)$ *and* $(Q \rightarrow_\sigma D_i)$: the component does not satisfy any more the exhaustivity criterion: we have been too deep in the structure and the component does not satisfy the whole query (it is too restrictive). The optimal response is either D, the parent component of D_i, or an other descendant D_j of D.

4. $Q \rightarrow_\sigma D$ *and not*$(D \rightarrow_\sigma Q)$ (see the solid arrow in figure 5): in this case , the document is relevant to the query considering the second criterion only. According to our general strategy, D is not relevant. The query then necessarily also implies all the document's components (due to the transitivity of $rightarrow_\sigma$): D being too restrictive for Q, the same applies for its components.

So, using both implications as filters, we may find structural objects (index objects) that are optimal relatively to the proposed optimization strategy. What

is important to notice is that this strategy works because the index objects are assigned index expressions that satisfy the dependency constraint. Without this property, it would be impossible to assert any property about possible relevance of related components, and hence it would have been impossible to optimize anything. A possible implementation of this approach could be based on a two-steps algorithm:

- Fetch: a preselection of *documents* is made using the exhaustivity criterion ($D \rightarrow_\sigma Q$). This is corresponds exactly to classical retrieval processes on unstructured documents. Note again that according to the index dependency constraint, if the index expression of a document does not imply the query, there cannot be any of its components that implies the query. Hence such documents may be discarded without any further investigation.
- Browse: the structural objects of the preselected documents are investigated: this is done in browsing within their logical structure, and using both implications to select optimal components as described before.

Going back to the query language, one may see how these two implications may be implemented using basic predicates:

- Fetch: here we have to select documents (i.e. structural objects of minimal types according to $\preceq_{tst}$, see section 2.2.1), based on the exhaustivity criterion. Using basic predicates, one may check if a given type t_2 is minimal by evaluating the query $\neg(\exists \mathrm{t}_1) InfTst(\mathrm{t}_1, t_2)$. One might then define a query implementing the fetch operation as:

$$(\neg\exists \mathrm{t}_1)(\exists \mathrm{t}_2) TypeSt(\mathrm{os}, \mathrm{t}_2) \wedge InfTst(\mathrm{t}_1, \mathrm{t}_2) \wedge$$
$$(\exists \mathrm{ie}) HasValue(\mathrm{os}, symbolic, \mathrm{ie}) \wedge MatchSymbolic(\mathrm{ie}, ie_q)$$

  Note that the only result variable is os.

- Browse: when checking the reverse implication, we do not have constraints about the type of document. The corresponding query is then:

$$(\exists \mathrm{ie}) HasValue(\mathrm{os}, symbolic, \mathrm{ie}) \wedge MatchSymbolic(ie_q, \mathrm{ie})$$

Note finally that the whole algorithm may be defined (though in a not optimal way) by the following query which selects every *os* satisfying the exhaustivity criterion and having no component that also satisfies this condition (hence being the optimal structural unit):

$$(\exists \mathrm{ie}) HasValue(\mathrm{os}, symbolic, \mathrm{ie}) \wedge MatchSymbolic(\mathrm{ie}, ie_q) \wedge$$
$$(\neg\exists \mathrm{os}_2) Compstr(\mathrm{os}, \mathrm{os}_2) \wedge$$
$$(\exists \mathrm{ie}_2) HasValue(\mathrm{os}_2, symbolic, \mathrm{ie}_2) \wedge MatchSymbolic(\mathrm{ie}_2, ie_q)$$

The query may of course retrieve several optimal components in the same document; this is natural considering the implemented optimization strategy. The problem of displaying them in the most appropriate way (from the users' point of view), though out of the scope of this study, is an important matter related to interfaces. As mentioned before, one have to remind here that:

- We are no longer in the closed world of querying: navigation is also available at any time, facilitating the access to information closely related to retrieved objects.
- All retrieved objects are structural objects. This means that the interface may make full use the logical structure, navigation links to adequately display the retrieved objects (for example in showing their structural/navigational relationship).

The approach presented above is applicable to any dynamic ascending attribute provided of course that the basic requirements about indexing strategy and query optimization are satisfied.

4.7 Conclusion

In this section we have defined a query language based on a set of predicates and a resolution mechanism that allow the retrieval of structured documents. The main features of multimedia data are imported within the framework of this upper-level model through the basic implication operators noted σ_α , where α stands for any of the five Content Attributes defined in section 2. According to the model, we have focused on hierarchical structures which are of major interest in the context of standards for document representation and also in the context of hypermedia systems. We have also investigated the problem of optimizing query responses in avoiding the production of redundant (relatively to the logical structure) information. Finally, we have proposed a model and a query language that integrates several complementary approaches for retrieving information: content-based querying, database querying and navigation. In our opinion this is extremely useful in the context of complex, multimedia information. Its major limitation at the moment is that it does not integrate uncertainty processing. Our purpose then is to complete the model within WP4 (Integration) in two directions: integration of uncertain information for Content Attributes, and integration of single-media models (i.e. definition of operators $\rightarrow_\alpha$). The first aim will be based on the approach developed by the University of Dortmund (probabilistic datalog) which is quite close to ours. The second goal will be developed mainly in our site for texts and graphics, and in collaboration with IEI-Pisa for the images.

References

[AC95] Anastasia Analyti and Stavros Christodoulakis. Multimedia object modelling and content-based querying. In *Advanced Course Multimedia Database in Perspective*, pages 213–240, Enschede, The Netherlands, June 1995.

[ACG91] M. Agosti, R. Colotti, and G. Gradenigo. A two-level hypertext retrieval model for legal data. *ACM*, pages 316–325, 1991.

[AMC95] M. Agosti, M. Melucci, and F. Crestani. Automatic construction of hypermedia for information retrieval. *Multimedia Systems*, 1995.

[BH94] Peter D. Bruza and T.W.C. Huibers. Investigating aboutness axioms using information fields. In *ACM SIGIR, tutorial*, Dublin, Ireland, July 1994.

[BLC] T.J. Berners-Lee and D. Connolly. *Hypertext Markup Language - 2.0.* MIT/W3C. http://www.w3.org/hypertext/WWW/MarkUp/html-spec/html-spec_ toc.html.

[BLC93] T.J. Berners-Lee and D. Connolly. Hypertext markup language - a representation of textual information and metainformation for retrieval and interchange. internet draft, July 1993.

[Boh95] Klemens Bohm. Building a configurable database application for structured documents. Technical Report 942, GMD-IPSI, September 1995.

[Bur94] Lou Burnard. What is sgml and does it help ? In *ACM SIGIR, tutorial material*, Dublin, Ireland, July 1994.

[CK96] Y. Chiaramella and A. Kheirbek. *Information Retrieval and Hypertext*, chapter An Integrated model for Hypermedia and Information Retrieval. Kluwer Academic Publ., 1996.

[CT89] W. B. Croft and H. Turtle. A retrieval model for incorporating hypertext links. In *Second ACM Conference on Hypertext (Hypertext'89)*, pages 213–224, Pittsburgh, USA, 1989. ACM.

[DR93] M. D. Dunlop and C. J. Van Rijsbergen. Hypermedia and free text retrieval. *Information Processing & Management*, 29(3):287–298, 1993.

[Erf94] Robert Erfle. Hytime as the multimedia document model of choice. *IEEE*, pages 445–454, 1994.

[Hal88] F. G. Halasz. Reflections on notecards: Seven issues for the next generation of hypermedia system. *Communication of the ACM*, 31(7):836–852, july 1988.

[Hor85] W. Horak. Office document architecture and office document interchange formats: Current status of international standardization. *IEEE Computer*, 18, November 1985.

[KC95] Ammar Kheirbek and Yves Chiaramella. Integrating hypermedia and information retrieval with conceptual graphs. In *Conference Hypertext, Information Retrieval and Multimedia (HIM)*, Konstanz, Germany, April 1995.

[LDH92] Z. Li, H. Davis, and W. Hall. Hypermedia links and information retrieval. In *14th Information retrieval Colloquium*, pages 169–180, Lancaster, 1992.

[MBC95] Mourad Mechkour, Catherine Berrut, and Yves Chiaramella. Using conceptual graph frame work for image retrieval. In *International conference on MultiMedia Modeling (MMM'95)*, pages 127–142, Singapore, November 1995.

[Mec95a] M. Mechkour. A conceptual graphics model for infformation retrieval. In *A model for the semantic content of multimedia data*, chapter 3. Deliverable 1, FERMI Esprit BRA project N. 8134, May 1995.

[Mec95b] Mourad Mechkour. Emir2: an extended model for image representation and retrieval. Technical report, Basic Research Action FERMI, n. 8134, 1995.

[Mec95c] Mourad Mechkour. A multifacet formal image model for information retrieval. In *MIRO final workshop, Glasgow, UK.*, page (to appear), september 1995.

[Meg95] Carlo Meghini. A model for image bases and its query facility. In *ACM SIGIR*, Seattle, USA, 1995.

[Nie90] J. Nie. An inormation retrieval model based on modal logic. *Information Processing and Management*, 25(5):477–491, 1990.

[Par95] François Paradis. Modeling textual information. Technical report, Basic Research Action FERMI, n. 8134, 1995.

[RF96] T. Rölleke and N. Fuhr. Retrieval of complex objects using a four-valued logic. Technical report, FERMI Esprit BRA project N. 8134, march 1996.

[Rij86] C.J. Van Rijsbergen. A non-classical logic for information retrieval. *The Computer Journal*, 29(1), 1986.

[Seb94a] F. Sebastiani. A probabilistic terminological logic for modelling information retrieval. In W. B. Croft and C.J. Van Rijsbergen Ed., editors, *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Developments in Information Retrieval*, pages 122–130. ACM, Springler Verlag, July 1994.

[Seb94b] F. Sebastiani. A probabilistic terminological logic for modelling information retrieval. Technical report, FERMI Esprit BRA project N. 8134, March 1994.

[Tha90] C. Thanos. *Multimedia Office Filling: The MULTOS Approach.* North-Holland, 1990.

[TR96] N. Fuhr T. Rölleke. Retrieval of complex objects using a four-valued logic. In *Proceedings of ACM-SIGIR Annual International Conference on Research and Developments in Information Retrieval*, Zurich, August 1996. ACM.

[WB90] T. P. Van Der Weide and P. D. Bruza. Two level hypermedia: An improved architecture for hypertext. In Springer Verlag, editor, *Database and Expert System Application (DEXA'90)*, Vienna, Austria, september 1990.