Foundations of a Multidimensional Query Language for Digital Libraries

Donatella Castelli, Carlo Meghini, Pasquale Pagano
Istituto delle Tecnologie e Scienze dell'Informazione
Consiglio Nazionale delle Ricerche
Area della Ricerca di Pisa, Pisa, Italy
e-mail:{castelli,meghini,pagano}@iei.pi.cnr.it

Abstract

A query language for Digital Libraries is presented, which offers access to documents by structure and sophisticated usage of metadata. The language is based on a mathematical model of digital library documents, centered around a multilevel representation of documents as versions, views and manifestations. The core of the model is the notion of document view, which is recursive, and captures the content and structure of a document. The metadata representation distinguishes between formats and specifications, so being able to accommodate different metadata formats, even for the same document. A query is a logical formula, and its result are the digital library documents satisfying the user query.

1 Introduction

Current Digital Libraries (DLs) strongly resemble traditional libraries based on on-line public access catalogs: the only added value offered by DLs is the on-line availability also of documents [2, 4, 3]. As a consequence, the information retrieval service offered to DL users boils down to queries expressing very simple conditions on descriptive metadata. From an information system point of view, this kind of functionality equates current DLs to pre-database information systems and, what is worse, prevents the full exploitation of DL contents, defeating the huge investments that are necessary to create such contents in the first place.

Emerging standards and well-consolidated technologies can contribute to overcome the present state of affair, permitting the realization of DLs offering information access services that are adequate to user expectations. Three aspects on which progress is possible are:

– more sophisticated usage of metadata[1]. A wide range of metadata formats

[1] throughout the paper, we use the term "metadata" as a synonym of "descriptive metadata".

exist and are actually employed for describing DL content [1, 13, 16]. Potentially, a DL could store information about the metadata models themselves, thus supporting in a principled way the co-existence of more standards in the same system, and even the exploitation of the metamodel in querying. For instance: "What are the fields available in the metadata format x?", or "What different formats are supported for a certain attribute value?".

- structure-based access. Digital formats highlighting document structure are available nowadays (*e.g.* Postscript, PDF), which means that, potentially, all the results obtained for querying structured (*e.g.* XML) or semi-structured data, could be put at work in a DL system. For instance: "What documents have a section titled x?", or "What documents have a reference to x in the bibliography?".

- content-based access. Typically, a DL stores its documents in digital format, which means that, potentially, all the results obtained in content-based multimedia information retrieval, could be put at work in a DL system.

The objective of the present work is to advance the state of the art in DL systems, by introducing a query language that addresses the first two aspects introduced above, while being amenable to be extended to the third one. This query language, which we call "multi-dimensional" as it addresses various dimensions of existence of a document, is to be intended as a foundation for DL systems, in the same sense that first-order predicate calculus is a foundation for relational database systems. Thus, we are not aiming at user-friendliness or efficient processing, but rather *expressivity.* Despite its theoretical nature, this work is carried out in the context of a project aiming at the implementation of a DL system [10, 5], as a feasibility study of a novel information access service that the project will eventually realize.

The paper is structured as follows. The semantic universe of the query language is introduced in 3 stages: the document model (Section 2), the metadata model (Section 3) and the DL model (Section 4). The query language itself is given in Sections 5 and 6. Finally, Section 7 concludes.

As notational conventions, we will let $\mathbf{N}$ stand for the set of natural numbers and $\mathbf{N}_+$ for $\mathbf{N} - \{0\}$. The set of the first i natural numbers, *i.e.* $\{0, 1, \ldots, i-1\}$, will be denoted $[i]$ for short. Given any set A, $A^\star$ denotes the set of finite sequences of A's elements, while $|A|$ stands for the cardinality of set A.

2 The document model

The document model is inspired to the IFLA-FRBR model [16]. It distinguishes four aspects of a document:

(1) *documents,* which model the more general aspect of a document, seen as a distinct intellectual creation. For example, the article "The SOMLib Digital Library System" by Andreas Rauber and Dieter Mekl, the book "Digital

Libraries and Electronic Publishing" by William Arms, the lesson "A dynamic Warehouse for XML Data of the Web" by Serge Abiteboul, the proceeding of the conference ECDL'99, are all modeled as documents.

(2) *document instances,* also referred to as *versions,* which are specific editions of documents. In fact, a document is recognized through its individual instances along the time dimension. The preliminary version of the paper "The SOMLib Digital Library System", the version submitted to the conference ECDL'99, the version published on the ECDL'99 proceeding, are examples of successive editions of the same document.

(3) *views,* which are specific intellectual expressions of document instances. A view is one specific intellectual expression of the version of a document. A view excludes physical aspects that are not integral to the intellectual perception of a document. A document version is perceived through one or more views. For example, the original version of the ECDL'99 proceedings might be disseminated in three different views: (a) a "structured textual view" containing a "Preface" created by the conference Chair, and the list of papers presented to the conference, b) a "textual view structured into thematic sessions", where each session contains the documents presented during that session, and c) the "presentation view", containing the list of the slides presented at the conference.

(4) *manifestations,* which model the physical formats under which a document is disseminated. Example of manifestations are: the MPEG file which maintains the video recording of the presentation made by Andreas Rauber at the ECDL '99 Conference, the AVI file of the same video, the Postscript file of the paper presented by Andreas Rauber at the ECDL '99, etc.

2.1 Documents

Each document is modelled, according to the above requirements, mainly as a sequence of versions, which are drawn from a set $\mathcal{V}$, to be defined next. Formally, the set of documents, $\mathcal{D}$, is the countable set:

$$\mathcal{D} = \{\langle n, h, V, v\rangle \in (\mathbf{N}_+ \times \mathcal{H} \times 2^{\mathcal{V}} \times (\mathbf{N} \to \mathcal{V})) \mid v \text{ is a bijection from } [n] \text{ to } V\}$$

where, for each document $d = \langle n_d, h_d, V_d, v_d\rangle$: (a) n_d is the document *order;* (b) h_d is the document *publishing authority*, which is the entity responsible for the document; as authorities play a major role in the definition of document collections, they are explicitly considered by the model, unlike the (rather large) number of attributes of documents that the model ignores for reasons of relevance; authorities are drawn form a countable set $\mathcal{H}$; (c) V_d are the document *versions;* (d) v_d is the document *version function,* which indexes the document versions with the first n_d natural numbers; v_d is required to be total (to make sure that all indexes have a version), injective and surjective (to make sure that each version is indexed exactly once).

2.2 Versions

As a document version is just a finite set of views, we can use the same style adopted for documents to model the set $\mathcal{V}$,*i.e.*:

$$\mathcal{V} = \{\langle n, W\rangle \in (\mathbf{N}_+ \times 2^{\mathcal{W}}) \mid |W| = n\}.$$

2.3 Views

The definition of views is more complex, thus it will be developed in stages: in the first one, we will describe how views are analyzed; in the second stage, we will provide an informal definition of views, based on the previous analysis; finally, in the third stage we will give the mathematical definition of the set $\mathcal{W}$.

2.3.1 View analysis

Let us consider the view shown in Figure 1.(a), which is a section titled "Views", has an opening text and a couple of subsections, titled "High Views" and "Low Views". Regularity can be imposed on this view by analyzing it as shown in Figure 1.(b), in which the view is seen as an object (the outer box), divided into two pieces: one, the inner top box, holds content, *i.e.* the section opening text; the other, depicted as an oval, holds the section structure, and consisting in the example of two parts: the section title and the section body (for better readability these two parts are separated in Figure 1.(b) by a dashed line). These parts can be analyzed (Figure 1.(c)) in the same way: the title part has content (the title itself) and no sub-parts; the body part, on the other hand, has no content but two sub-parts: the two subsections. The same pattern can finally be applied (Figure 1.(d)) to each subsection.

This example shows that the notion of view is inherently recursive: the basic elements of the recursion, technically the *atomic views,* have just content and no sub-elements, hence no structure (the title part of the previous example is one such element). All the other views are built by aggregating these elements into hierarchical structures, and endowing with content each so created view (*e.g.* the section in Figure 1).

For simplicity, our model will only consider contents of type text, even though the extension to other types is possible [14]. As far as structure is concerned, we consider ordered sets and sequences. The former kind is useful to model views whose elements are, albeit ordered for formal reasons, to be regarded as likewise; this could be the case of a list of references in a paper, which are in fact order even though their order does not really matter. Sequences are useful when structure implies order, as for instance amongst the chapters of a book.

Both structure and content can be enriched. As for the former, we add labels to views, in order to embody structural information that may be very useful in querying. Labels can be envisioned in Figures 1.(b)-(d) as tags attached to ovals; these labels could be *SECTION* for the oval in Figure 1.(b), *TITLE* and *BODY* for the ovals in Figure 1.(c) (from the top down), and *SUBSECTION*

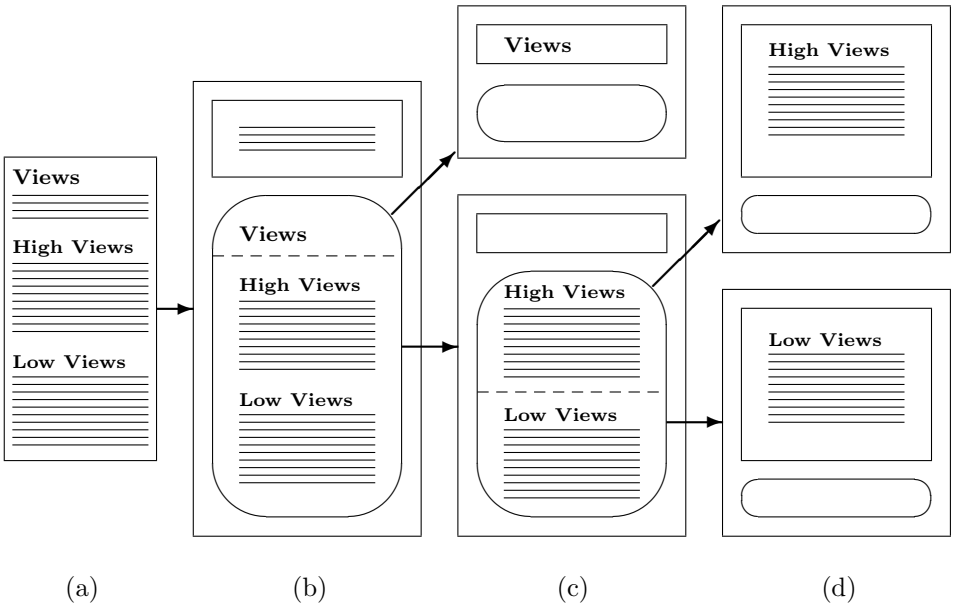

Figure 1: View analysis.

for both the ovals in Figure 1.(d). As for content, in the context of a digital library, content implies metadata, that is descriptions of content in standard formats. As there are many metadata formats, it is better to assume that each view may have several descriptions, in different formats, for its content.

2.3.2 Informal definition of views

Let $\mathcal{L}$ be a countable set of *labels,* and $\mathcal{T}$ a disjoint, countable set of *content elements*, *i.e.* text. Also, we will need one more object e, not belonging to any of the sets introduced so far, to represent the basic structure for building complex views, *i.e.* the *empty view.* Informally, views are defined as follows:

1. the empty view e is a view;
2. if $w_1, \ldots, w_k$, for $k \geq 1$, are views, $l \in \mathcal{L}$ and $t \in \mathcal{T}$, then:
   (i) the object having l as label, t as content and the ordered set $\{w_1, \ldots, w_k\}$ as structure, is a view;
   (ii) the object having l as label, t as content and the sequence $\langle w_1, \ldots, w_k \rangle$ as structure, is a view;
3. nothing else is a view.

For instance, the view shown in Figure 1.(a) is represented by the object $w \in \mathcal{W}$, defined as follows, letting t be the opening text of the section, t_1 and t_2 the

```
<?xml version="1.0" standalone="yes"?> <!DOCTYPE SECTION [
    <!ELEMENT SECTION (TITLE, BODY)>
    <!ELEMENT TITLE (#PCDATA)>
    <!ELEMENT BODY (SUBSECTION+)>
    <!ELEMENT SUBSECTION (#PCDATA)>
]> <SECTION>
     <TITLE> Views </TITLE>
     <BODY>
       <SUBSECTION> t1 </SUBSECTION>
       <SUBSECTION> t2 </SUBSECTION>
     </BODY>
   </SECTION>
```

Figure 2: XML serialization of a view.

contents of the two subsections (title included), and using parentheses to enclose all the elements of a view (*i.e.* label, content and structure):

$$w = (SECTION,\ t,\ \{(TITLE,\ \text{“}Views\text{”},\ e),\ (BODY,\ \lambda,\ \langle(SUBSECTION,\ t_1,\ e\ ),\ (SUBSECTION,\ t_2,\ e\ )\rangle\ )\ \}\ ).$$

Figure 2 shows an XML serialization of w, consisting of a DTD followed by the actual data. As in XML it is not possible to differentiate sequences and ordered sets, no bijective mapping between our model and XML exists: different views may result in the same XML data, thus in translating from XML back to our model, only views entirely structured as either sequences or ordered sets can be generated.

2.3.3 Formal definition of views

In order to make the above definition of view precise, we rely on the notions of constructors and closed sets [11]. Let U be any non-empty set. The *view constructors* on U are any two families of functions defined as follows:

$$A_U = \{\alpha_i :\ \mathcal{L} \times \mathcal{T} \times U^i \to U \mid i \in \mathbf{N}_+\}$$
$$B_U = \{\beta_i :\ \mathcal{L} \times \mathcal{T} \times U^i \to U \mid i \in \mathbf{N}_+\}$$

and satisfying the following conditions, for all $i, j \in \mathbf{N}_+$, $i \neq j$: (1) $ran(\alpha_i) \cap ran(\alpha_j) = \emptyset$, where $ran(\alpha_i) = \{u \in U \mid u = \alpha_i(\mathbf{u}) \text{ for some } \mathbf{u} \in U^i\}$; (2) $ran(\beta_i) \cap ran(\beta_j) = \emptyset$; (3) $ran(\alpha_i) \cap ran(\beta_i) = \emptyset$; (4) α_i is injective, that is for all $\mathbf{u}, \mathbf{u}' \in \mathcal{U}^i$ $\alpha_i(\mathbf{u}) = \alpha_i(\mathbf{u}')$ implies $\mathbf{u} = \mathbf{u}'$; (5)β_i is injective. Taken all together, these conditions amount to say that the functions in $(A_U \cup B_U)$ are injective; this guarantees unique readability of views.

Let $\mathcal{U}$ be a countably infinite and let $\mathcal{B} = \{e\}$ be a subset of $\mathcal{U}$. The set $\mathcal{W}$ of views is the closure of $\mathcal{B}$ under $A_\mathcal{U}$ and $B_\mathcal{U}$, that is:

1. $\mathcal{W}$ contains $\mathcal{B}$;
2. $\mathcal{W}$ is closed under $A_{\mathcal{U}} \cup B_{\mathcal{U}}$;
3. if $\mathcal{W}'$ is a subset of $\mathcal{U}$ that is closed under $A_{\mathcal{U}} \cup B_{\mathcal{U}}$, then $\mathcal{W} \subseteq \mathcal{W}'$.

It can be shown [11] that $\mathcal{W}$ exists, is unique, and is given by:

$$\mathcal{W} = \mathcal{B} \cup \bigcup_{R \in (A_{\mathcal{U}} \cup B_{\mathcal{U}})} R(\mathcal{W}^{n_R})$$

where for each $R \in (A_{\mathcal{U}} \cup B_{\mathcal{U}})$, R is n_R-ary. $\mathcal{W}$ is said to be *freely generated* by $(A_{\mathcal{U}} \cup B_{\mathcal{U}})$ over $\mathcal{B}$, which guarantees uniqueness and existence of the recursive functions on $\mathcal{W}$ to be defined next.

2.3.4 View selectors

For ordered sets, we want to be able to denote the association between a view and the members of the ordered set in which the view is structured. Accordingly, we define the relation among views $\epsilon \subseteq \mathcal{W} \times \mathcal{W}$ as follows:

$$\begin{aligned} \epsilon \quad = \quad & \{(w, w') \in \mathcal{W}^2 \mid \text{for some } l \in \mathcal{L},\ t \in \mathcal{T}, \text{ and } i, j \in \mathbf{N}_+, \\ & w' = \alpha_i(l, t, w_1, \ldots, w_{j-1}, w, w_{j+1}, \ldots, w_i)\} \end{aligned}$$

For sequences, we want to be able to select the k-th element of the sequence that makes up a view structure. Accordingly, we define the function $\sigma : (\mathbf{N}_+ \times \mathcal{W}) \rightarrow \mathcal{W}$ as follows:

$$\sigma(k, w) = \begin{cases} w' & \text{if for some } l \in \mathcal{L},\ t \in \mathcal{T}, \text{ and } i \in \mathbf{N}_+, \\ & w = \beta_i(l, t, w_1, \ldots, w_{k-1}, w', w_{k+1}, \ldots, w_i) \\ e & \text{otherwise} \end{cases}$$

Finally, we make available a relation that captures view composition regardless of the constructor. This is the relation among views $\pi \subseteq \mathcal{W} \times \mathcal{W}$:

$$\pi = \epsilon \cup \{(w, w') \in \mathcal{W} \times \mathcal{W} \mid \text{for some } k \in \mathbf{N}_+,\ w = \sigma(k, w')\}.$$

3 The metadata model

In order to develop a model of metadata able to incorporate as many metadata formats as possible, we look at the area of conceptual modeling [8], and specifically at that of semantic data modeling. The model we are going to propose next is inspired to a class of models known as object-oriented semantic data models [12, 15]. The model is divided in two parts: one addressing metadata formats and the other addressing metadata specifications.

Let us be given two countable sets: $\mathcal{S}$, the *classes,* and $\mathcal{P}$, the *properties.* Informally, a *metadata format* (MF) is a 4-tuple $\langle S, P, def, \Rightarrow \rangle$, where:

– $S \subseteq \mathcal{S}$ is the finite set of the MF classes;

Table 1: Properties and their definitions for the Dublin Core format

	$def(C,P) = D$	
Property	(C,P)	D
`Title`	(`DublinCore`, `Title`)	`String`
`Creator`	(`DublinCore`, `Creator`)	`String`
`Subject`	(`DublinCore`, `Subject`)	`SubjectScheme`
`Description`	(`DublinCore`, `Description`)	`String`
`Publisher`	(`DublinCore`, `Publisher`)	`String`
`Contributor`	(`DublinCore`, `Contributor`)	`String`
`Date`	(`DublinCore`, `Date`)	`DateScheme`
`Type`	(`DublinCore`, `Type`)	`DCMITypeVoc`
`Format`	(`DublinCore`, `Format`)	`String`
`Identifier`	(`DublinCore`, `Identifier`)	`URI`
`Source`	(`DublinCore`, `Source`)	`URI`
`Language`	(`DublinCore`, `Language`)	`LanguageScheme`
`Relation`	(`DublinCore`, `Relation`)	`URI`
`Coverage`	(`DublinCore`, `Coverage`)	`CoverageScheme`
`Rights`	(`DublinCore`, `Rights`)	`String`

- $P \subseteq \mathcal{P}$ is the finite set of the MF properties;
- def: $(S \times P) \to S$ is the *property definition function;*
- $\Rightarrow \subseteq S \times S$ is the MF *subclass* relation.

Classes group objects of the same kind, *i.e.* having the same properties, or attributes. These properties are defined by the partial function *def,* which assigns to a class and a property at most one domain, that is the class from which the property may assume values. The subclass relation models an intuitive generalization/specialization criterion. Formally, the set of metadata format $\mathcal{F}$ is defined to be:

$$\mathcal{F} \subseteq 2^{\mathcal{S}} \times 2^{\mathcal{P}} \times ((\mathcal{S} \times \mathcal{P}) \to \mathcal{S}) \times 2^{\mathcal{S} \times \mathcal{S}}$$

such that, for all $\langle S, P, def, \Rightarrow \rangle \in \mathcal{F}$: (a) def: $(S \times P) \to S$; (b) $(S, \Rightarrow)$ is a partial order; (c) (structural inheritance) for all $c, c', d \in S$ and $p \in P$, if $def(c,p) = d$ and $\Rightarrow (c', c)$ then there exists $d' \in S$ such that $def(c', p) = d'$ and $\Rightarrow (d', d)$.

Tables 1 and 2 show a formulation of the Dublin Core as a metadata format $m_{DC} = \langle S_{DC}, P_{DC}, def_{DC}, \Rightarrow_{DC} \rangle$, where: S_{DC} is given by the class `DublinCore` and the classes shown in the third column of Table 1; P_{DC} consists of the properties listed in the first column of Table 1; def_{DC} is defined in the second and third column of Table 1; $\Rightarrow_{DC}$ is shown in Table 2.

For metadata specifications (MS) we will need a countable set of *objects* $\mathcal{O}$. Given a MF $m = \langle S, P, def, \Rightarrow \rangle$, a *metadata specification for m,* is a triple $\langle O, \rightarrow, prop \rangle$, where:

Table 2: The specialization relation of the DC metadata format

$\Rightarrow (C, D)$		$\Rightarrow (C, D)$	
C	D	C	D
LCSH	SubjectScheme	ISO639-2	LanguageScheme
MeSH	SubjectScheme	RFC1766	LanguageScheme
DDC	SubjectScheme	DCMIPoint	CoverageScheme
LCC	SubjectScheme	ISO3166	CoverageScheme
UDC	SubjectScheme	DCMIBox	CoverageScheme
DCMIPeriod	DateScheme	TGN	CoverageScheme
W3C-DTF	DateScheme	DCMIPeriod	CoverageScheme
		W3C-DTF	CoverageScheme

- $O \subseteq \mathcal{O}$ is the finite set of MS objects;
- $\rightarrow \subseteq O \times S$ is the MS *instance* relation;
- *prop*: $(O \times P) \rightarrow O$ is the *property* function.

The objects in O are the entities that are used to represents metadata values. An object may be a simple value such as a string or a number, or may be complex, that is endowed with properties, such as dates or any other user-defined object. The *prop* function assigns to each object the properties which it is entitled to, in accordance to the property definitions in *def*.

There are three conditions that we require on a MS: *instantiation*, that is each object must be instance of at least one class; *property induction,* that is each property of an object must be induced by an appropriate definition; and *extensional inheritance,* that is if an object is an instance of a class, then it is also instance of all the superclasses of that class. Formally, we will capture these conditions in two stages. In the first one, the set of metadata specifications $\mathcal{E}$ is defined to be: $\mathcal{E} \subseteq 2^{\mathcal{O}} \times 2^{\mathcal{O} \times \mathcal{S}} \times ((\mathcal{O} \times \mathcal{P}) \rightarrow \mathcal{O})$ such that, for all $\langle O, \rightarrow, prop \rangle \in \mathcal{E}$ (instantiation): $dom(\rightarrow) = O$. In the second stage, the set of *metadata values,* $\mathcal{M}$, is defined to be $\mathcal{M} \subseteq \mathcal{F} \times \mathcal{E}$ such that, for all $(\langle S, P, def, \Rightarrow \rangle, \langle O, \rightarrow, prop \rangle) \in \mathcal{M}$, the following hold:

- $range(\rightarrow) \subseteq S$;
- (property induction) for all $o, o' \in O$ and $p \in P$, $prop(o, p) = o'$ implies that there exists classes $c, c' \in S$ such that: $\rightarrow (o, c)$, $def(c, p) = c'$ and $\rightarrow (o', c')$,
- (extensional inheritance) for all $o \in O$ and $c, c' \in S$, $\rightarrow (o, c)$ and $\Rightarrow (c, c')$ imply $\rightarrow (o, c')$.

A metadata value is thus a pair whose first member is a MF giving the "type" of the second member, a MS. For instance, a Dublin Core metadata value of the Joyce's *Ulysses* could be the pair (m_{DC}, s_{DC}) where m_{DC} is as given in the

previous Section and s_{DC} includes one object, `ulysses`, representing the whole specification; `ulysses` is an instance of the class `DublinCore` (*i.e.* , the pair (`ulysses`,`DublinCore`) is in $\rightarrow$) and as such specifies a value for each of the 15 properties of the `DublinCore` class, as desired. As mandated by the property induction principle, each such value must be an instance of the class that the function *def* assigns as range of the property.

4 Digital Libraries Information Space

A digital library information space (DLIS) consists of two main things: the documents and the metadata. Technically, a DLIS δ is a pair $\delta = (\Delta, \Lambda)$ where Δ is a *document base* and Λ is a *metadata base.*

A document base Δ is given by $\Delta = \langle D, A, r \rangle$ where:

- $D \subseteq \mathcal{D}$ is a set of documents satisfying two constraints: (1) no two documents can share the same version, and (2) no two authorities can have the same document;
- $A \subseteq \mathcal{A}$ is a set of manifestations, where $\mathcal{A}$ stands for the set of all manifestations;
- r is a relation that associates the views in D with the manifestations in A. Formally, letting:

  $$V_D = \cup_{d \in D} V_d \qquad \text{and} \qquad W_D = \cup_{v \in V_d} W_v,$$

  $r \subseteq W_D \times A$. We further require r to be injective, so that the same manifestation cannot be associated to two different views.

Unlike documents, metadata are not independent entities, in that they exist in order to describe something else. Accordingly, given a document base Δ, a metadata base Λ for $\Delta = \langle D, A, r \rangle$ is given by $\Lambda = \langle F, E, g \rangle$ where:

- $F \subseteq \mathcal{F}$ is a set of metadata formats;
- $E \subseteq \mathcal{E}$ is a set of metadata specifications;
- g is a function that assigns metadata specifications in E to document entities in D, according to the metadata formats in F. Formally, letting $\Omega_D = D \cup V_D \cup W_D \cup A$, $g : (\Omega_D \times F) \rightarrow E$ such that, for all $x \in \Omega_D$, $f \in F$, $g(x, f) = e$ implies $(f, e) \in \mathcal{M}$.

Ω_D collects all the entities to which metadata can be attached: documents (in D), their versions (in V_D), the views of these versions (in W_D) and the manifestations in the DLIS. $g(x, f)$ gives the metadata specification, e, assigned to an element x in Ω_D for a given metadata format $f \in F$. The condition that (f, e) be in $\mathcal{M}$ guarantees that e is a specification in the appropriate format f.

Table 3: Syntax and semantics of document predicate symbols

Predicate sym.	Semantics
`Document`	$(\delta, \varphi) \models$ `Document(d)` iff $d \in D$
`Version`	$(\delta, \varphi) \models$ `Version(v)` iff $v \in V_D$
`View`	$(\delta, \varphi) \models$ `View(w)` iff $w \in W_D$
`Manifestation`	$(\delta, \varphi) \models$ `Manifestation(a)` iff $a \in A$
`HasAuthority`	$(\delta, \varphi) \models$ `HasAuthority(d,h)` iff $d = \langle n, h, V, v\rangle \in D$
`HasVersion`	$(\delta, \varphi) \models$ `HasVersion(d,v)` iff $d = \langle n, h, V, v\rangle \in D$ and $v \in V$
`HasView`	$(\delta, \varphi) \models$ `HasView(v,w)` iff $v = \langle m, W\rangle \in V_D$ and $w \in W$
`HasManifest`	$(\delta, \varphi) \models$ `HasManifest(w,a)` iff $w \in W_D$ and $(w, a) \in r$

5 The query language

This section presents a language for querying digital libraries. For reasons of space, familiarity with first-order syntax is assumed.

5.1 Sorts and semantic structure

A many-sorted logic is the most appropriate tool to take advantage at the syntactic level of the structure of the semantic universe. Thus, our sorts are the sets which have been defined in the previous part of the paper. For each of these sort, we will have in the language: (a) a countable set of constants, which are intended to be names for the objects in the sort; (b) a countable set of variables, which are intended to range over the objects in the sort; and (c) an existential quantifier, which will be denoted by subscripting the symbol $\exists$ by the sort name; thus, for instance, the existential quantifier for metadata formats will be $\exists_{\mathcal{F}}$. The intuitive reading of $(\exists_{\mathcal{F}} x)$, where x is a variable of sort $\mathcal{F}$, is "there exists a metadata format x".

The language has no function symbols, and several predicate symbols, which are illustrated below, categorized by the dimension of DLIS objects they refer to. In order to introduce these symbols and their semantics we fix the semantic structure of the language, that is the class of mathematical objects which will be used to assign meaning to queries. A *denotation function* is a bijection, mapping each constant symbol of each sort to an object of the corresponding set. On keeping with simplicity, amongst the denotation functions, we chose as *the* denotation function φ of our semantics, the one which equates the query language and the metalanguage ; so, since we have used throughout the paper the (meta)symbol d to denote a generic document, we will use the constant `d` to denote the same document in queries, *i.e.* $\varphi($`d`$) = d$. A *DLIS structure* is a pair (δ, φ), where δ is a DLIS. Notice that the function φ does not change from structure to structure.

Table 4: Syntax and semantics of structure predicate symbols

Predicate s.	Semantics
`In`	$(\delta, \varphi) \models$ `In(w, w')` iff $w, w' \in W_D$ and $(w, w') \in \epsilon$
`Of`	$(\delta, \varphi) \models$ `Of(w, n, w')` iff $w, w' \in W_D$ and $w = \sigma(n, w')$
`PartOf`	$(\delta, \varphi) \models$ `PartOf(w, w')` iff $w, w' \in W_D$ and $(w, w') \in \pi$
`Root`	`Root(w)` iff `View(w)` $\wedge \neg(\exists_{\mathcal{W}} x)$`PartOf(w,`$x$`)`
`Leaf`	`Leaf(w)` iff `View(w)` $\wedge \neg(\exists_{\mathcal{W}} x)$`PartOf(`$x$`, w)`
`HasParent`	`HasParent(w,w')` iff `PartOf(w, w')`
`HasChild`	`HasChild(w,w')` iff `PartOf(w', w)`
`HasAncestor`	$(\delta, \varphi) \models$ `HasAncestor(w, w')` iff $w, w' \in W_D$ and $(w, w') \in \pi^*$
`HasDescendant`	$(\delta, \varphi) \models$ `HasDescendant(w, w')` iff $w, w' \in W_D$ and $(w', w) \in \pi^*$
`IsLabelled`	$(\delta, \varphi) \models$ `IsLabelled(w, l)` iff $w \in W_D$ and for some $i \in \mathbf{N}_+$, $w = \alpha_i(l, t, w_1, \ldots, w_i)$ or $w = \beta_i(l, t, w_1, \ldots, w_i)$

5.2 Predicate symbols

Table 3 shows syntax and semantics of the predicate symbols for querying the various entities of a document and their relationships. The first column of the table gives the symbol, while the second gives its semantics. Semantics is defined relatively to a DLIS structure, and is given by the conditions that make an assertion involving the predicate symbol *true* in the DLIS structure. For instance, the assertion `Document(d)` turns out to be true in a DLIS structure (δ, φ) just in case the document d, which is the object denoted by the name `d`, is in the set of documents D, which is part of the document base Δ, which is in turn part of the DLIS δ. By the same semantics, the assertion $(\exists_{\mathcal{D}} x)$`Document(`$x$`)` reads as "there exists an object of sort $\mathcal{D}$ that is a document" and is true in a DLIS structure just in case the set D is not empty. If we remove the existential quantifier, we obtain the open formula `Document(`x`)` so called because in it the variable x is free, *i.e.* not bound to any quantifier. The last formula denotes all the objects u in the domain of interpretation whose names would make the formula true, if used in place of x. More simply, we say that the object u (a document, in the present case), *satisfies* the query. Thus, the above formula is satisfied by all the documents in the DLIS.

The usage of free variables is essential for querying. Since each free variable "returns" an object, one uses one free variable for each object desired in the answer to a query. Thus, in oder to retrieve all the documents *and* their versions, the following formula must be used `HasVersion(`x, y`)`.

5.3 Structural predicate symbols

Table 4 shows the set of predicate symbols for specifying conditions on view structure. These symbols are divided into three sets. From the top down, we have:

Table 5: Syntax and semantics of metadata predicate symbols

Predicate s.	Semantics
`HasMetadata`	$(\delta, \varphi) \models$ `HasMetadata(b,f,e)` iff $g(b, f) = e$
`HasClass`	$(\delta, \varphi) \models$ `HasClass(f,c)` iff $f = \langle S, P, def, \Rightarrow \rangle \in F$ and $c \in S$
`HasProperty`	$(\delta, \varphi) \models$ `HasProperty(f,p)` iff $f = \langle S, P, def, \Rightarrow \rangle \in F$ and $p \in P$
`HasDef`	$(\delta, \varphi) \models$ `HasDef(f,c,p,d)` iff $f = \langle S, P, def, \Rightarrow \rangle \in F$ and $def(c, p) = d$
`IsA`	$(\delta, \varphi) \models$ `IsA(f,c,d)` iff $f = \langle S, P, def, \Rightarrow \rangle \in F$ and $(c, d) \in \Rightarrow$
`HasObject`	$(\delta, \varphi) \models$ `HasObject(e,o)` iff $e = \langle O, \rightarrow, prop \rangle \in E$ and $o \in O$
`Instance`	$(\delta, \varphi) \models$ `Instance(e,o,c)` iff $e = \langle O, \rightarrow, prop \rangle \in E$ and $(o, c) \in \rightarrow$
`Prop`	$(\delta, \varphi) \models$ `Prop(e,o,p,o')` iff $e = \langle O, \rightarrow, prop \rangle \in E$ and $prop(o, p) = o'$

- The predicate symbols addressing the structure of a view; thus, in order to state that the desired view is the 3rd component of a known view `w`, one uses the formula $\texttt{Of}(x, 3, \texttt{w})$.
- The symbols allowing to "navigate" a view structure on the basis of the parent-child relationship. Notice that the ancestor and descendant relationships are the transitive closure of the parent and child relationships, respectively, and this fact is captured semantically by using the transitive closure of the relation π, denoted π^*.
- A special symbol to exploit labels as node markers. For instance, by combining the last two sets of symbols, one can request a chapter with a figure by means of the formula

  $$\texttt{IsLabelled}(x, \texttt{Chapter}) \wedge ((\exists_{\mathcal{W}} y)\texttt{HasAncestor}(y, x) \wedge \texttt{IsLabelled}(y, \texttt{Figure}))$$

It has already been pointed out that there exists a strong relation between views and XML data. At query level, this relation can be exploited in two ways: one would be to serialize views as shown in Section 2.3.2 and query them by using any of the several languages that are being studied for XML, such as XQUery [7]. The other way would be to import in our query language significant portions of XML query languages, making them specialized sub-languages; a good candidate for this role would be [6], a language for addressing parts of an XML document which could be cast in the model ontology.

5.4 Metadata predicate symbols

Table 5 illustrates the predicate symbols for querying metadata. These can be divided in 3 sets:

- the symbol `HasMetadata`, which links a document entity in Ω to its metadata values;

– the symbols for querying metadata formats, which are: (a) `HasClass`, which is true of a format and a class if the latter is part of the former; (b) `HasProperty`, which does the same for properties; (c) `HasDef`, which mirrors property definitions; and (d) `IsA`, which permits to query the subclass hierarchy of a given format:

– the symbols for querying metadata specifications, which are the remaining symbols in the Table, and whose role is perfectly analogous to those of the previous set of symbols.

Thus, in order to know the fields available in the metadata format f one uses the formula:

$$(\exists_{\mathcal{C}} x_c)(\exists_{\mathcal{C}} x_d)\texttt{HasDef}(f, x_c, x, x_d),$$

while for the classes that are in the metadata formats associated to a certain known document d, one uses the formula:

$$(\exists_{\mathcal{F}} x_f)(\exists_{\mathcal{E}} x_e)\texttt{HasMetadata}(\texttt{d}, x_f, x_e) \wedge \texttt{HasClass}(x_f, x).$$

For the range of the property `date` in the same metadata formats of the previous formula:

$$(\exists_{\mathcal{F}} x_f)(\exists_{\mathcal{E}} x_e)(\exists_{\mathcal{C}} x_c)\texttt{HasMetadata}(\texttt{d}, x_f, x_e) \wedge \texttt{HasDef}(x_f, x_c, \texttt{date}, x).$$

In the same context, the specializations of the class `author` can be found by:

$$(\exists_{\mathcal{F}} x_f)(\exists_{\mathcal{E}} x_e)\texttt{HasMetadata}(\texttt{d}, x_f, x_e) \wedge \texttt{IsA}(x_f, x, \texttt{author}).$$

As far as querying metadata specifications, the following formula returns the views that have the field `Creator` of the Dublin Core metadata format equal to "John Doe":

$$(\exists_{\mathcal{F}} x_f)(\exists_{\mathcal{E}} x_e)(\exists_{\mathcal{O}} x_o)\texttt{View}(x) \wedge \texttt{HasMetadata}(x, x_f, x_e) \wedge \texttt{HasClass}(x_f, \texttt{DublinCore}) \wedge \texttt{Prop}(x_e, x_o, \texttt{Creator}, \texttt{JohnDoe}).$$

6 Querying Digital Libraries

For organizational reasons, the documents in a DLIS are organized in *collections,* which are virtual subsets of the document base, collecting documents on the basis of their authorities. Collections are defined via a special kind of queries, called *collection definition predicates* (CDPs). A CDP is a formula of the query language, having the form:

$$(\texttt{HasAuthority}(x, \texttt{a}_1) \vee \ldots \vee \texttt{HasAuthority}(x, \texttt{a}_n)) \wedge \phi(x)$$

for $n \geq 1$, where $\phi(x)$ is a query having x as only free variable and including only metadata predicate symbols. In practice, a CDP denotes a set of documents

in the document base, namely those having as authority one of $a_1, \ldots, a_n$, and satisfying the query $\phi(x)$.

An *extended DLIS* is a tuple $\Gamma = (\delta, \Phi_1(x), \ldots, \Phi_k(x))$ where δ is a DLIS and $\Phi_1(x), \ldots, \Phi_k(x)$ are CDPs, for $k \geq 0$. A *query* to an extended DLIS Γ is given by $(\Upsilon, \Psi(x_1, \ldots, x_m))$, where Υ is a (possibly empty) subset of the collections in Γ, the *target* of the query, and $\Psi(x_1, \ldots, x_m)$ is a formula of the previously introduced language whose only free variables are $x_1, \ldots, x_m$, for $m \geq 1$. The *predicate* of a query $(\Upsilon, \Psi(x_1, \ldots, x_m))$ is the formula $\Xi(x, x_1, \ldots, x_m)$ given by

$$(\bigvee_{\psi(x) \in \Upsilon} \psi(x)) \wedge \Psi(x_1, \ldots, x_m).$$

In practice, the predicate of the query is the formula that denotes all the documents x belonging to anyone of the target collections (as required by the first disjunction of the above formula), *and* satisfying $\Psi(x_1, \ldots, x_m)$. In case Υ is empty, the semantics of the language guarantees that all the documents in the document bases are considered.

The *result set* of a query $(\Upsilon, \Psi(x_1, \ldots, x_m))$ to the extended DLIS $\Gamma = (\delta, \Phi_1, \ldots, \Phi_k)$, is the set of $(m+1)$-tuples $\langle u_0, u_1, \ldots, u_m \rangle$ that satisfy the query predicate $\Xi(x, x_1, \ldots, x_m)$, that is:

$$(\delta, \varphi) \models \Xi(\mathtt{u}, \mathtt{u}_1, \ldots, \mathtt{u}_m).$$

7 Conclusions and future work

The presented query facility permits structure- and sophisticate metadata-based document retrieval, and as such advances current DL models. In order to arrive at a precise definition of this query facility, a mathematical model of Digital Libraries has been developed, and used to assign meaning to user queries. This level of precision makes the language universally available, implementable and, also important, extensible with more capability, notably content-based access.

As already pointed out, this work, albeit of a foundational nature, is carried out within a project, aiming at the development of a DL system. We plan to implement a significant fragment of the language presented. Specifically, our work programme is to cut down the expressivity of the language down to a subset of first-order logic, which is powerful enough to express all significant queries but also simpler from the computational point of view. The idea is to partition the language presented here into a set of sub-languages, each addressing a different dimension of documents, and use the general framework of logic for combining these sub-languages. In this respect, the inclusion of content-based retrieval in the language presents no conceptual problem, as a logical framework supporting this kind of functionality has already been developed [14]. An architecture for implementing the language is described in [9].

References

[1] Dublin core metadata element set, version 1.1: Reference description. http://dublincore.org/documents/dces/.

[2] NCSTRL: Networked computer science technical reference library. http://www.ncstrl.org.

[3] Networked digital library of theses and dissertations. http://www.theses.org.

[4] The new zealand digital library. http://www.nzdl.org.

[5] SCHOLNET: A digital library testbed to support networked scholarly communities. http://www.ercim.org/scholnet.

[6] XML path language (XPath) 2.0. http://www.w3.org/TR/xpath20.

[7] XQuery 1.0: An XML query language. http://www.w3.org/TR/xquery.

[8] M. Brodie, J Mylopoulos, and J. Schmidt, editors. *On Conceptual Modelling*. Springer Verlag, 1984.

[9] D. Castelli and P. Pagano. OpenDLib: a digital library service system. In *Proc. of ECDL2002, 6th European Conf. on Research and Advanced Technology for Digital Libraries*, Rome, I, 2002.

[10] D. Castelli and P. Pagano. Global system architecture report. Technical Report D2.2.1, Scholnet Project Deliverable, 2001. http://www.ercim.org/scholnet/results.html.

[11] P. Fejer and D. Simovici. *Mathematical Foundations of Computer Science. Volume 1*. Springer-Verlag, New York, 1991.

[12] R. Hull and R. King. Semantic database modeling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–259, 1987.

[13] Network Development Library of Congress and MARC Standards Office. MARC standards. http://lcweb.loc.gov/marc/.

[14] C. Meghini, F. Sebastiani, and U. Straccia. A model of multimedia information retrieval. *JACM*, 48(5):909–970, 2001.

[15] J. Mylopoulos and A. Borgida. Some features of the Taxis data model. In *Proc. of the 6th VLDB*, pages 399–410, Montreal, CA, 1980.

[16] K.G. Saur. Functional requirements for bibliographic records. Final report. Technical report, Munchen, 1998. http://www.ifla.org/VII/s13/frbr/frbr.pdf.