On the Effective Manipulation of Digital Objects: A Prototype-Based Instantiation Approach

Kostas Saidis, George Pyrounakis and Mara Nikolaidou

Libraries Computer Center
Department of Informatics and Telecommunications
University of Athens
University Campus, Athens, 157 84, Greece
saiko@di.uoa.gr, forky@libadm.uoa.gr, mara@di.uoa.gr

Abstract. This paper elaborates on the design and development of an effective digital object manipulation mechanism that facilitates the generation of configurable Digital Library application logic, as expressed by collection manager, cataloguing and browsing modules. Our work aims to resolve the issue that digital objects typing information can be currently utilized only by humans as a guide and not by programs as a digital object type conformance mechanism. Drawing on the notions of the Object Oriented Model, we propose a "type checking" mechanism that automates the conformance of digital objects to their type definitions, named *digital object prototypes*. We pinpoint the practical benefits gained by our approach in the development of the University of Athens Digital Library, in terms of code reuse and configuration capabilities.

1 Introduction

In the context of Digital Libraries, a digital object can be conceived as a human generated artifact that encapsulates underlying digital content and related information [7, 13]. Although variations on representation and encoding issues may exist, this information is used to describe, annotate, link and manipulate the object's digital content. However, the term "object" is used in a far richer context in the field of Software Engineering: in the Object Oriented (OO) model an object acts as the container of both data (its state) and behavior (its functionality) and conforms to a type definition, named class.

The Metadata Encoding and Transmission Standard (METS) [11] refers to digital objects in terms of XML documents, providing detailed specifications of its sections, comprised of descriptive and administrative metadata, files, structural maps and links. Moreover, METS supports behaviors attached on digital content, through the *Behavior* section of the METS object, "that can be used to associate executable behaviors with content" [11]. Even though METS uses the notion of Profiles to refer to "classes" of digital objects, a METS document's Profile [10] is "intended to describe a class of METS documents in sufficient detail to provide both document authors and programmers the guidance they require to create and process METS documents conforming with a particular profile". The absence of an effective type-conformance mechanism forces (a) programmers to write custom code to implement digital object manipulation mechanisms in an ad-hoc and not reusable fashion and (b) cataloguing staff to carry out all the digital object typing arrangements "by hand", since DL system is not able to perform them automatically in a transparent manner.

Our work, presented in this paper, refers to the design and implementation of a sufficient digital object manipulation mechanism, that facilitates the conformance of digital objects to their type definitions, named *digital object prototypes*, in an automated manner. Section 2 provides a thorough discussion on digital object manipulation requirements of University of Athens Digital Library (UoA DL) in detail, also presenting UoA DL architecture. Section 3 introduces the notion of digital object prototypes, used as a means to express type-dependent customisations on the overall structure and behavior of digital objects. Section 4 demonstrates the benefits of utilising prototypes in the development of UoA DL modules, especially for the case of cataloguing and browsing interfaces and Section 5 clarifies the use of prototypes in the context of various collections, by setting up scopes of prototypes. Finally, discussion on related and future work resides in section 6.

2 The University of Athens Digital Library

2.1 Motivation

In [15] we presented a high-level overview of the University of Athens Digital Library (UoA DL) architecture. UoA DL will host several heterogeneous collections in terms of content type, structure, metadata and user requirements, containing both digitised and born digital material. Some of them are: the University's Historical Archive, Theatrical collection, Folklore collection [9], Byzantine Music Manuscripts collection, Medical collection and Ancient Manuscript collection. UoA DL System is implemented in Java and uses Fedora [18] as a digital object repository.

The Libraries Computer Center of the University is responsible for both UoA DL System development and the management of digitisation and cataloguing processes of the aforementioned collections, under a strict period of time. Under these conditions, we did not consider viable to develop custom functionality for each collection, in terms of metadata handling, user interfaces or any other modules. On the contrary, our design approach has been based on the concept of reusing configurable functionality, in order to cope with the various constraints and requirements imposed by each collection. Thus, we focused on the development of a general-purpose, parameterisable DL System that should be easily configured to accommodate to each collection's specific requirements, exhibiting code reuse.

Our primary focus has been given to collection management, cataloguing, browsing and user interface issues. Most collections consist of heterogeneous digitised material that require detailed cataloguing, given that free-text search facilities cannot be provided. Thus, it is of great significance to support detailed descriptions of the nature and structure of digital content in a configurable manner. Moreover, the majority of people participating in digitisation and cataloguing process will be active members of the Academic Community (such as scholars and postgraduate students) with expertise on the field of the specific collection. In order to increase productivity and facilitate cataloguing, while achieving configurability and code reuse, we focused on generating a unified configurable cataloguing interface, that should adapt to each collection's idiosyncrasies, while hiding from users internal representation, implementation and storage details. The system should "hide" underlying notions such as datastreams, XML documents, or even the use of specific metadata sets and only if the cataloguer requires a more technical cataloguing interface, the system should disclose such "internal" details.

2.2 Fedora Repository

In [15] we describe the reasons we have chosen Fedora for the development of UoA DL. Specifically, we use Fedora Repository for handling concerns related to storage, preservation and versioning, searching and indexing, along with metadata interoperability through the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [8].

Within Fedora framework, digital objects are comprised of datastreams, used to store all digital object related information. Fedora imposes no restrictions on datastream content; it may be XML metadata or arbitrary external content, such as locally stored files or remote URLs. Fedora Digital Object Model (FDOM) has been based on a METS variant in 1.x versions. In Fedora 2.0, released on late January 2005, a Fedora-specific representation of digital objects is introduced, named Fedora Object XML (FOXML) [5]. The concept behind METS *Behavior* section is implemented in FDOM in terms of *disseminators*. Disseminators associate datastreams to specific behaviors, through the use of special digital objects, namely *Behavior Definition Objects and Behavior Mechanism Objects*. Fedora Behaviors provide one or more methods that get associated to selected datastreams of a digital object and are automatically exposed in terms of Web Services [19], providing a standard-based, service-oriented mechanism for generating distributed and interoperable systems.

2.3 Digital Object Manipulation Requirements

In a higher level of abstraction, a digital object refers to a human generated artifact that encapsulates underlying digital content and related information [7, 13]. Although variations may exist on serialisation and encoding issues (METS [11], FOXML [5], RDF [16]), this information is used to describe, annotate, link and manipulate the object's internal content. Under this perspective, a digital object is conceived as an aggregation, consisting of four parts: its metadata sets, files, structure and behaviors, as presented in Figure 1.

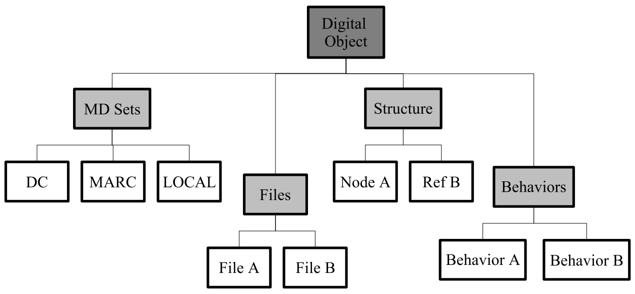


Fig. 1. An abstract representation of a Digital Object and its constitutional parts

In this context, an effective digital object manipulation mechanism should provide the following capabilities:

(a) *With regard to Metadata Sets:* (i) Allow the use of multiple Metadata sets to characterise digital objects, (ii) support mappings between fields of different Metadata Sets, in order to minimise redundancy and (iii) allow the localisation and customisation of each Metadata Set in order to cope with the special needs of different kinds of material, since, in practice, no single metadata standard can cover all possible needs, especially in the case of digitised content or digital culture content.

(b) *With regard to Files:* A digital object may contain zero or more files. File existence should not be mandatory, since a digital object could be used as a means to express the structure of "real world" objects.

(c) *With regard to Structural Information:* The representation of both structural and general-purpose linking information should be easily expressed.

(d) *With regard to Behaviors:* Facilitate DL modules and services to effectively manipulate and compose digital objects.

The detailed specification of each of these attributes depend on the digital object's nature; that is, the object's type. Consider, for example, the Theatrical collection consisting of albums that contain photos from theatrical performances of the National Theater. All Photo digital objects should behave in the same manner, *being themselves aware* about which parts comprise them and what each one represents. However, this fundamental "is-a" information is not properly expressed in neither METS nor FDOM. Fedora supplies digital objects with a *Content Model*, resembling the METS *Profile* metadata attribute. METS Profiles provide descriptions of "classes" of digital objects in order to be used by humans as a guide and not by programs as an actual type checking mechanism. In essence, digital objects are practically "typeless", since the knowledge of the types of objects is utilisable only by humans and not by the DL system. The absence of automatic type checking forces (a) programmers to write custom code to implement digital object type-conformance in an ad-hoc and not reusable fashion and (b) cataloguing staff to carry out all the digital object typing arrangements "by hand", since the system is not able to perform them automatically in a transparent manner.

According to the OO paradigm, each object corresponds to a type definition, named class. The same should stand for digital objects as well; digital objects should conform to a specification of their constitutional parts, wrapped in a separate entity. This entity, named *digital object prototype*, should contain all the corresponding specifications, in a manner independent of their realizations, such as class in the OO model provides the specification of its instantiations. Under this perspective, behaviors should not be assigned on digital objects directly; they should be assigned on the definition of their type. In the OO paradigm, functionality expressed in terms of methods is defined *once and in one place*, in the class definition. Objects, defined as class instances, are automatically supplied with this functionality as a result of their "is-a / instance-of" relation with their type definition; it is the class (that is, the type) that makes this functionality available to all its instances, it is not the user that provides it to each of them separately.

2.4 UoA DL System Architecture

From a software design perspective, the digital object type checking issues can be explained using the notion of separation of concerns [14]. There are three separate concerns that need to be resolved in order to provide an efficient digital object manipulation mechanism: (a) how the object is stored (storage and serialization concern), (b) what are the object parts and what each one represent (object specification and typing information), (c) how the object is internally represented, handled and composed according to its typing specification, in terms of coding facilities and APIs. Figure 2 depicts UoA DL architecture, where an intermediate layer is inserted between Fedora Repository and DL application logic, named *Digital Object Dictionary*, that copes with the typing and DL system internal representation concern of underlying Fedora digital objects.

3 Digital Object Prototypes and Instances

A *Digital Object Prototype* provides a detailed specification of a digital object's constitutional parts. The process of generating a digital object based on a prototype is called *instantiation* and the resulted object is called an *instance of the prototype*. The digital object instantiation process ensures that the acquired object instance will conform with the specifications residing in the prototype, making it automatically behave in the prototype-specified manner. Our implementation has focused on expressing "is-a / instance-of" relations between objects and respective prototypes, resembling the OO instantiation mechanism. Future work includes the ability to utilise OO inheritance in the context of digital object prototypes.

Figure 3 represents a Theatrical Collection Photo object as an instance of the corresponding Photo prototype. The latter is defined in terms of XML, depicted

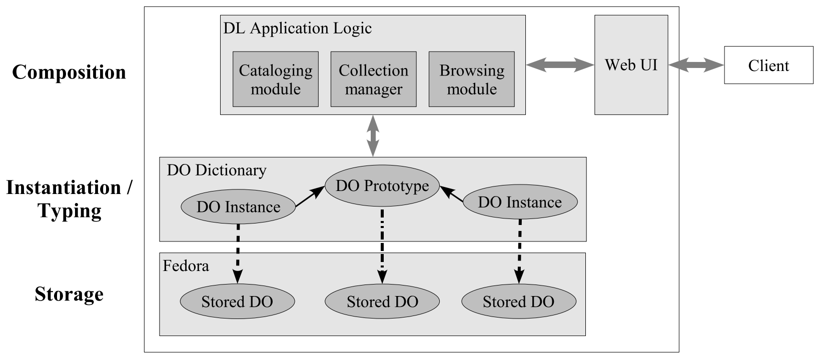


Fig. 2. Uoa DL architecture, using a Digital Object Dictionary to facilitate Compositions of Digital Objects

in Figure 4, providing the specifications of the constitutional parts of all the Photo digital object instances. When a Photo object instance is acquired by a DL module, the Dictionary utilises information residing in the prototype in order to load its constitutional parts from Fedora repository.

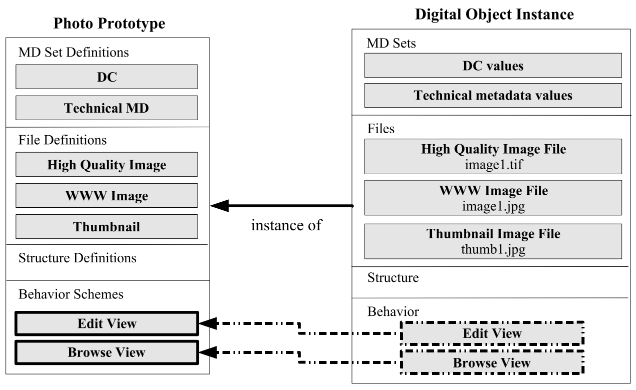
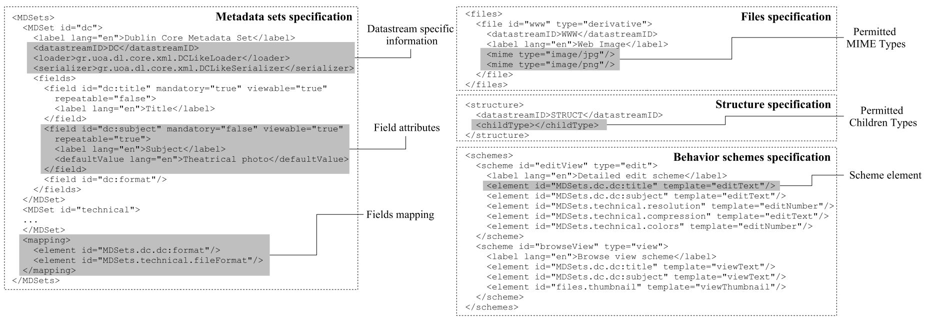


Fig. 3. A Photo Object as an Instance of the Photo Prototype

In particular, the Photo prototype specifies that:

(a) Photo object instances should contain two metadata sets, namely: (i) DC, used to hold the fields of the Dublin Core Metadata Element Set [3] for content description purposes and (ii) TechnicalMD, holding the fields of the NISO technical metadata for Digital Still Images [12] for holding digitisation information. There exist common DC and NISO fields (e.g. file format), that are mapped internally by the object instance, as specified in the Photo Prototype. Thus,



```
<MDSets>
  <MDSet id="dc">
    <label lang="en">Dublin Core Metadata Set</label>
    <datastreamID>DC</datastreamID>
    <loader>gr.uoa.dl.core.xml.DCLikeLoader</loader>
    <serializer>gr.uoa.dl.core.xml.DCLikeSerializer</serializer>
    <fields>
      <field id="dc:title" mandatory="true" viewable="true"
        repeatable="false">
        <label lang="en">Title</label>
      </field>
      <field id="dc:subject" mandatory="false" viewable="true"
        repeatable="true">
        <label lang="en">Subject</label>
        <defaultValue lang="en">Theatrical photo</defaultValue>
      </field>
      <field id="dc:format"/>
    </fields>
  </MDSet>
  <MDSet id="technical">
  ...
  </MDSet>
  <mapping>
    <element id="MDSets.dc.dc:format"/>
    <element id="MDSets.technical.fileFormat"/>
  </mapping>
</MDSets>
```

```
<files>
  <file id="www" type="derivative">
    <datastreamID>WWW</datastreamID>
    <label lang="en">Web Image</label>
    <mime type="image/jpg"/>
    <mime type="image/png"/>
  </file>
</files>
```

```
<structure>
  <datastreamID>STRUCT</datastreamID>
  <childType></childType>
</structure>
```

```
<schemes>
  <scheme id="editView" type="edit">
    <label lang="en">Detailed edit scheme</label>
    <element id="MDSets.dc.dc:title" template="editText"/>
    <element id="MDSets.dc.dc:subject" template="editText"/>
    <element id="MDSets.technical.resolution" template="editNumber"/>
    <element id="MDSets.technical.compression" template="editText"/>
    <element id="MDSets.technical.colors" template="editNumber"/>
  </scheme>
  <scheme id="browseView" type="view">
    <label lang="en">Browse view scheme</label>
    <element id="MDSets.dc.dc:title" template="viewText"/>
    <element id="MDSets.dc.dc:subject" template="viewText"/>
    <element id="files.thumbnail" template="viewThumbnail"/>
  </scheme>
</schemes>
```

Fig. 4. Photo Prototype XML Definition

even if the file format field is stored within the DC section, it is "aliased" as both a DC and NISO field by the Photo Instance. This way, field mappings are automatically handled by object's type, such as a class in the OO paradigm uses encapsulation and information hiding to enclose "internal / private" functionality.

(b) Photo instances should contain a high quality TIFF image, a WWW quality JPEG image and a thumbnail. Allowed MIME types and descriptive attributes are also provided.

(c) Photo object instances do not contain other types of objects; the Structure section of the Photo instance is empty, following the specification residing in the structure definition part of the Photo Prototype. On the other hand, Albums are used to host photos and the respective structure definition of the Album Prototype (not presented herein), specifies that Photo object instances can be "inserted" into Album instances. Structural information is utilised by user interface modules, in order to inform the users of the DL System about the allowed children types of each object instance.

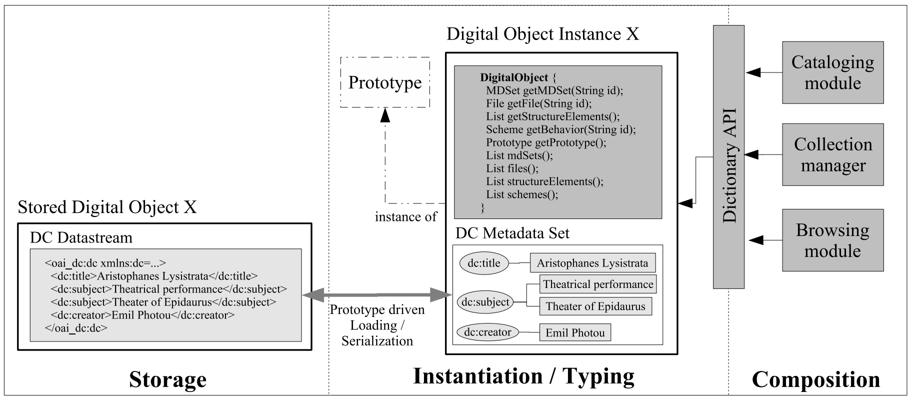
(d) When a Photo object instance is created, it automatically exposes the behavior schemes defined by the Photo prototype, namely *editView* and *browseView*.

A prototype may contain various behavior schemes for different purposes. However, two basic types of behavior schemes are currently supported: `edit` and `view`. The application logic indicates if the acquired digital object instance should be used for read-only or editing purposes. In the latter case, the object instance is acquired in editing mode and the user interface adjusts accordingly, by displaying the scheme elements through an editable web form. The *editView* behavior scheme depicted in Figure 4 determines the components of a low-detail cataloguing interface, comprised of the photo's title, subject, resolution, compression ratio and number of available colours. Respectively, the *browseView* scheme defines the metadata used for browsing photo objects, comprised of the photo's title, subject and thumbnail image. The distinction between an "editable" view or a "read-only" view is made internally by the object's prototype, that “knows” how to guide the low-level Web UI engine to display each element for editing or presentation purposes. The result is configurability and code reuse; the Dictionary API makes no distinction between end-user and cataloguer forms, allowing the development of modules in a uniform and reusable fashion. Cataloguer interfaces are generated in the same way with “common” content display end-user interfaces.

A behavior scheme represents a composition of the digital object constitutional parts. It resides in the prototype and it is dynamically bound to all the prototype instances. This dynamic binding resembles the notion of Fedora disseminator. However, behavior schemes are based on well-known concepts of the OO paradigm, with well understood semantics. In particular, a Behavior scheme can be conceived as a method of class operating on the latter' s “internal” fields. The behavior scheme is defined once in the digital object prototype and is made available to all its corresponding instances by the typing mechanism, resembling the OO model's dynamic method dispatching. Moreover, Fedora behaviors are parameterised on the datastream level of digital objects, while Prototype-based behavior schemes can be parameterised in a more fine-grained manner, supporting arbitrary compositions of digital object atomic elements, as contained in the metadata sets, files or structural / reference parts. Behavior schemes are able to identify the required atomic elements (e.g *MDSets.dc.dc:title, MDSets.dc.dc:subject, files.thumbnail* in the `browseView` behavior scheme) independent of their storage representation or datastream location. Finally, Fedora behaviors rely on the a-priori existence of the digital object and its constitutional datastreams; it is not possible to attach behavior on a digital object or its datastreams, if they do not yet exist. This means that they are not suitable for performing cataloguing effectively; all datastreams should be present and all behaviors should be defined and bound, in order to be able to utilise them. On the contrary, our approach gathers all object-related behaviors in the prototype and thus, we are able to treat a newly created prototype instantiation in a type-defined manner, before it has been ingested in Fedora Repository. Fedora Web Service behaviors will be of value after collections and related objects have been inserted into the system and DL application logic services have been developed.

4 The Benefits of Digital Object Type Conformance

The point put forward by this paper is that digital object composition and manipulation is performed more effectively if digital objects are supplied with a high-level type conformance mechanism, that resolves internal object details in a transparent manner. Fedora satisfies several of the requirements identified in section 2.3, such as the support of many metadata sets and files, along with the effective storage and indexing of structural and reference information, added in its new version. Based on its rich feature set, we have set up the digital object “type conformance” extension, that can be conceived as a realization of the *Content Model* notion, in programming and practical terms. Figure 5 depicts this functionality, performed by the intermediate Dictionary layer.



```
<oai_dc:dc xmlns:dc=...>
  <dc:title>Aristophanes Lysistrata</dc:title>
  <dc:subject>Theatrical performance</dc:subject>
  <dc:subject>Theater of Epidaurus</dc:subject>
  <dc:creator>Emil Photou</dc:creator>
</oai_dc:dc>
```

```
DigitalObject {
  MDSet getMDSet(String id);
  File getFile(String id);
  List getStructureElements();
  Scheme getBehavior(String id);
  Prototype getPrototype();
  List mdSets();
  List files();
  List structureElements();
  List schemes();
}
```

Fig. 5. A detailed overview of our proposed "type conformance" mechanism

The use of digital object prototypes allowed us:

– to gather all digital object manipulation functionality in a unified programming API, exposed in terms of the digital object dictionary, that abstracts underlying serialisation and storage details of digital content. This allowed us to centralise and reuse the code that handles datastream handling and XML parsing concerns, advancing the overall system's modularity and increasing the independence of DL System modules from low-level Fedora datastream serialisation details.

– to customise digital object instances "internally" and automatically, using prototype-driven digital object instantiation. The concept of OO encapsulation is based on entities carrying out "private" functionality in a type-specific manner (e.g. metadata field mappings, validation of containment relationships and allowed files), while exposing unified external manipulation interfaces. The notion of prototype-based *behavior schemes* aims at accomplishing this functionality – each behavior scheme is executed differently by each object, albeit used in a unified manner. For instance, Album instances are also supplied with *editView* and *browseView* schemes that are executed by the Album prototype in a different manner with regard to the corresponding schemes of the Photo prototype.

– to enable type-defined introspection of digital object structure and behavior. A UoA DL module is capable of querying a digital object instance for its constitutional parts, made available to the instance through its corresponding prototype. This way, a module can supply the user with the list of supported behaviors and let him or her select the desired behavior scheme to execute. In software engineering, this concept is named *reflection*. Work presented in [4] enables introspection of underlying object structure and behaviors. A prototype can be conceived as an introspection guide for its digital object instances and we argue that prototype-driven introspection is richer in semantics terms.

5 Digital Collections and the Scope of Prototypes

A collection refers to a set of items with similar characteristics, such as type, owner, subject area and like. A digital collection aims at providing this grouping in the context of digital content. In the case of the University of Athens, most digital collections contain digitised material representing real world complex objects, as the Photo Albums of Theatrical collection. Furthermore, various metadata sets and/or local extensions or combinations of them are used to characterise their content. Thus, the elements of a digital collection vary significantly in terms of digital content structure, the metadata sets used to describe it and the user requirements for cataloguing and browsing. For these reasons, digital object prototypes are defined with a *collection-pertinent scope*, affecting collection-specific digital object instances. This allows us (a) to support fine-grained definitions of collection specific kinds of material and (b) to avoid type collision problems – Theatrical Collection Photo prototype may coexist with the Medical Collection Photo in the UoA DL. Each prototype is supplied with composite identifiers, such as `dl.theatrical.photo`, that differs from `dl.medical.photo`. The composite identifier is attached on respective prototype instances, through the Fedora *Content Model* metadata attribute.

For uniformity reasons, all information is stored in Fedora Repository, in terms of digital objects. This stands for collections, too. In order to be able to represent, manage and manipulate all digital content in a unified manner, collections are treated as digital object instances conforming to the *collection prototype*. By representing everything in terms of digital object instances, flexible and effective collection management capabilities can be generated. Collections, sub-collections and the Digital Library itself are represented as a hierarchy of digital object instances, as depicted in Figure 6. This representation scheme provided the following benefits:

– New collections can be easily added in UoA DL, through the creation of a new instance of the collection prototype. This has been of significant importance, providing us the ability to work out the details of each collection independently, but yet in a unified fashion.

– Support collection / sub-collection hierarchies in a configurable manner. A sub-collection is simply a collection instance added in another collection instance.

– Support the characterisation of collections and sub-collections, supplying them with metadata sets, as any other common digital object instance.

– Easily identifiable prototype definitions. The Content Model fully qualified identifier is used to supply the Dictionary with the path of the DL hierarchy that leads to the specific collection and the definition of the specific prototype in that collection.

6 Discussion

UoA DL System is currently being used for the cataloguing process of the UoA Historical Archive collection. Several extensions are under consideration, such

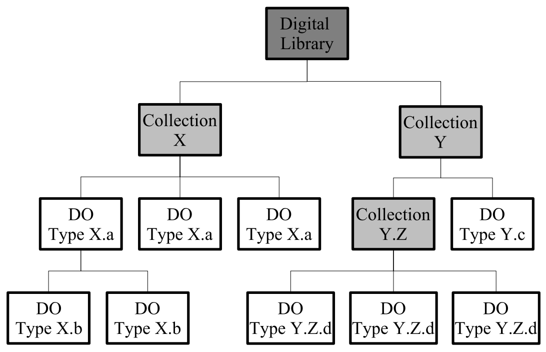


Fig. 6. A Digital Library as Hierarchy of Digital Object Instances with collection-scoped Prototypes

as the generation of a prototype construction interface, that could assist on the creation of digital object prototypes, since, currently, the XML prototype specification need to be issued by direct XML editing. Moreover, the structure part of a digital object should be supplied with general purpose linking capabilities, in order to be able to cope with relations that extend structural containment. We are taking under consideration the use of the Fedora Metadata for Object-to-Object Relationships, introduced in Fedora 2.0, for this purpose. Finally, our current effort is focused on generalising the current definition of behavior schemes in order to support prototype-driven: (a) automatic conversions of *primitive* file types to their prototype-defined *derivatives*, in order to facilitate cataloguing staff (e.g convert a TIFF image to a low quality JPEG image and a thumbnail) and (b) cataloguing form validation (e.g. metadata fields validation).

The greatest challenge, however, relies in extending Digital Object Prototypes in order to supply them with Object Oriented inheritance capabilities. We argue that it is of great importance to bridge the gap between digital objects and OO objects even further. Approaches on formalisation of inheritance semantics [2, 1, 17] treat an object as a record of fields, where each field contains a method. Our representation treats objects as aggregations of their four constitutional parts. In a high level of abstraction, these two representations present significant similarities, indicating that it should be possible to incorporate inheritance in the context of digital object prototypes, albeit supplied with the appropriate semantics. Our second long-term goal is to support the concept of OO polymorphism, having digital object instances participate in more that one "is-a" relations. Definition reuse through inheritance has been discussed in [6], although targeted on information retrieval enhancements. Our aim is to use prototype inheritance for enhancing the reuse and configuration capabilities of the Dictionary digital object manipulation mechanism.

References

1. L. Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types*, pages 51–68, 1984.
2. W. Cook and J. Palsberg. A denotational semantics of inheritance and its correctness. In *Proceedings of the ACM Conference on Object-Oriented Programming: Systems, Languages and Application (OOPSLA)*, pages 433–444, 1989.
3. *DCMI Metadata Terms.* Dublin Core Metadata Initiative, January 2005. Available at http://www.dublincore.org/documents/dcmi-terms/.
4. N. Dushay. Localizing experience of digital content via structural metadata. In *Proceedings of the Joint Conference on Digital Libraries (JCDL '02)*, pages 244–252, 2002.
5. *Introduction to Fedora Object XML.* Fedora Project. Available at http://www.fedora.info/download/2.0/userdocs/digitalobjects/introFOXML.html.
6. N. Fuhr. Object-oriented and database concepts for the design of networked information retrieval systems. In *Proceedings of the 5th international conference on Information and knowledge management*, pages 164–172, 1996.
7. R. Kahn and R. Wilensky. *A Framework for Distributed Digital Object Services.* Corporation of National Research Initiative - Reston USA, 1995. Available at http://www.cnri.reston.va.us/k-w.html.
8. C. Lagoze and H. V. de Sompel. The open archives initiative: Building a low-barrier interoperability framework. In *Proceedings of the Joint Conference on Digital Libraries (JCDL '01)*, 2001.
9. I. Lourdi and C. Papatheodorou. A metadata application profile for collection-level description of digital folklore resources. In *Proceedings of the Third International workshop on Presenting and Exploring Heritage on the Web (PEH'04), 15th International Conference and Workshop on Database and Expert Systems Applications DEXA 2004*, pages 90–94, August 2004.
10. *METS Profile Documentation.* Library of Congress.
11. *METS: An Overview & Tutorial.* Library of Congress, September 2004. Available at http://www.loc.gov/standards/mets/METSOverview.v2.html.
12. *Data Dictionary - Technical Metadata for Digital Still Images.* NISO Standards Committee, June 2002.
13. *Reference Model for an Open Archival Information System (OAIS).* Consultative Committee for Space Data Systems (CCSDS), 2002. Blue Book, Issue 1.
14. D. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.
15. G. Pyrounakis, K. Saidis, M. Nikolaidou, and I. Lourdi. Designing an integrated digital library framework to support multiple heterogeneous collections. In *Proceedings of the 8th European Conference on Digital Libraries (ECDL 2004)*, pages 26–37, 2004.
16. *Resource Description Framework (RDF).* World Wide Web Consortium. Available at http://http://www.w3.org/RDF/.
17. U. Reddy. Objects as closures: Abstract semantics of object-oriented languages. In *Proceedings of the ACM Conference on Lisp and Functional Programming*, pages 289–297, 1988.
18. T. Staples, R. Wayland, and S. Payette. The fedora project: An open-source digital object repository management system. *D-Lib Magazine*, 9(4), April 2003.
19. *Web Services Activity.* World Wide Web Consortium. Available at http://www.www.org/2002/ws/.