On Foundations of Typed Data Models for Digital Libraries

Leonardo Candela, Donatella Castelli, Paolo Manghi,
Marko Mikulicic, and Pasquale Pagano

Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo"
Consiglio Nazionale delle Ricerche
Via G. Moruzzi, 1 – 56124, Pisa – Italy
{candela, castelli, manghi, mikulicic, pagano}@isti.cnr.it

Abstract. This work discusses the foundational aspects of a compound object data model for Digital Libraries Management Systems whose novelty is to re-introduce the notion of types as a mean of supporting safe, optimized and efficient implementations of Digital Library Systems.

1 Introduction

According to the DELOS Digital Library Reference Model [2] a Digital Library (DL) is an organization of human and technological resources necessary to serve the needs of user communities willing to share, preserve, publish, search, ingest digital objects. The users of a DL, ranging from librarians and administrators to end-users such as researchers and students, interact through a Digital Library System (DLS). A DLS is a software system designed, developed and deployed to implement DLS *applications* required to manage the digital objects conforming to the DLS *document model*, i.e. the required digital document structure. The realization of DLSs is assigned to developers who rely on Digital Library Management Systems (DLMS), software tools specially devised to assist and ease the implementation of DLS document models and relative applications. DLMSs vary from those delivering flexible tool-kits for customized DLS construction to those offering pre-packaged and easy-to-install DLSs. In particular, the Reference Model classifies DLMS platforms in terms of quality of service, architecture, users, policies, content, and functionality parameters. In this paper, we focus on content and functionality of DLMSs, with attention on the *compound object data models* supported by their underlying storage system. Compound objects are generally intended as "graphs" of digital objects associated with each other through relationships whose "label", i.e. name, expresses the nature of their association. In this respect, DLMSs differ by the *modeling primitives* they offer, that is the way they characterize the structure of the digital objects and the relationships into play, and thus by the kind of DLS document models they are able to represent.

Data models can range from *rigid models*, where the model basically expresses "one" DLS document model allowing for light customizations (e.g. DSpace [7], Greenstone [8], Eprints [6] data models), to *flexible models*, where the model can potentially describe "any" DLS document model (e.g. Fedora data model [5]). "Rigid" and "flexible" DLMSs tend to offer different kind of modeling primitives, for defining DLS document models and for managing or querying their digital objects. DLMSs supporting rigid models focus on efficient storage of pre-defined DLS document models that developers may only configure in some aspects; e.g. label vocabularies. In such DLMSs, modeling primitives are provided directly as ready-to-install DLS applications, with pre-defined set of ingestion, access and search functionalities, whose graphical appearance can be customized to the user community needs. DLMSs implementing flexible models expose to developers the modeling primitives they need to create, store and search digital objects according to the structure of the DLS document model desired by the user community. Developers then realize the relative DLS applications as a separate piece of software, "on top" of the DLMS.

An organization willing to set up a DL requires a DLS matching the needs of its user communities. As a consequence, the organization has to cope with the trade-off between the inherent cost of realizing and sustaining the "DLS of their dreams" and the relative realization cost. In this respect, DLSs obtained from rigid DLMSs can be installed with simple configurations steps; realization costs are very low to the organizations, which, for some DLMS platforms, may also count on free technical support and frequent code updates and patches. On the other hand, if the user community demands for document models distant from those supported by existing rigid DLMS, organizations will likely rely on flexible DLMSs. In this case, costs are higher: developers, from local labs or from external companies, should be hired for customizing the DLMS to support the required DLS document model and then complete the DLS with the implementation of the missing applications.

In this paper we focus on construction of flexible DLMSs. The main motivations behind their realization can be found in the common requirements of developers facing the design of DLSs that match special document model needs. Typically, such developers require to manage digital objects according to similar structural and functional patterns, independently from the peculiarities of the DLSs they target. Flexible DLMSs were conceived to supply developers with general-purpose functionalities for compound object management they should otherwise re-implement at their own cost.

The Fedora Repository [5] is the only DLMS known to offer a flexible compound object data model, which describes compound objects as graphs of digital objects connected by relationships. In particular, the DLMS offers primitives for storing objects, each consisting of one mandatory Dublin Core record and a number of payloads. Objects are versioned and can be connected through relationships labeled with values from a controlled and extendible vocabulary; e.g. *isPartOf*, *isMetadataFor*, etc. The DLMS offers query languages to retrieve objects matching given Dublin Core record field values or objects matching a navigational query (RDF query) in the graph of relationships. Designers can thus develop their DLS applications on top of a Fedora DLMS, which is used for storing and querying the graph of objects corresponding to the DLS document model. In this respect, Fedora DLMSs store graphs of objects and are "unaware" of the DLS document model notion, whose nature is embedded in the DLS applications business logic, i.e. in the peculiar way applications store objects and relationships into the DLMSs.

In this paper we propose a new flexible data model for compound objects. The novelty of the model is that of being typed in the traditional database sense: the existence of an object must be preceded by the definition of its structure, called type, i.e. the description in a formal language of the object structure. DLMSs implementing the model will support construction of DLSs starting from the type definition of the DLS document model within the DLMS – as well as DBMSs require the definition of database tables prior the realization of database applications managing the records therein. In this case, the DLMS is "aware" of the DLS document model, which is declared as a type by the developers. The DLMS can thus take advantage of the type information to support safe, optimized and efficient management of the DLS document model at hand.

To summarize, the data model proposed in this paper aims at:

(*i*) Including the data abstractions necessary for describing any digital library document model in terms of compound objects;
(*ii*) Inspiring the design of DLMSs supporting safe, efficient and optimized storage, access and search of compound objects.

In the following we motivate the foundations of a typed compound object data model by means of real case scenario requirements, then conclude by illustrating our model proposal.

2 Model Requirements

As explained in the previous section, flexible DLMSs support compound object data models devised to meet the requirements of developers willing to realize DLSs. An important feature of such models is that of being both "fully expressive" and "minimal", that is (*i*) the set of primitives they provide should be capable of describing an document model and (*ii*) removing one of such primitives would compromise the expressivity of the data model language, i.e. leave a subset of DLS document models out of our solution domain. To identify such sets of primitives, a study of common DL behavioral patterns is necessary. Consider the following DLS real-case scenarios.

Real-case 1 (*Catalogues*). DLSs for management of metadata record catalogues, for example in standard library administration. In this case the records are describing entities, i.e. publications, whose digital payload is not stored within the DLS. The metadata records may obey to a standard bibliographic metadata format, such as Dublin Core or MARC, or to a proprietary format of preference to the DLS user community. The DLS offers efficient search functionality over the metadata records, based on the given format.

Real-case 2 (*Archives*). DLSs for management of multi-media digital objects coming with their metadata description. In this case, the digital objects are stored in the DLS back-end, can be searched through their metadata, and eventually accessed by proper protocols; e.g. streaming for video digital objects. In principle, the same digital object may be described by several metadata records, conforming to metadata formats specific to relative application scenarios. The DLS offers efficient format-based search functionality over the metadata records. Note that the same scenario is reflected by Institutional Repositories, with publications and bibliographic metadata.

Real-case 3 (*Enhanced publication management*). A special DLS for management of enhanced publication objects, intended as graphs of digital objects consisting of one publication object, with one metadata record description and with reference relations to other publication objects. The DLS is capable of (i) ingesting or importing publication objects, i.e. metadata records and/or payloads, from other domains in the form of "simple" compound objects (digital objects with no relationships) and (ii) allowing the user community to construct enhanced publications by specifying reference relationships over such pool of simple compound objects. It is important to observe that: the same simple objects can be part of several enhanced publications; relationships are specified in a second stage and are kept apart from the objects.

Real-case 4 (*DLS Extensions*). In a later stage, when the DLS in Real-case 3 (RC3) has been used for some time, a user community requires a DLS capable of sharing/reusing the publication objects portion of RC3 document model so as to include its content as part of a further and separate DLS document model. This user community is interested in growing experiment-oriented enhanced publications, consisting of publication objects in used relationship with special data source objects, that is metadata records describing an external experimental data source; e.g. a database, a Web Site, data files. The new DLS is capable of (i) ingesting or importing publication objects in the same collection of publications defined by RC3 and data source objects from other domains in the form of simple compound objects and (ii) allowing the user community to construct new enhanced publications by specifying the appropriate used relationships and connecting publication objects with data source objects.

DLMSs capable of supporting all the above DLS real-case scenarios should respect the following requirements:

Requirement 1 (*Metadata records*) The DLMS should support management of metadata records of arbitrary metadata formats and also deal with them in "isolation", since these may not necessarily come with the digital object they describe. DLS developers should be able to configure the DLMS to provide efficient storage and search of objects based on the formats required by the DLS model at hand.

Requirement 2 (*Digital objects*, i.e. *payloads* or *files*) Since payloads may obey to different media types the DLMS should provide different ways of efficiently storing and accessing such objects depending on their type. To this aim, the DLMS should be configured prior objects ingestion by DLS developers, who should specify the kind of digital objects required by the DLS document model.

Requirement 3 (*Object relationships*) The DLMS should enable the creation of relationships after the digital objects were, thus regard relationships as independent DLMS entities, conceptually parted from the objects they connect. Relationships should be "labeled", i.e. suggest and include their semantic nature, and such labels should be customizable, i.e. defined by the DLS developers. Finally, a given labeled relationship is supposed to link two objects of a given typology, not any two objects in the system.

Requirement 4 (*Collections*) The DLMS should be able to support management of collections of objects, i.e. groups of objects, so as to satisfy some aggregative logic specified by the DLS requirements. DLS applications should be able to ingest, search and access digital objects into a given collection.

Requirement 5 (*Data integrity*) The DLMS should be able to support management of document models of several DLSs at the same time. Furthermore, when such document models share part of the objects, the DLMS should prevent DLS applications from interfering with each other and compromise the consistency, i.e. integrity, of the document model structure.

The Fedora DLMS matches only parts of such requirements. For example, Fedoras data model does not include the notion of arbitrary metadata record and metadata format management. It is instead assumed that all objects have a mandatory Dublin Core record associated with them and efficient search and access is available only for that format. Fedora digital objects, i.e. payloads, cannot be stored or accessed depending on their media type, since the model does not include the notion of "type of objects" nor a notion of "object collection". For the same reasons, data sharing and integrity cannot be supported. Different DLSs operating on top of the same Fedora DLMS find themselves managing a common graph of objects, whose consistency depends on how careful and aligned have been the developers in implementing the respective DLS applications. In recent implementations of Fedora, the notion of *Content Model* has been introduced to overcome some of these rather dangerous issues. Content models are represented and stored as special Fedora objects, whose payload bears an XML description of the structure of other objects, i.e. the payload types they are supposed to contain. In this respect part of the benefits of type information are recovered, but other are still unsolved. For example, some form of automatic application correctness checking is possible, but type-dependent optmized and efficent data storage is still not achievable.

3 Typed Compound Object Data Model

In this section we present the foundation of a typed compound object model that meets all requirements presented in the previous Section. We can summarize such requirements in three main conclusions:

– Object kinds: metadata records, digital objects and relationships are three independent entities, i.e. object kinds, in the data model. Developers should be able to combine them most appropriately to match the DLS document model at hand.
– Object collections: in order to organize the variety of objects present in a DLS document model, the notion of group of objects, i.e. collection, is crucial. Objects of the same kind and structure, e.g. Dublin Core metadata records, may not necessarily belong to the same intuitive pool, but be separated in conceptually different collections; e.g. "my Dublin Core records", "your Dublin Core records".
– Object structure: in order to enable optimized and efficient storage and access of objects, their structure/typology should be specified prior their ingestion to the DLMS. The DLMS should be "aware" of the parts composing the DLS document model at hand. In particular, metadata records require their format to be specified, digital objects their media type and relationships their label and the type of objects they are supposed to connect.

The typed data model proposed here adheres to these conclusions. To this aim the model supports the notions of *Type*, *Set* and *Object*. Types define the *abstract* structure and the operators of the Object entities in our data model. Sets are instead the instantiation of Types, i.e. the *concrete* structure, that is containers of the Objects they describe. More specifically, the relation between the three entities is:

– Types can be *instantiated* to create new Sets conformant to the type properties; A Set is therefore the result of the instantiation of one Type and also acts as "generator" of Objects of that Set.
– Sets have unique names and are referred to add, delete or update the Objects therein; in that sense, each Set defines a new unique *concrete type* for all Objects it will contain, whose structure and operators will be that of the Sets Type; Sets are therefore (possibly empty) containers of "structurally homogenous" Objects.
– Objects have unique identifiers and they conform/belong to the concrete type, i.e. the Set, through which they were created.

Figure 1 depicts how a Type *T* can be instantiated in a number of Sets, here with unique names *A* and *B*. Such Sets will have type *T (A::T)* and will be able to generate and contain Objects *o* with concrete types *A* and *B* respectively (*o:A* and *o:B*). Objects in *A* and *B* share the same structure and behavior, but have different concrete types.

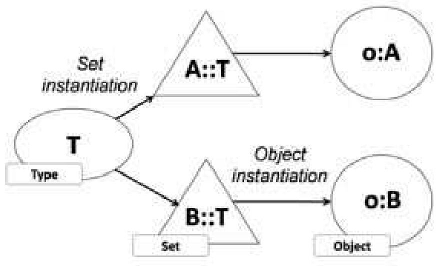

Fig. 1. Relationships between Types, Sets and Objects

Sets can be of three main Types:

– *Atom Type*: Atoms are intended as "simple digital objects", i.e. files. A Set of atoms can be specialized to contain Objects of a specific format/media type;
– *Description Type*: Descriptions are human/machine readable descriptions of digital or physical entities in the real world, thus they match the notion of metadata records as well as Description Types correspond to the metadata formats. Description Types are defined in terms of "properties", i.e. tuple types as sequences of ¡attribute-value domain¿ pairs, or more complex structures, such as trees of records;
– *Relation Type*: Relations represent binary relationships between Objects of two given Sets, i.e. they consist of two pointers at two existing objects in the given target sets. As all other objects, relation objects have an identity (they can be themselves target of Relations); unlike other objects, relation objects cannot exist without the

relative target Objects to exist. Relation Types depend on two Sets, relative to the objects that can be possibly associated by the Relation objects in the Relation Set of the given Type. Note that the name of the Set is the “label” to be associated to all Relation objects therein.

The Object Model attempts to capture the essence of modern Digital Libraries, whose content may be the result of the combination of new content with existing content, in turn possibly heterogeneous and not locally available. In this scenario, Atom objects might be available from different sources and might come or not come with a human/machine readable “description”; e.g. metadata description. Equally, Description objects might exist in the system without for the objects they describe to be present; e.g. metadata catalogs. Finally, given a DLS populated with objects, relations between such objects are and should be defined independently by DLS communities, based on their needs of connecting the objects. This is why relation objects are independent from the objects they connect, i.e. they are always added in a “second stage” and can be removed with no implicit impact on the objects they were binding together.

Figure 2 depicts the relation between Set Types, where the triangles represent Sets and the other shapes the different kinds of Objects they may contain. The picture illustrates how Relation, Atom and Description Sets are all specializations (can be casted to) of Object Sets; Relation Sets depend on two other Sets; and relation Objects relate two Objects in such two Sets. In particular, for any object *o* that is “linked” to a description object *d* through one relation object *r*, we say that *o is described by d*, or that *d is a description of o*.

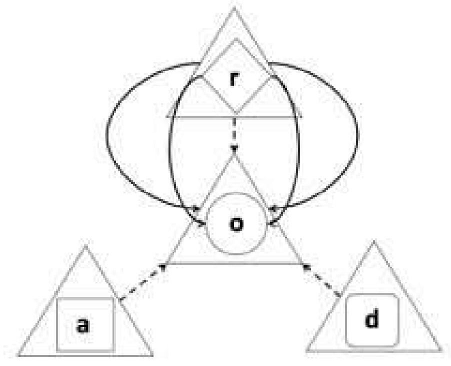

Fig. 2. Object types interrelation

For example, consider the archive scenario depicted in RC2, where the DLS handles a document model where metadata records of format X are accompanied by the digital objects of media Y they describe. Such document model is described in our data model with three Sets: one Description Set named *myMetadata* of the Description Type that matches X, one Atom Set named *myPayloads* of the Atom Type Y, and one Relation

Set named *describedBy*, whose objects can combine one object in *myMetadata* to one object in *myPayloads*. If the DLS requires the Atoms in *myPayloads* to be described by metadata records of different formats, the DLS document model should be extended by creating the relative Description Sets, together with the Relation Sets.

Query Language. The data model supports a query language whose queries start from a Set of objects and return the Set of objects that satisfy a given predicate. Queries can be navigational and/or description based. For example, it is possible to search all Objects in *myPayloads* that are connected via objects in *describedBy* to an object in *myMetadata* that respects a given CQL query over fields in X.

4 Conclusions

In this work we discussed the motivations behind the realization of DLMSs and illustrated the requirements of DLMS data models through DL real-case scenarios. We have shown how existing DLMS models fail at satisfying part of such requirements and then presented the foundations of a typed compound object model that manage to fulfill them. In the future, we plan to provide a formal definition for such data model and relative query language, and undertake the development of a DLMS that supports it.

Acknowledgments This work is partially supported by the INFRA-2007-1.2.1 Research Infrastructures Program of the European Commission as part of the DRIVER-II project (Grant Agreement no. 212147).

References

1. Dublin Core Metadata Initiative. `http://dublincore.org`.
2. L. Candela, D. Castelli, N. Ferro, Y. Ioannidis, G. Koutrika, C. Meghini, P. Pagano, S. Ross, D. Soergel, M. Agosti, M. Dobreva, V. Katifori, and H. Schuldt. *The DELOS Digital Library Reference Model - Foundations for Digital Libraries*. DELOS: a Network of Excellence on Digital Libraries, February 2008. ISSN 1818-8044 ISBN 2-912335-37-X.
3. L. Candela, D. Castelli, P. Manghi, and P. Pagano. Item-Oriented Aggregator Services. In *Third Italian Research Conference on Digital Library Systems*, Padova, Italy, January 2007.
4. L. Candela, P. Manghi, and P. Pagano. An Architecture for Type-based Repository Systems. In *Foundations of Digital Libraries II, Pre-proceedings of the Second Workshop on Foundations of Digital Libraries, Budapest, Hungary, September*, 2007.
5. C. Lagoze, S. Payette, E. Shin, and C. Wilper. Fedora: An Architecture for Complex Objects and their Relationships. *Journal of Digital Libraries, Special Issue on Complex Objects*, 2005.
6. P. Millington and W. J. Nixon. EPrints 3 Pre-Launch Briefing. *Ariadne*, 50, 2007.
7. R. Tansley, M. Bass, D. Stuve, M. Branschofsky, D. Chudnov, G. McClellan, and M. Smith. The DSpace Institutional Digital Repository System: current functionality. In *Proceedings of the third ACM/IEEE-CS joint conference on Digital libraries*, pages 87–97. IEEE Computer Society, 2003.
8. I. H. Witten, D. Bainbridge, and S. J. Boddie. Power to the People: End-user Building of Digital Library Collections. In *Proceedings of the first ACM/IEEE-CS joint conference on Digital libraries*, pages 94–103. ACM Press, 2001.