# 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.** Digital Library Systems (DLS) are software applications implementing the functionalities to operate over the (possibly compound) objects of a Digital Library. In the past, the development of DLSs has been mainly characterised by a *from-scratch* approach. Only in the recent period Digital Library System developers started adopting *Digital Library Management Systems* (DLMS), special software systems easing the development process by supporting facilities common to DLSs. The Digital Library community has not yet reached a formal agreement on the detailed functionality these systems must implement in terms of content management support. This work focuses on the foundational aspects of content management for DLMSs by discussing a data model whose novelty is that of (*i*) identifying modeling primitives capable of expressing the nature of the compound objects of any DLS and (*ii*) re-introducing the notion of *object type* as a mean of supporting safe, optimized and efficient DLS implementations. The model will inspire the realization of the first DLMS supporting the expressiveness of compound objects and the traditional capabilities of static typing.

## 1 Introduction

According to the DELOS Digital Library Reference Model [1] a *Digital Library* (DL) is "*an organisation, which might be virtual, that comprehensively collects, manages and preserves for the long term rich digital content, and offers to its user communities specialised functionality on that content, of measurable quality and according to codified policies*".

The operation of any DL is guaranteed by an operational software system, the *Digital Library System* (DLS). A DLS is a software system developed and deployed to implement every aspect of the DL, i.e. to manage the content the DL is devised for, to take care of the users the DL is conceived to serve, to realise the functionality the DL has been designed for, to put in place the policies governing every aspect of the DL, to guarantee the DL expected quality of service.

The development of such kind of systems as well as their deployment and operation has been performed *from scratch* for many years, i.e. different approaches and technologies have been developed to satisfy the same needs arising in the various application scenarios. In the recent years, the notion of *Digital Library Management Systems*

(DLMS) [2,1], i.e. software system specially devised to assist and ease the implementation of DLSs, has been introduced in the DLSs development arena. DLMSs range from software systems implementing specific aspects of the DLs (e.g. content management functionality in the case of Repository systems like DSpace [3] and Fedora [4]) that once enriched with a set of *extensions* lead to the software needed to implement the expected DLS, to Extensible Digital Library Systems (e.g. DelosDLMS [5]), i.e. a component-oriented software systems implementing complete DLSs that thanks to the openness of the architecture can be easily enriched with additional capabilities, and Digital Library Systems Generators (e.g. gCube [6] and MARIAN [7]), i.e. open systems promoting a development model based on (*i*) a declarative definition and configuration of the expected system and (*ii*) an autonomic deployment and management of the needed constituents.

Beyond the internal architecture and the DLS construction facilities they support, DLMSs are characterized by the data models they implement, also called *compound object data models*. In this paper, we focus on such data models, discussing the modeling primitives they should minimally comprise to describe the *information object model* of any DL. According to the Reference Model, Information objects conceptually represent DL content in terms of a "graph" 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 characterise DLS digital objects, and thus by the kind of information objects they are able to represent.

Information object models can range from *rigid data models*, where the model basically expresses "one" information object model allowing for light customizations (e.g. DSpace [3], Greenstone [8], Eprints [9] data models), to *flexible models*, where the model can potentially describe "any" information object model (e.g. Fedora data model [4]). DLMSs equipped with "rigid" and "flexible" models tend to offer different kind of modeling primitives, for defining personalised information object models and for managing or querying their digital objects. DLMSs supporting rigid models focus on efficient storage of pre-defined information object 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 components, 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 information object model desired by the user community. Because of this genericity of the data model, the rest of facilities needed to produce the specific DLS have to be implemented by developers as extensions of the selected DLMS.

An organization willing to set up a DL requires a DLS matching the needs of its user communities, typically expressed in terms of an information object model and the relative functionalities. As a consequence, the organization has to cope with the trade-off between the benefits 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 information object 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 information object model and then complete the DLS with the implementation of the missing features.

In this paper we put the bias on 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 information object model needs. Typically, such developers end up managing digital objects matching similar structural and functional patterns, independently from the peculiarities of the DLSs they target; e.g. they deal with files, metadata records, collections. 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 [4] 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. DLS designers can thus develop their extensions on top of a Fedora DLMS, which is used for storing and querying the graph of objects corresponding to the DL information object model.

In this respect, Fedora's data model exploits the flexibility of labeled graphs to represent any instance of DL information object models, hence the development of any DLS. However, such graphs of objects are "unaware" of the information object model notion supported at the DL level, whose structure is embedded in the DLS business logic, i.e. in the peculiar way software extensions (e.g. user interfaces) ingest objects and relationships into the DLMS. This lack of information about the objects structure in the DLMS leads to well known software development drawbacks and does not allow the enforcement of storage optimization and efficient access techniques.

In this paper we propose a new flexible data model for compound information objects. The novelty of the model is that of being *statically 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. As well as DBMSs require the definition of database tables prior the realization of database applications managing the records therein, in such a scenario, developers will first define in the DLMS the types matching the structure of the DL information object model and then construct the DLS using the customized DLMS instance. In this case, the DLMS is "aware" of the DLS information object model structure, which is declared as a type by the developers. The system can thus take advantage of the type information to support safe, optimized and efficient management of the DLS information object 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 information object 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. A desirable 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 any information object model and (*ii*) removing one of such primitives would compromise the expressivity of the data model language, i.e. leave a subset of DLS information object 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 zero or one Dublin Core metadata record description and with relations to other publication objects, cited by the publication. 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 therefore kept apart from the objects.

*Real-case 4* (*Federated DLS*). 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 information object model so as to include its content as part of a further and separate DLS information object 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 allows the user community to (*i*) ingest publication objects together with metadata records in the same collections defined by RC3 (*ii*) ingest data source descriptions and (*iii*) building experiment-oriented enhanced publications by connecting publication objects with data source objects.

DLMSs capable of supporting all the above DLS real-case scenarios should offer modeling primitivessatisfying 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. 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 to objects ingestion by developers, who should specify the kind of digital objects (e.g. mime type) required by the DLS information object 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 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 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 information object models of several DLSs at the same time. Furthermore, when such information object models share part of the objects (e.g. RC4: the publication and metadata records collections), the DLMS should prevent DLSs from interfering with each other and compromise the consistency, i.e. integrity, of the information object model structure (e.g. RC4: the second DLS may, by mistake, ingest records of a different metadata format and compromise RC3).

**Requirement 6 (*Structure*)** From all requirements above, it appears that the DLMS should be aware of the structure of the metadata records, digital objects and relationships, so as to optimize their storage, ensure their efficient access and their safe manipulation (e.g. sharing between different DLSs) and safe grouping into collections.

Fedora's DLMS matches only parts of such requirements. For example, Fedora's 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 lack-of-structure 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 drawbacks. 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 others are still unsolved. For example, some form of automatic application correctness checking is possible, but type-dependent optimized and efficient 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 information object model at hand.
- *Object collections*: in order to organize the variety of objects present in a DLS information object 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 information object 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 *Object*, *Set* and *Type*. Objects belong to (and are created by) Sets, which are instantiations of Types. Accordingly, we say that Types define the *abstract* structure of objects, while Sets the concrete structure of the objects they contain. More specifically, the relation between the three entities is:

– Types have unique names and can be *instantiated* to create new Sets, whose Objects will conform to the type properties; a Set also acts as "generator" of Objects of that Set.
– Sets have unique names and are used 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 Set's Type; Sets are therefore (possibly empty) containers of "structurally homogeneous" 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 *Cat* can be instantiated in a number of Sets, here with unique names *myCats* and *yourCats*. Such Sets will have type *Cat (myCat::Cat)* and will be able to generate and contain Objects *myCat* and *yourCat* with concrete types *myCats* and *yourCats* respectively (*myCat:myCats* and *yourCat:yourCats*). Objects in *myCats* and *yourCats* share the same structure and behavior of the type *Cat*, but have different concrete types.

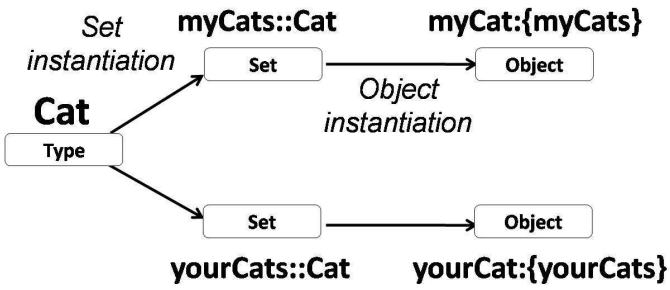


**Fig. 1.** Relationships between Types, Sets and Objects

In particular, 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;
– *Structure Type*: Structures 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. Structure 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 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 illustrates a formalism through which we can describe DL information object conceptual models in terms of our data model – as well as the Entity-Relationship model is used to describe the data model of a reality of interest. According to the model, circles represent Sets of Atoms, rectangles Sets of Structures, whose Types are described by double-lined rectangles to which they are connected with a dashed arrow. Rhombuses represent Sets of Relations: the totality (partial or total relation) and the cardinality (one to one, one to many, many to many) of the relationship it defines are represented through different arrows. In the figure, one Atom PDF in *MyPapers* can be related to none or one structure object in *MyDublinCore*, while each structure object in *MyDublinCore* must be associated to exactly one Atom object in *MyPapers*.

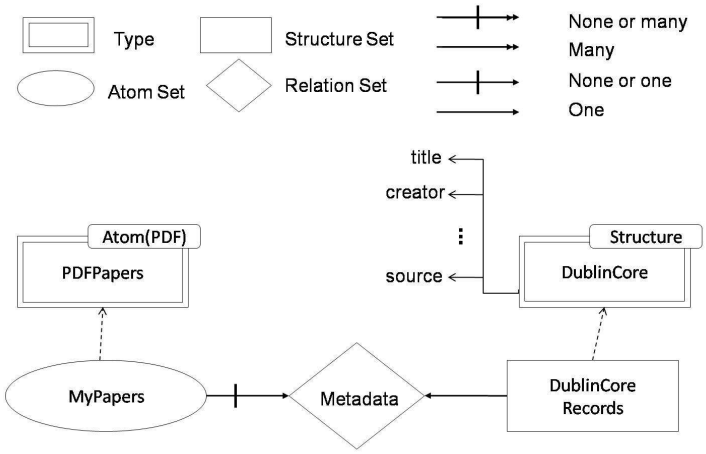


**Fig. 2.** Information Object Models formalism

Figure 3 shows the information object model arising from the specification of RC3 and RC4. The data model formalism clearly states the structure and content of the Objects to be managed by the two DLSs as well as their shared portion, namely the Sets *MyPapers* and *DublinCoreRecords*.

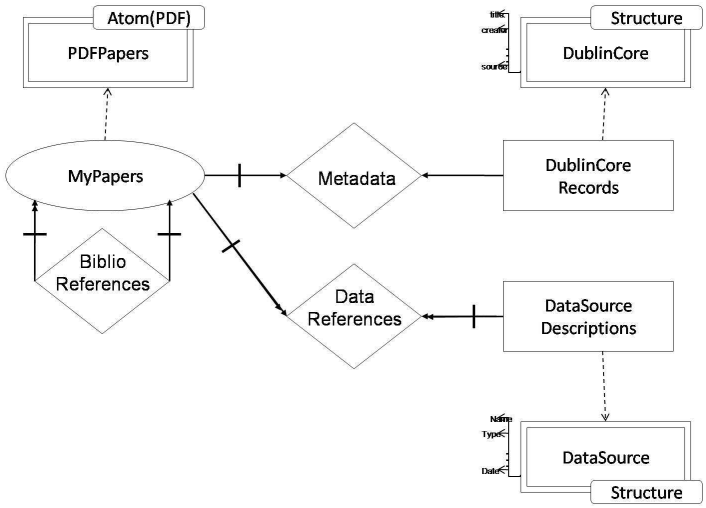


**Fig. 3.** Information object model for the Real-Cases 3 and 4

## 4 Typed flexible DLMSs: Doroty

Such a model offers a way for designers to unambiguously specify the features of the information object model for their DL. In our research, however, this formalism also shows the minimality and completeness of the compound data model it captures. This proof of concept was crucial before starting the design and development activities of a DLMS implementing the data model. Currently, we begun the development of a prototype of a DLMS called *Doroty* (*DigitaL Object RepOsitory with Types*), offering APIs for creating Sets according to a Type and creating and querying Objects into Sets using an X-Path based query language. The DLMS exploits type information for two main reasons:

– Checking type-correctness of the DLS software operating on-top of Doroty: programs must respect the Type constraints imposed by the DL designer and thus cannot mis-behave when manipulating the Objects of a Set;

– Optimizing the storage of objects based on Type information: Structures are stored into Sets endowed with indexes; Doroty storage has a modular architecture, which will be incrementally enriched by modules exploiting Type information to store Atoms according to properties such as their mime type, size and access mechanisms.

The design and development of Doroty is based on the architectural principles described in [10] and on the language and grammar presented in [11].

## 5 Conclusions

In this work we discussed the motivations behind the realization of DLMSs and illustrated the modeling requirements of information object models by means of four DL real-case scenarios. We have shown how existing flexible data models for DLMSs fail at satisfying part of such requirements and then presented the foundations of a typed compound object model that appears to fulfill them. Currently we are developing a prototype of a DLMS, called Doroty, which supports such data model so as to offer a practical proof of concept of the typed data model approach to DLMS construction.

**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. Candela, L., Castelli, D., Ferro, N., Ioannidis, Y., Koutrika, G., Meghini, C., Pagano, P., Ross, S., Soergel, D., Agosti, M., Dobreva, M., Katifori, V., Schuldt, H.: The DELOS Digital Library Reference Model - Foundations for Digital Libraries. DELOS: a Network of Excellence on Digital Libraries (2008) ISSN 1818-8044 ISBN 2-912335-37-X.
2. Ioannidis, Y., Maier, D., Abiteboul, S., Buneman, P., Davidson, S., Fox, E., Halevy, A., Knoblock, C., Rabitti, F., Schek, H., Weikum, G.: Digital library information-technology infrastructures. International Journal on Digital Libraries **5** (2005) 266–274
3. Tansley, R., Bass, M., Stuve, D., Branschofsky, M., Chudnov, D., McClellan, G., Smith, M.: The DSpace Institutional Digital Repository System: current functionality. In: Proceedings of the third ACM/IEEE-CS joint conference on Digital libraries, IEEE Computer Society (2003) 87–97
4. Lagoze, C., Payette, S., Shin, E., Wilper, C.: Fedora: An Architecture for Complex Objects and their Relationships. Journal of Digital Libraries, Special Issue on Complex Objects (2005)
5. Ioannidis, Y.E., Milano, D., Schek, H.J., Schuldt, H.: DelosDLMS. International Journal on Digital Libraries **9** (2008) 101–114
6. Assante, M., Candela, L., Castelli, D., Frosini, L., Lelii, L., Manghi, P., Manzi, A., Pagano, P., Simi, M.: An Extensible Virtual Digital Libraries Generator. In Christensen-Dalsgaard, B., Castelli, D., Jurik, B.A., Lippincott, J., eds.: 12th European Conference on Research and Advanced Technology for Digital Libraries, ECDL 2008, Aarhus, Denmark, September 14-19. Volume 5173 of Lecture Notes in Computer Science., Springer (2008) 122–134

7. Gonçalves, M.A., Fox, E.A.: 5SL - A Language for Declaratively Specification and Generation of Digital Libraries. In: Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'02), Portland, Oregon (2002) 263–272
8. Witten, I.H., Bainbridge, D., Boddie, S.J.: Power to the People: End-user Building of Digital Library Collections. In: Proceedings of the first ACM/IEEE-CS joint conference on Digital libraries, ACM Press (2001) 94–103
9. Millington, P., Nixon, W.J.: EPrints 3 Pre-Launch Briefing. Ariadne **50** (2007)
10. Candela, L., Manghi, P., Pagano, P.: 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)
11. Castelli D., Candela L., M.P.M.M.P.P.: Typed Compound Objects Models for Digital Library Repository Systems. Technical report, Istituto di Scienze e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche (2008)
12. Candela, L., Castelli, D., Manghi, P., Pagano, P.: Item-Oriented Aggregator Services. In: Third Italian Research Conference on Digital Library Systems, Padova, Italy (2007)