A Storage Model for Supporting Figures and Other Artefacts in Scientific Libraries: the Case Study of Invenio

Piotr Praczyk[1,2], Javier Nogueras-Iso[2], Samuele Kaplun[1], and Tibor Šimko[1]

[1] CERN, Geneva, Switzerland
[2] Universidad de Zaragoza, Zaragoza, Spain

Abstract. Current digital libraries for scholar publications are facing new challenges to facilitate discovery and access to digital objects distinct from the traditional full-text documents, e.g. figures, data sets or software related to scientific developments.
This work presents an extension of the data storage model of Invenio, a digital library platform developed at CERN. We concentrate on fulfilling requirements arising while extending INSPIRE, the information resource in High Energy Physics, with storage of figures and preservation of data files on which publications are based.

Keywords: digital library, data preservation, data storage, metadata, MARC, Invenio, INSPIRE

1 Introduction

Over recent years we observed rapid development of digital libraries and the software allowing management, retrieval and preservation of data. This included amassed efforts to increase the interoperability between different software platforms by standardising data exchange formats and protocols involved in this process [1]. The development of data-mining techniques allowed more extensive usage of content stored in digital library systems. This includes the automatic reasoning based on full texts of publications, but also the automatic discovery of internal structure of documents and providing access and search with finer granularity. The latter includes the interest in automatic extraction of figures and tables 'locked' in scientific publications [2].

INSPIRE is a project aiming at constructing a digital library of all publications of High Energy Physics (HEP). INSPIRE is a successor of SPIRES [3] database, created since 1970 at the Stanford Linear Accelerator Center (SLAC), providing metadata about preprints. With the advent of WWW, a web interface to the existing SPIRES resource was created, making SPIRES the first web site in North America. Later, the birth of arXiv.org allowed SPIRES to provide not only metadata, but also to display links to full-text documents. These two services are often perceived by users as a single system [3], SPIRES providing a search engine based on the metadata and being equivalent to a paper-catalogue

in libraries, and arXiv.org being a storage of articles linked from search results. The SPIRES software written in mid 1970s was still running at the beginning of the XXI century, becoming increasingly difficult to maintain. In the era of content-based search engines like Google and user-oriented services, metadata-only based search engines stopped providing the best available service.

The INSPIRE project arose as a collaboration of SPIRES database containing manually curated, high quality records and the Invenio [4] software platform for digital library repository, developed at CERN. INSPIRE, being a service based on a more modern software platform, provides searches based not only on metadata but also on full text. Later developments concentrate on creation of intelligent content-aware tools allowing automatic keywording of records, disambiguation of authors having similar names, or storage and search of figures. INSPIRE serves a large community of users consisting of the researchers in high-energy physics. Content of INSPIRE comes from various sources. The main corpus of data is being harvested from different digital libraries using the OAI-PMH [5] protocol or obtained directly from publishers. This data is then automatically improved and if necessary, manually curated. INSPIRE also enables its content to be harvested. These characteristics make INSPIRE an example of Very Large Digital Library [6].

One of the challenges that INSPIRE faces during its development is the storage of different artefacts that are not publications but that are closely related to the publication process. An example of such an artefact is a data file or a software code of a simulation leading to the discovery described in the scientific paper. Because of a lack of standard platform, this type of information is usually either published on private websites by authors, or not published at all. At the same time, data files have great value for scientists wanting to reproduce results of their colleagues or to perform similar calculations.

Complete preservation of data artefacts is a very complex process. In this paper we concentrate only on the part of establishing infrastructure for storing custom data objects inside the digital library and on providing necessary APIs to different parts of the system. We omit discussion of all issues connected with the possibility of reusing preserved custom data in the future. These aspects are very important, but not dependent on the underlying data model. There exists extensive research addressing these problems [7–9]. The data model implemented in the current version of Invenio, although flexible, has some intrinsic limitations that lead to proposed extensions that would satisfy the requirements for a more general data artefact storage.

The paper is structured as follows. In Section 2 we discuss the current storage system of Invenio. Section 3 describes new requirements that cannot be satisfied by the existing system. In Section 4 we describe the new data model that is backwards compatible with the existing storage system and allows for new use cases. Later in Section 5 we discuss issues of uploading data in Invenio and the interoperability between Invenio installation and other digital libraries with respect to the exchange of data objects. Finally, in Section 6 we briefly com-

pare our work with other digital library systems and in Section 7 we present conclusions and outlook for the future work.

2 The current storage and description model

The Invenio subsystem responsible for managing documents is split into three layers. Data model varies between the end-user view, the API (Application Programming Interface) layer and the data storage layer. The description provided in this section has been prepared according to the data model available at the API level.

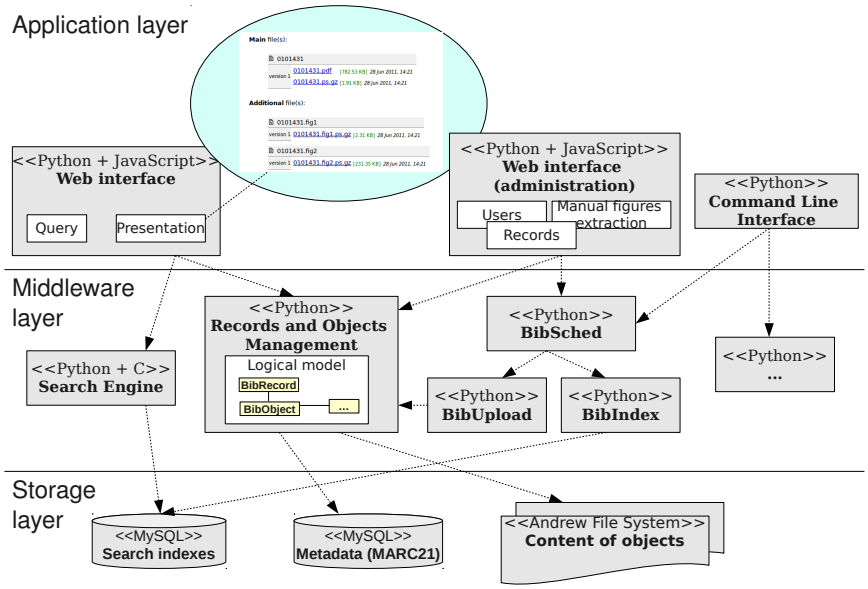

Fig. 1. Multi-layered architecture of Invenio.

Every bibliographic record can have a set a documents attached to it. The view presented to the end user is organised in a tree structure with roots corresponding to particular abstract documents, nodes aggregating available versions, and versions aggregating available formats for the given version.

On the API level, versions do not have status of separate entities. The document is an object having two dimensions: version and format. These two parameters can be provided by the end user in order to obtain a particular file instance. If they are not specified, the newest version is retrieved.

The storage layer utilises a database and the file system. Those entities from the API model that require remembering additional data associated with them are stored in the database. Entities that are simple enough are stored directly in the file system.

The data model currently used by Invenio concentrates on document objects related to bibliographic records. This approach carries several consequences. All the user-visible metadata is stored in MARC format representing the bibliographic record. While one bibliographic record can reference many documents, a single document cannot be attached to multiple bibliographic records. Figure 2 depicts relations between entities appearing in Invenio when dealing with the storage of attached documents.

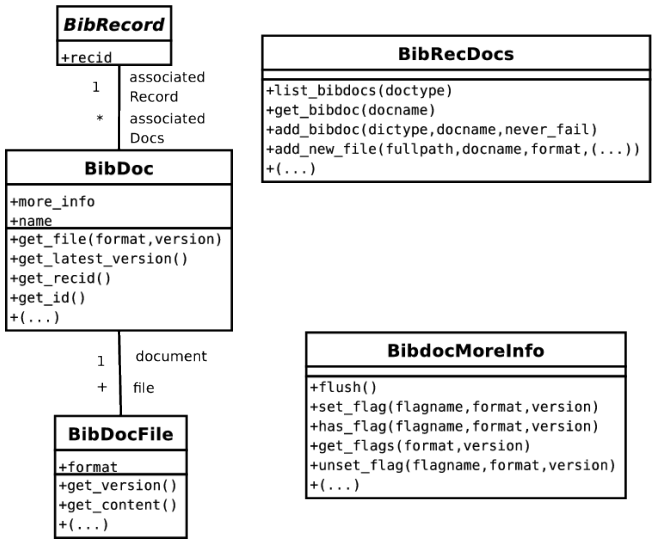

Fig. 2. Data model currently used in Invenio.

BibDoc is a class representing document at the most abstract level. It is agnostic to the concrete encoding of the files representing an object. In the case of fulltext representation, a *BibDoc* instance represents one document regardless of whether it is encoded in PDF or Postscript formats or as a text file. The class is capable of representing the document in different versions, allowing selective access to them.

BibDoc allows storage of additional pieces of information that should not be presented in the MARC record of the publication. This includes different flags used by Invenio software internally. Additional data is stored in an internally-managed *MoreInfo* instance. *MoreInfo* has a dictionary structure and is serialised and stored in a BLOB field of the database table representing documents. An example of the data stored in *MoreInfo* is a flag marking the document as 'hid-

den'. Documents having this flag are not displayed in the MARC representation of a record, but they can be accessed and processed internally. The *BibDoc* API does not allow accessing the *MoreInfo* dictionary freely, but rather provides separate methods for modifying different types of properties stored there. *BibDoc* instances can be identified by the control number of the record to which they are attached, and a name that has to be unique within the scope of the record. It is assumed that exactly one connection to a record will exist for each *BibDoc*.

BibDocFile is a class representing the concrete representation of a document. Each of its instances encodes a particular format and version. The notion of format is related to the ordinary file format, though it extends it. Besides the main format type (such as jpg or pdf) it is possible to specify a 'derived format' or a 'sub-format' as an arbitrary string that is useful for example to store the same master graphical file in different derived resolutions. In this case, a document will have more files with the same format and different sub-formats.

BibRecDocs is a class providing an interface to retrieve documents associated with a particular record. An instance of this class can be constructed by providing the control number of the bibliographic record. Such an instance provides methods manipulating *BibDoc* instances representing particular documents.

3 Need for a new storage structure

As mentioned in the introduction, we concentrate on the following use cases: (i) preserving scientific results and data files; (ii) storing figures, notably in situations where the same data object is reused by multiple bibliographic records.

The implementation of these two use cases implies managing not only bibliographical records but also custom objects as first-class citizens in the Invenio data ecosystem. This includes providing means of storing, identifying and accessing objects without reference to records to which they are attached.

When translating to the lower level, these requirements become:

1. Storing standalone objects not attached to records.
2. Assigning unique persistent identifiers to objects.
3. Allowing to establish relations between custom objects.
4. Attaching objects to possibly many records rather than exactly one.

Currently, the documents are assigned a unique identifier by Invenio, although it is not visible outside of the implementation layer. Satisfying the second listed requirement can be done simply by making the existing mechanism more prominent. Other requirements require deeper modifications.

In order to allow storage of different types of objects, the storage system should be as flexible as possible, allowing the storage of completely custom metadata. Because of its structure of a tree with a maximal depth of two, the MARC format is neither very effective nor natural as a format for describing non-record objects.

The use case of figures as a specific version of attaching custom objects may be considered as a useful testing scenario for the metadata organisation. Basic

metadata elements that describe figure include: caption, text inside figure, exact location of figure and its caption within the fulltext document, fulltext document from which the figure has been extracted, list of references to the figure from the text of the article, or the text present inside the figure itself. This list is not closed and can grow in the future when methods of treating figures will progress. Some of possible future metadata elements include: type of figure (plot, photograph, diagram), information about quantities represented on axis and scales. We can also store links to the data used to create the plot, which might exist as a different object stored in the system.

4 Proposal for an extension of the storage model

Figure 4 depicts the proposed new structure of data organisation inside Invenio. In order to reduce the need for modifications on the way Invenio treats records, the model extends entities known from the current implementation.

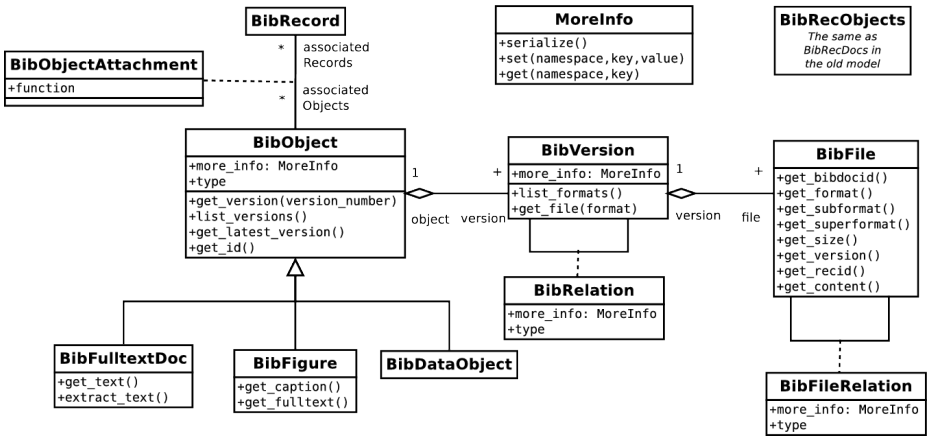

Fig. 3. UML diagram of the proposed data model.

The *BibObject* class is equivalent to the old *BibDoc* class. The name change emphasises the usage of the entity to represent arbitrary objects. In the new model, *BibObject* instances allow direct access to a more general *MoreInfo* object resigning from managing every type of information with dedicated methods. The type property of the link between the object and the bibliographic record is no more promoted to property of *BibObject* instances.

*BibObject*s are still identified by unique names on the level of the record. However, as many records may reference the same document using different names, this information is no more an attribute of the *BibObject* instance. In the proposed model, the name of a document inside a record has been shifted to the relation between object and the bibliographical record. Instead of using

a name to identify documents, a persistent and unique identifier in the scope of an Invenio installation is assigned to each custom object.

Currently, many aspects of the *BibDoc* class are oriented towards dealing with full text documents. In the new model, a similar specialisation of class behaviour may be achieved by permitting subclassing of the *BibObject* class. When creating an instance representing a document, a type property might be read and used to instantiate an appropriate subclass. Subclasses could be provided using the Invenio plug-in system.

BibObjectAttachment class is an association class representing all properties of the link between an object and a bibliographical record. The instances of this class store information about the function under which the object is attached to the record. As aforementioned, also a unique name inside a record is stored.

BibRecObjects is an interface similar to *BibRecDocs*. Instead of retrieving *BibObject* instances, each *BibRecObjects* instance provides an interface to retrieve instances of *BibObjectAttachment*. Previously, every *BibDoc* could be retrieved using an instance of *BibRecDocs*. Now it is not possible because there may exist objects that are not associated with any record.

In order to provide a more general metadata storage system, *MoreInfo* instances can be attached to more elements of the structure. Because of this, the *BibVersion* entity has been introduced. The *BibObject* class provides access to a collection of its versions. Each version provides access to the *BibDocFiles* that behave similarly to the old scheme, but in addition it provides access to the *MoreInfo* data field. Besides the above advantages, the explicit treatment of versions is more uniform with the mechanism used by Invenio to communicate the structure of attached documents to end users.

BibRelation is an entity modelling the concept of generic relations between documents. This relation is established between particular versions of documents. The relation remembers, and contains information about, its type, source and target *BibObject* versions, and *MoreInfo*. Each *BibObject* instance provides an API allowing to retrieve its incoming and outgoing relations. *BibRelations* are useful to model, for instance, that one document has been derived from another.

The structure of *MoreInfo* has been extended in the new model. Many Invenio modules might need to store different pieces of information related to the same object. This might happen for example when a module processing figures needs to store some portions of metadata and a module dealing with access and copyright control needs to do the same. In order to avoid name clashes, the dictionary structure has been extended by name spaces. In practice, this means that the dictionary has been equipped with an additional level. Each module has access to one name space which should be characteristic for the module. Inside this name space, arbitrary key-value pairs can be stored. Keys should be encoded as strings, but values can be arbitrary cPickle[3] serialisable objects. If two different modules want to save data using the same key, the conflict will be avoided because they belong to different name spaces. The low-level storage of *MoreInfo* can be implemented in several manners. As long as the amount of

[3] http://docs.python.org/library/pickle.html

data stored inside one structure is small, the current solution using the serialisation and storage inside a BLOB field of a database is efficient. This solution may become too slow if the amount of metadata grows considerably. It is also more difficult to index data stored in *MoreInfo* BLOB fields implemented in this manner. The indexing of metadata fields may be simplified by providing derived logical fields calculated based on the data environment of the record. An alternative implementation could involve the use of NoSQL technologies such as MongoDB [10]. Although this solution seems to be suitable for managing data without known schema, yet additional software dependencies would be imposed on Invenio.

The interface described in this section does not guarantee that every object will correspond to at most one instance of *BibObject* class. This would be difficult to achieve, because an Invenio installation may be distributed across multiple computing nodes. When an instance is created, it reflects the state of the object at the moment of instance creation. All the modifications to objects have to be explicitly saved using appropriate flush functions. This is not a problem as long as all the data modifications are performed by a special bibliographic task (*BibUpload*) that is executed in a serial way by a dedicated queuing service (*BibSched*).

In the case of figures, an abstract figure can be stored as an instance of *BibObject*. Particular formats, like PNG or SVG should be stored as *BibFiles*. If a figure was extracted from the fulltext of the document, a *BibRelation* should be established between a particular version of *BibObject* representing the figure and a particular version of *BibObject* representing the fulltext. When a fulltext document changes, figures extracted from it may change their location, which indicates that certain types of metadata should be stored together with representations of *BibRelation*. This metadata includes: the location inside of the document, the location of the caption, the caption or text references.

Certain types of metadata are properties of the object itself and should be stored either in *MoreInfo* associated with *BibObject* directly or in *MoreInfo* of a particular *BibVersion*. These portions of metadata contain text extracted from the figure but also all the possible future metadata elements such as types of physical quantities described by the plot. Additionally, the links to the objects representing the data underlying the figure should be stored in the association with *BibObject* instances.

5 Importing and exporting data

5.1 Current extensions to MARC format supporting uploading of attachments

In Invenio, metadata records are represented internally in MARCXML[4] format. MARCXML is also used as primary format for importing information. Storing data objects transcends the capabilities of MARC.

[4] http://www.loc.gov/standards/marcxml/

In order to insert data in a consistent manner, the use of MARC had been extended in Invenio by a custom artificial MARC field called FFT (Fulltext File Transfer) which allows specifying information about external data to be attached to the record. Table 5.1 shows the complete list of subfields of an FFT field and explains their usage.

code	description
$a	location of the docfile to upload (a filesystem path or a URL)
$n	docfile name (if not set, deduced from $a)
$m	new desired docfile name (used for renaming files)
$t	docfile type (e.g. Main, Additional)
$d	docfile description
$f	format (if not set, deduced from $a)
$z	comment
$r	restriction (used only when deleting or reverting files)
$v	version (used only when deleting or reverting files)
$x	url/path for an icon

Table 1. Custom FFT MARC field and its subfields.

The data uploading process is in Invenio managed by a special daemon called *BibUpload*. When an FFT field is encountered during the uploading process, it is interpreted instead of being directly saved in the database like it is the case with regular MARC fields.

The $a subfield encodes the location of the file to be uploaded. *BibUpload* reads the content of this subfield and transfers the referred file into internal Invenio storage system. Other FFT subfields are also read and used to build instances of classes described in Section 2. These objects are also moved to the persistent storage. Subsequently, an occurrence of 856 field[5] containing a persistent link to the file inside Invenio is created. Subfields of the 856 field are populated using the content from some subfields of FFT.

The described schema encounters a number of limitations with new data needs: for example, the upload of arbitrarily structured data to be uploaded to *MoreInfo* dictionaries is difficult in MARC, and the current syntax does not permit creating links between already existing *BibDocs* and bibliographic records.

The described model allows several different scenarios of attaching objects. We can either create completely standalone objects and attach them to multiple records, or we can also create a single MARC record describing the object and referencing an underlying *BibObject*. The way of attaching objects can be decided together with details of the MARC format.

In subsection 5.2 we describe a METS-based format that could be adopted to upload data objects. Currently the upload using this format is not implemented, because for the primary use case of INSPIRE it is enough to use extended current FFT syntax allowing to upload serialised entities of the proposed data model.

[5] http://www.oclc.org/bibformats/es/8xx/856.shtm

5.2 Uploading objects with new METS-based format

As described in the previous section, the syntax provided by MARC and extended with the FFT tag is not sufficient to allow the upload of standalone objects and relations between them. This creates a need for more expressive language. METS is an XML-based standard [11] for describing structured objects and their metadata. For the purposes of Invenio we can use a customised subset of METS reflecting internal data structures. The METS input can be processed by the uploading daemon along with the existing MARC input. In this scenario, the distinction between uploading publication metadata and objects preserved in Invenio becomes much clearer. At the same time, the FFT syntax remains intact with a different understanding as being a syntactic sugar for more general METS description. Internally, FFT tags can be translated to corresponding METS.

```
<structMap>
  <!-- uploading new object -->
  <div type="BibObject" id="tmp:NewObject1">
    <div type="version" id="tmp:NewObject1:newVersion"
         dmid="identifier_of_metadata">
      <fptr ... />
      <fptr ... />
    </div>
  </div>

  <!-- uploading relation between new version of NewObject and
       existing object -->
  <structLink xlink:from="tmp:NewObject1:newVersion"
              xlink:to="objId:232314:3"
              xlink:arcrole="is_extracted_from"
              dmid="link to metadata in serialised moreinfo" />

  <!-- establishing relation between two existing objects -->
  <structLink xlink:from="objId:2334234:5"
              xlink:to="objId:232314:6"
              xlink:arcrole="is_extracted_from"
              dmid="link to metadata in serialised moreinfo" />

</structMap>
```

Fig. 4. Example of a METS structural map section describing the upload of one new object and establishing its relation with an existing object.

A full description of the METS format can be found in [11]. METS is structured in different sections. From the point of view of our use case, the sections of METS representation that are the most important are descriptive metadata,

files section and structural map. The section describing files would allow Invenio daemon to locate files scheduled to be uploaded in the local file system. The descriptive metadata section could provide real metadata of the uploaded object. This metadata might be encoded in one of metadata description formats (MODS, MARC or some custom format, serialised *MoreInfo* dictionary). The structural map section is the most interesting from the point of view of uploading data. We can use *div* elements to encode entities of the proposed Invenio data model. The *div* elements are used to establish a hierarchical structure within described digital object.

Figure 5.2 depicts a sample structural map section of an input METS file. Every top-level division describes a *BibObject*. If the identifier is provided, the description can be used to update an existing entity. If the identifier is not provided, or if it is not present in the system, a new object should be created.

A *BibObject* division should contain a subdivision describing a particular version of the document. The identifiers of versions are constructed by concatenating the identifier of the *BibObject* and the number of the version. Inside every version *div*, we can specify references to particular files forming part of it.

Every object in the hierarchy may be linked to an entity from the descriptive metadata section. This description will be encoded inside the *MoreInfo* field.

6 Related work

In this section we review several existing approaches to storing objects in digital libraries and discuss connections and differences with respect to our proposal for the Invenio model.

DSpace [12] is an open-source platform for institutional repositories. The data model of DSpace emphasises the usage of Dublin Core [6] for describing all possible types of resources, while Invenio is built around MARC for bibliographical records. Our proposed data model allows more flexibility in the case of custom objects, because it does not assume any particular metadata model, only defines data access interfaces and imposes very basic requirements on stored data types. This is particularly important in large systems consisting of many modules having different data requirements, such as INSPIRE. Different modules can treat data differently and store different types of metadata fields.

Another feature of the DSpace approach is that it allows describing the internal structure of documents and methods of building objects from separate data streams. The Invenio model also allows describing relations between objects for construction of more complex objects, but only on a level higher than the data model.

OpenDLib [13] is a modular repository service intended to be used as a building block for digital library systems. The DoMDL data-model proposed in the solution is very general and makes few assumptions about the nature of handled documents [13]. However, the extended data model of Invenio has been

[6] http://dublincore.org

designed to meet more specific requirements of storing figures and data behind articles. Some notions of the DoMDL model, for example *View*, are not necessary as referring to the internal structure of the document and are completely omitted in our work.

Flexible and Extensible Digital Object and Repository Architecture (FEDORA) [14] is a standard for describing digital objects. The data model used in Invenio is much simpler, not going in the direction of objects composed of many parts stored as separate content streams. On the other hand, FEDORA data model does not define the notion of an object version, which is crucial for application in INSPIRE.

In summary, the existing storage systems concentrate on managing bibliographic materials having complicated internal structures. For instance, storing scanned pages of publications separately and making the data model to describe the method of assembling the whole publication from parts. However, in the case of INSPIRE, this type of use case does not exist as the data is stored primarily in atomic, consistent PDF documents. The difficulty that INSPIRE is facing consists of storing additional materials complementary to publications, remembering relations between them and metadata associated with all these entities. Therefore the model proposed in this work extends the existing implementation in Invenio to include minimal modifications that increase the flexibility to incorporate different digital objects associated with publications.

7 Conclusions

In this paper we have proposed a new model for the storage of custom objects inside Invenio. It enables to manage figures, data files, software source code, and other data artefacts that may be associated to existing publications or that may exit independently. Proposed flexible infrastructure facilitates development of new applications for searching and accessing digital objects.

There are several issues that must be addressed during development of the proposed model:

1. The first issue involves deciding if the data should be stored in a database or partially in the file system. The underlying database could follow a relational model or be a key-value store[7].
2. The second issue is more complex and relates to the integration of extended objects with the rest of the Invenio platform. The integration of custom object model with Invenio requires to allow searching for custom objects and displaying information about objects. This can be achieved either by creating small, almost empty MARC records that would point to objects, reusing parts of current record facilities, or by providing splash pages rendering information directly from *MoreInfo* structures.
3. The third issue is connected to the need of assigning Digital Object Identifiers (DOI) to stored data objects. These identifiers should store the persistent state of a data object, allowing to manage a particular version of an object.

[7] http://en.wikipedia.org/wiki/NoSQL#Key-value_store

Acknowledgements

We would like to thank Salvatore Mele and the whole INSPIRE team for their comments and insights. This work has been partially supported by the Spanish Government through the project TIN2009-10971.

References

1. Rebecca Guenther and Sally McCallum. New Metadata Standards for Digital Resources: MODS and METS. *Bulletin of the American Society for Information Science and Technology*, 29(2):12–15, 2003. ISSN 1550-8366. URL `http://dx.doi.org/10.1002/bult.268`.
2. Saurabh Kataria, William Browuer, Prasenjit Mitra, and C. Lee Giles. Automatic extraction of data points and text blocks from 2-dimensional plots in digital documents. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, pages 1169–1174. AAAI Press, 2008. ISBN 978-1-57735-368-3.
3. Anne Gentil-Beccot, Salvatore Mele, Annette Holtkamp, Heath B O'Connell, and Travis C Brooks. Information Resources in High-Energy Physics: Surveying the Present Landscape and Charting the Future Course. oai:cds.cern.ch:1099955. *J. Am. Soc. Inf. Sci. Technol.*, 60(arXiv:0804.2701. CERN-OPEN-2008-010. DESY-08-040. DESY-2008-040. FERMILAB-PUB-08-077-BSS. SLAC-PUB-13199. 1): 150–160. 27 p, Apr 2008.
4. Jerome Caffaro and Samuele Kaplun. Invenio: A Modern Digital Library for Grey Literature. oai:cds.cern.ch:1312678. Technical Report CERN-OPEN-2010-027, CERN, Geneva, Dec 2010.
5. Herbert Van de Sompel, Michael L. Nelson, Carl Lagoze, and Simeon Warner. Resource Harvesting within the OAI-PMH Framework. *D-Lib Magazine*, 10(12), 2004. ISSN 1082-9873.
6. Paolo Manghi Yannis Ioannidis and Pasquale Pagano. Report of the Second Workshop on Very Large Digital Libraries VLDL 2009. *D-Lib Magazine*, 15(11/12).
7. André G. Holzner, Peter Igo-Kemenes, and Salvatore Mele. First results from the PARSE.Insight project: HEP survey on data preservation, re-use and (open) access. *CoRR*, abs/0906.0485, 2009.
8. Richard Mount et al. Data Preservation in High Energy Physics. 2009.
9. Margaret Hedstorm. Digital Preservation: A Time Bomb for Digital Libraries. *Computers and the Humanities*, 31:189–202, 1998.
10. Kyle Banker. *MongoDB in Action - A document database for the modern web*. Manning, 2011. 375 pp.
11. Digital Library Federation. *METS - Metadata Encoding and Transmission Standard: Primer and reference manual*. 2010.
12. Robert Tansley, Mick Bass, David Stuve, Margret Branchofsky, Daniel Chudnov, Greg McClellan, and MacKenzie Smith. The DSpace Institutional Digital Repository System: Current Functionality.
13. Donatella Castelli and Pasquale Pagano. A Flexible Repository Service The OpenDLib Solution. *VWF proceedings*, 2002.
14. Sandra Payette and Carl Lagoze. Flexible and Extensible Digital Object and Repository Architecture (FEDORA). *ECDL '98, LNCS*, 1513:49–59, 1998.