Towards a Unified Runtime Model for Managing Networked Classes of Digital Objects*

Kostas Saidis and Alex Delis
{saiko,ad}@di.uoa.gr

Department of Informatics
University of Athens, 157 84, Athens, Greece

Abstract. In this paper we propose a unified runtime model for managing diverse and heterogeneous digital material in terms of network classes/types of digital objects. Our proposal aims to offer a general-purpose, reusable system that may act as a common runtime layer for the development of any digital library. We identify the fundamental requirements that such a runtime layer should fulfill, namely (a) storage independence, (b) service neutrality and (c) digital object modeling capabilities. Then we present how our proposal meets these requirements, while offering a great deal of expressiveness and ease of use to the DL application developer.

1 Introduction

Digital Libraries (DLs) need to provide the infrastructure for enabling users to access a broad range of intellectual works originating from various interconnected and heterogeneous sources. DL development can be viewed from many perspectives including but not limited to information retrieval and integration, web information and data management, human/computer interaction and software engineering. Apart from this plethora of disciplines involved in DL development, modern DL systems have to cope with diverse uses and deployment scenarios. For example, two DL systems may: (a) employ different storage solutions, (b) operate on different types of intellectual works, (c) may use different metadata schemes to document such works and finally, (d) may synthesize digital object information in diverse service provision environments, governed by different protocols, services and user interfaces. These variations depend on the rules and constraints of the particular DL-specific environment. In this paper we elaborate on the essential features that should be offered by a DL Management System (DLMS)[1]. Our objective is to realize such a system to act as the "core" for developing any DL regardless of the variations that may occur in the deployment environment of DLs.

We view the DLMS as an intermediate runtime layer between (a) the services employed by a DL and (b) the sources of material managed by a DL. We supply the DLMS with a unified foundation, identifying the critical requirements that should be met by the DLMS runtime layer and its underlying logical model. We also focus on the abstractions and expressiveness the DLMS should offer to the DL service developer. For example,

* This work was supported by a European Social Funds and National Resources Pythagoras Grant with No. 56-90-7410 and the Univ. of Athens Research Foundation.

databases offer a great deal of expressiveness to application developers through the use of a domain-specific language, namely SQL. As a domain-specific language can provide the ultimate abstraction of a problem [2], in practical terms, a developer can view a database as an interpreter of SQL statements, assisting her to realize complex data management tasks with a little effort. Unfortunately, a corresponding analogy is not available in DL service development; DL developers have to realize complex digital material management tasks "manually". The DLMS should not only offer an effective unified representation of networked, semantically diverse and heterogeneously stored digital material but it should also empower DL developers with expressiveness and ease of use. Thus, we need to supply the DLMS with a language pertinent to the digital object management domain. Such a language will automatically "translate" incompatible physical/storage representations into the DLMS' unified runtime model, abstracting high-level services from the diversity and heterogeneity of underlying networked sources of material.

The remainder of this paper is organized as follows. In Section 2 we elaborate on the design of a DLMS and the essential requirements of its unified model. In Section 3 we propose an approach to implement the DLMS using a domain-specific embeddable language [3] of network classes/types of digital objects. Finally, in section 4 we conclude the paper.

2 Requirements of a Unified Model for Digital Object Management

Various terms and concepts have been used in the literature to refer to DL material including but not limited to *content*, *digital object*, *information object* and *data object* [1,4,5,6,7,8,9,10]. To narrow down the ambiguity, we use a single term, *digital object*, to refer to any digital artifacts managed by DLs such as e-prints, digital assets, archival material and so forth, independently of the particular representations and mechanisms involved in storing such artifacts. We call the storage artifact a *stored digital object* and we use the term *digital object store* (DO store) to refer to any system that can hold stored digital objects. As a database system constitutes a reusable, general-purpose tool for managing data in terms of tuples, respectively, we need to realize a DLMS as a reusable, general-purpose system that can be used to manage data in terms of digital objects. In the remainder of this section, we elaborate on the design of a unified runtime model for the DLMS, identifying the requirements that such a model should fulfill.

• **Storage Independence:** A storage-independent logical model of digital objects will allow the DLMS –and the higher-level DL services– to operate atop any DO stores such as custom database solutions or XML-based repositories. The digital object representations employed by the DLMS should be independent of any physical digital object storage representations as the latter may differ between a DO store and another. Moreover, as it often occurs in practice, digital objects may need to be moved from one DO store to another and the DLMS should be able to effectively cope with such a need.

The challenge is to enable the DLMS to actually supply high level services with a unified logical view of any kind of digital objects originating from any underlying DO stores. Given that the DLMS should be (re)usable in any DL deployment context, no

assumption can be made about whether such DO stores will either be local/remote or heterogeneous/homogeneous. Therefore, the DLMS should operate atop any physical/storage digital object model without requiring any modifications to be performed in the latter. Finally, in terms of digital object integration/interoperation, the DLMS should assist us to make any DLs to appear as remote DO stores for each other, effectively wrapping any source of digital objects in an inexpensive manner.

• **Service Neutrality:** The DLMS will employ appropriate structures to stage digital object information at runtime based on its unified model. It will also allow high-level services to manage this information through an API. We believe that the DLMS API should be exposed to higher-level services in a service neutral manner, allowing DL services to synthesize digital object information in any service provision environment of choice. Should we consider that the DLMS exposes its API in terms of web services, although such a decision best fits to an heterogeneous networked environment such as the WWW, it will probably be inefficient for developing a personal digital library system, for example. Moreover, in the case of a DL system used in a private intranet –acting as a document management system, for example– the organization should be allowed to use its proprietary communication protocols and existing service infrastructure for reasons of backward compatibility with legacy applications.

Similar service-neutrality exists in databases, where the result set returned by an SQL query is made available to the calling application through appropriate data structures that wrap involved database table/field data. In the context of a database system, no assumptions are made about the actual usage of data at the application level. An application may use this data to dynamically render a web page, feed a web service response or generate a PDF report, for example. Respectively, the DLMS API should only pertain to supplying high-level DL services with access to runtime representations of digital object information, as the latter is internally held by the DLMS runtime structures. The particular communication protocols and mechanisms involved in exposing digital object information to other systems and/or users should be designated by the higher-level DL application logic and its services.

• **Digital Object Modeling Capabilities:** The DLMS unified digital object model should support the fundamental requirement to represent semantically diverse digital objects in a unified manner. Such a model should supply DL designers with a common "language" for expressing a variety of digital object structural arrangements. In this context, we believe that the DLMS model should allow the DL designer to use all four established abstraction principles, namely aggregation/decomposition, grouping/individualization, classification/instantiation and generalization/specialization [11,12,13] transcending network and digital library boundaries. More specifically, the DLMS unified model should allow DL designers to express the aggregation nature of digital objects, while at the same time it should allow services to effectively decompose such aggregations for their service provision needs. Support for grouping/individualization will allow "incompatible" objects to be grouped together –i.e., for realizing a collection of digital objects or for linking two objects with each other. Classification/instantiation will allow the DLMS to make individual digital objects comply with their structural and behavioral specifications automatically. Finally, generalization/specialization will

allow the DL designer to reuse and/or refine digital object specifications across varying DO sources.

3 Our Approach

In our approach to deal with the DLMS requirements we view a digital object as an identifiable unit of information which may contain various types of digital content and multiple metadata schemes. It may also develop several relationships with other objects and it may expose behavior in terms of logical views of the above. As Figure 1 shows, we view a digital object as any combination of the following set of attributes: *Metadata Set*, *Metadata Field*, *Stream*, *Relation Context* and *Behavior Scheme*. Any digital object, at any point in time, may contain zero or more *Metadata Sets*, each one containing one or more *Metadata Fields*. The latter can hold multiple multi-lingual values. A digital object may also contain zero or more *Stream* attributes, used as handles for the underlying locally or remotely stored digital content including files, bitstreams or URIs. Moreover, a digital object may contain zero or more *Relation Contexts*, each one outlining a particular relationship between the object and others. A *Relation Context*, at any point in time, may contain zero or more members, referenced using their unique identifiers. Finally, a digital object may contain zero or more *Behavior Schemes*, each one providing a different view of the object's structure.

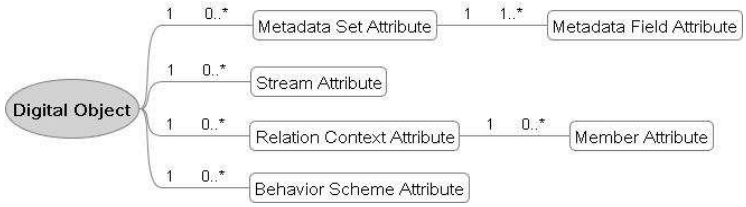

Fig. 1. The Logical View of a Digital Object and its Attributes

Each attribute type is used by the DL designer to model a particular component of a digital object, regardless of each component's storage representation employed by underlying DO stores. Consider two heterogeneous DLs hosting *books* and *articles* digital objects respectively. Articles full text is available in terms of PDF documents, while books' pages are digitized and made available in terms of digital images. Moreoever, different descriptive metadata are used to document each type of material. Finally, consider that *articles* originate from a custom database solution, while *books* and *pages* are stored in terms of a less-rigid XML format. Figure 2 shows the representation of such *article*, *book* and *page* digital objects, displaying the ability of our proposed model to generate logical views of the information held in a digital object, regardless of the digital object storage details.

By exploiting the above logical views that transcend network and DL boundaries, the DL designer can combine various attribute specifications to generate a definition

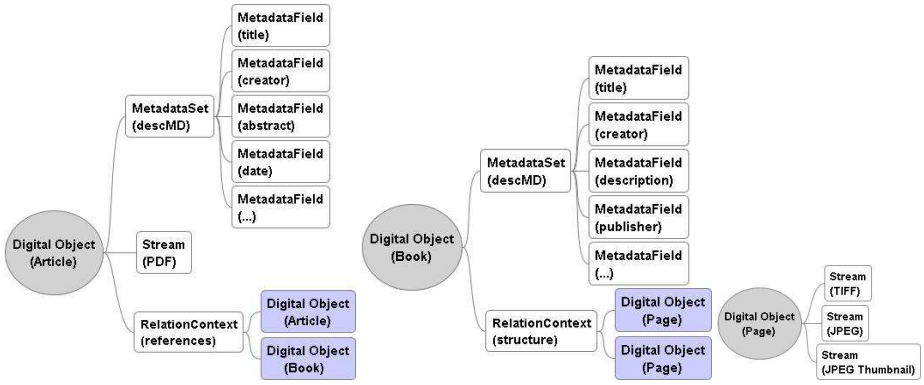

Fig. 2. Articles, Books and Pages digital objects

of the structure and behavior of a digital object. We call such digital object definitions *digital object prototypes*. With prototypes, the DL designer is able to express diverse classes of digital material containing numerous kinds of digital content, metadata and relationships such as the *books* and *articles* of Figure 2, while also supporting exceptional cases in which digital objects may contain (a) metadata values only, acting as metadata records, (b) the members of single relationship, acting as "relationship" objects (c) streams of bytes, acting as "file" objects. Formally, a digital object prototype outlines a namespace of identifier/attribute bindings. For example, the *article* prototype definition provides the following hierarchical namespace:

article : *prototype* (
  descMD : *MetadataSet* (**title** : *MetadataField* , **creator** : *MetadataField* , . . .) ,
  PDF : *Stream* ,
  references : *RelationContext*)

We treat each different prototype's namespace as generating a different digital object class and/or type [14]. At runtime, we guarantee that any two objects created via the same prototype will be of the same type by making them contain an identical namespace of identifier/attribute pairs. Supporting the classification/instantiation abstraction, we call a runtime representation of a digital object, *which automatically complies with its prototype*, digital object instance. Digital object instantiation represents the ultimate abstraction of our proposal. This refers to the process of "translating" an object's physical/storage representation into the logical model defined by the object's prototype at runtime[1]. Such a process is automatically carried out by our framework "behind the scenes", making a digital object instance appear to high-level services as being loaded automatically. Digital object instances are exposed to high-level services through the DLMS API which realizes our unified digital object model and "knows" of *Metadata Set*, *Metadata Field*, *Stream*, *Relation Context* attributes.

Figure 3 presents the overall picture of our proposed architecture of the DLMS. Each layer in the Figure abstracts its upper level from specific details, advancing sep-

[1] Our simple XML notation for defining prototypes is not presented herein due to space limitations. The XML syntax of the prototypes "language" is described in [15].

aration of concerns and modularity [16]. In particular, the distribution of knowledge among these layers is as follows. Underlying DO stores internally "know" how to store digital object information. The DLMS "knows" how to generate a unified logical view of the diverse and heterogeneous digital objects held beneath. Finally, the DL application logic "knows" the details of the particular DL deployment context and uses the DLMS' logical view to implement service provision.

To "translate" any underlying physical/storage digital object model into the DLMS unified model, we use the *DO store connectivity/driver* mechanism, as Figure 3 shows. The DLMS defines a DO store connectivity interface, providing the signatures of the operations it requires to access and/or modify underlying digital object information in terms of our proposed logical model. Consequently, each participating DO store in a DL should provide its own driver, offering an appropriate implementation of the DLMS connectivity interface that "knows" the details of the underlying stored digital object information. For example: (a) a DO store driver for an XML-based digital object repository will use the particular repository protocol to realize the DLMS interface, (b) a driver for a DO store that employs a relational database will use the appropriate SQL queries to realize the DLMS interface and so forth.

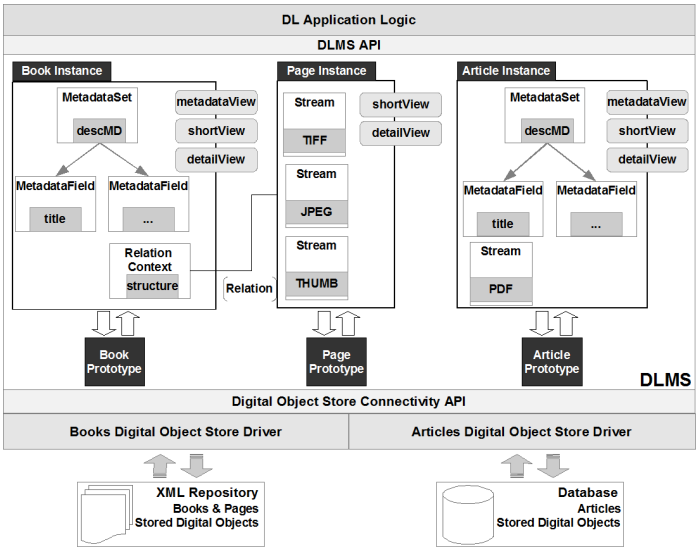

Fig. 3. Our proposal for realizing the DLMS

To manage the information of a stored digital object at runtime, a high-level service needs to acquire its corresponding digital object instance. To do so, the service provides the object's unique identifier which can follow any local or global naming scheme. Our runtime will then identify the stored artifact and retrieve the name of its prototype. It will then use the prototype's definition to create a corresponding digital object instance.

Lastly, the instance –being a runtime structure that holds stored object's data in terms of the DLMS prototype-based unified model– is returned to the service, hiding all digital object identification, location and storage details. For example, a book digital object instance will automatically load metadata values from the database and place them into its respective *Metadata Set* and *Metadata Field* attributes. Respectively, an article digital object instance will load its XML-encoded metadata into its own *Metadata Sets* and *Metadata Fields*, hiding from services DO store heterogeneity.

We finally use *Behavior Schemes* to realize digital object behavior in a storage-independent and service-neutral manner. As Figure 3 shows, all books, pages and articles instances contain their own behavior schemes such as *metadataView*, *detailView* and *shortView*. In practical terms, a behavior scheme defines a projection/view of an object's namespace and can contain any subset of an object's identifiable attributes, including any combination of the object's *Metadata Sets*, *Metadata Fields*, *Streams*, *Relation Contexts* and their members. For example, Figure 4 shows the execution of the *shortView* behavior scheme of a book instance, providing the book's title, creator and publisher along with the thumbnail of the book's first page. In general, a behavior scheme supplies the calling service with access to a subset of the digital object instance's runtime structures, offering an effective service-neutral view of the encapsulated digital object information. The service can then process the data held in these structures according to the particular service provision requirements – i.e., generate a web page, feed a web service or create an XML document.

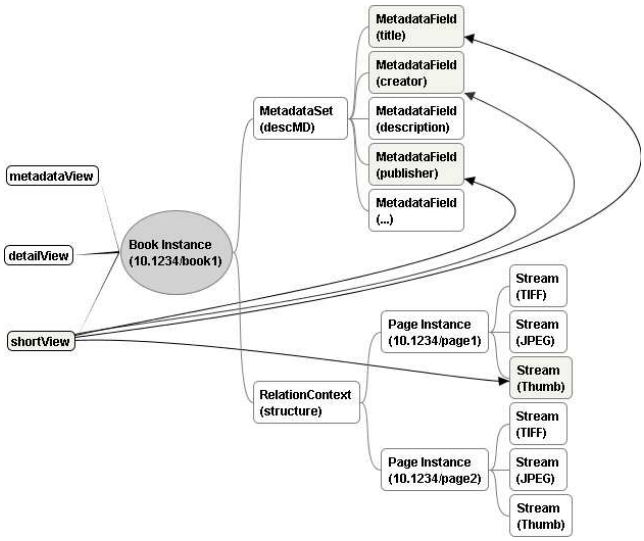

Fig. 4. An example of a behavior scheme

Behavior schemes are (a) defined in the context of a digital object's prototype, allowing the DL designer to provide the views that such a type of objects will generate at

runtime, (b) attached to digital object instances at runtime, during the prototype-based instantiation process. In this way, we enable digital object behavior to retain storage independence by being part of the DLMS unified model. To this effect, DL designers model digital object behavior independently of any physical/storage details of the objects held beneath. Furthermore, such storage independence permits objects to maintain their type-specific views when moved between different DO stores.

4 Summary

In this paper we presented our proposal to realize a unified runtime layer for the DLMS using a domain-specific "language" of network classes and/or types of digital objects. We have also discussed how our approach can be used as the foundation for making the DLMS a reusable integral component for realizing any DL.

References

1. Candela, L., Castelli, D., Pagano, P., Thanos, C., Ioannidis, Y., Koutrika, G., Ross, S., Schek, H.J., Schuldt, H.: Setting the Foundations of Digital Libraries: The DELOS Manifesto. D-Lib Magazine **13**(3/4) (March/April 2007) [doi:10.1045/march2007-castelli].
2. Hudak, P.: Building domain-specific embedded languages. ACM Computing Surveys **28**(4es) (1996)
3. M. Mernik and J. Heering and A.M. Sloane: When and how to develop domain-specific languages. ACM Computing Surveys **37**(4) (2005) 316–344
4. R. Kahn and R. Wilensky: A Framework for Distributed Digital Object Services. International Journal on Digital Libraries **6**(2) (2006) 115–123
5. Arms, W.Y., Blanchi, C., Overly, E.A.: An architecture for information in digital libraries. D-Lib Magazine **3**(2) (February 1997)
6. Gonçalves, M., Fox, E., Watson, L., Kipp, N.: Streams, Structures, Spaces, Scenarios, Societies (5s): A Formal Model for Digital Libraries. ACM Transactions on Information Systems (TOIS) **22**(2) (2004) 270–312
7. Consultative Committee for Space Data Systems (CCSDS): Reference Model for an Open Archival Information System (OAIS) Blue Book, Issue 1, `http://public.ccsds.org/publications/archive/650x0b1.pdf`.
8. Nelson, M.L., Maly, K., Zubair, M., Shen, S.N.T.: SODA: Smart Objects, Dumb Archives. In: ECDL '99: Proceedings of the 3^{rd} European Conference on Digital Libraries. (1999) 453–464
9. de Sompel, H.V., Bekaert, J., Liu, X., Balakireva, L., Schwander, T.: aDORe: A Modular, Standards-Based Digital Object Repository. The Computer Journal **48**(5) (2005) 514–535
10. T. Staples and R. Wayland and S. Payette: The Fedora Project: An Open-source Digital Object Repository Management System. D-Lib Magazine **9**(4) (April 2003)
11. Borgida, A., Mylopoulos, J., Wong, H.K.T.: Generalization/Specialization as a Basis for Software Specification. In: On Conceptual Modelling: Perspectives From Artificial Intelligence, Databases and Programming Languages, Springer Verlag (1982) 87–117
12. Mattos, N.M.: Abstraction concepts: The basis for data and knowledge modeling. In: Proceedings of the Seventh International Conference on Entity-Relationship Approach, North-Holland Publishing Co. (1989) 473–492
13. Taivalsaari, A.: On the notion of inheritance. ACM Computing Surveys **28**(3) (1996) 438–479

14. Saidis, K., Delis, A.: Type-consistent Digital Objects. D-Lib Magazine **13**(5/6) (May/June 2007) [doi:10.1045/may2007-saidis].
15. Saidis K.: Digital Object Prototypes Framework `http://www.dops-framework.net`.
16. Parnas, D.: On the criteria to be used in decomposing systems into modules. Communications of the ACM **15**(12) (1972) 1053–1058