Typing OpenDLib Repository Service: Strengths of an Information Object Type Language

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

Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo" – CNR
Via G. Moruzzi, 1 - 56124 PISA - Italy
{leonardo.candela, donatella.castelli, paolo.manghi, pasquale.pagano}@isti.cnr.it

End-users perceive a Digital Library (DL) as a set of *Functionality* ($\mathcal{F}_{DL}$) operating over the digital objects of an *Information Space* ($\mathcal{I}_{DL}$). At this level of abstraction, objects represent entities of the end-users application domain; for example, *conference proceedings*, i.e. collections of article objects. The structure of an $\mathcal{I}_{DL}$ is typically given in terms of high-level modeling primitives, e.g. a combination of *classes* of objects, each class describing domain-specific structure, namely *properties*, *constraints*, and *behavior*, of the *objects* it contains.

DL designers main task is the definition of an appropriate structure for the $\mathcal{I}_{DL}$. For example, assume the scientists of the *European Space Agency* (ESA) work on an Earth observation project, aimed at observing and measuring environmental modifications of some planet sites. To this aim, scientists need to store a chronological history of site *observations*, each consisting of a satellite `avi` movie and sensor-gathered data in `txt` format. Given a site, the time of its next observation is established by elaborating the sensor-data of the last site observation. Scientists need to organize observations into site *investigations* and must be able to access the history of the observations relative to a site. The structure of an $\mathcal{I}_{DL}$ for a DL supporting such investigation may include a class of *investigation* objects, each representing the history of observations relative to a planet site, and a class of *observation* objects, each representing satellite observations, i.e. pairs of an `avi` file and `txt` file. The association between an investigation object and one of its observation objects is characterized by a *version* number, which establishes the position of the former in the history of site observations. A minimal $\mathcal{F}_{DL}$ would feature an ingest interface, to handle investigation and observation objects, and a search interface, to retrieve observations by site or by version number through an investigation. Inserting an observation object requires uploading the two files and specifying both the associated investigation object and its version number.

DL's $\mathcal{I}_{DL}$ and $\mathcal{F}_{DL}$ are usually realized as software *components* extending and/or customizing the functionality supported by a DL *Repository Service* (RS). An RS maintains a *Repository Information Space* ($\mathcal{I}_R$), i.e. a storage unit for digital objects, and provides the primitives for *storing*, *accessing*, and *searching* digital objects into the $\mathcal{I}_R$ [8] [5] [3]. Intuitively, DL developers would implement their $\mathcal{F}_{DL}$ in terms of RS primitives and represent objects in $\mathcal{I}_{DL}$ with persistent objects in $\mathcal{I}_R$. Unfortunately, unlike $\mathcal{I}_{DL}$'s, $\mathcal{I}_R$'s are generally not organized into sets of objects with the same user-defined structure. Typically, an $\mathcal{I}_R$ is a flat space of digital objects, all structured according to the *Digital Object Model* (DOM) of the RS. DOMs describe objects as general purpose storage units, whose structure and relationships with other objects can be flexibly tailored to describe "any" possible DL high-level entity.

Due to this modeling "misalignment", $\mathcal{I}_{DL}$'s and $\mathcal{F}_{DL}$ can hardly be directly "interpreted", thus implemented, in terms of $\mathcal{I}_R$'s objects and primitives. As a consequence, DL developers "emulate" $\mathcal{I}_{DL}$ abstractions outside of $\mathcal{I}_R$ boundaries, by "embedding" high-level modeling primitives into the engineering of components, and rely on $\mathcal{I}_R$ only for storage issues. Specifically, components will allocate and manage a number of local data structures in order to emulate the notion of classes, objects conforming to a class, and relationships between objects. For example, the ESA DL user interface (UI) component would present the scientists an $\mathcal{I}_{DL}$-based information space. To this aim, the UI would rely on an RS to store digital objects, and then (*i*) implement *investigations* and *observations* classes as independent containers of objects; (*ii*) support the notion of objects conforming to such classes (e.g. by implementing versioning of observation objects w.r.t. investigation objects), by performing structural controls at object insertion/retrieval time (e.g. uploading should be allowed only for `avi` and `txt` files) and implementing class methods in terms of $\mathcal{I}_R$ primitives; and (*iii*) implement the relationship between investigation and observation objects.

The problem of such scenario is that the $\mathcal{I}_R$ alone is "unaware" of its "real" content. $\mathcal{I}_{DL}$ structure is not stored along with the corresponding digital objects, but is instead encoded into the components together with its relationship with the underlying $\mathcal{I}_R$. Accordingly, components are hard to develop, maintain, and integrate with others as their implementation is inspired by a logical $\mathcal{I}_{DL}$, and only $\mathcal{I}_R$ primitives are visible to them. Indeed, no automatic tool prevents $\mathcal{I}_{DL}$ structural programming errors, thereby leaving data consistency up to developers skills and precision. For example, consistency of multiple views with respect to an $\mathcal{I}_{DL}$ cannot be automatically checked: components handling objects of the same class may mistakingly provide diverse interpretation of the $\mathcal{I}_{DL}$ structure; the same holds for components handling objects of the same class according to different interpretations (subclasses).

We believe DLs should move one step forward, toward the realization of RS supporting the creation of domain-specific $\mathcal{I}_R$'s, directly matching $\mathcal{I}_{DL}$ designers intuition. Based on the experience of developing DLs serving different application areas, the *OpenDLib* project [4] is experimenting the realization of a new RS based on T-DoMDL [1][2][6], a rich and flexible DOM. T-DoMDL defines a type language whose constructs can be combined to define typed-sets of objects, i.e. collections of digital objects with homogeneous user-defined structure. Each typed-set of objects defines a different $\mathcal{I}_R$, storing objects with type-specific and user-specified properties and behavior.

T-DoMDL type language offers an exhaustive list of type abstractions, identified as common $\mathcal{I}_{DL}$ classes in the field of DL design: *raw*, *aggregation*, *relation*, *version*, and others. For example, all objects of an $\mathcal{I}_R$ T of type *version* have the property *list of versions*, where each version in the list is identified by a version number and features a user-defined label and an object reference. Objects in T have primitives for inserting/deleting versions into/from the list, getting the last version, and others; and share the behaviors, if any, declared in T by the DL designer. DL designers first devise their $\mathcal{I}_{DL}$ directly in terms of one or more types of T-DoMDL, then issue a request to the repository for the instantiation of the corresponding $\mathcal{I}_R$'s, i.e. a list of domain specific typed-sets. For the ESA DL, the designer may consider the following $\mathcal{I}_R$ typed-sets: *MovieMaps* and *Measurements*, both of type *raw* with format `avi` and `txt` respectively, *Observations* of type *aggregation* referring to two objects in *MovieMaps* and *Measurements*, and *Investigations* of type *version* referring to objects in *Observations*.

In OpenDLib, the new RS can thus store sets of typed objects and enable their management only through type-specific primitives. As a consequence, the RS guarantees data consistency and facilitates component development. In a further investigation, we are exploring how typed $\mathcal{I}_R$'s can be used for the automatic generation of components and for the optimization of storage facilities: for example, from the $\mathcal{I}_R$ *Investigations* of type *version*, the RS may be able to automatically generate a user interface for the management of the relative objects, including observations versioning management; from the $\mathcal{I}_R$ *MovieMaps* of type *raw*, which defines precise `avi` format constraints, the RS could configure a storage unit that optimizes disk space, by adopting *avi* compression techniques, and access performance, by providing streaming server primitives.

To our knowledge, only Saidis et Al.[7] addressed typing for DLs. In their approach they introduced the notion of *prototypes*. Informally, prototypes are user-configurable components which virtualize the notion of class of objects. They reside on top of an $\mathcal{I}_R$ and extend it in order to provide high-level class primitives to DL components developers.

References

1. L. Candela, D. Castelli, P. Manghi, and P. Pagano. A Typed Repository for OpenDLib. ISTI - CNR Technical Report 2006-TR-08, 2006.
2. L. Candela, D. Castelli, P. Pagano, and M. Simi. From Heterogeneous Information Spaces to Virtual Documents. In E. A. Fox, E. J. Neuhold, P. Premsmit, and V. Wuwongse, editors, *Digital Libraries: Implementing Strategies and Sharing Experiences, 8th International Conference on Asian Digital Libraries, ICADL 2005*, Lecture Notes in Computer Science, pages 11–22, Bangkok, Thailand, December 2005. Springer.
3. D. Castelli and P. Pagano. OpenDLib: A Digital Library Service System. In M. Agosti and C. Thanos, editors, *6th European Conference on Research and Advanced Technology for Digital Libraries, ECDL 2002*, Lecture Notes in Computer Science, pages 292–308, Rome, Italy, September 2002. Springer-Verlag.
4. D. Castelli and P. Pagano. A System for Building Expandable Digital Libraries. In *ACM/IEEE 2003 Joint Conference on Digital Libraries (JCDL 2003)*, pages 335–345. Springer-Verlag, 2003.
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. OpenDLib. A Digital Library Service System. `http://www.opendlib.com/`.
7. K. Saidis, G. Pyrounakis, and M. Nikolaidou. On the Effective Manipulation of Digital Objects: A Prototype-Based Instantiation Approach. In *Research and Advanced Technology for Digital Libraries: 9th European Conference, ECDL 2005, Vienna, Austria, September 18-23, 2005. Proceedings*, 2005.
8. R. Tansley, M. Bass, and M. Smith. DSpace as an Open Archival Information System: Current Status and Future Directions. In T. Koch and I. Sølvberg, editors, *Research and Advanced Technology for Digital Libraries, 7th European Conference, ECDL 2003, Trondheim, Norway, August 17-22, 2003, Proceedings*, Lecture Notes in Computer Science, pages 446–460. Springer-Verlag, 2003.