D-Lib IST-2001-32587
Digital Library Competence Center

Open Access to DLs Testbed – Report D2.1.1

Deliverable Type: REPORT
Number: D2.1.1
Nature: Public
Contractual Date of Delivery: month 12
Actual Date of Delivery :month 19
Task WP2.1

Name of responsible: Costantino Thanos, ISTI-CNR

Authors: Donatella Castelli, Pasquale Pagano, Manuele Simi, ISTI-CNR

contact info
donatella.castelli@isti.cnr.it

Abstract

This report details the organization of the course on the Open Access to DLs Testbed and the manuals written for and distributed at the course itself. The complete documentation has been made available to the public through the D-Lib Center web site (http://dlibcenter.iei.pi.cnr.it/).

Executive Summary

The aim of the course on the open access to DLs testbed is twofold:

- · to illustrate the possibilities provided by “new generation” digital libraries, in terms of new types of content and new functionality;
- to teach how to create and maintain a (new generation) digital library.

The Scholnet digital library service system (http://www.ercim.org/scholnet), developed within a project funded by the IST Programme of the European Commission, will be used as a testbed to demonstrate the concepts introduced during the course. Scholnet is an open digital library system that can be customised to meet the specific needs of diverse user communities. In its basic version, Scholnet provides services to support:

· acquisition, description, archiving, search, access, and dissemination of multimedia, multilingual, structured digital documents
· virtual organization of the information space
· handling of annotations on documents
· multilingual access
· personalised information dissemination

These features render it capable of supporting new forms of remote scholarly communication. In particular, they enable the dissemination of not only conference papers, technical reports, project deliverables, but also annotated seminars, lectures, demos, etc.

The structure of the course is the following:

- Tutorial (part 1) – Methodology for building a digital library
- Introduction to Scholnet and demo
- Training on how to use a Scholnet digital library
- Tutorial (part 2)- Methodology for digital library set-up and maintenance
- Case studies

System documentation is attached to the present document and was also made available to the participants.

Teaching Staff:
Donatella Castelli (castelli@iei.pi.cnr.it)
Pasquale Pagano (pagano@iei.pi.cnr.it)
Technical Staff:
Carlo Carlesi

Courses on the open access to DLs testbed were held in February 19-20, 2003 and in May 15-16, 2003 and will be repeated in the Autumn 2003.

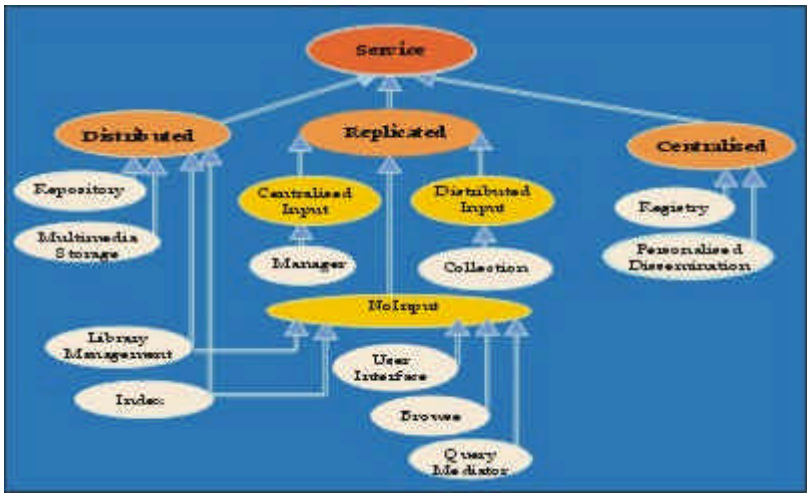
Table of Contents

OpenDLib: a Digital Library Service System

Architecture and Services

OpenDLib is a software toolkit that can be used to create a digital library easily, according to the requirements of a given user community, by instantiating the software appropriately and then either loading or harvesting the content to be managed. OpenDLib consists of a federation of services that implement the digital library functionality making few assumptions about the nature of the documents to be stored and disseminated. If necessary, the system can be extended with other services to meet particular needs.

OpenDLib was built as a distributed digital library, according to the notion of individually defined services located anywhere on the Internet. When combined, these services constitute a digital library. The functionality of the OpenDLib digital library includes the storage of and access to multimedia and multilingual resources, cross-language search and browsing, user registration and



personalized information dissemination of new incoming documents. The OpenDLib federation of services will communicate through an established protocol.

The figure shows a conceptual model that specifies the notion of "OpenDLib service". The ovals represent classes of services, the double arrows the specialization relationship. A OpenDLib architecture consists of a set of instances of the leaf classes (service types) of this model.

This model is central to the digital library system design since it highlights the properties (descriptive metadata) that define each service. Each service instance is known by the other instances through the values of these properties, which are disseminated on demand. These properties are modeled using structured attributes that are associated with each entity of the model. As a natural consequence of the relation between entities, a child entity inherits all attributes of its parent entity. The whole set of attributes characterize an instance of a service in the OpenDLib architecture.

A generic service of the OpenDLib architecture is modeled by the entity Service. A service can be distributed over different servers, replicated, or if necessary centralized. The model has an entity for each service type.

The **distributed services** are those that implement the same service through multiple instances, each of which manages data stored on a different server. The data are distributed according to a set of criteria that may differ from service to service. Note that each instance of a distributed service does not need to know anything about the other instances of the same service. In the version of OpenDLib that we are illustrating, the services that are distributed are those that maintain a huge amount of data and/or those that are strictly related to the document publishing institutions. These institutions usually prefer to maintain their own documents on their own server to have a physical control over them. Moreover, each institution usually has its own rules for document submission/withdrawal, or content management, and therefore prefers to maintain these procedures also in a common shared environment.

The **replicated services** are those implemented by a set of service instances, possibly located on different servers, where each instance is able to cover completely the service functionality over the entire set of data. OpenDLib distinguishes three kinds of replicated services: NoInput, CentralisedInput, or DistributedInput. A service is of NoInput type if it is instantiated by pure replications, i.e. the different instances are never distinguishable since they handle the same data and behave in the same way. A service of CentralisedInput or DistributedInput type has one replication, which acts as a master, and a set of replicates which act as slaves. In the case of the CentralisedInput, the master is a special instance of the service whose only purpose is to maintain and distribute on demand an updated version of the information handled by the service. The slave instances update their content by periodically invoking the master. Both the master and slave replicates of a DistributedInput service can accept new information and serve information requests. The master maintains the global state of the service information: each time a slave updates its local information; the slave communicates the change to the master, which merges the new information with its own information. Periodically, each slave updates its state by invoking the master. The role of the instances of a replicated service, i.e. master and slave, is not statically assigned but can be changed in order to achieve a better connectivity or to overcome temporary crash. In the present version of OpenDLib we have chosen to replicate those services that are either not constrained by any proprietary (see distributed services), security or privacy constraint (see centralized). This replication makes it possible to improve service efficiency and to increase its robustness. This is, for example, the case of the replication of indexes to improve content access, or the replication of meta information to improve digital library service access, and so on.

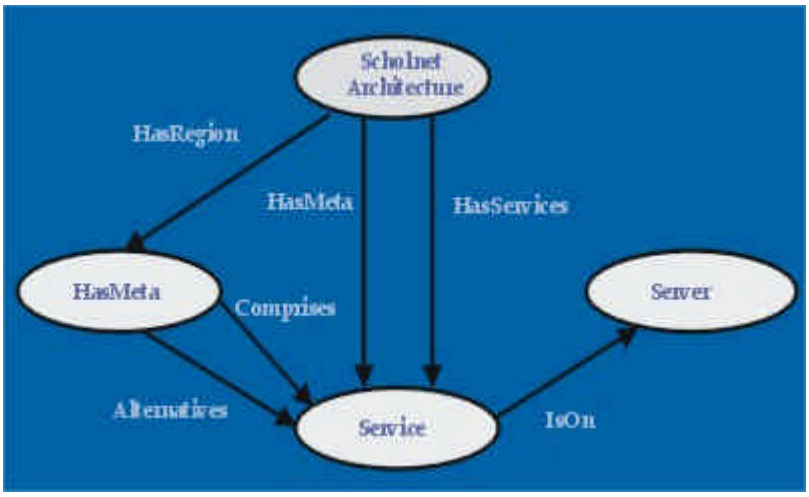
A **centralized service** has always a single instance in the digital library. Each time the security and the privacy of the content of a service is an issue, the centralized solution is preferred to the distributed and replicated ones.

Although the presence of multiple instances of a service increases fault tolerance, reduces the overload of each instance, and makes it possible to dynamically reorganize the environment when a server hosting a service instance is not reachable, the replication and distribution of the services is not mandatory and therefore each of the services outlined in Figure can be instantiated as a single instance. This means that the level of distribution and replication, and the physical location of the service instances may be freely chosen to better satisfy the needs of the specific digital library context.

The services provided in a OpenDLib digital library are instantiations of the listed service types. Each service has thus all the attributes declared for its type.

The federation is instantiated when the digital library is created and can change dynamically during the digital library lifetime. For example, a new server may be added or removed, a Query Mediator can send its search requests to a different Index; a service instance can change its role from master to slave, etc.

An OpenDLib digital library architecture is described by the OpenDLib model. The OpenDLib Model specifies the legal configurations of the OpenDLib digital library system, i.e. the digital libraries that can be created with this system, and their possible evolutions.



An OpenDLib digital library architecture model uses three concepts: Service (and its specializations, as introduced in the previous section), Server, and Region. A Server is a network device that is able to provide services to the network users by managing shared resources. It can host different service instance types. A Region is an abstract notion which stands for a dynamic set of service instances which cover the complete functionality of the digital library and which represent the optimal choice with respect to some set of optimization criteria, e.g. their mutual connectivity. A region thus consists of the entire set of centralized and distributed service instances and a set of instances of the replicated services, one for each service type. For each region, the set of replicated service instances changes over time in order to always implement the best choice, e.g. according to the state of the connection.

Any configuration that agrees with this model is a legal configuration of an OpenDLib digital library system, i.e. an OpenDLib digital library.

The Service entity and its specializations have been described above. Note that, in order to simplify the figure, we have not included the whole service hierarchy. Each of the Server and Region entities has an attribute, Address and Name, respectively, which identifies it. Specific relationships link a service instance with the server that hosts it (IsOn), and a region with both its current service instances (Comprises) and with the possible alternative replicated service instances (HasAlternatives). This relationship indicates the service instances that can be used to replace the current ones when the optimization criteria are not met anymore. For each region, more than one alternative of the same service type can be indicated. A priority value is associated with each pair (region, service instance) as a measure of the quality of participation of the service in the region with respect to the set of optimization criteria selected. The service with the highest priority belongs to the region. Note that the same service instance can belongs to, and be an alternative in, more than one region. This means that the number of replications can be chosen freely and is not constrained by the number of established regions.

The OpenDLib Architecture entity models a digital library architecture created by instantiating the OpenDLib system. Each digital library has a name and participates in three relationships, which specify the digital library functionality and distribution. The relation HasService expresses the composition of the federation, i.e. the service instances that working together provide the digital library functionality; HasRegion models the organization of the instances into a cluster of services

that satisfy optimization criteria; HasMeta identifies the services that control of the architecture and are responsible over time for its consistency. Note that the only constraint on the number of participants in a relationship is that they must be sufficient to cover the digital library functionality. This means that OpenDLib digital library architecture is completely flexible in terms of the number of instances, regions and servers. These can be freely chosen when the digital library is created and they can be modified dynamically during the digital library lifetime. Note also that the part of the OpenDLib model depicted in the figure does not impose any constraint on the type of services implemented by the federation. In presenting the model we have assumed the service types implemented by the current version of OpenDLib, however other choices are equally supported.

The entities introduced above are subjected to a number of rules that must be satisfied in any state of any OpenDLib digital library architecture. Two rules are reported below as an illustrative example.
Rule 1:
The Query Mediator service is parametric with respect to the type of search supported, the query language, the metadata formats, and the format of the returned result set. All the replicated Query Mediator instances usable in a region must select the same search types, result set formats, and the same sub-set of the metadata formats that are handled by the Repository service instances. As a consequence of this, different Query Mediator instances can only offer different search types, or return different result set formats only if they belongs to different regions;
Rule 2:
The Index service is distributed and is parametric with respect to the distribution criteria, to the retrieval engine supported, to the metadata formats, etc. Clearly an Index Service instance has to manage a metadata format that is used by the whole set of Repository instances identified by its distribution criteria. Nevertheless, as it is replicated, all Index instances that are usable in a region must refer to the same result set formats, and to the same indexed fields. For example, if an Index Service instance indexes documents in Italian, it will use a stop-word list for Italian, and Italian stemming rules, As a consequence, only an Index Service instance with the same configuration can be set-up as its replication.

The functionality of each service type is the following:

- Repository - stores documents that conform to the DoMDL document model. It can be distributed on multiple servers because a Repository server can store documents published by different authorities and different authorities can be hosted by different Repository servers.

- Multimedia Storage - stores video documents (according to the DoMDL document model), and supports their dissemination either as whole documents or as aggregations of scenes, shots and frames. It can be distributed on multiple servers because a Multimedia Storage server can store documents published by different authorities and different authorities can be hosted by different Multimedia Storage servers.

- Library Management - supports the submission, withdrawal, and replacement of documents. It can be distributed on multiple servers because a Library Management server can manage documents of different authorities published in different Repositories and different authorities can be managed by different Library Management servers. It is replicated because the same authorities can be managed by different LibMgt servers.

- Index - accepts queries and returns lists of document identifiers matching those queries. It can be distributed and replicated on multiple servers because an Index server can index documents published by different authorities stored in different Repositories and because the document

published by different authorities can be indexed by different Index servers.

- Query Mediator - dispatches queries to appropriate index servers. It can be replicated on multiple servers.
- Browse - supports the construction of browsing indexes and the actual browsing of those indexes on library contents. It can be replicated on multiple servers.
- Registry - supports the storage and access of information about authors, individual users, and user groups. Hosted by one server.
- Personalised Dissemination - supports the storage and execution of persistent queries, subscriptions, on a collection. Hosted by one server.
- Regional Manager Service (Meta) maintains the information pertaining to the regions. It is replicated on multiple servers, one for each region.
- Collection Service - provides a virtual organisation of the documents stored in the repositories. It supplies the information necessary to manage these virtual document aggregations. This information is used by the other services in order to handle the collection objects allowing, for example, the Query Mediator to perform a query on a specified collection, or the Browse to perform a browse on a collection, and so on. It can be replicated on multiple servers.

Each type of service has descriptive attributes that specify its content. The table below lists the specific attributes associated with each service type:

Service type	Specific Attributes
Repository	contentDescription, authorities, sets, metadataFormats, documentStructure
Library Management	authority,set,documentStructure, metadataFormat
Index	metadataFormat, indexedFields, resultFormats, language
Multimedia Storage	compositeDocumentFormats, nonCompositeDocumentFormats, multimediaDocumentFormats
Query Mediator	searchMethods, resultFormats
Regional Meta Service	serviceDescriptionFormat
Collection	collectionMetadataFormat
Browse	metadataFormat, browsableFields, resultFormats
Registry	userProfileFormat, groupProfileFormat
Personalized Dissemination	topicProfileFormat

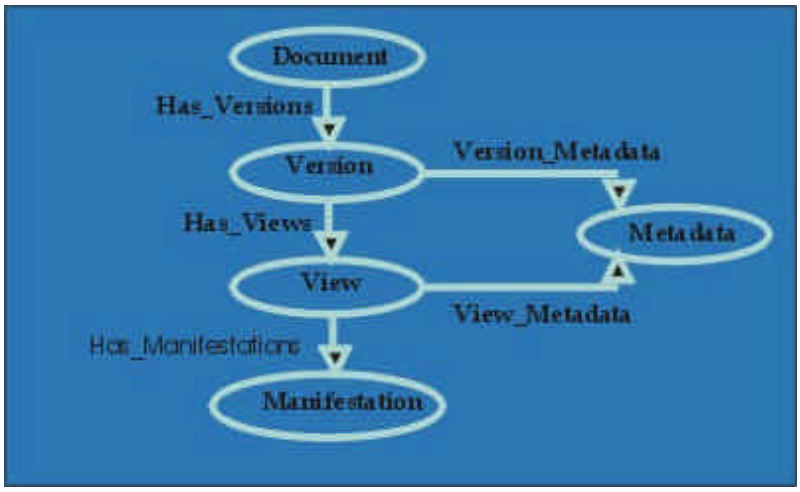
The Manager Service covers a special role because always maintains updated information about the OpenDLib federation status. This service is completely parametric with respect to the type and number of managed services. The only prerequisite for these services is that they must be able to respond to an established set of OLP protocol requests. The information maintained by the Manager is partially provided by the digital library administrator when he/she instantiates the digital library service system; the rest is collected by the Manager Service by periodically sending appropriate protocol requests to the service instances.

Each service instance responds to the Manager by providing information about its state, structure, and functionality. A service instance, in turn, may ask the Manager for information about the location and characteristics of any other instance of the federation. This exchange of information between the Manager and the other services makes it possible to create co-operating services that have no built-in knowledge of the location of their instances. It also permits a dynamic allocation of the service instances according to the needs and availability of the servers in the architecture. Note that this co-operation is more complex than a simple client-server application. A service can act both as a provider and as a consumer, and sharing relationships can exist a priori among any subset of the services. Services, in fact, may be combined to support different functionality, and the same services may be used in different ways, depending on the restrictions placed on the sharing and the goal of the sharing.

The Document Model for Digital Libraries

The structure of the documents that populate the information space of different digital libraries is usually different. For example, a digital library of conference proceedings may contain documents (the proceedings) that are aggregates of other documents (the preface and the articles). Each article, may be disseminated in different ways, for example it can be disseminated both as a text in postscript format (the readable content of the article) and as a videos in MPEG3 format (the speaker presentation). A digital library of project deliverables is likely to have documents with a completely different structure. For example, these may be textual reports, structured into sections, and demos of the project prototypes. Different digital libraries may also support different metadata formats. For example, a MARC format can be used in a digital library of journals built by library professionals but will not be certainly used in a digital library of geographical information or in a digital library of self-published documents. A digital library service system that wants to be generic with respect to the underlying information space must be able to support different organizations of the information space, and the storage and dissemination of a wide variety of document types and descriptive metadata formats. OpenDLib was designed to satisfy this requirement. In particular, it was designed to support a very powerful document model, called **DoMDL**, that can be customized at the start-up of the system to represent the specific structure of the application documents and their descriptive metadata formats.

DoMDL distinguishes four aspects of document modeling, each of them is seen as a different kind of entity: Document, Version, View, and Manifestation.

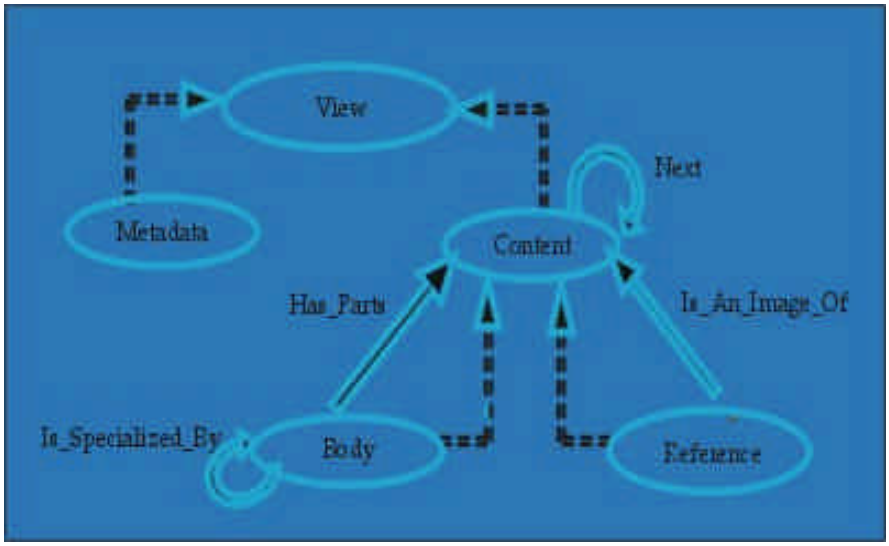


Entity

A **Document** entity represents the more general aspect of a document, i.e. the document as a distinct intellectual creation. For example, the article "The SOMLib Digital Library System" by Andreas Rauber and Dieter Mekl, the book "Digital Libraries and Electronic Publishing" by William Arms, the lecture "A dynamic Warehouse for XML Data of the Web" by Serge Abiteboul, the proceeding of the conference ECDL'99, can all be modeled as entities Document. Each entity of this type is named by a URN (Unified Resource Name). Unlike a URL, a URN is location independent.

A **Version** represents a specific edition of the distinct intellectual creation, i.e. an instance along the time dimension. The preliminary version of the paper "The SOMLib Digital Library System", the version submitted to the conference ECDL'99, the version published in the ECDL'99 proceeding, are examples of successive editions of the same document. Versions are linear and numbered. The first version is version 1, subsequent versions are 2, 3, etc.

A **View** models a specific intellectual expression of an edition of a document. A view excludes physical aspects that are not integral to the intellectual perception of a document. A document edition is perceived through one or more views. For example, the original version of the ECDL'99 proceedings might be disseminated under four different views:
a) a "structured textual view" containing a "Preface" created by the conference Chair, and the list of papers presented to the conference
b) a "textual view structured into thematic sessions", where each session contains the documents presented during that session
c) a "presentation view", containing the list of the conference presentation slides
d) a "metadata view", containing a structured description of the proceeding.
The entity View is specialized in two sub-entities: *Metadata* and *Content*. The former view perceives a version through the conceptualizations given by its metadata representations. These can be a flat list of pairs (fields, values), as with the Dublin Core metadata records, or more complex conceptual structures, such as IFLA-FRBR descriptions. Typically this is the view indexed to support fielded querying and browsing, but it can have other uses. For example, it may be the view disseminated free of charge in an environment in which documents are covered by access rights, or the view disseminated on a mobile device.
The Content is the view of the document content. It has two sub-entities: Body, and Reference. Each of them has a different representational semantics characterized by different constraints and relationships (see Figure).



The Body entity represents a view of the document content that can be either perceived as a whole or as the aggregation of other views. For example, the ECDL'02 proceedings can have a textual view that is built as the aggregation of the textual views of its components article. Similarly, the "black and white (B&W) view" of the "Herald Tribune" newspaper can be modeled as the aggregation of a certain number of "B&W page views". Each of these pages, in turn, can be represented as the aggregation of particular "B&W" views of articles and pictures, each authored by a different person, and possibly, published elsewhere. The relation Has_Parts maintains the link between a Body view and its components views. If the parts are ordered, i.e. they document is a sequence of parts, then two consecutive parts are linked by the Next relation.
A body view can be specialized, i.e. there may be other views that represent more detailed perceptions of the same content. For example, the view "slides of the Andreas Rauber presentation" might be specialized by: "B&W view of the slides", and by "Colored view of the slides". A view is related to all its specializations though the relation Is_Specialised_By.

A Reference entity represents a view that, being equal to an already registered one, does not need to be explicitly stored. It has been introduced in the model to avoid duplication of information and it has no conceptual counterpart. This view can be used, for example, when a new version of a composite document is submitted. All the views that have not been changed with respect to the previous version can be specified as reference and linked to the original through the relation Is_An_Image_Of. There is thus no need to load and store them again. As a consequence of its definition, this type of view has no manifestation and no assigned explicit property, except the link to the existing view. This link is sufficient to derive all the information that is required to retrieve and disseminate the complete description of the view modeled as reference.
Note that Body and Reference do not partition the Content view. For example, the views Summary and Keyframe are content views that do not belong to any of these subclasses.

A **Manifestation** models the physical formats under which a document is disseminated. Examples of manifestations are: the MPEG file which maintains the video recording of the presentation made by Andreas Rauber at the ECDL '99 conference, the AVI file of the same video, the poscript file of the paper presented by Andreas Rauber at the ECDL '99, etc.

Relation

The relations Has_Versions, Has_Views, and Has_Manifestations link the different aspects of a document. Note that these relations are multiple, i.e. there can be several objects in the range associated with the same object in the domain. This means that there can be multiple versions of the

same documents, multiple views of the same version and multiple manifestations of the same view. Versions and views may be described by one or more metadata records in different formats. This records are accessible through the relations Version_Metadata and View_Metadata. Each of the entities described above has a set of attributes, which model properties of the corresponding aspect of the document. For space reasons we cannot list them here. We only want to mention that each entity has an attribute that specifies the rights on the modeled document aspect. This makes it possible, for example, to model possibly different rights on different editions, or different rights on different parts of the same view, and so on. By combining the rich document model with the possibility of having multiple metadata descriptions for each aspect, the DoMDL document model provides a very powerful representational mechanism that can satisfy the needs of many different information spaces. This is one of the key aspects that enables the Repository to serve different application contexts. In the next section we will describe the other specific feature of the Repository Service that make it widely applicable.

Communication Protocol

Communication with and among individual OpenDLib services takes place via the OpenDLib Protocol (OLP). The OpenDLib Protocol is an evolution of the Dienst protocol. It inherits from Dienst the basic rules and conventions, and many protocol requests. This section illustrates the protocol rules inherited from Dienst. A detailed description of the protocol requests that are served by each single service is given in the next sections.

Verbs and Versions

OLP protocol requests are called "verbs". Each service supports a set of verbs.
A service may support more than one version of a verb, and each version may differ in syntax or semantics. A version takes the form of two integers, separated by a period. This version applies to the individual verb, not the protocol as a whole. Including a version number in the message allows for backward-compatible extensions to the OLP system.
A server offering one or more OLP services might support verbs in various versions. A service receiving a message with a version number which is previous to the current one must reply either using the pertinent syntax and semantics, or with an error. If a service receives a message with a newer version number, then it must return an error.
Software supporting the OLP protocol may or may not be versioned. If a software version number exists, that number is independent of the OLP protocol verbs and versions of those verbs that the software supports. OLP protocol requests are expressed as URLs embedded in HTTP requests . Except where noted, these are all HTTP GET requests. A typical implementation uses a standard Web server, such as Apache, that is configured to dispatch OLP URLs to the appropriate OLP service. The remainder of this section describes the aspects of the protocol that are specific to the HTTP embedding.

Message Format

All messages are encoded in URLs where the path portion of the URL consists of the following tokens, in the following order:

- OLP: This token appears literally in the URL.
- Service Name: The name of the service that can handle the message, e.g. Repository.
- Version: The version of the verb being invoked.
- Verb: This is the name of the message, e.g. Structure. A verb is unique within a Service.

- Fixed arguments: Each verb can have a certain number of fixed arguments, which must always be supplied, and must appear in the order cited.
- Fixed_post arguments: Each verb can have a certain number of fixed arguments, which must always be passed as HTTP POST request. The fixed_post syntax arguments take the form key=value.
- Optional arguments: the optional syntax arguments take the form key=value. If there is more than one optional argument, they are separated by an ampersand. Arguments may appear in any order. Unless otherwise specified, optional arguments are always optional and need not be repeated.
- Optional_post arguments: the optional_post syntax arguments take the form key=value. Arguments may appear in any order and must always be passed as an HTTP POST request. Unless otherwise specified, optional_post arguments are always optional and need not be repeated.

The separator between tokens in the path is the slash, except for the separator before the Optional arguments, which is a question mark.

Example: If the Repository Service implemented the Structure verb, and if version 1 of that verb accepted one fixed argument and two optional optional arguments (version and view), then an example request is: /OLP/Repository/1.0/Structure/handlecorp/docid?version=2%26view=book. The full URL for this request at a particular Web server might be: http://xx/OLP/Repository/1.0/Structure/handlecorp/docid?version=2%26view=book

Special Characters

The syntax rules for URIs give special roles to a few characters in certain contexts. If these characters are used in any other way they must be written as an escape sequence: a percent sign followed by the character code in hexadecimal. The special characters are:

- "/" - Path Component Separator: %2F
- "?" - Query Component Separator: %3F
- "#" - Fragment Identifier: %23
- "=" - Name/Value Separator: %3D
- "&" - Argument Separator in Query Component: %26
- ":" - Host Port Separator: %3A
- ";" - Authority/Set Namespace Separator: %3B
- " " - Authority/Set List Separator: %20

Message Responses

Responses to messages are formatted as HTTP responses, with appropriate HTTP header fields. The return type specified for each message in this document will, therefore, be the MIME type included in the HTTP Content-Type header field (if a wrapper is applied to the content, the Content-Type will correspond to the type of the wrapper). Likewise, any encoding applied to the content will be

included in the HTTP Content-Encoding header field. Responses to OLP protocol requests vary among the following MIME types:

1. "text/plain" is used for responses that contain unstructured information.
2. "text/xml" is used for responses that contain structured information (such as the verb requesting the internal structure of a digital object). This document lists the DTD (Document Type Definition) for every verb that returns a text/xml response. All XML responses to OLP protocol requests have the following uniform features. The first tag output is an XML declaration where the version is always 1.0 and the encoding is always UTF-8. The remaining content is enclosed in a root element that has the same name as the verb of the respective request. The element has a single attribute named version, which has a value that is the version of the verb of the respective request. For example, a Disseminate verb with version 2.0 will produce text/xml content with a tag like
3. "text/html" is used in response to user interface service requests (that are intended for rendering by a browser).

Content specific types such as application/postscript and image/gif are reserved for disseminations from digital objects.

Status Codes

Status codes and error returns correspond to those defined for HTTP (see the following list). A normal response from an OLP message in HTTP is signaled with the 200 reply code. Error returns are signaled with the appropriate 4xx or 5xx code as specified in the HTTP protocol. The use of HTTP error codes is as follows:

- "400" - if the OLP request is malformed; for example, illegal arguments or the values of arguments are invalid.
- "401" - if the client is unauthorized to make the request.
- “402” – if the user is not authorized
- "404" - if a document specified in a OLP repository request is not in the repository.
- "415" - if the format, encoding, or binder requested for a document is not available or cannot be generated.
- "501" - if the OLP service, verb, or version is not supported by this server.
- "503" - if the server is able to automatically generate a requested content type, but does not have a copy on hand, and does not wish to keep the connection open while it is generated, it should return the HTTP status code 503 (Service Unavailable), along with a suitable Retry-After header.

For each error return, the HTTP reason-phrase returned with the code should provide additional useful information to a human reader.

Date

All dates in the protocol requests and responses are encoded using the "Complete date" variant of ISO8601. This format is CCYY-MM-DD where CC is the century, YY is the year, MM is the

month of the year between 01 (January) and 12 (December), and DD is the day of the month between 01 and 31.

Service Information Verbs

Each OpenDLib service implements three verbs that provide information about the service to the rest of the architectural components: Identify, ListVerbs and DescribeVerb. These are termed Service Information verbs.
Identify returns the name of the service and other specific information about the service.
ListVerbs lists the name of the verbs defined by that service.
DescribeVerb takes as input a verb of the service and returns a description of it.
These verbs have been introduced in order to allow each service to "introduce itself" to other services that may want to use it. They are intended as a way of supporting the openness of the architecture and the reusability of the single architectural components.