

GDFR

Registry Node

System Design

PROJECT NUMBER: 538

By: Andreas Stanescu

Date: December 17, 2006

Non-Disclosure Statement

This document has been prepared by OCLC Online Computer Library Center, Inc. and the information contained herein is proprietary and confidential. OCLC reserves all rights to the document and its contents and any reproduction, republication, or other unauthorized use thereof, in whole or in part, without the express written consent of OCLC is prohibited. This document may be copied in full with this notice for internal use only for OCLC's intended purpose.

1.0 Product Purpose

The registry node is the basic service component needed by the Global Digital Format Registry. The design is based on the IWSA-Reg implementation and the SRU server implementation from the Office of Research.

For a complete overview of the system purpose, objectives and requirements please see the GDFR grant proposal and the system analysis and architecture. Furthermore, additional requirements were developed by the Registry Framework project.

The goals for this component are:

1. Develop a registry component capable of hosting multiple record collections.
2. Ensure that widely-known standard protocols are made available to clients.

3. Develop a method and protocol for distributing record and synchronizing record updates in a peer-to-peer network.
4. Develop a method and protocol for auditing record holdings in the peer-to-peer network.
5. Ensure that the implementation is capable of deployment arbitrary record collections easily, efficiently and with minimal operator intervention.

2.0 Product Design Constraints

The component could be widely used to host new record collections of varying schemas, hence the need for minimal operator or human intervention.

The current Service Level Agreement will not be affected.

Reusing existing components, frameworks and protocols is highly desirable.

3.0 High Level Design

The components devised here perform the following actions:

- 3.1 Search, retrieve and reformat collection records using a standard naming scheme.
- 3.2 Add and update collection records.
- 3.3 Export, harvest, distribute and audit collections.
- 3.4 Create, update and deploy new collections, based solely on the schema of the records managed by the collection.

4.0 Detailed Design

The model below describes the interfaces and the services built by this project, as well as their relationship to other components - in this case the IWSA front-end services and the SRU server.

Registry Node System Design v0.1

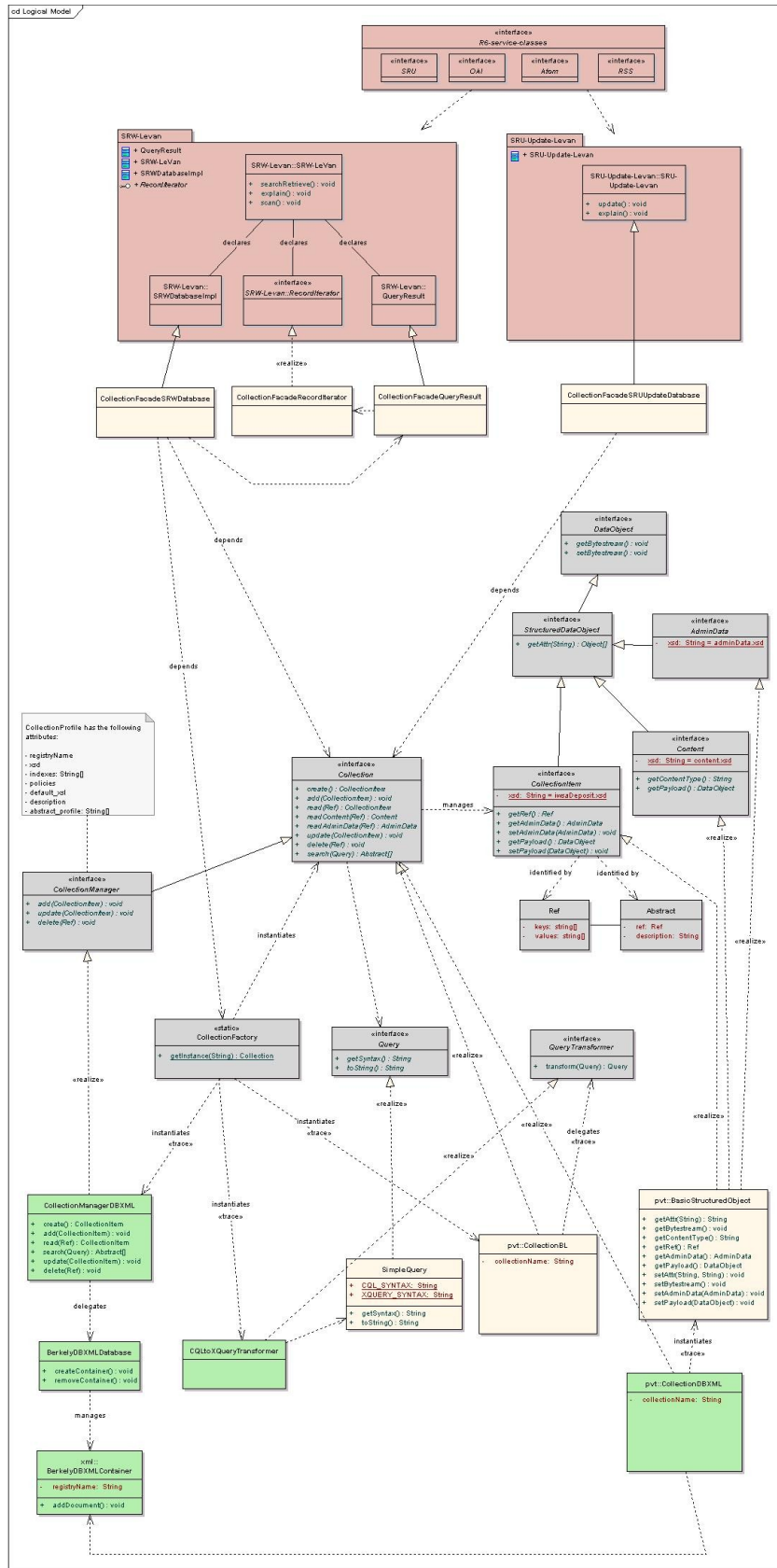


Figure 1: Design Model

The packages shown in deep red are generic representation of existing components, developed by Office of Research: IWSA services, SRU/SRW server and SRU Update server.

The classes shown in gray comprise a set of interfaces capable of abstracting away any implementing database, including new or existing ones, or remote services needing a façade to take advantage of the standard IWSA front-end services. In essence, by implementing the 6 methods (create, add, read, update, delete, search) in the Collection interface and the query translation method (translate) in the QueryTranslator interface, a new database implementation can gain all the upper level capabilities.

The classes shown in yellow represent the classes implemented in this design. Just under the SRW server, the few classes are built to plug into the SRW server and delegate calls to implementations of Collection interface. Just under the Collection interface, those classes are used to hold query requests and implement necessary data validation business logic, regardless of database implementation.

The classes shown in green represent a particular database technology implementation. This design uses Berkeley DB XML database to implement all the capabilities described above and implements a CQL-to-XQuery translator capable of searching the stored XML documents. Note that SQL database can be plugged in, but the capability to create collections on the fly may be hampered by the need for a database administrator to design and deploy relational tables holding “shredded” XML attributes.

Design paradigm

The pattern used here is called Entity-Manager pattern. The concept is very easy to grasp: Entities are managed by an EntityManager. Entities are uniquely identified by a Ref. An Abstract is an extension of the Ref, also uniquely identifying the object, but additionally containing a simple description needed to quickly describe the Entity in a list.

Anything needed to be done with an Entity, such as creating, reading, updating, is done to the entire Entity and it is solely controlled by the EntityManager. This way, the business rules governing data in the object reside in only one place (the Entity Manager), while the Entity is responsible for only one thing: to hold and transport attributes of the object.

Manager methods

Method	Arguments	Return Type	Notes
create	n/a	<i>CollectionItem</i>	A StructuredDataObject of type <i>CollectionItem</i> , the type of every item, in any collection. The identifier is either generated by the system or left blank for the user to fill it in. This method implements the Factory pattern, allowing servers to vary the implementation of the class. No persistent object is actually created, only an instance needed to transport data to and from clients and servers.
add	<i>CollectionItem</i>	void	Add a new item to the collection, but only if the validation is successful and another item with the same identifier isn't found in the database.
update	<i>CollectionItem</i>	void	Update an existing item in the collection, but only if the validation is successful and an item with the same identifier is found in the database. Different server implementations can implement versioned backends or simple databases.
readAdmin	Ref	<i>AdminData</i>	Read the admin data of a given collection item, as identified by the key held in the Ref.
readContent	Ref	<i>Content</i>	Read the content of a given collection item, as identified by the key held in the Ref.
read	Ref	<i>CollectionItem</i>	Read both admin data and content (in other words, the whole collection record), as identified by the key held in the Ref.
delete	Ref	void	Delete the collection record identified by the key held in the Ref.
search	Query	<i>Abstract[]</i>	Search the collection. The Query is written in CQL. The return is a list of <i>Abstract(s)</i> – the key of the collection item and a brief

			description – for all records matching the query.
--	--	--	---

ContentItem attributes

Attribute	Type	Notes
adminData	<i>AdminData</i>	A DataObject of type AdminData containing data about the collection item: identifier(s), author, content type, URI, version. Additionally, an extension has been defined by GDFR: provenance (defined using OAI provenance schema and SAML assertion schema).
Content	<i>Bytes</i>	Base64 encoded content stream. Anything can be packaged in this fashion.

The schemas for these data objects are defined elsewhere. TODO: add link here.

Berkeley DB XML implementation.

The technology choice in this design allows dynamic creation of collections, including storage of data, indexing of terms and searching, simply by providing the XSD, or the schema, of the records held in a collection. One collection contains only one type of item – to create complex arrangements between objects, data from different collections is combined to implement such data requirements.

While searching is defined in terms of CQL queries, this implementation offers a class to map the CQL query into XPath, therefore automatically generating search queries across the XML documents. Again, no human intervention is necessary since the query is generated by using only the XSD schema of the collection item.

“Collections” collection.

To facilitate the dynamic creation and discovery of collections, a special collection named “Collections” will hold XML documents describing each collection. This document is named Collection Profile. It contains the name of the collection, the permanent identifier, the indexes defined by the users, the XSD schema for the items in the collection, etc.

To create or update a collection, the exact same interface is used, where the document being modified is the Collection Profile. However, the current implementation using the Berkeley DB XML has the added capability of creating the document database and its necessary indexes of the fly. No database administrator is needed, no special rights are

needed, no human intervention is needed. As soon as the XML database is created and the Collection Profile is saved, items can be stored in the new collection immediately.

This standard collection allows a standard implementation of SRU specific capabilities such as records per page, persistent result sets, creation of the Explain response, etc. Once this layer is implemented, new database implementations (or façades to existing systems) only need to implement the 6 methods in the Collection interface and the CQL query translator. All business rules and standard interface protocols will automatically work.

5.0 Data

- N/A

6.0 Open Issues

- N/A

7.0 Definitions

- None

8.0 References

- Business Plan: The GDFR grant proposal serves as the Business Plan for this activity.
- Grant contract.
- Analysis document.
- Data model design.
- Architecture document.
- IWSA and IWSA-Ref interface definitions.
- AdminData and Content XSD schemas.