Learning Object Repository
Interoperability
Framework
# Table of Contents

1. Introduction to Learning Object Repository Interoperability ............................................ 1
2. Query Service ..................................................................................................................... 3
3. Authentication and Session Management .......................................................................... 4
4. Related Work ...................................................................................................................... 4
5. Open Work Items .............................................................................................................. 6
References .................................................................................................................................. 7
About this Document

Title: Learning Object Repositories Interoperability Framework

Editor: Bernd Simon (bernd.simon@wu-wien.ac.at)
David Massart (david.massart@eun.org)
Frans Van Assche (frans.van.assche@eun.org)
Stefaan Ternier (stefaan.ternier@cs.kuleuven.ac.be)
Erik Duval (erik.duval@cs.kuleuven.ac.be)

Status: Public Draft

Version (Date): 1.0 alpha (2004-12-30)

Summary: This paper introduces an interoperability framework for Learning Object Repositories.

Document Location: http://www.prolearn-project.org/lori

Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Minor textual improvements.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>First version created.</td>
</tr>
</tbody>
</table>
Acknowledgements

This work is partly sponsored by the CEN/ISSS Workshop on Learning Technologies. We also would like to acknowledge contributions from the following initiatives: Ariadne, Celebrate, Educanext, Edutella, Elena, EduSource, ProLearn, Universal, Zing.

List of Contributors

Bernd Simon, Vienna University of Economics and Business Administration & EducaNext
Christian Werner, Learning Lab Lower Saxony
Erik Duval, Katholieke Universiteit Leuven & Ariadne
Daniel Olmedilla, Learning Lab Lower Saxony
Dan Rehak, Carnegie Mellon University
David Massart, European Schoolnet
Frans Van Assche, European Schoolnet
Griff Richards, Simon Fraser University
Gerhard Müller, IMC
Julien Tane, Universität Karlsruhe
Marek Hatala, Simon Fraser University
Matthew J. Dovey, Oxford University
Michel Arnaud, University of Paris X & AFNOR
Peter Dolog, Learning Lab Lower Saxony
Stefaan Ternier, Katholieke Universiteit Leuven
Stefano Ceri, Politecnico Milano
Stefan Brantner, BearingPoint Infonova
Simos Retalis, University of Piraeus
Theo van Veen, Koninklijke Bibliotheek
Zoltán Miklós, Vienna University of Economics and Business Administration

Context

Currently, there are three SQI related documents:

i. “Learning Object Repository Interoperability Framework”: this document provides the “big picture” and should probably be read first if you want to understand the context of this work;

ii. “Simple Query Interface Specification”: this is the core of the specification for querying learning object repositories in an interoperable way;

iii. “Authentication and Session Management”: this document focuses on specific issues related to authentication and session management.
1 Introduction to Learning Object Repository Interoperability

The Web puts a huge number of learning resources within reach of anyone with Internet access. However, many valuable resources are difficult to find in an efficient manner, because resources are either not clearly identified as learning resources or lack a sufficient description in order to make them reusable in learning situations. Sometimes, valuable resources are hidden in the closed and proprietary worlds of learning (content) management systems, streaming media servers and online collaboration tools.

Such systems are commonly referred to as learning object repositories. Learning object repositories, sometimes also called “repositories for learning”, hold information on learning objects (i.e., metadata), in order to describe educational artefacts such as courses, online tutorials, lecture notes, electronic textbooks, tutoring sessions, quizzes, etc. Frequently, these systems allow user access only through proprietary interfaces and data formats, which hinders re-use and quality improvements through collaborative authoring and exchange.

In brief, there is lack of interoperability. Interoperability can be defined as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged” [1]. To a user, the lack of interoperability means:

- Applications and their data are isolated
- Redundant data entry is required.

By contrast, interoperability ensures that data is entered only once in one application, and that it can automatically be propagated to other applications. Such a scenario requires a common semantic model, so that data from repository A can also be interpreted by repository B. Additionally, interoperable systems are based on common protocols, which control the interactions in order to make repositories exchange data effectively.

To achieve interoperability, different kinds of protocols can be used. The Learning Object Repository Interoperability Framework presented in Figure 1 distinguishes between core services and application services. Core services are needed, for example, to agree on a common procedure for uniquely identifying learning objects [1]. Other core services are related with authenticating users and repositories, or with creating and managing sessions for interaction between applications. Typical applications that make learning repositories interoperable are the provision service and the query service, for example. The provision service – also referred to as replication service – allows repository A to forward metadata about a learning object or a metadata update to repository B. The query service allows repository A to query repository B for suitable learning resources. A contracting service assigns access rights to a learning object stored at a remote repository. The delivery service interacts with the repository where the learning resource is stored and delivers an electronic learning resource to the end user.

Application services make use of core services. For example, an identifier service might be used in a query or session management may be required before metadata about a learning resource is replicated between repository A and repository B using the provision service.
Both core and application services, require a common messaging infrastructure, which enables repositories to interact. XML RPC, Java RMI, or WSDL/SOAP are examples of such messaging services. A messaging service is based on a common network infrastructure and lower level protocols such as TCP/IP, HTTP, etc. Figure 2 depicts the various layers of Learning Object Repository Interoperability as described above.

Learning Management Systems (LMS), Learning Content Management Systems (LCMS), Knowledge Pools, or Brokerage Platforms are the kind of information technology the interfaces proposed herein are designed for. With its current focus on the query interface, this document targets architects of educational networks, managers of learning resource repositories, stakeholders in learning object re-use, as well as researches in web services and system interoperability.

This document is the first of three documents investigating the issue of learning object repository interoperability. The other two documents are Authentication and Session Management and the Simple Query Interface Specification.

The remainder of this document is structured as follows: While Section 2 gives an introduction to the Query Service, Section 3 is devoted to Authentication and Session Management. Section 4 reviews related work.
2 Query Service

The Query Service is the focus of a separate document, called “Simple Query Interface Specification”. In order to enable finding resources, a query interface is needed. Although we refer to learning repositories interoperability with a special focus on learning object metadata, this query interface can also be used within other domains and application scenarios (A first discussion on this issue is provided in the Open Work Items Section of the Simple Query Interface Specification).

An Application Program Interface (API) for query services needs to specify a number of methods a repository can make available in order to receive and answer queries from other applications. Since one design objective of the API is to keep the specification simple and easy to implement, the API is labelled Simple Query Interface (SQI).

In our scenarios, learning repositories can be equal peers, where each repository is able to submit queries to the other. To distinguish the requestor from the answering system, the term “source” is introduced in order to label a system which issues a search (the source of the query). The term “target” labels the system which holds the metadata queried (the target of the query). Alternatively, the “source” can also be referred to as “requestor” and “target” as “provider”.

Metadata can be stored using different means, such as file-based repositories, (distributed) relational databases, XML repositories, or RDF tool kits, which are based on heterogeneous query and modification languages. In order to make learning repositories interoperable, not only a common interface needs to be defined, but also a common query language together with a common results format for learning object descriptions needs to be agreed on. Neither of these latter two issues are currently within scope for the work documented here.

The query service is used to send a query in the common query language to the target. Then, the query results, represented in the common results format, are transported via the query service to the source. On the implementation level, wrappers may need to be built to convert a query from a common query language X to a local query language Y and transform the query and the query results from a proprietary format to a common one and vice-versa.

Figure 3 illustrates an exchange process, where Learning Repository A (the source) submits a query to Learning Repository B (the target). It is assumed that both systems have agreed upon a common query language beforehand. The concepts used in the query statement are part of a common schema. At Repository B, the interface component might need to transfer the query from the common query language to the local one. Also some mappings from the common to the proprietary schema might be required before submitting the search. Once the search has yielded results, the results-set is forwarded to the source, formatted according to the common results format.

Figure 3: Communication between two Simple Query Interface Components
3 Authentication and Session Management

The services for authentication and session management are the focus of a separate document, called “Authentication and Session Management”. The application interfaces make abstraction from authentication and access control issues. However, there is a need to authenticate the source in order to allow a target, for example, to link query policies to a source repository. For instance:

- Repository A is allowed to query Repository B without any limitations,
- Repository C is only allowed to retrieve 1000 query results per day from Repository D at a maximum.

Ideally, authentication is performed only once for a series of interactions. To accomplish this, a session token needs to be returned after successful authentication that can be used to identify the system in the subsequent communication.

Session management needs to be understood as a higher-layer management of configuration settings and authentication. The session ID serves as a mandatory element in the application interfaces in order to identify the requestor/source in all query commands.

4 Related Work

OpenURL [2] as well as the Content Object Repository Discovery and Resolution Architecture (CORDRA) [1] are initiatives that investigate the “Identifying” problem of Figure 1. The work on SQI is “orthogonal” to this, in that queries and results can refer to identifiers of arbitrary nature.

Z39.50-International: Next Generation (ZING) covers a number of initiatives by Z39.50 implementers to make Z39.50 [6, 9] more broadly available and to make Z39.50 more attractive to information providers, developers, vendors, and users. SRW is the Search/Retrieve Web Service protocol, which is developed within ZING and aims to integrate access to various networked resources, and to promote interoperability between distributed databases, by providing a common utilization framework. SRW is a web-service-based protocol [10]. SRW takes advantage of CQL ("Common Query Language"), a powerful query language, which is a human-readable-string query-representation.

SRW has many similarities with the Simple Query Interface Specification, but also some differences. SRW is purely synchronous (source-initiated), i.e. query results are returned with the response. Additional query results can be retrieved later from the results set stored at the target for a pre-defined amount of time. SRW foresees a mapping to Web Services and URL encoding (the later is called SRU). SQI currently does not support something similar. SRW encourages the use of Dublin Core, but is in general schema neutral (like SQI). SRW packs everything in few functions and does not adhere to the “Command-Query separation principle”. Hence, SRW cannot be considered as easily extensible. SRW does not provide hooks for authentication and access control nor is it based on a session management concept. SRW defines an Explain operation, allowing a client to easily discover the capabilities and facilities available at a particular server. SRW uses a rich set of XML-encoded application level diagnostics for reporting errors. SQI uses method exceptions.

The purpose of the IMS Digital Repository Interoperability (DRI) Specification [5] is to provide recommendations for the interoperation of the most common repository functions.
The DRI specification presents five core commands, i.e. search/expose, gather/expose, alert/expose, submit/store, and request/deliver, on a highly abstract level. The specification leaves many design choices for implementers. For example, while recommending Z39.50 (with its own query language) it also recommends XQuery as a query language. Hence, DRI is not query language neutral. The query service does distinguish between asynchronous and synchronous query mode.

The EduSource project [4] aims to implement a holistic approach to building a network for learning repositories. As part of its communication protocol - referred to as the EduSource Communication Language (ECL) -, the IMS Digital Repository Specification was bound and implemented. A gateway for connecting between EduSource and the NSDL initiative, as well as a federated search connecting EduSource, EdNA and Smete serve as a first showcase. EduSource opted for a document-style web service implementation of the Search and Expose method instead of an RPC SOAP-based approach. The document-style approach of Web services is used, which the authors claim to have advantages in terms of higher flexibility.

Google offers a web-service based API for retrieval [3]. Although a retrieval API has many similarities with a query API, some differences do exist. For example, no underlying, common data model does exist. The allowed parameters for the doGoogleSearch method are "key" (userid/password), "q" (query text), "start" (where to start returning in the results), "maxResults" (number of allowed results), "filter" (filter out very similar results), "restrict" (country or topic restrictions), "safeSearch" (pornography filter), "lr" (language restrictions), "ie" (input encoding) and "oe" (output encoding). Each parameter can be set via a dedicated method before the doGoogleSearch method is invoked (key and query must be set others are optional). Additionally the two methods doSpellingSuggestion and doGetCachedPage are supported. doSpellingSuggestion requests a spelling correction suggestion. doGetCachedPage requests a cached page from Google. The query results can be handled via the GoogleSearchResult class, which supports a wide variety of methods in the meantime (e.g. getEstimatedTotalResultsCount, getSearchComments, getSearchTime, getResultElements, etc.). Google does not support a semantic search based on schemas and query languages. Google’s move to web services is nicely discussed in [8].

OKI (Open Knowledge Initiative) is a development project for a flexible and open system to support on-line training on Internet [7]. OKI has issued specifications for a system architecture adapted to learning management functions. One of the main characteristics of the project is its commitment to the open source approach for software component development. OKI supplies specifications for a model of functional architecture and an API called Open Service Interface Definition (OSID). OKI OSID main aspects are:

- To supply specifications for a flexible and open source model of functional architecture
- Service Interface Definitions (SIDs) organize a hierarchy of packages, classes and agents and propose Java versions of these SIDs for use in Java-based systems and also as models for other object-oriented and service-based implementations.
- Components developed by OKI are compliant with specifications issued by IMS and ADL SCORM.
5 Open Work Items

- The requirements of the messaging layer (basis for API) shall be defined in the document, good example: Knowledge Query and Manipulation Language (KQML) specification (see http://www.cs.umbc.edu/kqml/)
- Core and Application Services other than the ones described above should be investigated in more detail.
- Implementation Guidelines for Building Wrappers can be another document, which might be beneficial for the Learning Object Repository Interoperability Framework.
- Related Work needs to be investigated in more detail. For example, what lessons can be learned from OKI OSID? Is KQML relevant in our context and what can be learned from it?
- Discuss: Investigate difference between provision and query. Why do we focus on query? Provide rational.
- Discuss: The API definition is spread out into two documents (at the moment). Does this give the implementers an overview of the work that has to be carried out at one glance?
References


