



Creating a Smart Space for Learning

Learning Management Network Specification –

Version 1.5

This document follows up the “System Interface Framework for Building Educational Brokerage Networks” document, Version 1.3.

You can download this document from:

<http://nm.wu-wien.ac.at/e-learning/interoperability/>

Deliverable Number:	D1.2b
Contractual Date of Delivery:	Feb 2004
Actual Date of Delivery:	08/03/2004
Work Package(s) Contributing:	WP1
Nature of the Deliverable:	Report
Status:	Final
Security (Distribution Level):	Public
Editor(s):	B. Simon (WUW), S. Brantner (INF)

The ELENA Consortium consists of:

Centre for Social Innovation (CSI)	Financial Coordinator	Austria
Wirtschaftsuniversität Wien (WUW)	Scientific Coordinator	Austria
Infonova (INF)	Contractor	Austria
National Centre for Scientific Research “Demokritos” (NCSR)	Contractor	Greece
Allweb Solutions S.A. (ALW)	Assistant Contractor	Greece
Institute “Jozef Stefan” (IJS)	Contractor	Slovenia
Universidad Politécnica de Madrid (UPM)	Contractor	Spain
Landssimi Island (LSI)	Contractor	Iceland
Universität Hannover (UHA)	Contractor	Germany
CDI Deutsche Private Akademie für Wirtschaft (CDI)	Contractor	Germany
imc information multimedia communication (IMC)	Contractor	Germany

Amendment History

Vers.	Date	Editor(s)	Modification
1.5	8.3.2004	Zoltán Miklós Bernd Simon Stefan Brantner	<ul style="list-style-type: none"> • Section 3.2.3: Changed paramter "resourceURL" to "resourceID" • Section 3.3: New section on session management • Section 3.5.3: <ul style="list-style-type: none"> - Changed parameters of "createUser" call: removed parameter "brokerInstitutionID", removed parameter "homepage" - New method "deleteUserAccount". This is necessary in order to avoid inconsistencies, when the broker user account is deleted for example. • Section 4.1.4: <ul style="list-style-type: none"> - Changed parameters of method "requestSynchronisation": Instead of "userID" and "password" now a "sessionID" must be provided. - New exception introduced: "AuthorisationFailedException" is thrown when the given session has not the right authorization to provide a new resource. - New exception introduced: "NoSuchSessionException" is thrown if the sessionID does not exist - Removed exception "NoSuchUserException" • Section 4.3.3: <ul style="list-style-type: none"> - Changed paramter "resourceURL" to "resourceID" for methods "createAuthorization", "withdrawAuthorization" and "isAuthorised". - Changed exception "AuthorizationDoesAlreadyExistException" of method "createAuthorization" to "NoSuchAuthorisationException". • Section 4.4 - Resource Delivery: This section has been reworked completely. <ul style="list-style-type: none"> - New method "getResourceAccessURL". This method must be called in order to get an access URL of a resource.

Vers.	Date	Editor(s)	Modification
			<ul style="list-style-type: none"> • Section 4.5 - Resource Management: This section has been reworked completely. - New upload process, that does NOT directly involve the broker anymore. Resources are directly uploaded to the delivery system. - Also a basic version management has been introduced for delivery systems, that support upload. - New methods "reportUploadedResource", "getPossibleStartFiles", "getStartFile", "setStartFile", "confirmUpload", "revokeUpload", "getVersions", "removeResource" and "removeVersionOfResource". • Appendix A: Conventions and Data Types Used: This appendix is not valid anymore and has been removed.
1.4	6.2.2004	Bernd Simon Stefan Brantner	<ul style="list-style-type: none"> - Simple Query Interface (SQI) added - Smart Space for Learning explained
1.3	1.4.2003	Bernd Simon Stefan Brantner	<ul style="list-style-type: none"> - Schema-neutral Provision Specifications - Provision Specification revised based on feedback from implementers - Incorporation of feedback of reviewers (Tomaz Klobucar and Joaquin Salvachua)
1.2	24.1.2003	Bernd Simon Stefan Brantner	<ul style="list-style-type: none"> - Query interface specified - Implementation layer removed - Data model for describing delivery systems revised - First WSDL Mapping
1.1	19.9.2002	Bernd Simon Stefan Brantner	<ul style="list-style-type: none"> - Management Summary introduced
1.0	12.9.2002	Bernd Simon Stefan Brantner	<ul style="list-style-type: none"> - First consolidated version created

Contributors

<u>Name</u>	<u>Institution</u>
Antoine Dubost	Campus of Europe
Antoine Kah	Campus of Europe
Arno Wagner	ETH Zürich
Bernd Kammlander	Wirtschaftsuniversität Wien
Bernd Simon	Wirtschaftsuniversität Wien
Effie Law	ETH Zürich
Joaquin Salvachua	Universidad Politécnica Madrid
Nikos Papazis	National Research Center Demokritos
Symeon Retalis	University of Cyprus
Stefan Brantner	Bearingpoint Infonova
Sven Kayser	IMC
Thomas Zillinger	Bearingpoint Infonova
Tomaz Klobucar	Institute “Jozef Stefan”
Erik Duval	Katholieke Universiteit Leuven
Daniel Olmedilla	Learning Lab Lower Saxony
David Massart	European Schoolnet
Frans Van Assche	European Schoolnet
Griff Richards	Simon Frasur University
Gerhard Müller	IMC
Marek Hatala	Simon Frasur University
Matthew J. Dovey	Oxford University
Peter Dolog	Learning Lab Lower Saxony
Stefaan Terrier	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

*“The whole power of science is the power of shared ideas,
not the power of hidden ideas.”*

Paul Jones

Foreword

Learning resources are increasingly stored in closed systems. While first web-based learning management environments were open to the public, off-the-shelf learning management systems and other on-line collaboration tools keep valuable knowledge within a restricted environment. This specification is inspired by the idea of carefully opening these systems for the purpose of building learning management networks.

A first working group has been founded within the UNIVERSAL Project (IST-1999-11747). This working group will continue its work not only in the context of the UNIVERSAL project, but will also reflect on the work performed in the projects Ten-A (Ten-Telecom Project, C27777) and ELENA (IST-2001-37264). Additionally, the working group has become open to any kind of new member interested in this subject matter. As a consequence Version 1.0 of this specification was presented at a meeting between dSpace, OKI, and UNIVERSAL held at the MIT (August 2002, Boston). A public invitation for joining this initiative was also made at a Cen/Isss Workshop Meeting (December 2002, Copenhagen). Since then one subsection of this specification, the simple query interface has drawn the attention of a wider audience and is now further developed as a CEN/ISSS activity. This activity is currently co-lead by Bernd Simon, Erik Duval and Frans Van Assche.

Vienna and Graz, 6/2/2004

Bernd Simon

Stefan Brantner

Management Summary

This specification introduces a system interface framework for building learning management networks. Learning Management Networks are understood as networks of interoperable educational nodes. One particular example of a learning management network this specification emphasizes on, is an educational brokerage network, which aims to make learning resources from heterogeneous sources available via one central node.

Stakeholders in education and training institutions are interested in supporting the re-use, distribution and dissemination of learning resources. One way of exchanging learning resources that they are announced at a central market place, where users (and systems) can place query requests. Besides announcing, brokers also have to support contracting and delivery of learning resources. Brokers are important building blocks of the Learning Management network; they often play the role of educational nodes in the network.

This specification introduces an Application Program Interface (API) for brokers for the exchange of learning resources, so that learning resources available via heterogeneous delivery systems can be distributed via a broker. Examples for delivery systems are simple web servers, streaming video servers, video conferencing tools, or learning management systems.

The API comprises an application layer and an administration layer. The administration layer introduces services for registering users at delivery systems and at the broker and presents a model for describing delivery systems. In addition a basic interface for inspecting delivery systems and the learning resources stored at them is introduced. In the application layer the following services are defined:

- Provision
- Learning Resource Management
- Querying
- Access Control
- Delivery

Provision covers the use case for synchronizing learning resource metadata descriptions between two systems (e.g. a broker and a learning management system). Learning resource management introduces a specification for remotely creating, modifying and deleting learning resources. Querying covers a query-language independent service for remotely searching a learning resource repository. Access control introduces commands for granting and withdrawing a user's access rights to a particular learning resource. Delivery is concerned with the presentation of the learning resource to the user.

WSDL Bindings of this specification can be downloaded from: <http://nm.wu-wien.ac.at/e-learning/interoperability/services.html>.

Special attention is given to the query interface in this specification. We introduce the so called Simple Query Interface (SQI) for learning repositories. The specification is a common effort of several leading providers and designers of learning repositories both from the commercial and academic area.

The latest version of the SQI interface specification is available at <http://nm.wu-wien.ac.at/e-learning/interoperability/sqi/sqi.pdf>.

List of Acronyms

API	Application Programme Interface
BSF	Bean Scripting Framework
CA	Certification Authority
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computer Environment
ebXML	Electronic Business Extended Markup Language
ECDM	Edutella Common Data Model
EJB	Enterprise Java Bean
HTTP	Hypertext Transfer Protocol
IPSec	Internet Protocol Security
LOM	Learning Objects Metadata
P2P	Peer-to-Peer
QEL	Query Exchange Language
RDF	Resource Description Framework
RFC	Request for Comment
RPC	Remote Procedure Calls
RQL	RDF Query Language
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQI	Simple Query Interface
SQL	Structured Query Language
SSL	Secure Socket Layer
UBP	Universal Brokerage Platform
ULRIF	Universal Learning Resource Interchange Format
URI	Uniform Resource Identifier
URP	User Registration Program
UTC	Coordinated Universal Time
UUID	Universal Unique Identifier
VSQI	Very Simple Query Interface
WSDL	Web Service Description Language
XML	eXtensible Markup Language

Table of Contents

1	The Corporate Learning Spaces Today.....	10
2	“Smart Spaces for Learning” Defined.....	12
3	Brokers in the Learning Management Network.....	14
4	Architectural Framework of Brokers.....	18
5	Educational Node Administration Services	20
5.1	System Registration	20
5.1.1	System registration functionality.....	20
5.1.2	System authentication.....	22
5.2	Inspection.....	22
5.2.1	Usage scenario.....	22
5.2.2	Functionality.....	23
5.2.3	Inspection Interface	23
5.3	Session Management	24
5.3.1	Functionality.....	24
5.3.2	Session Management Interface.....	25
5.4	Broker User Management.....	26
5.4.1	Usage scenario.....	26
5.4.2	Functionality.....	26
5.4.3	Broker User Management Interface	27
5.5	Delivery System User Management	28
5.5.1	Usage scenario.....	28
5.5.2	Functionality.....	28
5.5.3	Delivery System User Management Interface.....	28
6	Educational Node Application Services.....	31
6.1	Provision	31
6.1.1	Usage scenario.....	31
6.1.2	Functionality.....	31
6.1.3	Additional User Interface Requirements	32
6.1.4	Synchronisation Interface	33
6.1.5	Examples of Interface Calls.....	34
6.2	Querying	34
6.3	Access Control	35
6.3.1	Usage scenario.....	35
6.3.2	Functionality.....	35
6.3.3	Access Control Interface	35
6.4	Resource Delivery.....	37
6.4.1	Usage scenario.....	37
6.4.2	Functionality.....	37
6.4.3	Delivery Interface	37

6.5	Resource Management.....	38
6.5.1	Usage scenario.....	38
6.5.2	Functionality.....	38
6.5.3	Resource Management Interface.....	38
7	Simple Query Interface.....	43
7.1	Rational and Usage Scenario.....	43
7.2	Requirements and Scenarios.....	44
7.3	Functionality Overview.....	48
7.4	Query Interface Methods.....	49
	Query Configuration.....	49
	Synchronous Query Methods.....	51
	Asynchronous Query Methods.....	53
	Results Retrieval.....	54
7.5	Limitations.....	54
7.6	SQI Profile “Very Simple Query Interface”.....	55
7.7	Related Work.....	56
8	References.....	59

1 The Corporate Learning Spaces Today

Over the past few years, corporations have made significant progress in linking learning processes with the employee's work environment. Today's knowledge workers are served by Internet access through their desktop and mobile phone, business-unit specific knowledge repositories, e-learning tools, and customized education and training opportunities available through corporate intranets. Leading business organisations are offering its workforce a heterogeneous set of learning resources ranging from traditional seminars to knowledge management activities and e-learning content.

While such a sophisticated learning space creates competitive advantage by intellectually empowering a company's workforce, some shortcomings limit the benefits, mainly from the perspectives of decision effectiveness, process administration, and IT infrastructure management. The lack of interoperability of knowledge repositories, for instance, does not allow for a unique view on the learning services offered. As a result, a user's search costs increase and the transparency of learning resources offered is reduced with each repository added to the environment. However, such an environment not only lacks transparency in terms of learning service offerings, but also does not provide a customizable view of the learning processes undertaken by the work force. The latter constitutes important information for personnel developers and other mentors. In many cases, the electronic environments also lack decision and recommendation support. Neither potential learners nor their mentors have all the goal-driven business tools and information available to concisely select the right learning service for closing a particular knowledge gap. On the other hand, a series of wrong decisions (e.g. not taking a "required" learning service or registering for a "wrong" learning service) can have substantial impact on individual and corporate performance.

Until recently, setting up a corporate learning space consisting of monolithic components such as traditional course offerings, e-learning content (where appropriate), and knowledge management activities has been a major task in corporate work environments. However, this no longer seems to be the main concern. Companies are starting to focus on the integrated management of these heterogeneous components in what can be referred to as "Smart Spaces for Learning". Besides the integrate view on a company's human resources (HR) development process, institutions are now also selectively opening up there knowledge environments to incorporate also resources from other environments (e.g. book abstracts, courses offered through electronic market places, etc).

In Smart Spaces for Learning, semantic web technologies are used to provide enhanced, customizable and automated learning and administrative services. These include technologies such as the Resource Description Framework (RDF), the Query Exchange Language (QEL), TRIPLE, and ontologies that play a crucial role in achieving interoperability among repositories or recommending appropriate learning services. This paper reports on the ELENA project¹ and investigates and discusses how these

¹ <http://www.elena-project.org/>

technologies can be used to build systems like Smart Spaces for Learning. Smart Spaces for Learning are defined in Section 2, while Section 3 describes relevant design issues. Sections 4 and 5 respectively address two of the design issues mentioned: artefacts interoperability and personalisation. The paper concludes with a presentation of the ELENA Smart Space for Learning and discusses implications for the development of an Educational Semantic Web.

This document organized as follows. In Section 2 is the concept Smart Spaces for Learning explained. Section 3 und 4 concentrates on brokers in the Learning Management Network and their architecture. Section 5 describes the administration of an Educational Node. Section 6 explains the services of a Learning Management Network and Section 7 is devoted to a special service: the query services, and explains the Simple Query Interface (SQI) in detail.

2 “Smart Spaces for Learning” Defined

A Smart Space for Learning is a distributed system, which provides management support for the retrieval and consumption of heterogeneous learning resources. While "Space" is used as a synonym for "Network", "Smart" refers to the ‘intelligent’ mediation of learning resources (e.g. courses, e-learning content, etc) based on user profiles and artificial intelligence techniques.

Like any information system also a Smart Space for Learning consists of a human component and a technology component. Smart Spaces for Learning are built for supporting human resources development processes. Hence, learners, educators (e.g. teachers, instructors, trainers, professors, peers), and learning managers (e.g. parents, HR developers, team leaders) constitute the primary users of the system.

The two major technology components of Smart Spaces for Learning are the network of interconnected educational nodes (the Learning Management Network) and a Personal Learning Assistant (PLA), which provides a personalised access point to learning resources on the network (see Figure 1). The PLA supports learners in searching for, selecting, contracting with, and evaluating learning resources. It might also assess the learner’s pre-existing knowledge to better identify knowledge gaps and learning needs. By using personalisation techniques a PLA is capable of creating a personalised view of a Learning Management Network.

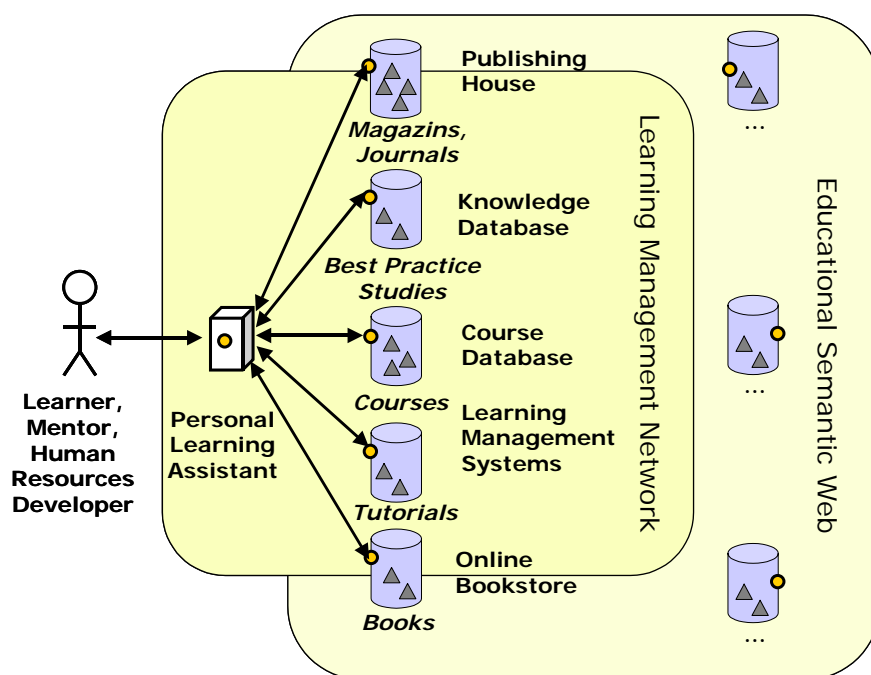


Figure 1: System components of a Smart Space for Learning

In a Learning Management Network, system interfaces provide means for exchanging information on educational artefacts such as courses, offer information and learner profiles. The information on educational artefacts (i.e. data on data) is commonly referred to as metadata and plays a crucial role for achieving interoperability among the

various educational nodes. A Learning Management Network is a “trusted” network in which users and systems are authenticated.

We envision learning management networks as sub-networks of a larger Educational Semantic Web – according to ELENA terminology also referred to as Artefacts and Service Network. The Educational Semantic Web facilitates the identification of educational nodes, both, in terms of network location as well as service types offered. The types of services offered comprise learning services and services that supplement learning services, which facilitate the preparation, generation, control, or evaluation of learning services. For example, a content brokerage service can be used for preparing the delivery of a course or for providing a learner with related information in a particular subject area. Assessment services can be used to identify knowledge gaps. Evaluation services provide information that helps to gauge the quality of a learning service. Reputation services attempt to quantify the reputation of a learning service provider within the network. Designers of Learning Management Networks can take advantage of the variety of educational services offered in the Educational Semantic Web by integrating external educational nodes into their Smart Space for Learning.

3 Brokers in the Learning Management Network

This section focuses on the use case of exchanging learning resources with a central broker as described below. The schemas used for describing the artifacts subject to change are not within the scope of this document.

Herein we describe interfaces required for building networks of educational nodes for the purpose of exchanging learning resources. The idea of building such learning management networks is about making learning resources delivered through dispersed delivery systems available via an educational broker.

Within the network the broker acts as a central system, providing facilities for managing exchange relationships between learning resources providers and consumers. As an educational mediator the broker combines, both, a collaboration facility for distributed educational activities and a market place for educational materials, in one system.

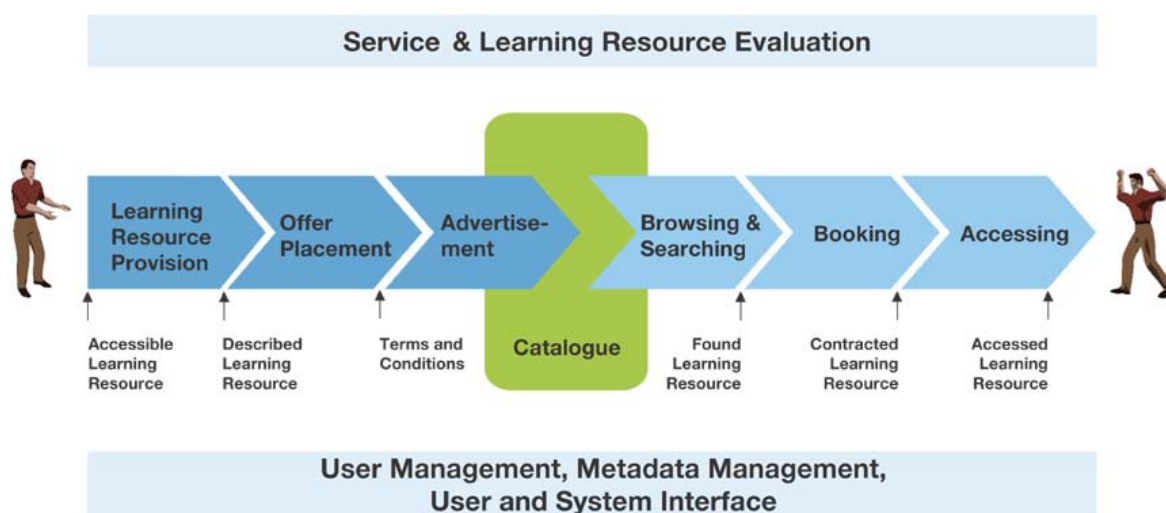


Figure 2: Exchange process supported by a broker

Users are able to provide learning resources to the broker and specify offers (i.e. usage terms), which consumers of those learning resources are asked to agree on before accessing the resource. Based on learning resource descriptions and usage conditions, learning resources are advertised through a catalogue and/or mailing lists. Users can choose and access learning resources from various kinds of delivery systems (e.g. video conferencing applications, learning management systems, streaming media servers, etc.), but have to agree on the offer terms before getting access to a resource. The process of agreeing on offer terms is referred to as “booking”. Figure 1 illustrates the exchange process as described above.

Although the broker provides its service via a central web site, the system architecture is based on a network of specialized educational nodes.

Table 1 gives an overview of the system types involved and the kind of service(s) they provide. The broker is an educational node interacting with various decentralized educational nodes, which hold educational material or provide educational activities. Within the learning management network the brokerage service is centralized at the broker, where as the content provision service is de-centralized and provided via the delivery systems.

Educational Node	Educational Service Provided
Broker	Holds descriptions of learning resources, provides booking exchange functionality via user interface
Basic Delivery System	Provides learning resources
Learning Resource Repository (Advanced Delivery System)	Provides text- and image-based educational material
Streaming Media Server (Advanced Delivery System)	Provides audio/video-based educational material
Video Conferencing System (Advanced Delivery System)	Provides educational activities via synchronous communication channels. Broker holds metadata of the educational activities provided
Learning Management System (Advanced Delivery System)	Provides educational material and educational activities via metadata replication, retrieves educational material from dispersed delivery systems
User Registration Program	Registers users at the broker
Reputation Service Provider	Provides reputation about learning services providers.

Table 1: System types and kinds of services offered

A delivery system can be a web server-based learning resource repository, a streaming media server, a video conferencing system or learning management system. The network can provide, both, static learning content (i.e. educational material in broker terms) and educational events with a specific time table and a virtual meeting place (i.e. educational activities in broker terms).

In order for the broker to announce, grant access and deliver learning resources stored at delivery systems, the delivery systems need to have interfaces providing services such as: learning resource provision, learning resource inspection, access control, delivery, etc.

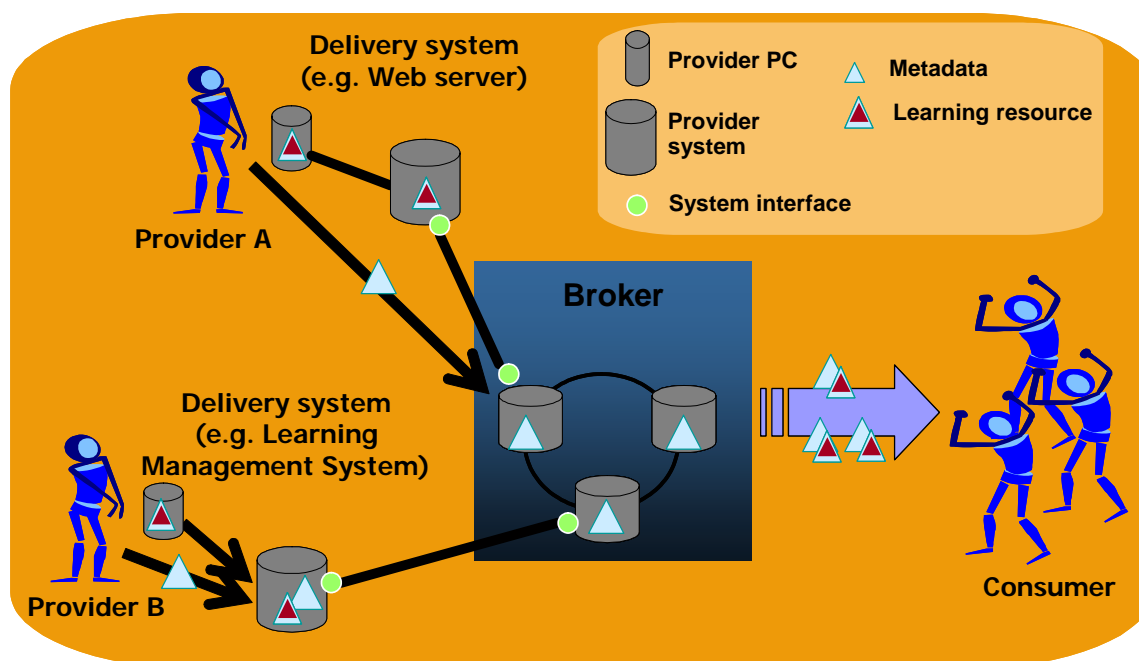


Figure 3: Learning Management Network with Broker

Figure 3 illustrates a learning management network based on a broker. In order for Provider A to share his set of lecture notes (i.e. educational material) via the broker, Provider A needs to log on to the broker, describe the learning resource at the broker and upload the material to a delivery system – in this case maybe a simple Apache web server running a delivery system application. If one of the consumers books the set of lecture notes the broker grants access rights to the consumer and delivers the learning resource.

The interface services required to realize this scenario are: learning resource management (upload of learning resource), delivery system user management (for the consumer account creation at the delivery system), access control (to add an access authorisation for the consumer), and delivery (to actually consume the learning resource).

Provider B is a user of a learning management system. She intends to announce her courses (i.e. an educational activity in terms of this document) at the broker to attract a higher target audience. Provider B uses a special functionality at the learning management system to forward the learning resource description to the broker. Once Provider B's course is booked a learner account is created for the consumer at the learning management system and the consumer is registered at the course. Once the course starts the consumer is notified by the learning management system via e-mail and directly accesses the learning resource there.

The interface services required to realize this scenario are: learning resource provision (forwarding of learning resource descriptions), delivery system user management (for the consumer account creation at the delivery system), and access control (to add an access authorisation for the consumer).

In the learning management network user accounts at the broker can also be registered de-centrally. User Registration Programs (URP) are third-party systems, which are empowered to register users at the broker.

4 Architectural Framework of Brokers

The framework is structured into two layers; *Application Layer* and *Administration Layer* (see Figure 4). The application layer contains a number of services facilitating the exchange of learning resources. The administration layer provides means for administering users and systems. These two layers can be encoded using various kinds of technologies, e.g. HTTP(S)-based URL-encoding or WSDL/SOAP transferred over HTTP(S)/SMTP.

The application layer provides application services along the exchange process presented in Figure 2. It relies on a “trusted” environment, where users and systems are authenticated by services provided by the administration layer.

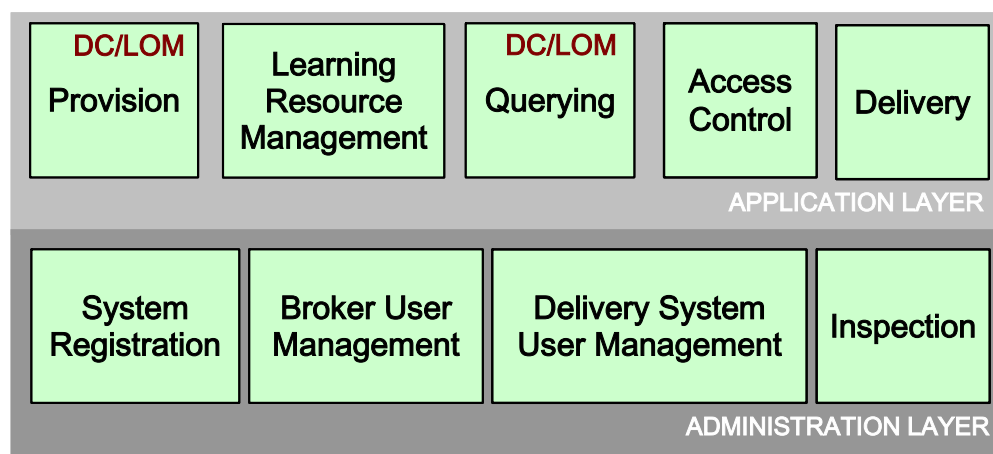


Figure 4: Interface framework

There is a *Provision* service, which replicates learning resource descriptions and offer information between the delivery systems and the broker. *Learning Resource Management* provides means for uploading an educational material to a delivery system. The *Inspection* service can be used to check the status (e.g. availability) of learning resources and systems. The *Querying* service provides an interface for realizing multiple, federated brokers. The *Access Control* service grants users access control and the *Access and Delivery* interface supports learning resource access via the broker. For provision and querying, metadata standards such as Dublin Core (DC) [11] and the IEEE Learning Objects Metadata (LOM) Standard [9] are relevant since both provide a uniform vocabulary, which can be used for describing and querying learning resources.

As there is no reasonable classification into server side and client side in the communication, the terms "broker-initiated communication" and "delivery-system initiated communication" will be used to specify which side is initiating a communication and which side is responding. The side that does not initiate the communication will be called the "target" of the communication.

In Section 3 and 4 administration and application interfaces are described in a variation of the (CORBA) Interface Definition Language. For encodings see Appendices.

5 Educational Node Administration Services

5.1 System Registration

Systems with all their administrative and technical characteristics must be registered at the broker. In order to register systems at the broker a user must have delivery system administrator or (brokerage) system administrator rights assigned. A system can be deactivated at the broker in the case it is (temporary) unavailable. As part of the system management process systems are authenticated as described below.

5.1.1 System registration functionality

Any system participating in the learning management network is registered at the broker. The broker stores detailed information about system characteristics as well as administrative data. All the information in the subsequent sections have to be provided before communication can take place between the broker and a system.

General

- *ID*: Each system has a unique ID. The ID is maintained by the broker and cannot be altered during system registration.
- *Name*: A name describing the system. The name can be freely chosen by the delivery system administrator.
- *Hosting institution*: The ID of the institution which hosts the system (this institution must be registered at the broker too).
- *Administrator*: The broker user ID of the delivery system administrator that maintains the information stored about the system. The user account at the broker holds the contact information.
- *Description*: Here additional information to the delivery system, e.g. a link to a site where it is described can be added.
- *Active Flag*: Denotes whether the system is currently active or not.
- *Last modified date*: The date when system characteristics have been changed the last time.

Interface Characteristics

- *Service versions supported*: Since new versions of the interface specifications might arise, this set of numbers indicates the version(s) of the interface

specifications supported, wildcards '*' and ranges '-' can be used. Examples: 1.0-1.2, 2.0, 3.*

- *Login mechanism supported*: The broker currently supports two login mechanisms, i.e. mechanisms how consumers at the broker actually access a resource at a system.
 - *FORM_BASED*: This is a simple automatic login mechanism, which automatically sends user ID and password to the system (see section 0 for details)
 - *DISPLAY_PWD*: When using this mechanism the broker displays user ID and password to the consumer at the broker when she attempts to access a resource at the system.
- *Learning resource management of static resources supported*: This flag denotes whether the system supports a resource upload mechanism as for example the one described in Section 6.5. The following values are allowed: "HTTP", "SOAP", "SSH" and "NONE". "HTTP" denotes direct upload to the system via HTTP (in that case an upload URL as described in the next parameter is needed), "SOAP" denotes an indirect upload via SOAP (with attachment), "SSH" denotes an indirect upload via SSH copy and "NONE" denotes that no upload mechanism is supported.
- *Learning resource management service URL*: In case a system supports the upload of static resources via the broker, this field holds the URL that accepts HTTP uploads.
- *Delivery system user management support*: This field denotes whether the system supports only temporary users or supports permanent user accounts. Permissible values:
 - *TEMPORARY*
 - *PERMANENT*
- *SOAP endpoint services URL*: Every system that is connected with the central broker has to have a unique SOAP services base URL.

An example of a SOAP endpoint services URL is the following:

```
http://www.educanext.org:8080/JDS/services
```

- *Provision format*: In the case a system provides learning resource descriptions a field in the system administration specifies the format of the XML files that are provided. The following formats are supported: DC_XML, DC_RDF, LOM_XML, LOM_RDF, ULRIF (NO_PROVISION_SUPPORT is also a valid

value here). In this section it is also specified whether the broker holds the master copy of the metadata record.

- *Security credentials*: For every system the broker stores a valid certificate.

5.1.2 System authentication

In order for a system to communicate with the broker in a secure way, the system has to obtain a valid digital certificate from the broker. Certificate is requested by a delivery system administrator in whatever informal way, e.g. by initiating procedure at administration area of the broker. The system administrator of the broker verifies the delivery system administrators' identity (e.g. using information from his broker account), validity of the public key and other information in the certificate request, and possession of the corresponding private key. In case there is no error and the delivery system admin is allowed to get a certificate for his system, a certificate is issued, and the delivery system administrator is notified where and how he can retrieve it. Validity of the broker's CA certificate should be verified, e.g. by out-of-band methods, by the delivery system administrator.

For verification of the broker's identity in communication between the broker and the delivery system, the delivery system administrator also needs to securely obtain from the broker administrator, verify, e.g. by out-of-band methods, and install broker's certificate.

A delivery system administrator can request his system's certificate to be revoked, for example if he suspects that the private key of his system has been compromised. The system administrator of the broker must verify validity of the request before certificate revocation. System's certificate can also be revoked by the broker without delivery system administrator's request, for example, if data in the certificate are false. The broker administrator must always inform promptly the delivery system administrator about the revoked certificate.

5.2 Inspection

5.2.1 Usage scenario

The inspection service allows the broker to query administrative information from a delivery system. It serves two types of queries:

- The broker can query the availability of certain resources. This is used each time a consumer decides to book a resource. In this case the broker first asks for availability of the resource at the delivery system. The broker might also periodically check for availability of the resources offered in the broker's catalogue.

- This service allows querying a system for its state and capabilities. This service is necessary in order to check whether systems are accessible or not.

5.2.2 *Functionality*

Resource availability during booking process

At the time a user decides to book a resource at the broker, the broker checks for availability of the resource first. In the case the learning resource is available the broker proceeds by presenting the usage conditions of the learning resource to the consumer. In case the resource is not available the consumer, the provider and the catalogue administrator are informed.

Regular check for availability of resources

During times of low traffic the broker regularly checks for availability of the resources offered in the broker's catalogue. In case a resource offered in the catalogue is not available at the hosting delivery system, the offer of the resource is temporarily deactivated and the provider of the resource notified via email.

5.2.3 *Inspection Interface*

Method isResourceAvailable

```
boolean isResourceAvailable(  
    String resourceID)  
throws  
    NoSuchResourceException;
```

The method `isResourceAvailable` must be implemented by every delivery system that delivers content that can be booked at the broker. The method simply checks for resource availability.

Method isSystemAvailable

```
boolean isSystemAvailable();
```

This method must be implemented by every delivery system as well as the broker as a means to check for availability of the system. The method returns the current status of the system. It is intended to determine whether a system has been (temporary) disabled by its operator (e.g. for maintenance reasons).

Method getSupportedServices

```
String getSupportedServices ();
```

This method allows for querying the supported services and must be implemented by every delivery system as well as the broker. The result format of the method is a

comma-separated list of the names of the supported services. Table 2 lists the possible service names of broker and delivery systems.

Service Name	Described in Section
Inspection	5.2
SessionManagement	5.3
BrokerUserManagement	5.4
DeliverySystemUserManagement	5.5
Provision	6.1
Query	6.2
AccessControl	6.3
ResourceDelivery	6.4
ResourceManagement	6.5

Table 2: Service Names

Method `getSupportedMethods`

```
String getSupportedMethods(  
    String service)  
throws  
    ServiceNotSupportedException
```

This method allows for querying the supported methods of a given service and must be implemented by every delivery system as well as the broker. The result format of the method is a comma-separated list of the names of the supported methods (see various sections of the service names mentioned in Table 2). In case the given service name is not supported a *ServiceNotSupportedException* must be thrown.

5.3 Session Management

5.3.1 Functionality

The interface introduced herein is based on a simple session management concept. It is assumed that a session has to be established before any further communication can take place. (username/password) authentication is not a requirement since also anonymous sessions can be created (`createAnonymousSession` method). Once a session has been established it is assumed that the establishing node has the right to communicate with the other.

A session is valid until it is destroyed. Hence, it continues to be active even after a session-based method invocation has been made. In order to destroy a session `destroySession` must be called, but it also automatically timeouts when no communication takes place within 30 minutes. However, a session might be valid much longer than 30 minutes and sometimes might even require manual destruction.

The Session IDs shall include an identifier of the educational node, so that session IDs are also most likely unique within the network of nodes. They have to be long numbers or combinations of characters and numbers. Session IDs are unique at the repository issuing them.

5.3.2 *Session Management Interface*

Method createSession

This method creates a session and requires *userID* and *password* as parameters. Passwords are MD5 encoded. The method returns a Session ID. If this method is called for the same user more than once during the time-out period the same session ID shall be returned, i.e. in this case no new session ID shall be created.

```
String createSession (  
    String userID,  
    ....String password)  
throws  
    ....WrongCredentialsException,  
    SessionCreationFailureException;
```

The *WrongCredentialsException* is thrown in case an invalid *userID* or *password* was provided. The *SessionCreationFailureException* is thrown in case the creation of the session failed due to any other reason (for example because of too many open sessions, etc.).

Method createAnonymousSession

This method creates a session without requiring an account at the system where the session will be created. The method returns a Session ID.

```
String createAnonymousSession ()  
throws  
    SessionCreationFailureException;
```

The *SessionCreationFailureException* is thrown in case the creation of the session has failed.

Method destroySession

The system initiating a session can use this method in order to destroy a session. The parameter *sessionID* identifies the session.

```
void destroySession (  
    ....String sessionID)  
throws  
    NoSuchSessionException;
```

The *NoSuchSessionException* is thrown when an invalid *sessionID* is supplied.

5.4 Broker User Management

5.4.1 Usage scenario

This service provides an interface for creating users at the broker by drawing upon an existing user base. This service serves the purpose of registering users at the broker without requiring human intervention.

5.4.2 Functionality

The process of automatically registering users at the broker is triggered by the user using a special registration system called "User Registration Program" (URP). If the URP is not a component of an already registered delivery system the stand-alone URP has to be authenticated at broker.

The remote creation of user accounts has to be logged both at the system initiating the action and at the broker. At the broker a flag has to indicate whether a user ID was created remotely and if so the system ID (see Section 5.1) has to be stored in addition.

The URP can have a policy implemented, which automatically assigns consumer and/or provider roles to registering users (e.g. students will be assigned consumer rights only, faculty get both, consumer and provider rights). The roles which a user can currently be assigned to via this interface are:

- consumer (of learning resources), and
- provider (of learning resources).

A user can have more than one role assigned. At the broker additional roles (e.g. local registration authority) can be assigned.

A URP carries out two distinct steps: First it identifies an applying user by using its local authentication infrastructure (e.g. Kerberos) and second it calls the broker user registration interface to create the user account at the broker. By default the broker user ID shall be equivalent to the local user ID, entered at the beginning of the registration process. As a first step the URP checks the availability of the broker user ID. The attempt to create a user account might fail in the case the local user has already created a user account or the broker user ID has been already taken (the latter is only the case if the URP allows for the alteration of the broker user ID).

As a variation of this functionality multiple user accounts can be created in a row based on user records, when iterations of the above process take place.

So far, the interface does not support synchronisation of user account alterations. However, the local user ID can be used to check back whether a broker user has still a valid local user account.

5.4.3 *Broker User Management Interface*

All the subsequent methods are called by a registering system and processed at the broker.

Method *checkUserAccount*

This method shall be called before a new user is created at the broker. It checks for availability of a certain user ID at the broker. If the broker user ID is already used the *BrokerUserAlreadyExistsException* is thrown. If the local user ID in combination with the institution ID has already been assigned to a broker user the *LocalUserAlreadyExistsException* is thrown. Hereby the URP can ensure that users can only apply for a broker account once. The institution ID is an optional element and can also be used for identifying a specific mandate. In case it is provided *NoSuchInstitutionException* is thrown when the institution ID can not be found at the broker.

```
void checkUserAccount(
    String brokerUserID,
    String localUserID,
    String institutionID)
throws
    NoSuchInstitutionException,
    LocalUserAlreadyExistsException,
    BrokerUserAlreadyExistsException;
```

Method *createUserAccount*

This method creates a new user account at the broker.

```
void createUserAccount (
    String brokerUserID,
    String password,
    String localUserID,
    String institutionID,
    String roles,
    String firstName,
    String lastName,
    String phoneNumber,
    String faxNumber,
    String street,
    String zipCode,
    String city,
    String country,
    String homepage,
    String email)
throws
    NoSuchInstitutionException,
    LocalUserAlreadyExistsException,
    BrokerUserAlreadyExistsException,
    IncompleteDataException;
```

Roles are encoded in a comma-separated list. Valid role labels are “Provider” and “Consumer” and can be provided in all caps, small caps or capitalized.

This method expects several fields to be non-empty in order to successfully complete. The mandatory fields are: "brokerUserID", "password", "localUserID", "institutionID", "roles", "firstName", "lastName", "country", "email".

If the broker user ID is already used the *BrokerUserAlreadyExistsException* is thrown. If the local user ID in combination with the institution ID has already been assigned to a broker user the *LocalUserAlreadyExistsException* is thrown. *NoSuchInstitutionException* is thrown when the institution ID provided is not found at the broker. The *IncompleteDataException* is thrown in case one of the mandatory fields is not provided (i.e. one of those fields is empty).

5.5 Delivery System User Management

5.5.1 Usage scenario

This service describes means for the broker to create and delete user accounts at a delivery system. These user accounts are needed to access resources that were booked at the broker.

5.5.2 Functionality

This service distinguishes between two different types of remote user accounts:

- Temporary user accounts: This type of user is used by simple delivery systems which only store user IDs and passwords as a piece of user information. This information can not be changed at the delivery system.
- Permanent user accounts: Advanced delivery systems, like an LMS, require more information than just user ID and password. Additional user information (first name, last name, email address, etc.) must be provided.

Temporary user accounts shall be used for booking and accessing educational material at simple delivery systems. User ID and password are administered at the broker. Temporary user accounts are automatically deactivated after the booking period has elapsed.

Permanent user accounts are administered at the delivery system. However, the user ID may not be changed by the delivery system. A synchronisation between the broker user account and the delivery system user account is not supported.

The remote user account creation has to be logged both at the broker and the delivery system.

5.5.3 Delivery System User Management Interface

The following methods are called at the delivery system by the broker.

Method *checkUserAccount*

This method shall be called by the broker at the delivery system before a new user account is created. In case the given user ID has been already taken the *UserAlreadyExistsException* is thrown.

```
void checkUserAccount(  
    String userID)  
throws  
    UserAlreadyExistsException;
```

Method *createTempUserAccount*

This method is called by the broker and attempts to create a new temporary user at the delivery system. The password is created by the broker and must be sent as an MD5 encoded String.

```
void createTempUserAccount(  
    String userID,  
    String passwordMD5)  
throws  
    UserAlreadyExistsException;
```

In case the given user ID has been already taken the *UserAlreadyExistsException* is thrown.

Method *createUserAccount*

This method creates a permanent user account at the delivery system. Here typical user data like first name, last name, email address, etc. are used in addition to userID and password (which must be sent as an MD5 encoded String).

```
void createUserAccount (  
    String userID,  
    String passwordMD5,  
    String brokerUserID,  
    String firstName,  
    String lastName,  
    String gender,  
    String phoneNumber,  
    String faxNumber,  
    String street,  
    String zipCode,  
    String city,  
    String country,  
    String email)  
throws  
    UserAlreadyExistsException ,  
    IncompleteDataException;
```

In case the given user ID has been already taken the *UserAlreadyExistsException* is thrown. In case one of the fields is left empty, the *IncompleteDataException* is thrown.

Method deleteUserAccount

This method is called by the broker to delete a previously created temporary or permanent user account on the delivery system. A call to this method becomes necessary as soon as a broker user account is deleted, then namely the respective user accounts at different delivery systems must be deleted as well.

```
void deleteUserAccount(  
    String userID)  
throws  
    NoSuchUserException;
```

In case the given user ID does not exist at the delivery system, the *NoSuchUserException* is thrown.

6 Educational Node Application Services

6.1 Provision

6.1.1 Usage scenario

This service is used to replicate and synchronize metadata records describing learning resources between delivery systems and the broker. It is assumed that

- a) learning resources are described via metadata records in a format, which is specified in advance,
- b) delivery systems are registered at the broker and a common communication framework is available,
- c) the providing user has an account at the broker.

The delivery system has a repository of metadata records. Metadata descriptions are created, modified and deleted throughout the lifetime of a learning resource. This protocol provides means to synchronize alterations of metadata between delivery systems and the broker.

As far as the level of completeness of the metadata records is concerned, two variations of this scenario are possible: Either XML records provide all mandatory elements including offer information as specified by the broker or the user interface of the broker has to be used to complete this information. However, only the system (broker or delivery system) holding the master copies is allowed to instantiate alterations of metadata records.

6.1.2 Functionality

A delivery system that intends to synchronize its metadata repository with the broker logs any change to its repository. Each alteration, creation and deletion of a metadata record results in a corresponding synchronisation request. The delivery system calls the broker to perform either an update, insert or delete command at the remote metadata repository. Synchronisation requests are bound to a user identity. Although a delivery system can use just one account for all records, multiple user accounts with individual provider areas can be used. A synchronisation request lists exactly one learning resource or one offer that has to be replicated.

In case an XML file is provided, the broker-specific fields such as educational objective can either be filled in via the user interface of the broker (see alteration requirements below) or are not filled in at all when alteration is not possible at the broker since the delivery system holds the master copy of the metadata record. Metadata records shall be encoded in UniCode (UTF-8). Further, metadata records must not contain special

characters that are not allowed in XML. These characters as well as their valid substitutions are listed in the table below.

Character	Substitution
&	&
"	"
'	'
<	<
>	>

Table 3: Special Characters and their Substitution

Once a learning resource has been provided it can be viewed in the provider section of the broker. Alteration of the learning resource description and its corresponding offers are only possible if the master copy of a learning resource is kept at the broker. This version of the interface framework only supports the synchronisation of metadata records stored, created and deleted at the delivery system, but does not cover a backwards synchronisation initiated by the broker.

The whole synchronisation process is logged both on the delivery system and at the broker. The log includes date and time of the synchronisation requests, request parameters (see below) and the exit status. In addition the delivery system ID is stored in the broker log.

6.1.3 Additional User Interface Requirements

In order to provide complete learning resource provision an additional user dialog might be introduced either at the delivery system or at the broker to provide means for completing mandatory information. Whenever a delivery system user provides a new learning resource, she will be asked whether she wants to announce this learning resource also at the broker. If she agrees on offering the learning resource the following dialogue will gather additional data as specified below. The system holding the master copy of the description record (broker or delivery system) also allows for the alteration of it. It is supported that the master copy of the learning resource description is stored at the delivery system while the offering is performed at the broker. The following paragraphs illustrate, which kind of data has to be gathered when synchronising with a broker.

For example in the case of Educanext, a broker dedicated to the exchange of educational material and educational activities in the context of higher education, three different kinds of data have to be gathered:

- Descriptive information: The following metadata fields are mandatory in the data model of the broker, in our case the Universal Brokerage Platform (UBP) deployed under Educanext.org (See help texts of Learning Resource Provision area for detailed information on the meaning of these attributes):
 - Description language
 - Title
 - Learning Resource Language

- Description
- Classification according to the Dutch Basis Classification [16]
- Learning Resource Type (Educational Material, Educational Activity)
- Educational Material / Educational Activity Type
- Media Type for educational activities
- Access Information

For the broker it is important to know the type of learning resource provided. In the case of Educanext the broker has to identify a learning resource either as educational activity or as an educational material. The (LOM) field learning resource type is used for that. Depending on the learning resource type different media types values and contributor roles are supported.

- Offer information: At educanext.org the provider can specify a customized offer which holds the usage conditions (e.g. price, duration, license agreement) of the learning resource. Per default all learning resources offered are available under the Universal License Agreement of Educanext.
- Evaluation information: Learning resource evaluation can be carried out at the LMS or at the broker. Similar to the broker, providers have the ability to specify how the evaluation of their learning resource shall be managed. At Educanext per default no evaluation of provided learning resources is carried out.

Once all the data is gathered a record describing the learning resource is created and stored.

6.1.4 Synchronisation Interface

This method is a delivery-system initiated method to request synchronisation at the broker.

```
String requestSynchronisation (  
    String sessionID,  
    String synchronisationCommandID,  
    String synchronisationCommand,  
    String resourceID,  
    String resourceDescriptionURL)  
throws  
    NoSuchSessionException,  
    AuthorisationFailedException,  
    NoSuchResourceException,  
    ResourceExistsException,  
    NoSuchResourceDescriptionException,  
    NoValidRecordException,  
    SynchronisationFailedException;
```

The `sessionID` identifies the user and her learning resource and must be previously obtained with a call to `createSession` (see section 5.3 above).

The `synchronisationCommandID` provides an ID of the synchronisation command. This ID is created by the delivery system and is unique for the live time of the delivery system. It comprises of the delivery system ID and a sub-sequent number.

`synchronisationCommand` holds the information, whether the synchronisation requires an Insert, Update or Delete. Hence, the permissible values of `SynchronisationCommand` are "Insert", "Update" or "Delete" (synchronisation command can be provided in all caps, small caps or capitalized).

`resourceID` identifies the learning resource. The ID is managed by the broker. When a learning resource is inserted the broker generates a unique resource ID, which is returned to the delivery system as a return value of the method. Hence, this parameter is not required in case of an insert operation. The delivery system associates the ID with the learning resource, so that in the case the learning resource description is updated or deleted the ID can be provided.

`resourceDescriptionURL` holds the location, which provides the record of the learning resource or the offer. This parameter is not required in case of an delete operation.

The method returns the new resource ID in case of an "Insert" synchronisation command. For the other two cases ("Update", "Delete") the method returns an empty string.

The `NoSuchSessionException` is thrown in case the given session ID does not exist and thus the provider cannot be identified. In case the given session does not carry the right to synchronise resource descriptions, the `AuthorisationFailedException` shall be thrown. In the case of an update or delete request the `NoSuchResourceException` is thrown if a learning resource or an offer with the provided `ResourceID` does not exist at the broker. The `ResourceExistsException` is thrown when an insert command is placed on an already existing resource.

The `NoSuchResourceDescriptionException` is thrown in case the resource description cannot be fetched from the delivery system with the URL provided. The `NoValidRecordException` is thrown in the case the record provided is not valid.

6.1.5 Examples of Interface Calls

The following interface call inserts an LR at the broker under the user sbrantner:

```
requestSynchronisation("sessionID", "124014", "insert", "",  
"http://clix-demo.im-c.de/lrs/lr-description-2003-02-12");
```

The request returns lr-sbrantner-214104 as return value, which refers to the ID of the learning resource at the broker. The following interface call updates the same LR at the broker:

```
requestSynchronisation("sessionID", "124059", "update",  
"lr-sbrantner-214104", "http://clix-demo.im-c.de/lrs/lr-  
description-2003-02-19");
```

The following interface call deletes the same resource:

```
requestSynchronisation("sessionID", "124059", "delete",  
"lr-sbrantner-214104", "");
```

6.2 Querying

A Simple Query Interface (SQI) has been specified to interoperate learning repositories. The description of SQI can be found in chapter 7.

6.3 Access Control

6.3.1 Usage scenario

Booking a learning resource is a process that has to be completed before a consumer is allowed to access this learning resource at a delivery system. During this process consumers have to agree on the usage conditions that were defined by the provider of this resource. As soon as all formal agreements between consumer and provider have been settled, the broker grants the consumer access to the learning resource by creating a user account at the delivery system and (via the "Remote User Management" service, see Section 5.5) and by assigning the authorisation to access this resource.

The broker is in charge of managing contracts and deriving access rights from them. Hence, the broker guides access control at the delivery system. If a contract expires the broker automatically withdraws the access rights for the learning resources effected. The same applies when a consumer decides to deactivate booking.

The functionality described below introduces a simple access control interface, which does not distinguish between different kinds of learning resource consumptions (e.g. print, view, play, copy, etc.) and the extent to which this consumption can take place (e.g. 10 times). However, extension towards a digital rights management interface are investigated.

6.3.2 Functionality

Access Authorisation Creation

A consumer has to agree on the usage conditions of a learning resource before it can be booked. In order to grant access rights for the consumer the broker must be authenticated by the delivery system. Then the broker creates a new user account for the consumer at the delivery system or a user account already exists. As a next step the broker grants access rights for the consumer for the learning resource at the delivery system. After this process the delivery system holds appropriate authorisation information so that the consumer is allowed to access the learning resource.

Access Authorisation Withdrawal

The broker withdraws the previously granted access rights for a consumer at the delivery system.

6.3.3 Access Control Interface

The methods for this service have real-time semantics. If a method is called the changes to the access permissions of a learning resource take effect immediately.

createAuthorisation

This method is used to create an authorisation for a certain user and learning resource at a delivery system. The learning resource is referenced by a resource ID, which is equal to the ID returned by the `requestSynchronisation` method of the provision service or the methods of the learning resource management service.

```
void createAuthorisation(
    String userID,
    String resourceID)
throws
    NoSuchUserException,
    NoSuchResourceException,
    AuthorisationExistsException;
```

NoSuchUserException is thrown in case the user does not exist. *NoSuchResourceException* is thrown in case the learning resource does not exist. In case the authorisation has already been granted, the *AuthorisationExistsException* is thrown.

withdrawAuthorisation

This method is used to withdraw an authorisation for a certain user and learning resource at a delivery system.

```
void withdrawAuthorisation(
    String userID,
    String resourceID)
throws
    NoSuchUserException,
    NoSuchResourceException,
    NoSuchAuthorisationException;
```

NoSuchUserException is thrown in case the user does not exist. *NoSuchResourceException* is thrown in case the learning resource does not exist. In case the authorisation does not exist, the *NoSuchAuthorisationException* is thrown.

Method isAuthorised

This method is used to check whether a certain user is authorised to access a learning resource.

```
boolean isAuthorised(
    String userID,
    String resourceID)
throws
    NoSuchUserException,
    NoSuchResourceException;
```

NoSuchUserException is thrown in case the user does not exist. *NoSuchResourceException* is thrown in case the learning resource does not exist.

6.4 Resource Delivery

6.4.1 Usage scenario

As soon as a consumer has booked a resource at the broker and an authorisation for the user has been granted at the delivery system, the resource can be accessed through the broker by redirecting to the delivery system, which provides the resource.

6.4.2 Functionality

In order to access a resource the broker must first call the `getResourceAccessURL` at the delivery system. In this method the broker identifies the resource and also the user who wants to access the resource. After receiving the access URL the broker redirects to this URL and the user who previously booked the resource can finally access it.

6.4.3 Delivery Interface

Method `getResourceAccessURL`

```
String getResourceAccessURL(
    String resourceID,
    String version,
    String userID,
    String passwordMD5)
throws
    NoSuchResourceException,
    NoSuchVersionException,
    WrongCredentialsException,
    AuthorisationFailedException;
```

This method returns the full access URL of a given version of the given resource for the given user. If the version parameter is left empty, the URL of the most recent version of the resource is returned. In case the given `resourceID` is not valid, the `NoSuchResourceException` is thrown. In case the given `userID/password` pair is not valid the `WrongCredentialsException` is thrown. In case the given user is not allowed to access the given resource, the `AuthorisationFailedException` is thrown.

The access URL returned can be built in a way, so that it includes a simple security mechanism. E.g. it could contain an ID that can be used only once to establish a session with the delivery system for the user given in the `getResourceAccessURL` call.

After receiving the access URL of the resource, the broker should redirect the user to this URL again providing the username and password as HTTP parameters. The example below illustrates this with a simple HTML form.

```
<form name="deliverLR" action="accessURL" method="POST">
  <input type="hidden" name="userID"
    value="<USERID>">
  <input type="hidden" name="password"
    value="<MD5PASSWORD>">
</form>
```

This form must contain the hidden form fields "userID" and "password" (the password itself must be MD5 encoded) in order to denote the user who wants to access the resource.

6.5 Resource Management

6.5.1 Usage scenario

The broker supports providers of resources with a remote resource management mechanism. Hereby providers can upload and replace resources at a delivery system via the user interface of the broker.

6.5.2 Functionality

By using for example an HTTP file upload mechanism, a resource provider working at the broker can directly upload a resource to a chosen delivery system.

Resource Upload

In this scenario the provider has a local file available that contains the resource to be uploaded. During the resource provision process the provider can select a delivery system and upload a resource to this delivery system. After the file has been successfully transferred to the delivery system, the delivery system informs the broker that a new resource has been installed (see method `reportUploadedResource` below).

Resource Deployment

In case the uploaded resource file is a file archive, the broker first queries the delivery system for a list of available files that are potential entry points to the resource and asks the provider to select a file from this list (see method `getPossibleStartFiles` below).

When the provider successfully finishes an upload of a resource file, the broker has to finally confirm or reject the upload by calling the respective methods (see methods `confirmUpload` and `revokeUpload` below).

After successful confirmation the resource has been installed at its final destination and is ready to be booked and accessed via the broker.

6.5.3 Resource Management Interface

Upload process

This section describes the upload mechanism which supports the direct upload of a new resource to a delivery system.

During the resource provision process the broker produces an HTML form to upload files directly to a selected delivery system. In order for the upload to work, several hidden form fields are needed as parameters of the upload (also see example below):

- The address where to redirect the user to after the successful upload (hidden form field "redirectURLSuccess").
- The address where to redirect the user to after an unsuccessful upload (hidden form field "redirectURLError").
- The form must also contain the SOAP endpoint address of the broker (hidden form field "brokerSoapEndPoint", so that the delivery system can invoke the needed methods, described below.
- The ID of the resource at the broker to which the uploaded file belongs to must be provided in the hidden-form field "resourceID".

The following example form uploads a selected file to the URL "<http://www.deliverysystem.com/Upload>".

```
<form name="upload"
  action="http://www.deliverysystem.com/Upload"
  enctype="multipart/form-data" method="POST">

  <input type="hidden" name="redirectURLSuccess"
    value="<the URL upon success>">

  <input type="hidden" name="redirectURLError"
    value="<the URL upon error>">

  <input type="hidden" name="brokerSoapEndPoint"
    value="<Broker's SOAP-Endpoint-Address>">

  <input type="hidden" name="resourceID"
    value="<The ID of the resource to create a new
      version of>">

  <b>Select a file:</b>
  <input type="file" name="file" size="70">
  <input type="submit"
    value="Upload Resource">
</form>
```

As soon as the delivery system receives the uploaded file together with the hidden form fields, it reports the uploaded file to the broker by invoking the `reportUploadedResource` method.

Method reportUploadedResource

```
void reportUploadedResource(
    String resourceID,
    String version)
```

This method is called by a delivery system that received a resource. It is used to inform the broker about a new uploaded resource. The parameters of the method call are the ID of the new uploaded resource, and a new version number. The version must be generated by the delivery system.

Method *getPossibleStartFiles*

```
String getPossibleStartFiles(  
    String resourceID)  
throws  
    NoSuchResourceException;
```

In case a file archive (e.g. zip) is uploaded, there must be one file that constitutes the entry point to the resource. In order for the broker to identify this entry file, this method is called by the broker during the upload process (see Figure XXX). The delivery system returns a comma-separated list of files that are possible candidates (e.g. files with filenames ending in .htm or .html) for entry files. The filenames can also contain path information.

Method *getStartFile*

```
String getStartFile(  
    String resourceID,  
    String versionID)  
throws  
    NoSuchResourceException,  
    NoSuchVersionException;
```

This method returns the currently selected start file (i.e. entry point) for the given resource and version. In case the version-ID is not provided (left empty or null) the latest version of the given resource is taken.

In case the given resourceID is not valid, the `NoSuchResourceException` is thrown. In case the given versionID is not valid, the `NoSuchVersionException` is thrown.

Method *setStartFile*

```
void setStartFile(  
    String resourceID,  
    String versionID,  
    String startFilename)  
throws  
    NoSuchResourceException,  
    NoSuchVersionException,  
    NoSuchStartFileException;
```

This method is called by the broker to inform the delivery system about the file that shall be delivered as entry point to the given resource and version. In case the version-ID is not provided (left empty or null) the latest version of the given resource is taken.

In case the given resourceID is not valid, the `NoSuchResourceException` is thrown. In case the given versionID is not valid, the `NoSuchVersionException` is thrown. The method must also check whether the given start file is valid. If the file name is not a valid file, the `NoSuchStartFileException` is thrown.

Method confirmUpload

```
void confirmUpload(
    String resourceID)
throws
    NoSuchResourceException,
    DeploymentFailedException;
```

After a resource has been successfully uploaded, this method must be called by the broker to confirm the new uploaded resource. If no confirm method is called, the delivery system should automatically delete the uploaded resource after some time (e.g. 60 minutes).

Upon confirmation of an uploaded resource the delivery system should deploy the resource and make it thus accessible. In case an error occurs during deployment of the resource, the `DeploymentFailedException` is thrown.

Method revokeUpload

```
void revokeUpload(
    String resourceID)
throws
    NoSuchResourceException;
```

Similarly to the above method, this method can be used to revoke (i.e. cancel) a previously uploaded resource which has not yet been confirmed. Upon receiving this method call a delivery system should delete all files connected to the given resource.

Method getVersions

```
String getVersions(
    String resourceID)
throws
    NoSuchResourceException;
```

This method returns all available versions of the given resource as a comma separated string. In case the given `resourceID` is not valid, the `NoSuchResourceException` is thrown.

Upon accessing a resource, these version numbers can be used to access a specific version of a resource (see section 6.4 above).

Method removeResource

```
void removeResource(
    String resourceID)
throws
    NoSuchResourceException;
```

This method completely removes the given resource. If the resource contains several versions all these versions are removed. In case the given `resourceID` is not valid, the `NoSuchResourceException` is thrown.

Method removeVersionOfResource

```
void removeVersionOfResource(
```

```
    String resourceID,  
    String version)  
throws  
    NoSuchResourceException,  
    NoSuchVersionException;
```

This method removes the given version of the given resource. In case the given resourceID is not valid, the `NoSuchResourceException` is thrown. In case the version given is not a valid version of the resource the `NoSuchVersionException` is thrown.

7 Simple Query Interface

A continuously updated version of this section is available at:
<http://nm.wu-wien.ac.at/e-learning/interoperability/sqi/sqi.pdf>

7.1 Rational and Usage Scenario

Triggered by the recent work on learning objects metadata on the one hand, and the Semantic Web initiative on the other, researchers in technology-enhanced learning have increasingly identified the need for making learning repositories interoperable. In this paper an Application Program Interface (API) for querying learning objects repositories is presented. While many related activities do already exist at the time of drafting this specification [20, 25, 27], none of them are sufficiently independent from network architectures, repository types, and query languages, so that also highly heterogeneous learning objects repositories (XML-based vs. RDF-based) are able to interconnect.

Learning objects repositories also referred to as "repositories for learning" hold information on learning objects (ie metadata), describing educational artefacts such as courses, online tutorials, lecture notes, electronic textbooks, tutoring sessions, quizzes, etc. Learning objects are scarce resources. To allow other systems to check the availability of learning objects in a certain repository a query interface is needed².

Hereby, a learning repository becomes an open system allowing other tools to take advantage of the electronic resources stored within. This document specifies a number of methods a repository can make available in order to receive and answer queries from other repositories. Since one design objective is to keep the specification simple and easy to implement, the interface is labelled Simple Query Interface (SQI).

Since learning repositories can become equal peers, where each repository is able to submit queries to the other, the term "source" is introduced in order to label a system which issues a search (the source of the query). Alternatively this repository can also be referred to as requestor. The term "target" labels the system which holds the metadata queried (the target of the query).

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 wrappers need to be built to convert a query from a common query language X to query language Y and transform the query and the query results from a proprietary format to a common one and vice-versa. In this specification it is assumed that target and source repository have agreed upon a common query language beforehand.

² Although in this document we refer to learning repositories interoperability, this query interface can also be used within other domains.

The design of wrappers and wrapping service interfaces is not within the scope of this document. Wrappers can be implemented in various ways. For example, the target repository can store a redundant copy of the metadata in a format that allows for processing the query in the common query language without having it transformed in the target's original query language. Alternatively, a wrapper's capabilities of processing queries can be limited to a predefined set of query templates, for which pre-defined mappings exist. Mapping arbitrary queries at the time of query execution remains another, challenging option for implementing a wrapper.

Figure 5 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 attributes/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 schema.

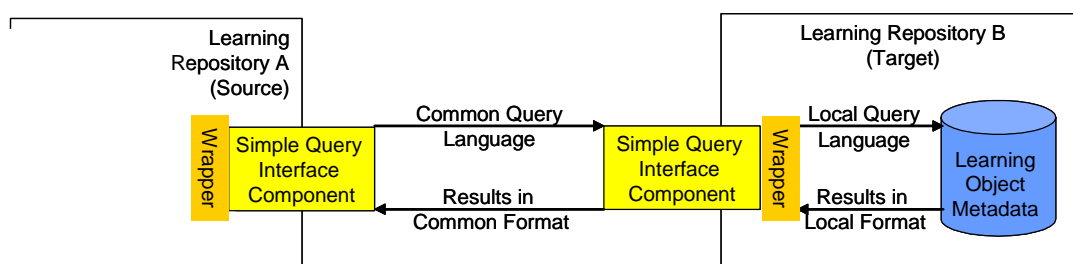


Figure 5 Communication between two Simple Query Interface Components

7.2 Requirements and Scenarios

Fehler! Verweisquelle konnte nicht gefunden werden. Figure 6 depicts the SQI stack, where each layer has a well-defined service it provides to the upper layer. The stack builds upon a common messaging system (e.g. SOAP, XML RPCs, JRM). These messages are transported via a common network architecture. On top of the messaging service an optional SQI session management layer is introduced providing authentication services. On top of session management layer the query management layer is placed supporting the exchange of queries and query results between source and target. Target and source are based on a common semantic model, which determines the semantic capabilities of the queries.

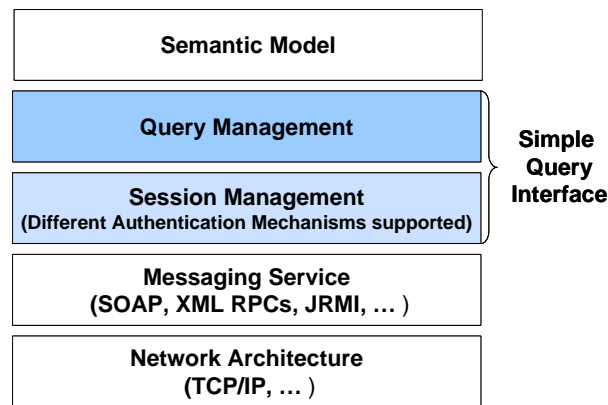


Figure 6 SQUI Stack

SQUI can be deployed in two different scenarios. In the synchronous scenario, the target provides the query results through the return parameter of the query method. The results retrieval is therefore triggered by the source through the submission of the query and through other methods allowing the source to access the query results set.

In the asynchronous scenario it is assumed that the target requires some time to collect the results. Therefore, the results retrieval is target-initiated. Whenever a significant amount of matching results is found, these results are forwarded to the source, which has a results listener implemented. The source must be able to uniquely identify a query sent to a particular target (even if the same query is sent to multiple targets). Otherwise the source is not able to distinguish the search results retrieved from various targets and/or queries previously submitted to a target.

Local course databases accessed through an intranet portal are more likely to take advantage of the synchronous scenario (see **Fehler! Verweisquelle konnte nicht gefunden werden.**). Source federating multiple repositories like a network gateway is more likely to be implemented according to the asynchronous scenario (see **Fehler! Verweisquelle konnte nicht gefunden werden.**).

Please note that the asynchronous query mode does not require an asynchronous handling on the messaging layer. It can also be implemented by two synchronous functions at the source and the target, respectively.

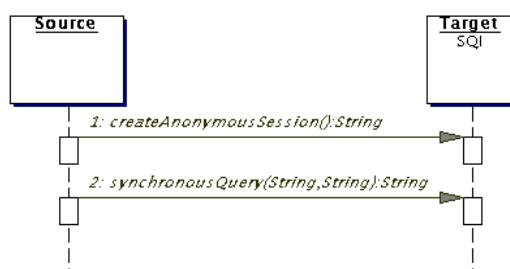


Figure 7 : Synchronous Query Mode used for querying a single Repository

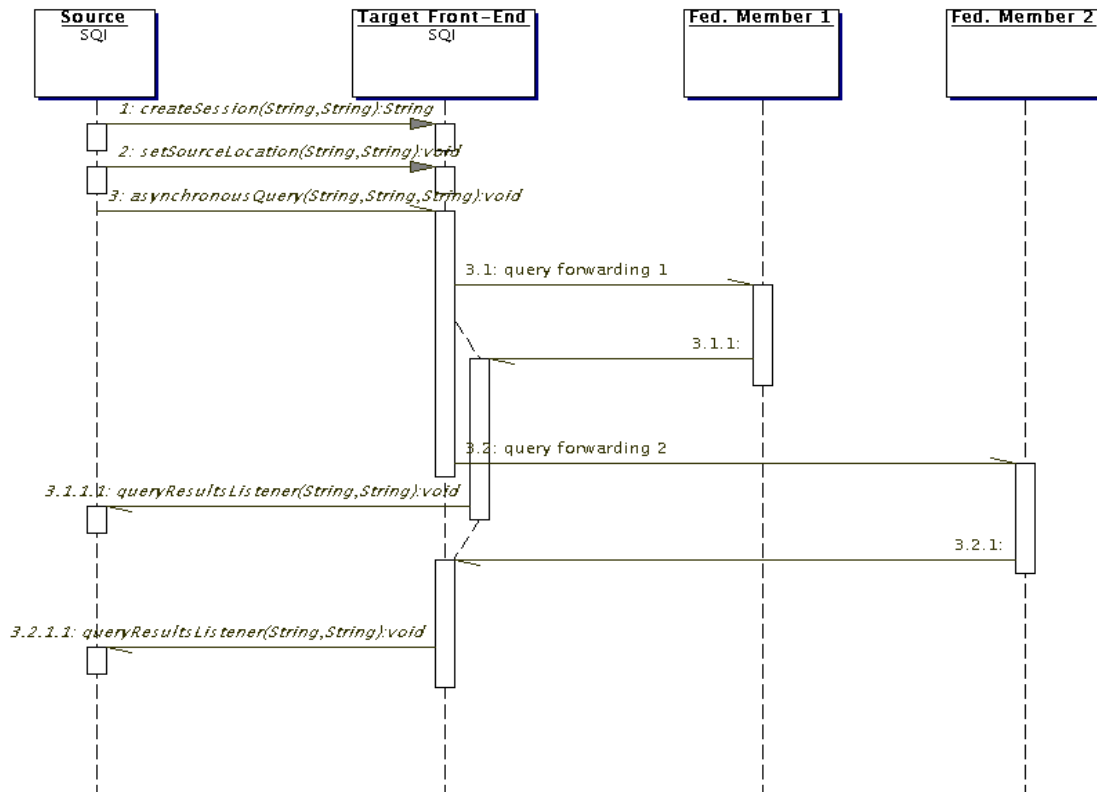


Figure 8 : Asynchronous Query Mode used for performing a Federated Search

The core of the query interface shall abstract from authentication and access control issues. However, there is a need to uniquely authenticate the source in order to allow a target to link query policies to a source repository. For example, while Repository I is allowed to query Repository II without any limitations, Repository III is only allowed to retrieve 1000 query results per day from Repository IV at a maximum. This requirement is covered by session management.

Session management needs to be understood as a higher-layer management of configuration settings and authentication. The session ID serves as a mandatory element, identifying the source in all query commands. However, session management can be implemented in a way that the token “session ID” is created manually or via a user (not system!) interface. Such session ID has the only purpose to uniquely identify the source and usually remains valid for a long period of time or until it is manually discarded.

Once a session has been established it is assumed that the establishing repository has the right to communicate with the other. In order to establish a session a user name and password or any other credential might be required, which is provided by the administrator of the respective repository. The identification of a repository shall mainly

prevent repositories from opening up their systems to unknown partners, but also prepares the grounds for implementing query policies. A query policy specifies the rights a repository's query rights at its counter-part, see Section 7.2 for examples on query policies.

In case of a synchronously operated query interface the source establishes a session at the target and uses the received session ID to identify itself during communication. This session ID is also referred to as *targetSessionID*, since it is issued by the target to identify the source. A query interface operated in synchronous mode can have only one query active per session. As a result the session ID identifies the query uniquely. In case of an asynchronously operated query interface, the source issues a query ID that allows it to link incoming results to with the submitted query (the source might query many targets). Multiple queries can be active within a session in asynchronous query mode.

At the time of writing a number of commonly defined schemas exist in the educational world. Examples of such schemas are:

- IEEE Learning Objects Metadata (LOM) Standard [21]
- IMS Learning Design Specification [16]
- European Schoolnet's Application Profile [34]
- Austrian Metadata Specification for Teaching and Learning Resources [33]
- Etc.

The interface shall be able to take advantage of these standardization initiatives by providing means for referring to those schemas. However, the set of permissible schemas shall not be limited to those specified by standardization bodies or similar institutions. Any schema that can be identified via a URI and that two repositories have agreed on is valid for being used. A SQI schema has two roles: On the one hand, it identifies the set of attributes and vocabularies that can be used in the query, and on the other, it provides a format in which learning object descriptions are returned to the source. Both, XML schemas and RDF schemas are valid in this context.

Since SQI is intended to be deployed by highly heterogeneous types of repositories (e.g., RDF engines, XML databases, relational databases, etc.) the interface cannot be based on one specific query language. Any query language chosen (e.g. XQUERY) would have a negative impact on the adoption of the interface by those repository types the language has not been designed for (e.g. RDF repositories in the case of XQUERY). Although the interface itself does not contribute to overcome the differences of the various paradigms in metadata management (Z39.50, XML-based approaches, RDF community), it aims to become an independent specification for all open educational repositories.

SQI design is based on the "Command-Query separation principle" [35]. This principle states that every method should either be a command that performs an action, or a query that returns data to the caller, but not both. More formally, methods should return a value only if they are referentially transparent and hence cause no side-effects. This leads to a style of design that produces clearer and more understandable interfaces.

In order to make the interface easily extensible while being backwards compatible we opted for an approach, which minimizes the number of parameters of the various methods rather than minimizing the number of methods. As a result, additional methods for setting query parameters like maximum duration and maximum number of returned search results were introduced. This design choice leads to simpler methods, but the number of interdependent methods is higher. However, for many of these query configuration and management methods defaults can be used.

7.3 *Functionality Overview*

First the source needs to create a connection with the target. Once a session has been established by using `createSession` or `createAnonymousSession`, the query interface at the target awaits the submission of a search request. As illustrated in the previous section, a “session” can also be established “off-line” (e.g. via e-mail or a web form). In such a case the session ID is provided as a configuration parameter for the target and there is no need to implement the API session management layer.

	Implemented at the target and called by the source	Implemented at the source and called by the target
<i>Query Configuration</i>		
<code>setQueryLanguage</code>	X	
<code>setResultsFormat</code>	X	
<code>setMaxQueryResults</code>	X	
<code>setMaxDuration</code>	X	
<i>Synchronous Query Interface</i>		
<code>setResultsSetSize</code>	X	
<code>synchronousQuery</code>	X	
<code>getTotalResultsCount</code>	X	
<code>getAdditionalQueryResults</code>	X	
<i>Asynchronous Query Interface</i>		
<code>asynchronousQuery</code>	X	
<code>setSourceLocation</code>	X	
<code>queryResultsListener</code>		X
<i>Results Management</i>		
<code>getResourceDescription</code>	X	
<i>Session Management</i>		
<code>createSession</code>	X	
<code>createAnonymousSession</code>	X	
<code>destroySession</code>	X	

Table 4: Overview of Simple Query Interface Methods

A number of methods allow for the configuration of the interface at the target. Query parameters such as the query language (`setQueryLanguage`), the number of results returned within one results set (`setResultsSetSize`), the maximum number of

query results (`setMaxQueryResults`), the maximum duration of query execution (`setMaxDuration`), and the results format (`setResultsFormat`) can be set with the respective methods. The parameters set via these methods are valid for the whole session unless they are set otherwise. If none of the methods is used before the first query is submitted, defaults are assumed.

Then the source submits a query, using either the `asynchronousQuery` or the `synchronousQuery` method. The query is then processed by the target and produces a set of records, also referred to as results set. The query is submitted using a common query language identified through a query parameter. In the query reference to a common schema is made.

To collect the query results from the source two methods are offered, but their usage depends on the query mode the interface operates in. The `getAdditionalQueryResults` method can be used by the source to collect additional query results in case of a synchronously operated query interface. The `getTotalResultsCount` method returns the total number for matching metadata records found by the target. In case of an asynchronously operated query interface the `queryResultsListener` method is called by the target to forward the query results to the source. The `getResourceDescription` method allow for retrieving a learning object's full metadata record.

Table 1 provides an overview of the various methods and whether they are implemented at the source or at the target.

7.4 Query Interface Methods

Query Configuration

Set Query Language

This method allows the source to control the syntax used in the query statement by identifying the query language. Values for the parameter *queryLanguageID* can be provided in upper case, lower case, or capitalized.

```
Void setQueryLanguage (  
    String targetSessionID,  
    String queryLanguageID)  
throws  
    NoSuchSessionException,  
    QueryLanguageNotSupportedException;
```

The *NoSuchSessionException* is thrown in case the given *TargetSessionID* is invalid. *QueryLanguageNotSupportedException* is thrown if the query language used in the request is not supported by the source.

Set Maximum Number of Query Results

This method defines the maximum number of results, which a query will produce. The maximum number of query results is set to 100 by default, but can be controlled via this method. *maxQueryResults* must be 0 (zero) or greater. If the maximum number of query results is set to 0 (zero), the source does not want to limit the number of maximum query results produced.

```
Void setMaxQueryResults (  
    String targetSessionID,  
    Integer maxQueryResults)  
throws  
    NoSuchSessionException,  
    NoValidMaxQueryResultsException;
```

The *NoSuchSessionException* is thrown in case the given *TargetSessionID* is invalid. *NoValidMaxQueryResultsException* is thrown if an invalid number is provided.

Set Maximum Duration

This method enables the source to set a time-out for the query in case of an asynchronously operated query interface. The values of *maxDuration* must be 0 (zero) or greater. A source delegates the time out management of the query to the target by setting *maxmiumDuration* to 0 (zero). The parameter *maxmiumDuration* is interpreted in milliseconds. The default value is 500.

```
Void setMaxDuration (  
    String targetSessionID,  
    Integer maxDuration)  
throws  
    NoSuchSessionException,  
    NoValidMaxDurationException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. *NoValidMaxDurationException* is thrown if an invalid number is provided.

Set Results Format

This method allows the source to control the format of the results returned by the target. The format according to which the results shall be formatted is specified in the *resultsFormat* parameter. The parameter is provided via a URI (e.g. “<http://ltsc.ieee.org/wg12/par1484-12-3/lom.xsd>” identifies the XML schema of the IEEE LOM standards) or via pre-defined values that can be provided in upper case, lower case, or capitalized.

```
Void setResultsFormat (  
    String targetSessionID,  
    String resultsFormat)  
throws  
    NoSuchSessionException,  
    ResultsFormatNotSupportedException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. The *ResultsFormatNotSupportedException* is thrown when the format provided via the *resultsFormat* parameter is not supported by the target.

Synchronous Query Methods

Set Results Set Size

This method defines the maximum number of results, which will be returned by single results set. The size of the results set is set to 25 records by default, but can be controlled via this method. *resultsSetSize* must be 0 (zero) or greater. A source asks for all results when the maximum number of results is set to 0 (zero).

```
Void setResultsSetSize (  
    String targetSessionID,  
    Integer resultsSetSize)  
throws  
    NoSuchSessionException,  
    NoValidResultsSetSizeException,  
    QueryModeNotSupportedException;
```

The *NoSuchSessionException* is thrown in case the given *TargetSessionID* is invalid. *NoValidResultsSetSizeException* is thrown if an invalid number is provided. The *QueryModeNotSupportedException* is thrown in case the target does not support synchronous queries.

Synchronous Query

This method places a query at the target. Since there can only be one active query per session, the *targetSessionID* also uniquely identifies the query. The query statement is provided via the *queryStatement* parameter. The resource attributes that have to be included in the results set are defined in the query statement.

This method returns a set of metadata records matching the query. The number of results returned is controlled by *setResultsSetSize* and its default value. The total number of results produced is limited by *setMaxQueryResults* and its default value.

```
String synchronousQuery (  
    String targetSessionID,  
    String queryStatement)  
throws  
    NoSuchSessionException,  
    QueryModeNotSupportedException,  
    NoValidQueryStatementException;
```

The *NoSuchSessionException* is thrown in case the given *targetSessionID* is invalid. The *QueryModeNotSupportedException* is thrown in case the target does not support synchronous queries. The *NoValidQueryStatementException* is thrown if the query statement does not comply with the syntax of the query language.

Get Additional Query Results

After a query has been submitted and returned the first set of results, this method can be called in case the query has produced more results than delivered by the first results set. The method returns a results set consisting of a list of additional metadata records. The number of results included in the results set is controlled by the `setResultsSetSize` method. The target returning the results ensures that the results comply with the results format specified.

The *targetSessionID* parameter identifies the previously defined query. The *startResult* parameter identifies the first record of the total results sets that will be returned. The method delivers the “next” result set in case *startResult* is set to 0 (zero). For example, in case of the first call after the `synchronousQuery` call, `getAdditionalQueryResults` delivers a results set starting with the record number ‘results set size + 1’. The index of the result set size starts with 1.

```
String getAdditionalQueryResults (  
    String targetSessionID,  
    Integer startResult)  
throws  
    NoSuchSessionException,  
    NoQuerySubmittedException,  
    QueryModeNotSupportedException,  
    NoValidStartResultException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. The *NoQuerySubmittedException* is thrown in case this method has been called without submitting a query beforehand. The *QueryModeNotSupportedException* is thrown in case the target does not support synchronous queries. *NoValidStartResultException* is thrown if an invalid number is provided for *startResult*.

Get Total Results Count

This method returns the total number of available results of the previously submitted query. The *targetSessionID* identifies the previously submitted query (including the associated session).

```
Integer getTotalResultsCount (  
    String targetSessionID)  
throws  
    NoSuchSessionException  
    QueryModeNotSupportedException,  
    NoQuerySubmittedException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. The *QueryModeNotSupportedException* is thrown in case the target does not support synchronous queries. *NoQuerySubmittedException* is thrown if no query has been submitted during the session.

Asynchronous Query Methods

Set Source Location

This method is required to be called before a query is submitted in asynchronous mode. The parameter *sourceLocation* specifies the location of the source's results listener in order for the target to be able to send the results.

```
Void setSourceLocation (  
    String targetSessionID,  
    String sourceLocation)  
throws  
    NoSuchSessionException,  
    QueryModeNotSupportedException,  
    NoValidLocationException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. The *QueryModeNotSupportedException* is thrown in case the target does not support asynchronous queries. The *NoValidLocationException* is thrown if *sourceLocation* is not valid.

Asynchronous Query

This method allows the source to submit a query to the target, while the results are returned in an asynchronous method. The query statement is provided via the *queryStatement* parameter. A query ID issued by the source is required in order to link the query results with the query, when they are later returned using the results listener. By using unique query IDs it is possible to submit an arbitrary number of queries per active session. The location of the source's results listener is needed and must be provided using *setSourceLocation* method.

Due to the asynchronous nature of this method, query results could still arrive from previous queries. The query is processed and results are forwarded within the timeframe specified in the *setMaxDuration* method.

```
Void asynchronousQuery (  
    String targetSessionID,  
    String queryStatement,  
    String queryID)  
throws  
    NoSuchSessionException,  
    NoSourceLocationException,  
    QueryModeNotSupportedException,  
    NoValidQueryStatementException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. The *QueryModeNotSupportedException* is thrown in case the target does not support asynchronous queries. The *NoSourceLocationException* is thrown in case no source location has been specified before submitting the query. The *NoValidSourceLocationException* is thrown if the location of the source has not been set (via the method *setSourceLocation*).

Query Results Listener

This target-initiated method forwards the results sets to the source. The *queryID* parameter is used for linking the query results to previously submitted query, when they are later return using the results listener.

The *queryResults* holds a results set consisting of a list of metadata records, which is formatted according to the schema specified in the query.

```
Void queryResultsListener (  
    String queryID,  
    String queryResults)  
throws  
    NoValidQueryResultsException,  
    NoSuchQueryException;
```

NoValidQueryResultsException is thrown in case the results set cannot be interpreted by the source. The *NoSuchQueryException* is thrown in case the given *queryID* is invalid.

Results Retrieval

Get Resource Description

This method returns the description of a learning resource. The parameter *resourceID* identifies the learning resource. The format according to which the results can be formatted is specified using the *setResultsFormat* method.

```
String getResourceDescription (  
    String targetSessionID,  
    String resourceID)  
throws  
    NoSuchSessionException,  
    NoSuchResourceException;
```

The *NoSuchSessionException* is thrown in case the given *targetSessionID* is invalid. *NoSuchResourceException* is thrown if the learning resource with the provided *resourceID* does not exist at the source.

7.5 Limitations

The following shortcomings have so far been identified:

- No methods supporting search status management (cancellation of search, query status reporting) are defined.
- The specification does not provide means for managing SQI repository profiles (Which query languages supported? Which schemas supported? Which query modes supported?), nor is a profile format specified.

- An asynchronous interface might become wrongly called using the synchronous methods and vice-versa. The interface is not prepared to handle any profile violation.
- The specification does not address any issues related to mappings of different metadata schemas besides allowing the provision of a schema according to which results shall be formatted.
- Data types should be specified at a higher level of detail.
- The only type of credential supported by session management is currently username/password.

7.6 *SQI Profile “Very Simple Query Interface”*

The query interface as it is specified above still leaves many choices for implementers. As a first starting point, this section proposes a “Very Simple Query Interface (VSQI)”, which can be used for achieving low-level interoperability between two repositories. The VSQI is based on the following assumptions:

- The administrator of the target repository provides the administrator of the source repository with a session ID, which is used for identifying the source repository and is valid until further notice.
- The query interface shall be implemented in synchronous mode.
- The schema for describing learning objects is based on IEEE LOM.
- The query shall only include a list of keywords, which the target repository can apply to the whole metadata and data set or to the most important fields. The keywords shall be submitted in the following format:

```
<simpleQuery>  
<term>search term 1</term>  
<term>search term 2</term>  
</simpleQuery>
```

A VSQI interface works as depicted in Figure 9. The query is submitted using a list of keywords as formatted above. The session ID is a static parameter which is provided via a configuration file of the API. The query returns a set for the first 25 results formatted according to the IEEE LOM XML schema. After a query has been submitted and returned the first set of results, the `getAdditionalQueryResults` method can be called in case the query has produced more results than delivered by the first results set.

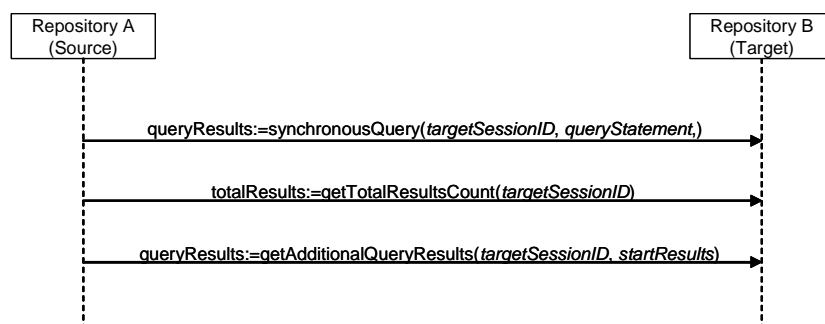


Figure 9: Work Flow SSI Application Profile for a Very Simple Query Interface

7.7 Related Work

The **IMS** has released the **Digital Repository Interoperability (DRI)** Specification [20]. The purpose of this document is to provide recommendations for the interoperability of the most common repository functions. The DRI specification presents five core commands (search/expose, gather/expose, alert/expose, submit/store, request/deliver) on a highly abstract level. The specification leaves too many design choices for implementers to effectively tackle the interoperability problem. While recommending Z39.50 (with its own query language) it also recommends XQUERY as a query language.

Z39.50 - properly "Information Retrieval (Z39.50); Application Service Definition and Protocol Specification, ANSI/NISO Z39.50-1995" - is an information retrieval protocol between computer systems primarily, library-related systems [19, 22]. Among other things Z39.50 defines a query language for specifying searches. The query message sent to the source uses the Reverse Polish Notation [22]. The queries are built upon registered definitions for attribute sets (e.g. the Bib-1 attribute set). Additionally, various record syntaxes (e.g. USMARC, UNIMARC) that can be used for transferring records from the target to the source are defined. The IMS Digital Repositories Specification [20] has extended the default attribute set with the IMS Metadata set, which is based on IEEE LOM [21]. Z39.50 has its roots in efforts dating back to the 1970s to allow standardized means of cross-database searching among a handful of (rather homogeneous) major bibliographic databases. As a consequence modern web technologies such as XML or XQUERY are not reflected in the standard. In 1995 the standard was updated for the last time (Version 3). At the time this specification was produced, work was going on a 200x version of Z39.50, which additionally, incorporates many clarifications, amendments, defect corrections, and implementer agreements, all of which have been endorsed by the Z39.50 Implementers Group [25]. The group is also working on work on incorporating XML schemas as returned record schemas [26].

Z39.50-International: Next Generation covers a number of initiatives by Z39.50 implementers to make the intellectual/semantic content of Z39.50 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 with 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 [27]. 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 specification presented in this paper. It introduces two main methods `searchRetrieveRequest` and `searchRetrieveResponse`, both, with quite a number of parameters. Return schemas are determined by the target system. No asynchronous query mode seems to be supported. CQL is a recommended query language in this specification.

Edutella is a schema-based peer-to-peer network for sharing learning resources [11]. It is based on the assumption that all resources available through the Edutella network can be described using RDF, and all functionality (for example querying) is mediated via RDF statements. Edutella provides an RDF- and Datalog-based query language QEL [23]. Querying is asynchronous – consumer peers are based on a results listener, which is able to handle continuously incoming results from different peers. For connecting repositories no API definition does exist at the time of writing. QEL is a recommended query language in this specification.

The **EduSource** project [29] 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 interpreted 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. Interestingly, 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 [31], which seems to have advantages in terms of higher flexibility and less resource-consumption.

The **CeLeBraTe** (Context eLearning with Broadband Technologies) project is developing a system to support a European Learning Network (ELN) of virtual learning environments (VLEs) capable of exchanging digitally stored learning resources [28, 30]. The ELN is built around a Brokerage System (BS) that manages exchanges between its members. Unlike other VLE networks based on a client-server architecture, CeLeBraTe is based on a mixed approach where, although the BS hosts a central metadata repository, each ELN member is also authorized to manage its own local metadata repository. Search requests are both handled centrally and propagated to local repositories. When a VLE receives a "Checked LOM Query Request", it processes it by querying its local LOM repository and sends the results set as a "LOM Query Result" back to the BS. Finally, the BS checks the result message before forwarding it as a "Checked LOM Query Result" to the source.

Ariadne offers a web-service based query interface [18]. The core class is called `QueryClient`. Ariadne uses a session-based, synchronous approach. Two methods for submitting queries are defined: `keywordQuery`, and `advancedQuery`. For the `advancedQuery` a proprietary query language is suggested, but not further defined. Results can be retrieved via the `getResults` method. A `getNumberOfResults` is also available.

Google offers a web-service based API [14] (Google's move to web services is nicely discussed in [15]). 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.

8 References

- [1] IEEE Learning Technology Standards Committee. IEEE Learning Objects Metadata Working Draft 6.1. <http://ltsc.ieee.org/wg12/index.html>, April 2001.
- [2] Dublin Core Metadata Element Set, Version 1.1. <http://dublincore.org/documents/1999/07/02/dces/>, July 1999.
- [3] Dublin Core Qualifiers. <http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/>, July 2000.
- [4] vCard. <http://www.imc.org/pdi/>, August 2002.
- [5] RDF. <http://www.w3.org/RDF/>, August 2002.
- [6] XML. <http://www.w3.org/XML/>, August 2002.
- [7] Dutch Basis Classification. http://www.kb.nl/kb/resources/frameset_kb.html?kb/vak/basis/bc98-en.html, August 2002. An XML version is available at <http://tristan.wu-wien.ac.at/rdf/taxonomy>.
- [8] Nejdil, W.; Wolf, B.; Qu, C.; Decker, S.; Sintek, M.; Ambjörn, N.; Nilsson, M.; Palmer, M.; Risch, T.: EDUTELLA: A P2P Networking Infrastructure Based on RDF. In: Proceedings of the 11th International World Wide Web Conference. Hawaii, USA. 2002.
- [9] W3C: XQuery 1.0: An XML Query Language, <http://www.w3.org/TR/xquery/>, December 2002.
- [10] W3C: XML Query Use Cases, <http://www.w3.org/TR/xmlquery-use-cases/>, December 2002.
- [11] Nejdil, W.; Wolf, B.; Qu, C.; Decker, S.; Sintek, M.; Ambjörn, N.; Nilsson, M.; Palmer, M.; Risch, T. (2002): EDUTELLA: A P2P Networking Infrastructure Based on RDF. In: Proceedings of the 11th International WWW Conference. Hawaii, USA. 2002.
- [12] W3C: XQuery 1.0: An XML Query Language, <http://www.w3.org/TR/xquery/>, December 2002.
- [13] W3C (2002): XML Query Use Cases, <http://www.w3.org/TR/xmlquery-use-cases/>, December 2002.
- [14] Google (2003): Google Web APIs (beta): <http://www.google.com/apis/>.
- [15] Prescod, P. (2002): Google's Gaffe. In: Webservices.Xml.Com. April 24, 2002, <http://webservices.xml.com/pub/a/ws/2002/04/24/google.html>.
- [16] IMS (2003). IMS Learning Design Specification, <http://www.imsglobal.org/learningdesign/index.cfm>.

-
- [17] Chamberlin, D. (2003): XQuery: A Query Language for XML. Presented at SIGMOD 2003,
http://www.almaden.ibm.com/cs/people/chamberlin/sigmod03_xquery.pdf
- [18] Duval, E.; Ternier, S. (2003): The Ariadne Web Service Interface,
<http://rubens.cs.kuleuven.ac.be:8989/ariadne/KPSCClient/doc/index.html>.
- [19] Lynch, C. A. (1997). The Z39.50 Information Retrieval Standard - Part I: A Strategic View of Its Past, Present and Future. D-Lib Magazine, (April) 1997,
<http://www.dlib.org/dlib/april97/04lynch.html>.
- [20] IMS (2003): IMS Digital Repositories Interoperability - Core Functions Best Practice Guide.
http://www.imsglobal.org/digitalrepositories/driv1p0/imsdri_bestv1p0.html.
- [21] IEEE (2002): 1484.12.1 - 2002 Learning Object Metadata (LOM) Standard.
- [22] Biblio Tech Review (2001): Z39.50: Part 1 - An Overview, http://www.biblio-tech.com/html/z39_50.html.
- [23] Nilsson, M.; Siberski, W.: RDF Query Exchange Language (QEL) - Concepts, Semantics and RDF Syntax, <http://edutella.jxta.org/spec/qel.html>.
- [24] CQL - Common Query Language, Z39.50 International: Next Generation.
<http://lcweb.loc.gov/z3950/agency/zing/cql/xcql.html>
- [25] Information Retrieval (Z39.50): Application Service Definition and Protocol Specification. <http://www.niso.org/standards/resources/Z39-50-200x.pdf>
- [26] Z39.50 Implementor Agreement: Requesting XML Records, August 2003.
<http://lcweb.loc.gov/z3950/agency/agree/request-xml.html>
- [27] Search/Retrieve Web Service. <http://lcweb.loc.gov/z3950/agency/zing/srw/>
- [28] Massart, D.: European Schoolnet's Comments On: The Elena Query Interface for Learning Repositories (v-0.4).
- [29] Hatala, M.; Richards, G.; Eap, T.; Willms J.: The Interoperability of Learning Object Repositories and Services: Standards, Implementations and Lessons Learned.
<http://www.sfu.ca/~mhatala/pubs/www04-interop-submit.pdf>
- [30] Massart D. and D.T. Le.: Federated search of learning object repositories: The CELEBRATE approach. In Proceedings of the 2nd International Conference of French-Speaking Vietnamese Computer Scientists (RIVF'04), Hanoi, Vietnam, February 2004 (also available at: <http://celebrate.eun.org/docs/rivf04.pdf>).
- [31] McCharty, J.: Reap the benefits of document style Web services.
<http://www-106.ibm.com/developerworks/webservices/library/ws-docstyle.html>
- [32] Nilsson, M.; Palmer, M.; Brase, J.: The LOM RDF binding - principles and implementation. In: Proceedings of the 3rd Annual Ariadne Conference,
<http://rubens.cs.kuleuven.ac.be:8989/ariadne/CONF2003/index.html>

- [33] Simon, B.; Rapf, K.: Österreichische Metadatenpezifikation für elektronische Lehr-/Lernressourcen (Austrian Metadata Specification for Teaching and Learning Resources),
http://elearning.bildung.at/statisch/bmbwk/de/elearning/metadatenmodellversion1_3_1.pdf.
- [34] Celebrate's/European Schoolnet's LOM Application Profile:
http://celebrate.eun.org/docs/CELEB_AP_v1.1_2003-12-15.pdf