



# PROLEARN

---

European Commission Sixth Framework Project (IST-507310)

<b>Deliverable</b>	<b>D4.8 SPI - The Simple Publishing Interface</b>
--------------------	---

**Editor** *S. Ternier, D. Massart, E. Demidova, D. Olmedilla, M. Memmel and E. Duval*

**Work Package** *WP 4*

**Status** *Final*

**Date** *09/01/2008*

### The PROLEARN Consortium

1. Universität Hannover, Learning Lab Lower Saxony (L3S), Germany
2. Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), Germany
3. Open University (OU), UK
4. Katholieke Universiteit Leuven (K.U.Leuven) / ARIADNE Foundation, Belgium
5. Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. (FHG), Germany
6. Wirtschaftsuniversität Wien (WUW), Austria
7. Universität für Bodenkultur, Zentrum für Soziale Innovation (CSI), Austria
8. École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
9. Eigenössische Technische Hochschule Zürich (ETHZ), Switzerland
10. Politecnico di Milano (POLIMI), Italy
11. Jožef Stefan Institute (JSI), Slovenia
12. Universidad Politécnica de Madrid (UPM), Spain
13. Kungl. Tekniska Högskolan (KTH), Sweden
14. National Centre for Scientific Research "Demokritos" (NCSR), Greece
15. Institut National des Télécommunications (INT), France
16. Hautes Etudes Commerciales (HEC), France
17. Technische Universiteit Eindhoven (TU/e), Netherlands
18. Rheinisch-Westfälische Technische Hochschule Aachen (RWTH), Germany
19. Helsinki University of Technology (HUT), Finland
20. imc information multimedia communication AG (IMC), Germany
21. Open Universiteit Nederland (OU NL), Netherlands
22. University of Warwick

## **Document Control**

**Title:** The ProLearn Query Language

**Author/Editor:** S. Ternier, D. Massart, E. Demidova, D. Olmedilla, M. Memmel and E. Duval

**E-mail:** [stefaan.ternier@cs.kuleuven.be](mailto:stefaan.ternier@cs.kuleuven.be)

## **Amendment History**

<b>Version</b>	<b>Date</b>	<b>Author/Editor</b>	<b>Description/Comments</b>
1	24/11/2007	Stefaan Ternier	The document is opened and organized
2	17/12/2007	Stefaan Ternier	Update
3	18/12/2007	David Massart	Various editings and comments
4	07/01/2008	Stefaan Ternier	Update after feedback reviewers

## **List of Contributors and Editors**

Ben Bosman, atmire.com

Stefano Ceri, Politecnico Milano

Elena Demidova, Learning Lab Lower Saxony

Erik Duval, Katholieke Universiteit Leuven

David Massart, European Schoolnet

Martin Memmel, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Daniel Olmedilla, Learning Lab Lower Saxony

Joaquin Salvachua, Universidad Politécnica de Madrid

Rafael Schirru, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Quentin Tremerie, European Schoolnet

Stefaan Ternier, Katholieke Universiteit Leuven

Michael Totschnig, Vienna University of Economics and Business Administration

Martin Wolpers, Katholieke Universiteit Leuven

## **Legal Notices**

The information in this document is subject to change without notice.

The Members of the PROLEARN Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the PROLEARN Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

# 1. Requirements and design principles

This deliverable provides the definition of the Simple Publishing Interface (SPI), a protocol through which an application (web-based or standalone) can deposit a learning object and/or its metadata in a learning object repository.

The Simple Publishing Protocol complements the Simple Query Interface (SQI) and the ProLearn Query Language (PLQL).

- The Simple Query Interface is a CEN ISSS standard for transporting queries to Learning Object Repositories (LORs). Through this search protocol, very heterogeneous repositories can be connected. SQI has been implemented on relational databases, XML databases, P2P networks and information retrieval systems.
- The ProLearn Query Language (ProLearn deliverable D4.3) is an abstract query language for interchanging queries. This language builds on the Contextual Query Language (CQL) [CQL], a well-established language used for library search. PLQL has been designed such that it supports querying hierarchical metadata schemas such as the Learning Object Metadata (LOM) standard.

Establishing interoperability for searching learning objects has enabled search access to a vast amount of learning objects. SQI has made it easier to find relevant learning objects.

The goal of SPI is to make it easier to make objects available for reuse. The traditional procedure for publishing learning objects to a repository is tightly integrated with a specific repository, whereas the publication to learners typically relies on a Learning Management System (LMS) or a Virtual Learning Environment (VLE). This complicates the workflow of the creator who will often publish to learners only, so that the object does not become available for reuse in the repository.

The design of the SPI API is based on the design principles of SQI. We have defined a simple set of commands that is extensible and flexible. By analogy with SQI, this protocol makes a distinction between semantic and syntactic interoperability.

- Syntactic interoperability is the ability of applications to deal with the structure and format of data. For instance, a language such as XML Schema Description (XSD) ensures the syntactic interoperability of XML documents as it allows for the parsing and validation of these documents.
- Semantic interoperability refers to the ability of two parties to agree on the meaning of data or methods. When exchanging data, semantic interoperability is achieved when data is understood by all the applications involved.

## *“By reference” and “by value” publishing*

Traditionally, two approaches allow for passing data from a source to a target.

1. “By value” publishing embeds a learning object, after encoding, into the message that is sent to a target.
2. “By reference” publishing embeds a reference (e.g. a URL) to a learning object to publish into the message that is sent to a target. Note that this is different from publishing metadata in a referatory. Publishing in a referatory involves publishing metadata that contains a reference to the learning object. When publishing learning objects “by reference”, a reference to the learning objects is used to transport the learning object. This reference is not added to the metadata instance that describes the learning object.

“By value” publishing is useful for a standalone application, which is generally not associated with a web server on which a target can obtain a learning object. In this case, embedding a learning object in a message passed to the target lowers the threshold for pushing a learning object. “By reference” publishing is particularly suited when larger amounts of data have to be published. As embedding large files into a single message may cause degraded performance, a need exists to use a distinct method (e.g., FTP, HTTP, SCP, etc.) for uploading learning objects.

Rather than imposing one of these approaches, the publish protocol will be designed to support both of them.

## *Flexible application*

Some of SPI design decisions were inspired by existing applications and practices.

- A learning object referatory manages metadata that refers to learning objects stored on separate systems. SPI must thus be supportable by repositories that do not manage learning objects.
- Some applications manage publishing learning objects without the metadata. PENS (see section on related work) is an example of such a protocol.
- Finally, SPI must allow for publishing to repositories that manage both learning objects and metadata.

In order to support these three scenarios, a separate method is used to associate a learning object to its metadata.

## *Objectives*

This publishing protocol meets the following objectives:

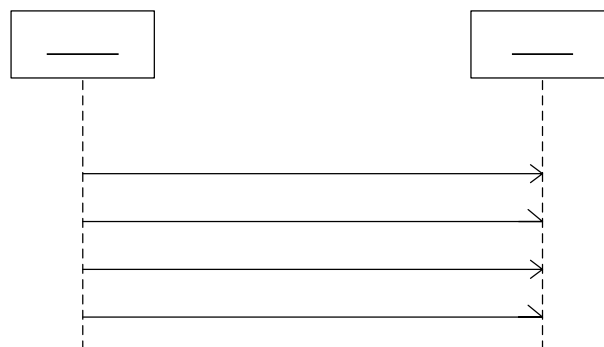
- SPI enables integrating publishing into authoring environments. This is beneficial for the authors' workflow, as they do not need to manually upload their learning objects using external publishing applications.
- SPI provides interoperability between applications that publish and applications that manage learning objects and metadata. Doing so, the effort of integrating publishing access into an authoring application can be reused on other learning object repositories, provided that they support SPI.

## Overview

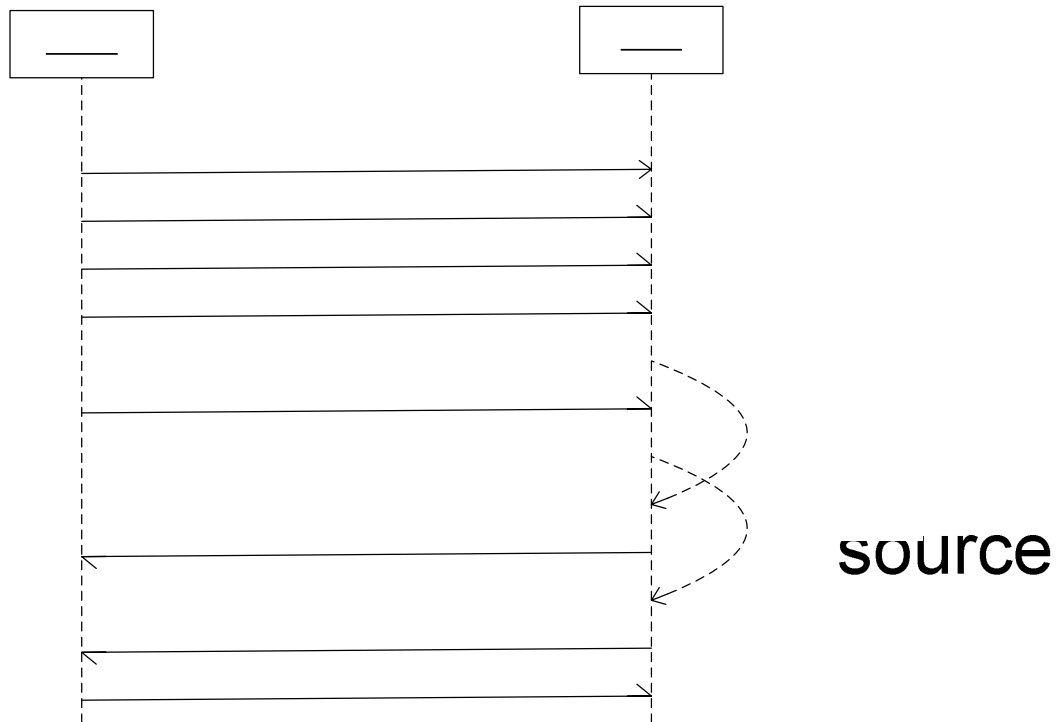
This deliverable is organized as follows. Section 2 introduces the methods necessary for publishing a learning object. Next, Section 3 groups methods for publishing metadata. Addressing learning objects and metadata separately implements a "separation of concerns". The next section introduces methods for connecting learning objects and metadata. Section 5 compares SPI with related work in publishing learning objects and metadata. Finally, Section 6 reports on experiences with SPI.

## 2. Submit & Delete Learning Objects

This section presents methods relevant for publishing learning objects. The first method, `setDataFormat`, sets the format of the learning object. Next, `setSourceLocation` is introduced. This method is relevant for transmitting learning objects "by reference". The next subsection introduces methods for submitting the learning object to a target. When learning objects are transmitted "by reference", a source asynchronously awaits the notification of retrieval. `NotifyRetrievalStatus` is a method for notifying the source and is discussed next. Finally, deleting learning objects is covered.



**Figure 1: Submit "by value": a sequence diagram**



**Figure 2: Submit “by reference”: a sequence diagram**

### *Set Data Format*

The setDataFormat method enables a source to indicate the type of objects that it will submit. With this method, one can set the data format to compound objects such as SCORM, AICC or IMS Content packages. Apart from specifying the package type, this argument can denote the type of archiving and compression format that is used for submitting the learning object. This method enables the target to process the resource that is submitted. For instance, a target can use this parameter to select a proper tool to decompress resources. Note that this parameter differs from the MIME media type that specifies the file type of a single object. The MIME media type of a learning object can be submitted as part of a metadata instance.

<i>Method name</i>	<b>setDataFormat</b>	
<i>Return type</i>	Void	
<i>Parameters</i>	<i>Name</i>	<i>Type</i>
	targetSessionID	String
	dataFormatID	String

<i>Fault</i>	NO_SUCH_SESSION DATA_FORMAT_NOT_SUPPORTED METHOD_FAILURE
--------------	--

Issuing setDataFormat, the following errors can occur:

- NO SUCH SESSION in case the given TargetSessionID is invalid
- DATA FORMAT NOT SUPPORTED if the format used in the request is not supported by the target
- METHOD FAILURE if the operation fails for another reason.

### *Set Source Location*

This method is called before a learning object is submitted in "by reference" mode. The parameter sourceLocation specifies the location of the source's notifyRetrievalStatus method and enables the target to send an acknowledge message.

<i>Method name</i>	<b>setSourceLocation</b>	
<i>Return type</i>	Void	
<i>Parameters</i>	<i>Name</i>	<i>Type</i>
	targetSessionID	String
	sourceLocation	String
<i>Fault</i>	NO_SUCH_SESSION SUBMISSION_MODE_NOT_SUPPORTED INVALID_SOURCE_LOCATION METHOD_FAILURE	

The following faults can occur:

- NO\_SUCH\_SESSION in case the given targetSessionID is invalid
- SUBMISSION\_MODE\_NOT\_SUPPORTED if the target does not support "by reference" submissions.
- INVALID\_SOURCE\_LOCATION if the location is incorrect or can not be parsed
- METHOD\_FAILURE if the operation fails for another reason

## Submit Resource

The submit resource methods allow for publishing a learning object to a learning object repository. In some scenarios, a target will be responsible for generating the identifier, while in other scenarios the source assigns an identifier to the learning object. Therefore, both `submitResourceByValue` and `submitResourceByReference` have a version that does not specify an identifier and a version that does specify an identifier as parameter. The methods on the right side do not receive an identifier from the source and require the target to generate and return an identifier, while the left-hand methods will use the identifier, received from the target. If a source does not support one of these scenarios, a `METHOD NOT SUPPORTED` exception is thrown.

"By value" publishing embeds a learning object in the request, whereas "by reference" publishing embeds a reference and does not deal with the actual transferring of the resource to the server. The sequence diagram in figure 1 presents a "by value" transmission of a learning object. This sequence diagram illustrates a scenario where the target is responsible for the creation of the identifier. The `submitResourceByReference` method implements the second scenario. Figure 2 shows that after the creation of a session and the invocation of the `setDataFormat` method, a source can set the location of its notification listener. Next, a source can use the `submitResourceByReference` method to submit a reference (e.g. a URL) to learning object. In this scenario, the source is responsible for the generation of the learning object identifier. The target retrieves the learning object. At the mean time, a source can submit another reference, with a different identifier. Finally, when retrieval is done, the target invokes the `notifyRetrievalStatus` method on the source.

<i>Method name</i>	<b>submitResourceByValue</b>	<b>submitResourceByValue</b>														
<i>Return type</i>	Void	String														
<i>Parameters</i>	<table border="1"> <thead> <tr> <th><i>Name</i></th> <th><i>Type</i></th> </tr> </thead> <tbody> <tr> <td>targetSessionID</td> <td>String</td> </tr> <tr> <td>data</td> <td>binaryData</td> </tr> <tr> <td>identifier</td> <td>String</td> </tr> </tbody> </table>	<i>Name</i>	<i>Type</i>	targetSessionID	String	data	binaryData	identifier	String	<table border="1"> <thead> <tr> <th><i>Name</i></th> <th><i>Type</i></th> </tr> </thead> <tbody> <tr> <td>targetSessionID</td> <td>String</td> </tr> <tr> <td>data</td> <td>binaryData</td> </tr> </tbody> </table>	<i>Name</i>	<i>Type</i>	targetSessionID	String	data	binaryData
<i>Name</i>	<i>Type</i>															
targetSessionID	String															
data	binaryData															
identifier	String															
<i>Name</i>	<i>Type</i>															
targetSessionID	String															
data	binaryData															
<i>Fault</i>	NO_SUCH_SESSION METHOD_NOT_SUPPORTED INSUFFICIENT_CREDENTIALS INVALID_RESOURCE_IDENTIFIER METHOD_FAILURE	NO_SUCH_SESSION METHOD_NOT_SUPPORTED INSUFFICIENT_CREDENTIALS METHOD_FAILURE														
<i>Method name</i>	<b>submitResourceByReference</b>	<b>submitResourceByReference</b>														

<i>Return type</i>	Void	String		
<i>Parameters</i>	<i>Name</i>	<i>Type</i>	<i>Name</i>	<i>Type</i>
	targetSessionID	String	targetSessionID	String
	reference	String	reference	String
	identifier	String		
<i>Fault</i>	NO_SUCH_SESSION NO_SOURCE_LOCATION METHOD_NOT_SUPPORTED INSUFFICIENT_CREDENTIALS INVALID_RESOURCE_IDENTIFIER METHOD_FAILURE	NO_SUCH_SESSION NO_SOURCE_LOCATION METHOD_NOT_SUPPORTED INSUFFICIENT_CREDENTIALS METHOD_FAILURE		

Issuing one of the submit resource methods, the following errors can occur:

- NO\_SUCH\_SESSION in case the given TargetSessionID is invalid
- METHOD\_NOT\_SUPPORTED if the submission method is not supported by the target. A target can for instance not support methods that enable a client to generate identifiers.
- INSUFFICIENT\_CREDENTIALS if the user is not authorized to execute this method
- INVALID\_RESOURCE\_IDENTIFIER when the identifier provided by the source already exists or has a structure that is not supported
- NO\_SOURCE\_LOCATION in case no source location has been specified before submitting the resource (via the method setSourceLocation).
- METHOD\_FAILURE if the operation fails for another reason.

### *Notify Retrieval Status*

After invocation of the submitResourceByReference method, the source asynchronously awaits a notification of transmission completion. The notifyRetrievalStatus method is a target-initiated method that informs the source on the status of the transfer. After completion of a "by reference" transfer, a target must send a notification with status "COMPLETE" to the source. If this message cannot be delivered (e.g. the source is not reachable), the target must delete the resource for which it tried to send a notification.

<i>Method name</i>	<b>notifyRetrievalStatus</b>
<i>Return type</i>	Void

<i>Parameters</i>	<i>Name</i>	<i>Type</i>
	targetSessionID	String
	resourceIdentifier	String
	Status	String
<i>Fault</i>	NO_SUCH_SESSION NO_SUCH_IDENTIFIER METHOD_FAILURE	

The following faults can occur:

- NO\_SUCH\_SESSION in case the given targetSessionID is invalid
- NO\_SUCH\_IDENTIFIER if an invalid identifier is provided.
- METHOD\_FAILURE if the operation fails for another reason

### *Delete Resource*

The deleteResource method deletes a learning object on the target.

<i>Method name</i>	<b>deleteResource</b>	
<i>Return type</i>	Void	
<i>Parameters</i>	<i>Name</i>	<i>Type</i>
	targetSessionID	String
	resourceIdentifier	String
<i>Fault</i>	NO_SUCH_SESSION INVALID_RESOURCE_IDENTIFIER RESOURCE_DOES_NOT_EXIST INSUFFICIENT_CREDENTIALS DELETION_NOT_ALLOWED METADATA_ASSOCIATED METHOD_FAILURE	

The following faults can occur:

- NO\_SUCH\_SESSION in case the given targetSessionID is invalid.
- INVALID\_RESOURCE\_IDENTIFIER if the identifier has an invalid structure.

- RESOURCE\_DOES\_NOT\_EXIST if the learning object associated with identifier does not exist.
- INSUFFICIENT\_CREDENTIALS if the user is not authorized to delete the learning object;
- DELETION\_NOT\_ALLOWED this target does not allow for deleting learning objects after they are published.
- METADATA\_ASSOCIATED the learning object is associated to a metadata instance. Dissociate both first.
- METHOD\_FAILURE if the operation fails for another reason

### 3. Submit & Delete Metadata

The following methods allow a source to submit a metadata instance to the target. Using the setMetadataSchema method, repositories can offer support for several metadata schemas or application profiles. After setting the schema, a source can submit a metadata instance to the target.

#### *Set Metadata Schema*

This method allows the source to control the schema of the metadata that will be submitted to the target.

The metadataSchema parameter is preferably a URI that identifies the metadata schema or the application profile. After invoking this method, all metadata instances that are submitted within the session identified by targetSessionID, must validate against the schema that was set.

Note that, when metadata is submitted as an XML instance, the metadata can contain a reference to a metadata schema. In such case, the metadata instance must validate against both the schema that is set with the setMetadataSchema method and the schema that the XML instance references to.

<i>Method name</i>	<b>setMetadataSchema</b>	
<i>Return type</i>	Void	
<i>Parameters</i>	<i>Name</i>	<i>Type</i>
	targetSessionID	String
	metadataSchema	String

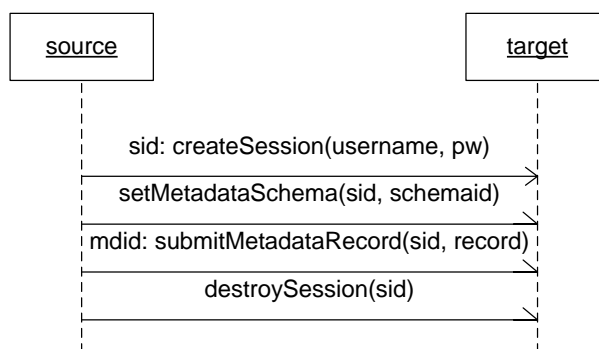
<i>Fault</i>	NO_SUCH_SESSION
	SCHEMA_NOT_SUPPORTED
	METHOD_FAILURE

The following faults can occur:

- NO\_SUCH\_SESSION in case the given targetSessionID is invalid;
- SCHEMA\_NOT\_SUPPORTED when the schema provided via the metadataSchema parameter is not supported by the target
- METHOD\_FAILURE if the operation fails for another reason.

### Submit Metadata

In analogy to submitResource, either the target or the source can generate an identifier for the metadata instance. After submission, the target must record this identifier with the submitted metadata.



**Figure 3: Submit a metadata record: a sequence diagram**

<i>Method name</i>	<b>submitMetadataRecord</b>		<b>submitMetadataRecord</b>	
<i>Return type</i>	Void		String	
<i>Parameters</i>	<i>Name</i>	<i>Type</i>	<i>Name</i>	<i>Type</i>
	targetSessionID	String	targetSessionID	String
	metadataRecord	String	metadataRecord	String
	identifier	String		

<i>Fault</i>	NO_SUCH_SESSION METHOD_NOT_SUPPORTED INVALID_METADATA_IDENTIFIER INSUFFICIENT_CREDENTIALS VALIDATION_FAILURE METHOD_FAILURE	NO_SUCH_SESSION METHOD_NOT_SUPPORTED INSUFFICIENT_CREDENTIALS VALIDATION_FAILURE METHOD_FAILURE
--------------	--	---

The following faults can occur:

- NO\_SUCH\_SESSION in case the given targetSessionID is invalid;
- METHOD\_NOT\_SUPPORTED when the target does not support a source creating identifiers or vice versa;
- INVALID\_METADATA\_IDENTIFIER when the identifier provided by the source is invalid. For instance the identifier already exists;
- INSUFFICIENT\_CREDENTIALS if the user is not authorized to publish metadata;
- VALIDATION\_FAILURE when the metadata submitted does not validate against the metadata schema that is set;
- METHOD\_FAILURE if the operation fails for another reason.

### *Delete Metadata*

The deleteMetadata method removes a metadata instance from the target.

<i>Method name</i>	<b>deleteMetadataRecord</b>	
<i>Return type</i>	Void	
<i>Parameters</i>	<i>Name</i>	<i>Type</i>
	targetSessionID	String
	metadataIdentifier	String
<i>Fault</i>	NO_SUCH_SESSION INVALID_METADATA_IDENTIFIER METADATA_RECORD_DOES_NOT_EXIST INSUFFICIENT_CREDENTIALS DELETION_NOT_ALLOWED RESOURCE_ASSOCIATED	

METHOD\_FAILURE

The following faults can occur:

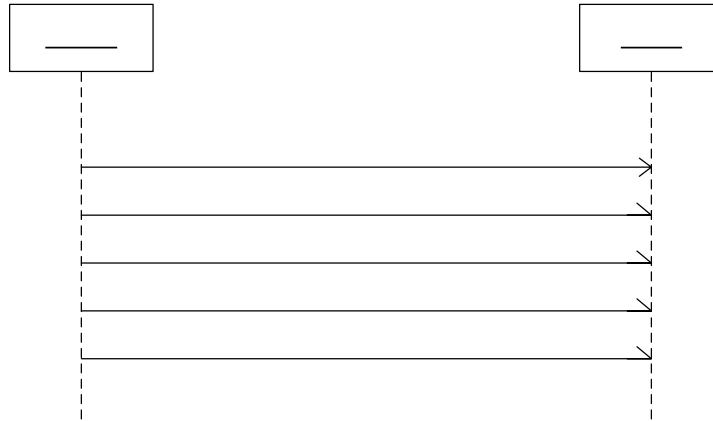
- NO\_SUCH\_SESSION in case the given targetSessionID is invalid.
- INVALID\_METADATA\_IDENTIFIER if the identifier has an invalid structure.
- METADATA\_RECORD\_DOES\_NOT\_EXIST if the identifier does not exist.
- INSUFFICIENT\_CREDENTIALS if the user is not authorized to delete metadata;
- DELETION\_NOT\_ALLOWED this target does not allow for deleting metadata instances after they are published.
- RESOURCE\_ASSOCIATED the metadata instance is associated to a learning object. Dissociate both first.
- METHOD\_FAILURE if the operation fails for another reason

## 4. Associating metadata with learning objects

After submitting both learning object and metadata, one can use the associate method to connect the metadata instance to a learning object. This section defines an associate method that enables connecting a learning object and a metadata instance. A dissociate method enables removing this connection. The methods that will be presented are only relevant when publishing to repositories that contain both a content and a metadata store. In a referatory, only the methods presented in Section 3 are necessary to publish a metadata instance. Learning objects are published elsewhere, prior to the submission of a metadata instance to a referatory. Therefore, the metadata instance can contain a reference to the learning object.

### *Associate*

In a learning object repository that contains both a context store and a metadata store, learning objects and metadata instances can be published in a random order. When a metadata instance is published prior to the learning object, it cannot refer to the learning object that is not yet available in the repository. Therefore, the associate method allows for explicitly adding this reference.



**Figure 4: Associate a resource and a metadata record: a sequence diagram.**

**source**

<i>Method name</i>	<b>associate</b>	
<i>Return type</i>	Void	
<i>Parameters</i>	<i>Name</i>	<i>Type</i>
	targetSessionID	String
	resourceIdentifier	String
	metadataIdentifier	String
<i>Fault</i>	NO_SUCH_SESSION INVALID_RESOURCE_IDENTIFIER INVALID_METADATA_IDENTIFIER RESOURCE_NOT_RETRIEVED INSUFFICIENT_CREDENTIALS METHOD_FAILURE	

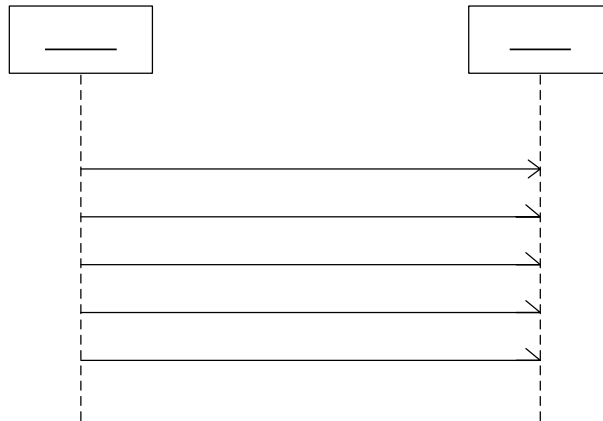
The following faults can occur:

- NO\_SUCH\_SESSION in case the given targetSessionID is invalid.
- INVALID\_RESOURCE\_IDENTIFIER if no resource is identified by the resourceIdentifier parameter.
- INVALID\_METADATA\_IDENTIFIER if no metadata instance is identified by the metadataIdentifier parameter.
- RESOURCE\_NOT\_RETRIEVED in case of a “by reference” retrieval, it can occur that retrieval of a resource has not yet completed.
- INSUFFICIENT\_CREDENTIALS if the user is not authorized to execute this method;

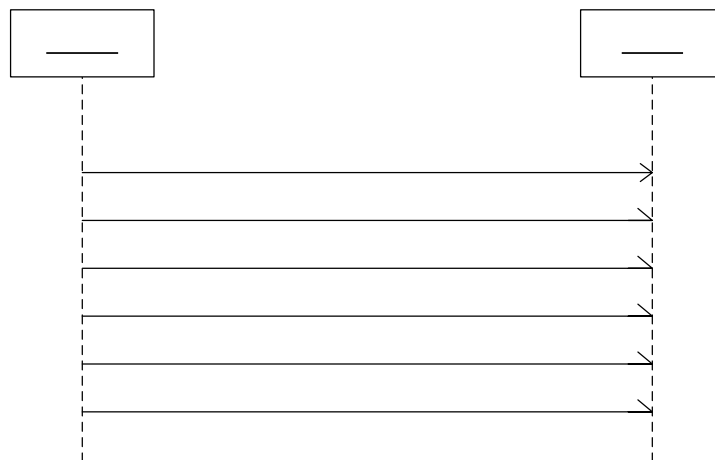
- METHOD\_FAILURE if the operation fails for another reason

### *Dissociate*

With the dissociate method, an association can be undone. This is necessary when metadata and/or learning object are to be deleted.



**Figure 5: Dissociate: a sequence diagram**



**Figure 6: Update an associated metadata instance: a sequence diagram**

<i>Method name</i>	<b>Dissociate</b>	
<i>Return type</i>	Void	
<i>Parameters</i>	<i>Name</i>	<i>Type</i>
	targetSessionID	String
	resourceIdentifier	String

	metadataIdentifier	String
<i>Fault</i>	NO_SUCH_SESSION	
	INVALID_RESOURCE_IDENTIFIER	
	INVALID_METADATA_IDENTIFIER	
	INSUFFICIENT_CREDENTIALS	
	NOT_ASSOCIATED	
	METHOD_FAILURE	

The following faults can occur:

- NO\_SUCH\_SESSION in case the given targetSessionID is invalid.
- INVALID\_RESOURCE\_IDENTIFIER if no resource is identified by the resourceIdentifier parameter.
- INVALID\_METADATA\_IDENTIFIER if no metadata instance is identified by the metadataIdentifier parameter.
- INSUFFICIENT\_CREDENTIALS if the user is not authorized to execute this method;
- NOT\_ASSOCIATED when an attempt was made to dissociate a resource and metadata that were not associated.
- METHOD\_FAILURE if the operation fails for another reason

## 5. Related work

In this section, the Simple Publishing Interface is compared to related specifications in the field of learning object repositories and digital libraries.

### *Fedora*

The Flexible Extensible Digital Object and Repository Architecture (Fedora) [Staples et al., 2003] [Payette and Lagoze, 1998] is an architecture for digital libraries, institutional repositories and learning object repositories. Fedora features a deposit API, through which objects can be deposited on a fedora server.

Persistent ID
Disseminators
System Metadata
Datastreams

**Fig 7: Fedora digital object**

The fedora digital object model packages the following information into a single object that is encoded with the Metadata Encoding and Transmission Standard (METS) [Cantara, 2005].

1. A persistent identifier is a unique identifier. This corresponds to the identifier of a learning object.
2. By using one or more data streams, the fedora object model supports representing compound objects. A data stream is a component that represents MIME-typed content and thus corresponds to a learning object. Using this model, one can for instance represent a book that is modeled using a data stream for each scanned page of the book.
3. System metadata corresponds to the metadata describing the package. These metadata and data are aggregated in one object. Having distributed metadata instances that refer to the same learning object is thus not possible in this model.
4. Disseminators are public services that enable accessing the datastreams. A disseminator can for instance render a graphic image representing content or translate the content into another language.

The fedora deposit API is intended to be deployed on fedora servers only and does not allow for manipulating objects other than fedora objects. The fedora API offers two methods to modify a datastream:

- The `modifyDatastreamByReference` method changes the location a datastream points to. One can use this method to set up a referatory. Unlike `submitResourceByReference`, this method does not retrieve the datastream from the URL that is provided.
- The `ModifyDatastreamByValue` method corresponds to the `submitResourceByValue` method and enables transporting a datastream.

With both methods, one can for instance dynamically add or remove datastreams to an instance.

A datastream in the fedora digital object model corresponds to a learning object. SPI does not specify how one should group different learning objects as one compound object. This however does not imply that with SPI one cannot support scenarios where managing compound information is necessary. The ALOCOM plugin for Microsoft Office exemplifies this. This tool builds on SPI for publishing individual slides, images as well

as entire presentations to a repository and uses the LOM relation field to express that an ALOCOM component is linked to a slide set.

## *PENS*

The Package Exchange Notification Services (PENS) protocol supports a notifications service for content packages [PENS, 2005]. This protocol only supports a sequence of interactions that is similar to the sequence diagram in figure 2. PENS does not support publishing metadata instances, neither does it support sending a content package "by value". Using this notification service a source can announce the location of a package that is available for transport. When an application like a learning management system receives a PENS notification, it can retrieve the package from the URL that is provided. Like SPI and SQL, this protocol makes a distinction between semantic and technical interoperability. The PENS specification contains an abstract data model and provides a binding to the HTTP protocol.

PENS relies on an intermediate service, that is not further specified, where the source can deposit a content package and where a PENS implementation can retrieve the package through a fully qualified URL that it received from the source. A fully qualified URL is a web address starting with a scheme (e.g. ftp:, http: or file:) and enables the PENS service to select a protocol for downloading the content package.

A PENS message contains a receipt parameter that is equivalent to the sourceLocation parameter in setSourceLocation SPI method. PENS specifies that this parameter must be URL and suggest a "mailto:" URL to send an acknowledgement via email. The "mailto:" URL is useful for desktop applications that publish to a PENS service. As these applications often cannot host a web server, the user of the application can receive an acknowledgement via email. With SPI, one can circumvent such a scenario by using "by value" publishing that does not require asynchronous communication.

## *SRU record update*

The SRU [McCallum, 2006] [SRU] Record Update service allows for creating, replacing and deleting metadata records. This specification does not deal with publishing resources and can thus be implemented on a metadata store component only.

SRU record update defines one request with a corresponding response. The request contains the following information. Mandatory parameters are bold faced.

1. **version**. The version of the request. With this parameter a source indicates to the target that it wants the response to be in a version of SRU update less than the submitted version.
2. **action**. Three actions (create, replace and delete) define how the target should interpret the information that is send. The create action corresponds to the submitMetadata method in SPI, the delete actions matches deleteMetadata. In SPI, replacing metadata is possible through a sequence of the following

methods: dissociate (from learning object), deletion (of metadata instance), submission (of a replacement) and associate (with the learning object).

3. `recordIdentifier`. The source is required to create an identifier for the metadata instance that is submitted. This parameter is optional, as SRU allows to include this information as part of the metadata record.
4. `recordVersions`. This parameter expresses versioning information, that allows for tracking changes to a single record. In SPI, versioning information is considered as part of the metadata that is submitted to a metadata store.
5. `record`. This optional parameter is used to transfer a record using either "by value" or "by reference" semantics. Record is optional as it is irrelevant when the action of the request is set to "delete". When a record is available, it includes `recordData` that corresponds to the metadata parameter in the `submitMetadata` method, a `recordSchema` that corresponds to `metadataSchema` parameter in the `setMetadataSchema` SPI method and a `recordPacking` element that defines how records are to sent:
  - The default value is `xml` and indicates that record is encoded as an XML metadata instance.
  - The value `string` indicates that an XML record is encoded as a string and implicates that special characters (e.g. `<` or `>`) are escaped to their entity forms. According to this specification, this is useful when records are not valid and would otherwise break the entire SRU message (that is also encoded in XML).
  - If the value of the `recordPacking` element is `url`, then the value of `recordData` should be a URL that links to the record. SRU record update indicates that submitting a reference is useful to allow a source to send a reference to large records, rather than submitting the record "by value". SPI does not enable "by reference" publishing of metadata as it is not the intent of SPI to submit large metadata instances.

The SRU record update protocol is only usable on a metadata store component. The functionality defined by this protocol is similar to the methods introduced in Section 3. With SRU record update, one can thus support a service that corresponds to the sequence diagram in figure 3.

### *EduSource Communication Layer*

The EduSource Communication Layer (ECL) [Hatala et al., 2004a] [Eap, 2003] [Eap et al., 2004] gives a concrete implementation to the messages proposed in IMS DRI [ims, 2003]. ECL implements a `submit` message for submitting learning objects or metadata to a repository. The store message is returned by a target, acknowledging the submission request. The ECL `submit` message contains the following parameters:

- The *action* parameter is similar to the action parameter in SRU update and defines two submission actions: `save` and `delete`.

- The *content\_type* parameter corresponds to the `setDataFormat` method in SPI and identifies the kind of content that will be submitted (SCORM, IEEE LOM, etc). Unlike the `setDataFormat` method, this parameter can be used to indicate that metadata will be submitted.
- The `username` parameter identifies the owner of the resource.
- A *permission* parameter enables submitting a UNIX based permission value, through which one can specify ownership properties.
- The *groupname* property is related to the latter parameter and enables submitting a UNIX-like group name that owns the content.
- The *content* parameter identifies either the filename of the object that is to be uploaded or contains one or more LOM XML instances.

The ECL way of handling learning object and submissions differs from SPI as follows:

- As ECL builds on IEEE LOM metadata only, it does not include parameters or methods to set the metadata schema or an application profile of LOM.
- ECL does not distinguish between uploading learning objects and metadata. Furthermore, there is no method in ECL for associating metadata and learning objects. [Eap, 2003] points out that ECL assumes the LOM metadata instance to refer to a Learning Object using a URL. This implies that ECL only works with learning objects that are stored on a web server. The `associate` method in SPI offers more flexibility, as a metadata instance can be bound to a learning object using other mechanisms. Furthermore, a source can submit the metadata instance prior to the learning object and does not require the metadata instance to be aware of the location of the learning object. The SPI target is however responsible for adding the identifier, that can be resolved to a location, to the metadata.

### *Open Knowledge Initiative*

The Open Knowledge Initiative (O.K.I) [Hatala et al., 2004b] has specified Open Service Interface Definitions (OSID) through which one can submit assets to a digital repository. The repository OSID defines a JAVA Asset interface that manages both content and records. The asset interface offers methods for adding and deleting records to an Asset. The asset interface defines an `updateContent` method, through which one can associate content to an asset.

Note that O.K.I. only specifies methods through which one can manipulate JAVA objects that implement the OSID interfaces. O.K.I does not specify how an implementation should implement and transport learning objects and metadata to a target.

The asset interface defines an `addAsset` method, through which one can add assets to a given asset. This allows for creating compound objects in O.K.I. SPI assumes that one will use metadata to express these compound relations. In practice for instance, a

LOM implementation can use the `lom.relation.kind` field which has values like "ispartOf". With this metadata field, one can relate learning objects and create compound structures.

Most of the methods defined in O.K.I., implement a `RepositoryException` method. Through using this exception, one can set the message to `UNIMPLEMENTED` and indicate that an OSID implementation does not offer an implementation for this method. With this feature, one can offer support for the flexible application scenarios that were described in Section 1. When offering publishing access to a repository with O.K.I., one can decide not to implement the methods `updateContent` and `getContent`.

The O.K.I. repository OSID relates to SPI as follows:

- O.K.I does not specify whether learning objects are published "by reference" or "by value". When communicating with the server, an implementation can support either of both. However, this is transparent for an O.K.I. client.
- O.K.I enables both publishing learning objects and metadata instances. As a learning object and a metadata instance are connected via an object that implements the asset interface, explicitly connecting them is not necessary. As a consequence O.K.I does not have methods that are equivalent to the `associate` or `dissociate` method, presented in Section 4.

## 6 Experiences

A first version of this specification was implemented in ARIADNE and has been published in [Ternier and Duval, 2003]. In this article, a web services based approach was introduced that facilitates integrating the publishing process into applications where learning objects are either consumed or produced. This deliverable presents a generalized version of this API that takes into account requirements from projects and organisations, such as ALOE, TENCompetence, MACE, MELT, EUN and ARIADNE.

Initial outcomes of this work have been implemented in the TENCompetence project [TENCompetence], ARIADNE [ARIADNE], MACE [MACE] [Prause et al., 2007] and MELT [MELT] and were published in [Demidova et al., 2007].

### *ARIADNE*

The ARIADNE finder, a search and indexation tool through which ARIADNE users, can upload their materials has been extended such that it can publish using the SPI protocol.

The screenshot shows the ARIADNE upload interface. At the top left is the ARIADNE logo with the text 'ARIADNE Foundation for the European Knowledge Pool'. To the right are links for 'Search' and 'Publish', and a user status 'test Logout'. Below this is a header 'Upload Area' and a sub-header 'Licensing details'. The form contains two sections for licensing: 'Allow commercial uses of your work?' with radio buttons for 'Yes' and 'No' (where 'No' is selected), and 'Allow modifications of your work?' with radio buttons for 'Yes', 'Yes, as long as others share alike' (selected), and 'No'. To the right of these sections is the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License logo and text. Below the licensing section is a text prompt 'Select the file you want to publish in the repository', a file input field with a 'Bladeren...' button, and an 'Upload' button.

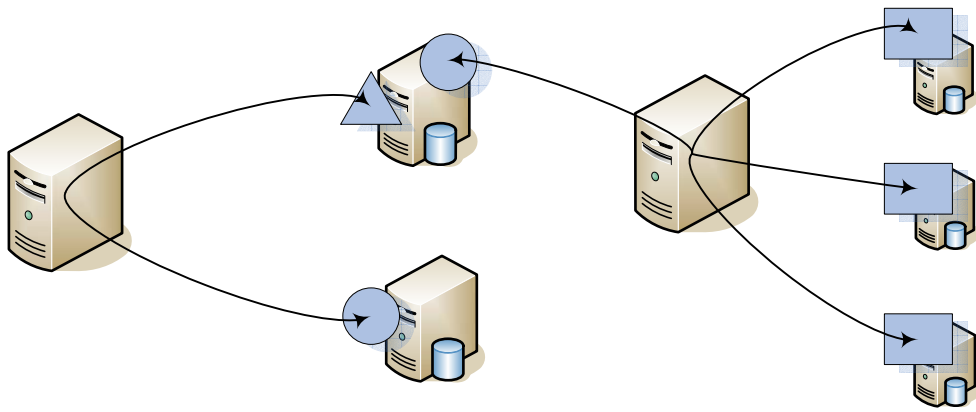
**Fig 8: The ARIADNE finder**

As a result of integrating SPI, this tool is no longer tightly coupled to the ARIADNE learning object repository for publishing learning objects and metadata. Future work with SPI in ARIADNE will focus on:

- Implementing SPI on other repository software. We are currently working on a DSpace implementation of SPI.
- Connecting LMSs (like blackboard and Moodle) to the ARIADNE repository through SPI. Outcomes of connecting LMSs to ARIADNE with an early implementation of a publishing protocol have been published in [Vandepitte et al., 2003] and [Broisin et al., 2005]

## MACE

The objective of the MACE eContentPlus project is to improve architectural education in Europe. This involves integrating a vast amount of learning objects from diverse repositories, the MACE content providers. In order to offer better services, the metadata that describes these existing learning objects requires enrichment. Enrichment involves adding more metadata fields to the metadata instances that describe learning objects. In MACE, for instance, a Global Positioning System (GPS) coordinate is one the metadata fields that has been added to the learning object metadata. This allowed for the creation of query clients that enable the user to search for learning objects via a map. For the purpose of metadata enrichment, an architecture (see figure 9) was designed that enables the MACE community to enrich learning objects.



**Fig 9: The MACE enrichment architecture**

In a first phase of this project, all metadata available at the different MACE content providers, was mapped to the common MACE application profile and exposed through an OAI-PMH [Lagoze and Van de Sompel, 2001] interface (ICONDA, DYNAMO and WINDS boxes in figure 9). A harvesting component harvests this metadata on a regular basis and stores the harvested metadata via SPI (circle in figure 9) in the MACE harvested metadata store. Next, the MACE community can enrich metadata through a graphical user interface (MACE enrichment toolkit). With this toolkit, users can select metadata items via SQL (triangle in figure 9) and push them into the enriched metadata store via SPI.

For this architecture, both the harvested metadata store and the enriched metadata store were realized reusing the ARIADNE Learning Object Repository components. Using SPI was beneficial for the MACE project as the protocol provides an interface that is not tied to a given learning object repository or technology. This enabled reusing the ARIADNE components and by reusing this software, it was not necessary to implement a new MACE metadata store from scratch.

# MACE enrichment toolkit

## 7. Conclusion

In this deliverable, a protocol was presented for publishing learning objects and metadata to learning object repositories. The Simple Publishing Interface is less mature than the Simple Query Interface. SQI already went through a formal CEN Workshop standardization process and has been endorsed by the experts of CEN. At the time of writing, SPI has been proposed to and was accepted by the CEN ISSS WS/LT workshop as a new work item. The current document will serve as an input for this work.

SPI enables both semantic and syntactic interoperability. Sections 2, 3 and 4 define an abstract model that provides semantic interoperability. A partial WSDL binding has already been created [SPI] and was implemented in MACE, MELT, TENCompetence and ARIADNE. This WSDL will be further extended such that all SPI methods can be supported.

The SPI protocol supports multiple flexible applications as specified in Section 1. Because of the separation between methods that transport content and metadata, SPI can support referatory scenarios, content only scenarios and scenarios where repositories manage both metadata and learning objects.

## 8. References

- [ARIADNE] The ARIADNE Foundation for the European Knowledge Pool.  
<http://www.ariadne-eu.org>.
- [CQL] CQL: Contextual Query Language.  
<http://www.loc.gov/standards/sru/specs/cql.html>
- [MACE] Metadata for Architectural Contents in Europe (MACE).  
<http://www.mace-project.eu/>.
- [MELT] Metadata Ecology for Learning and Teaching (MELT).  
<http://melt-project.eun.org> .
- [PENS, 2005] PENS (2005). Guidelines for Package Exchange Notification Services.
- [SPI] The SPI wiki.  
<http://ariadne.cs.kuleuven.be/lomi/index.php/SimplePublishingInterface>
- [SRU] Search/Retrieve via URL Record Update.  
<http://www.loc.gov/standards/sru/record-update/> .
- [TENCompetence]. TEN Competence. <http://www.tencompetence.org>.
- [IMS, 2003] (2003). IMS Digital Repositories Interoperability - Core Functions Information Model.
- [Broisin et al., 2005] Broisin, J., Vidal, P., Meire, M., and Duval, E. (2005). Bridging the Gap between Learning Management Systems and Learning Object Repositories: Exploiting Learning Context Information. *aict-sapir-elete*, pages 478–483.
- [Cantara, 2005] Cantara, L. (2005). METS: The metadata encoding and transmission standard. *Cataloging & classification quarterly*, 40(3-4):237–253.
- [Demidova et al., 2007] Demidova, E., Krger, P., Olmedilla, D., Ternier, S., Duval, E., Dicerio, M., Mendez, C., and Stefanov, K. (2007). Services for knowledge resource sharing & management in an open source infrastructure for lifelong competence development. *icalt*, 0:691–693.
- [Eap, 2003] Eap, T. (2003). The communication middleware for collaborative learning object repository network. PhD thesis, Simon Fraser University, Irvine.
- [Eap et al., 2004] Eap, T., Hatala, M., and Richards, G. (2004). Digital repository interoperability: design, implementation and deployment of the ecl protocol and connecting middleware. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 376–377, New York, NY, USA. ACM Press.

- [Hatala et al., 2004b] Hatala, M., Richards, G., Thorne, S., and Merriman, J. (2004b). Closing the Interoperability Gap: Connecting Open Service Interfaces with Digital Repository Interoperability. In Lorenzo Cantoni and Catherine McLoughlin, editors, *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2004*, pages 78–83, Lugano, Switzerland. AACE.
- [Lagoze and Van de Sompel, 2001] Lagoze, C. and Van de Sompel, H. (2001). The open archives initiative: building a low-barrier interoperability framework. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 54–62, New York, NY, USA. ACM.
- [McCallum, 2006] McCallum, S. H. (2006). A look at new information retrieval protocols: SRU, OpenSearch/a9, CQL, and XQuery. In *World Library and Information Congress: 72nd IFLA General Conference and Council*. IFLA, IFLA. Published at URL <http://www.ifla.org/IV/ifla72/papers/102-McCallumen.pdf>.
- [Payette and Lagoze, 1998] Payette, S. and Lagoze, C. (1998). Flexible and Extensible Digital Object and Repository Architecture (FEDORA). In *Research and Advanced Technology for Digital Libraries: Second European Conference, ECDL'98*, pages 41–59, Heraklion, Crete, Greece. Springer Berlin / Heidelberg.
- [Prause et al., 2007] Prause, C., Ternier, S., de Jong, T., Apelt, S., Scholten, M., Wolpers, M., Eisenhauer, M., Vandepitte, B., Specht, M., and Duval, E. (2007). Unifying Learning Object Repositories in MACE. In *First International Workshop on Learning Object Discovery & Exchange (LODE'07)*, Crete, Greece.
- [Staples et al., 2003] Staples, T., Wayland, R., and Payette, S. (2003). The Fedora Project. An Open-source Digital Object Repository Management System. *D-Lib Magazine*, 9(4).
- [Ternier et al., 2003a] Ternier, S., Duval, E., and Neven, F. (2003a). Using a P2Parchitecture to provide interoperability between learning objects. In *Proceedings of ED-MEDIA 2003 World Conference on Educational Multimedia, Hypermedia, and Telecommunications*, pages 148–151. AACE, AACE.
- [Vandepitte et al., 2003] Vandepitte, P., Van Rentergem, L., Duval, E., Ternier, S., and Neven, F. (2003). Bridging an LCMS and an LMS: a Blackboard building block for the Ariadne knowledge pool system. In *Proceedings of ED-MEDIA 2003 World Conference on Educational Multimedia, Hypermedia, and Telecommunications*, pages 423–424. AACE, AACE.