PEER Use of SWORD deposit protocol

Table of contents

Document information

Title: The SWORD Protocol
Subject:
Moderator:
Version:
Date published:
Excerpt: Write an excerpt here

(Optional information)
Type:
Format:
Identifier:
Language:
Rights:
Tags:

Document History

Date	Version	Owner	Changelog	PDF

Abstract

The abstract describes what the application profile is about. It should contain a problem definition, the standards described by the application profile and the goal of the application profile.

This document has been created within the EU funded PEER project as an appendix in deliverable 2.2 with the effort of many others and SURFfoundation.
The application for this standard has not been adopted by and implemented in the Dutch repositories. For the sake of reuse this document is put in the list of standards becouse it is being used by other repositories, and has the potential to increase in imlementations.

1 Introduction

In the PEER project, selected stage-2 material from publishers is being transferred to or deposited into the PEER Depot after which the content is being transferred from the depot to multiple, publicly available repositories.
The stage-2 material will be transferred in a Submission Information Package (SIP) containing the full-text publication, metadata and the complementary stage-2 source files. The SWORD AtomPub profile contains specific features that allows for an application-level deposit of material into repositories.
The PEER information model can be mapped onto the OAIS Reference Model and the DRIVER object model for Enhanced Publications.
Implementers may set up their own server conforming to these guidelines using one of repository specific implementations available from SourceForge, or write their own custom implementation either using the generic Java library, also available from SourceForge, begin their implementation from scratch.
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.
It is assumed that the reader of this document has knowledge of the PEER D2.1 report PEER D2.1 *Draft report on logfile harvesting systems and manuscript deposit procedures for publishers and repository managers*, http://www.peerproject.eu/reports/http://www.peerproject.eu/reports/, SWORD profile v1.3 Allinson, J et al 2008, *SWORD AtomPub Profile version 1.3*, viewed 25 March 2009 http://www.swordapp.org/docs/sword-profile-1.3.htmlhttp://www.swordapp.org/docs/sword-profile-1.3.html , the OAIS Open Archival Information System. Reference Model Consultative Committee for Space Data Systems 2002, *OAIS Reference Model* http://public.ccsds.org/publications/archive/650x0b1.pdfhttp://public.ccsds.org/publications/archive/650x0b1.pdf and the DRIVER Digital Repository Infrastructure Vision for the European Region. II Enhanced Publication object model and Functionalities Verhaar, P & Place, T 2008, *Report on Object Models and Functionalities*, DRIVER II D4.2..

1.1 SWORD overview

The SWORD AtomPub Profile is an application profile of the Atom Publishing Protocol (APP) (RFC 5023) Internet Engineering Task Force 2007, *The Atom Publication Protocol*, RFC 5023, Internet Engineering Task Force, http://tools.ietf.org/html/rfc5023http://tools.ietf.org/html/rfc5023 that contains specific features that allows for an application-level deposit of material into repositories.
The APP is based on the HTTP transfer of Atom-formatted representations. It is easy to think of APP as a way of publishing just Atom Syndication Format feeds. While it is true that APP provides the means to publish Atom Syndication Format Entries to collections (such as blogs), it also provides a mechanism for the publishing of binary formatted data called Media Resources in APP context (Internet Engineering Task Force 2007). While in the blog scenario this mechanism may be used to add attachments to a blog post i.e. images, audio, video, documents), SWORD exploits this for the publishing (or deposit) of material into repositories, usually in some form of content packaging in which data and descriptive metadata are being held together in one container (see Figure 13).

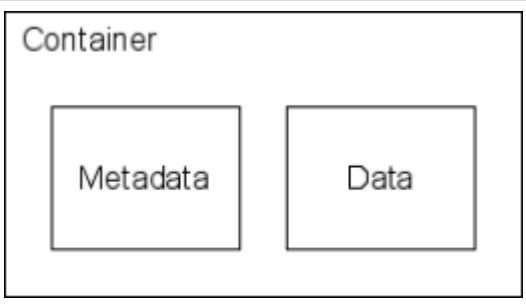

Figure 13: Content Package or Container

An example of an implementation of such a container would be a ZIP-file containing a full-text manuscript in the PDF/A-1 format and descriptive metadata in the TEI-XML format.
The container is being submitted by a client to a SWORD interface service (server) as a bit stream using a HTTP POST request consisting of a header containing information about authorisation and the bit stream (type and format of the container) in order for the server to be able to interpret the bit stream properly, and a body part containing the bit stream itself (see Figure 14). Upon reception, the server sends a HTTP response back to the client - again consisting of a header and a body part - with the header containing a HTTP status code indicating a success or failure of the attempted deposit according to regular HTTP semantics, and a response document containing additional APP/SWORD specific information about the deposit being made.

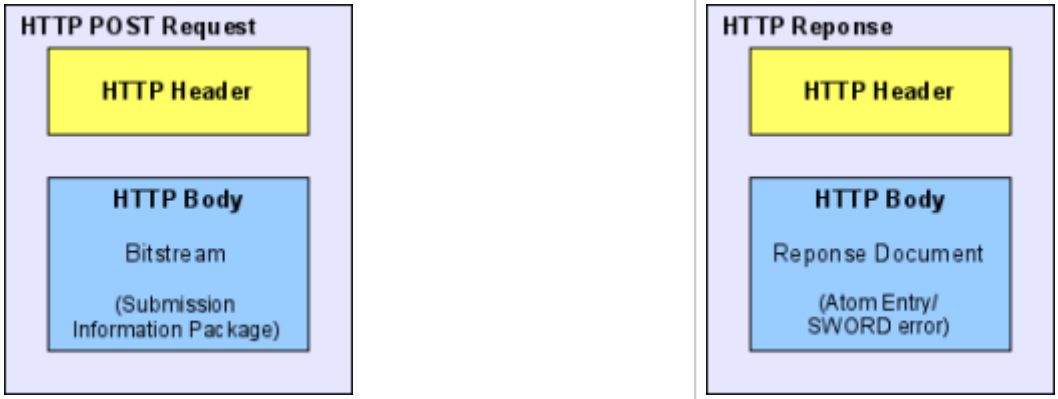

Figure 14: HTTP request and response structure in the SWORD context

1.2 Use of SWORD in PEER

In the PEER workflow there are two scenarios of deposits into the PEER repositories specified: deposit made by PEER and deposit made by authors (see Figure 15)

Publicly Available PEER Repositories

Eligible Journals / Articles
PublishersPEER DepotAuthorsSelect100 % Metadata50 % ManuscriptsPublishers Transfer50 % ManuscriptsPublishers DepositPublishers InformTransferDepositUGOEKTUMPGINRIAULDUNIBITransferLong-Term Preservation: LTP Depot(e-Depot, KB)

Figure 15: PEER Workflow

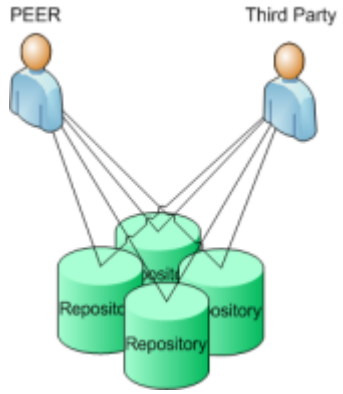

Figure 16: Deposit situation

This results in an n:n-relation between repositories and deposit sources either the PEER Depot or third party services operated by an author (see Figure 16). To prevent multiple tailored solutions and implementations it is important to define a standard process for the deposit of material into repositories.
The processes may be categorised into two types of mechanisms: **push and pull**. An example of the **pull** mechanism is the KB's mechanism of the eDepot harvesting repositories through OAI-PMH and pulling content using a webclient (see Figure 17) which downloads the objects specified in the location entries in the metadata.

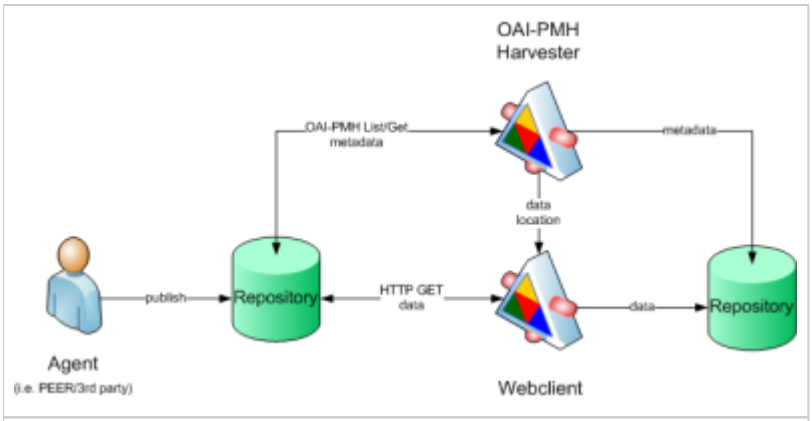

Figure 17: OAI-PMH data harvest

An example of the **push** mechanism is the SWORD deposit mechanism where the data is being pushed by an agent (i.e. a webservice or desktop application representing a user) to the SWORD interface of a repository which then accepts or rejects the deposit (see Figure 18).

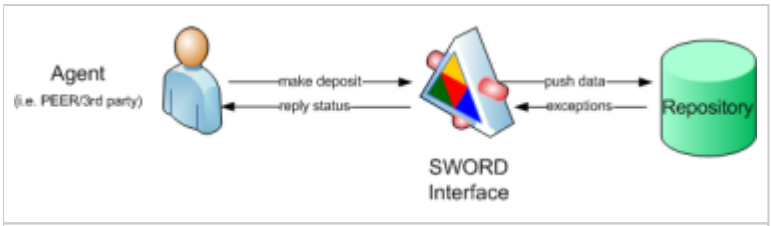

Figure 18: SWORD data deposit

Finally, a third, hybrid mechanism can be created by setting up an FTP server to which deposits can be uploaded (pushed) by an agent. A repository may then pull the FTP content which is then being pulled into the repository (see Figure 19).

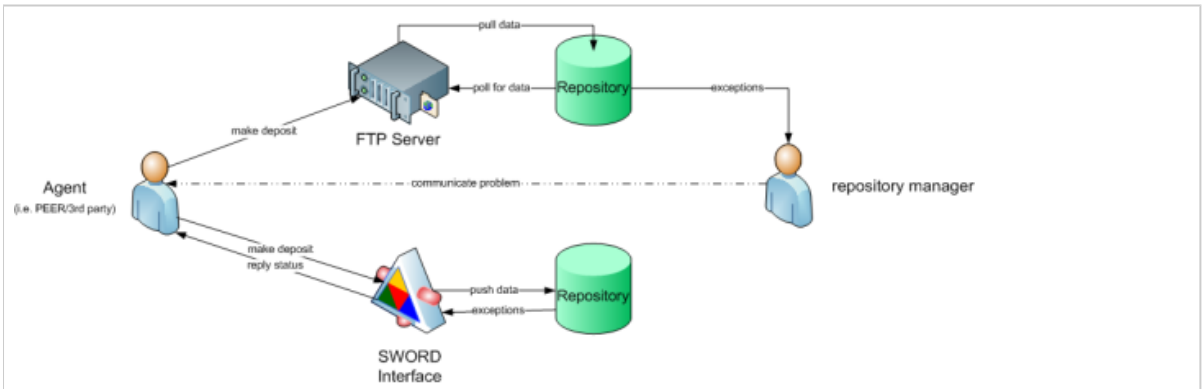

Figure 19: SWORD versus FTP

A disadvantage of this mechanism is that this only provides direct feedback to the agent about status of the upload, not of the status of the actual deposit into the repository. This may lead to the situation when an agent successfully uploads data to the FTP server, but the data is being rejected by the repository afterwards because it does not adhere to rules the repository enforces on its contents without the agent being informed about this rejection - something that is not the case when using SWORD.
Figure 20 provides a schematic overview of the use of SWORD in the PEER deposit scenario. Here a publisher transfers manuscripts and metadata into the PEER Depot where the manuscripts and metadata are being converted and crosswalked to the formats specified for the PEER deposit process. The converted and crosswalked manuscripts and metadata are then being packaged into a container and sent to the SWORD interface service of a repository where the contents are being unpacked from the container. Upon reception these MAY be converted and crosswalked into an internal storage format before they are being archived into the repository.

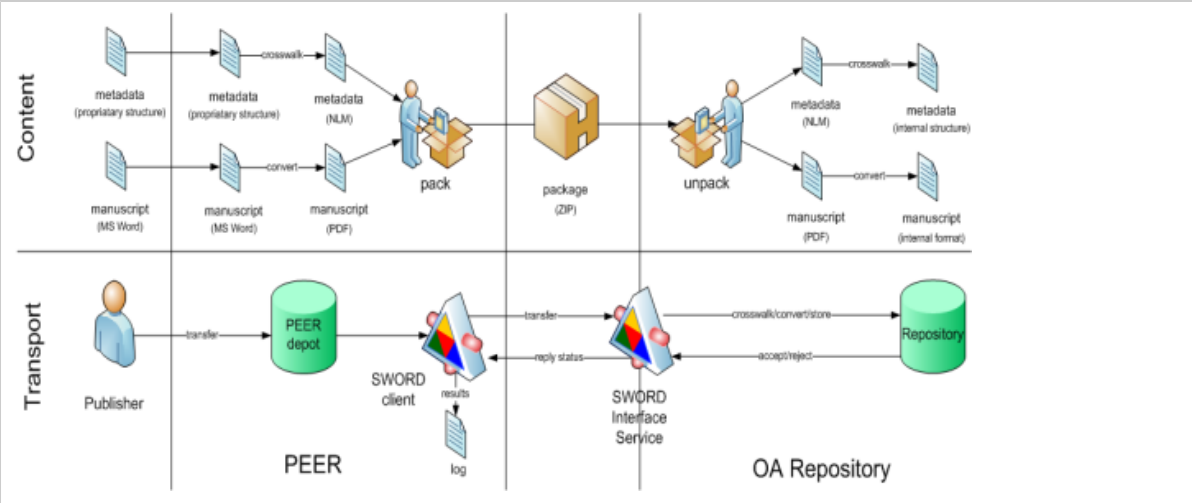

Figure 20: SWORD use in PEER for PEER Depot

2 Use of SWORD features

2.1 About this section

This section will describe the use of the SWORD profile in the context of the PEER project. The contents are organised according and supplementary to the document SWORD Atom Pub Profile version 1.3 part A. If a SWORD profile section or feature is omitted, implementations MUST behave as defined in SWORD profile.

2.2 Package Support

The PEER Submission Information Package (SIP) MAY be expressed using (a combination of) different formats (i.e. XML containers or RFC 1951 compliant ZIP archives) and/or serialised using different structural models (i.e. DIDL, METS, ORE, TEI, NLM, MODS, DC). The mappings between the SIP, its components and the formats and structures will be defined and expressed using specialised application profiles developed in the PEER context.

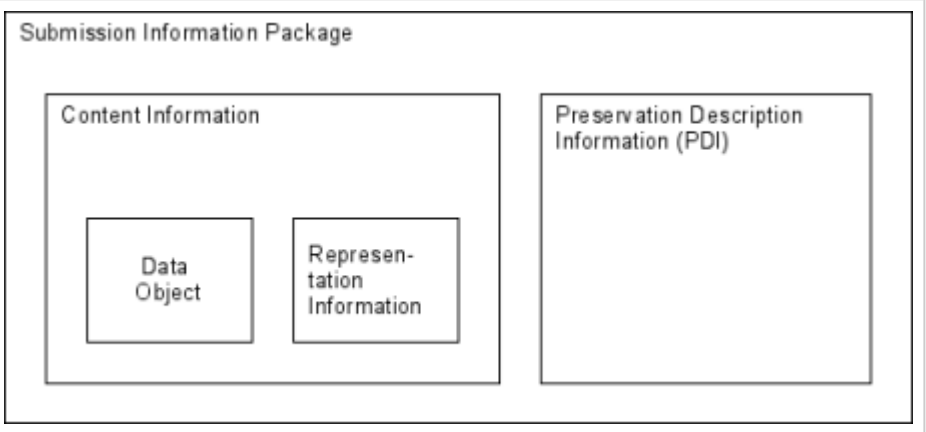

Figure 21: Submission Information Package structure

The SWORD profile offers the possibility to enumerate multiple packaging formats in the Service Document and supply a Quality Value attribute indicating a preference and level of support for a designated package format.

2.2.1 Package support in Service Description

The server MAY support multiple packaging formats with varying quality values according to the support of the PEER Submission Information Package (SIP).
The server MUST support at least one package format with Quality Value "1.0", indicating full support where all components supplied within the SIP will be processed and understood when using the designated package format.
All supported formats MUST be listed in the Service Document.
All formats listed in the Service Document MUST have a Quality Value attribute assigned.
The value used in the <sword:accepted Packaging> element MUST NOT overload any values enumerated in the SWORD Content Package Types.
The server MAY use the <sword:service> element in the Service Document to indicate the existence of other service interfaces supporting additional package formats.
The server SHOULD NOT accept a specific package format across multiple interfaces with different levels of support as indicated by the Quality Value attribute in the Service Document.

2.2.2 Package Support during Resource Creation

If a server receives a POST request with a format that is not listed as an accepted format in the Service Document, the server MUST reject the package by returning an HTTP status code of 415 (unsupported media type).

2.2.3 Package description in entry documents

When describing packaged resources in Media Entry documents, the server SHOULD add sword:packaging elements to the entry.

2.3 Mediated Deposit

The following paragraph is considered informative, but is included for clarity in the use of the SWORD profile outside the PEER project.
The PEER workflow offers two ways a manuscript can be deposited into one of the publicly available PEER repositories: either by publisher deposit (through the PEER Depot) or by author deposit (where the publisher informs the author who deposits his/her article(s) in the actual publicly available repository PEER D2.1 Draft report on logfile harvesting systems and manuscript deposit procedures for publishers and repository managers, p.8, http://www.peerproject.eu/reports/.).
For the author deposit, the author MAY make the deposit by proxy through a web service (i.e. by filling in a form to provide the metadata and upload a file containing the full-text material) after which the web service is making the actual deposit. The web service MAY not be used for the PEER project exclusively in which case the web service MAY use its own credentials to authenticate at the server (at the repository side).
Figure 22 depicts an example of the use of this mechanism in the PEER context. Note that the greyed out parts of the figure are considered outside the scope of the PEER project.

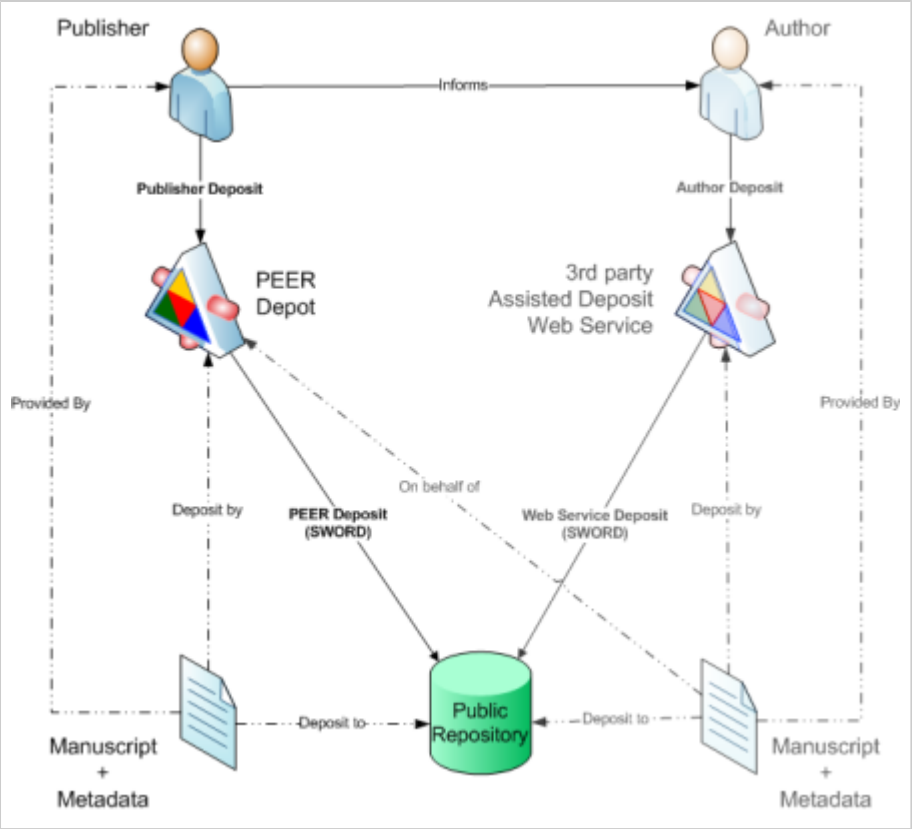

Figure 22: PEER deposit workflow

It is recognized that the repository MAY want to keep track of data that is being deposited within the PEER context by creating a single user account to the PEER Depot. This then covers the publisher deposit workflow, but does not provide for a solution for the case of author deposit through another web service which MAY use different credentials.
A possible solution MAY be the use of mediated deposit where a client authenticates using its assigned credentials on behalf of another known user (e.g. a web service authenticates using its own credentials and makes the deposit on behalf of the PEER user which is used by the PEER Depot).
This method MAY also be used to authenticate on behalf of other users (i.e. authors, librarians, data stewards, research assistants, etc.) that already have a valid user account at the repository.
The use of mediated deposit is considered OPTIONAL and is currently not implemented in the application of the SWORD profile within the PEER project.

2.3.1 Mediation in Service Description

Servers supporting mediated deposit MUST indicate this by including a SWORD:mediation element with a value of "true" in the Service Document as defined in the SWORD profile version 1.3 section 2.1.
For servers that do not include a SWORD mediation element in the Service Document, a default value of "no" SHOULD be assumed by clients.

2.4 Auto-discovery

AtomPub makes no recommendations on the discovery of Service Documents.
The SWORD profile states that it is RECOMMENDED that server implementations use an <html:link rel="sword" href="[Service Document URL]" /> element in the head of a relevant HTML document to assist with service discovery.
In addition, it is RECOMMENDED to also include an <atom:link rel="sword" type="application/atomsvc+xml" href="[Service Document URL]"/> element in relevant response documents such as Error Documents.

2.5 Nested Service Descriptions

Nested Service Descriptions MAY be used to specify alternative collections for both organisational (i.e. generic collection with a nested PEER specific collection) and technical purposes (i.e. a specific interface or service instance to cater for specific types of content packaging).

3 Use of APP features

The contents of the following section are organised according and supplementary to the document SWORD Atom Pub Profile version 1.3 part B. If a SWORD profile section or feature is omitted, implementations MUST behave as defined in SWORD profile.

3.1 Securing the Atom Publishing Protocol

The SWORD profile states servers SHOULD support the use of HTTP Basic Authentication over TLS. Because from a trust perspective it is important to confirm the identity of the PEER Depot during the deposit proces, this statement is considered insufficient for the purposes of the PEER project.Therefor this requirement has been restated as follows:
Servers implementing SWORD MUST support HTTP Basic Authentication (RFC 2617) over TLS (RFC 2818).

3.2 Creating and Editing Resources

When depositing resources using SWORD, resources are created by a server when a client makes an HTTP POST request with the resource in the HTTP request body. If the deposit is made successfully, the server then gives a HTTP reponse with the HTTP 201 Status code in the header of the response indicating the resource has been successfully created at the repository side.
Servers returning a HTTP 201 status code after a deposit MUST preserve the resource deposited.
Clients receiving a HTTP 201 status code MUST consider the resource deposited as being accepted for storage by the repository.

3.2.1 Asynchronous treatment of resources

It MAY however be the case that the repository implements an additional asynchronous validation process after which a resource MAY or MAY NOT be accepted. This for instance is the case when a repository uses an intermediate repository where resources deposited through the SWORD interface are temporarily stored, after which they will be moved to a final location within the repository when they are properly validated by a repository manager. When a resource is then being rejected by the repository during the validation process after the server has sent an HTTP 201 response to the client, the situation MAY arise where the client considers the resource as being successfully deposited into the repository, while in fact the resource is NOT being stored into the repository. This situation is viewed as undesirable.
Servers implementing an asynchronous validation process MUST return an HTTP 202 Accept response code indicating the request has been accepted for processing, but the processing has not been completed.
Clients receiving a HTTP 202 status code upon deposit of a resource MUST consider the resource deposited as NOT being stored into the repository.
RFC2616 states that there is no facility for the re-sending of status codes. Therefore, a client will not receive a notification of the outcome of the processing carried out by the server. In order to allow clients to retrieve the outcome of the deposit, the sword:treatment element MAY contain the status of the processing of the deposited resource.
Servers implementing HTTP 202 status codes MUST supply a permanent link to the Atom Entry document of the response.
Servers implementing HTTP 202 status codes MUST update the sword:treatment element of the Atom Entry document of the resource with the status of the processing of the deposited resource.
Client SHOULD implement a mechanism to confirm the successful deposit by periodically checking back at the server with an HTTP GET request to the permanent link supplied by the server, in order to check the contents of the sword:treatment element of the Atom Entry describing the deposited resource when a HTTP 202 status code has been received upon deposit.

4 PEER Object Model

In order to future proof agreements and guidelines for a technical model, it is important to detach the technical implementation from the abstract object and information model. Furthermore it is important to keep this abstract model aligned with other developments in the area the model will be used in. For PEER, there are two of such developments:
OAIS Reference Model for its use by the KB
DRIVER object model for Enhanced Publications for its use in DRIVER context
In PEER, manuscripts and metadata will be transferred between authors, publishers, the PEER Depot, Open Access repositories and an LTP repository exploited by the KB.
This results in a PEER object consisting of a manuscript object which is being described by one or more metadata objects (see Figure 23).
The D2.1 report provides an exhaustive metadata field set to be used in the PEER project (see Table 5, below).

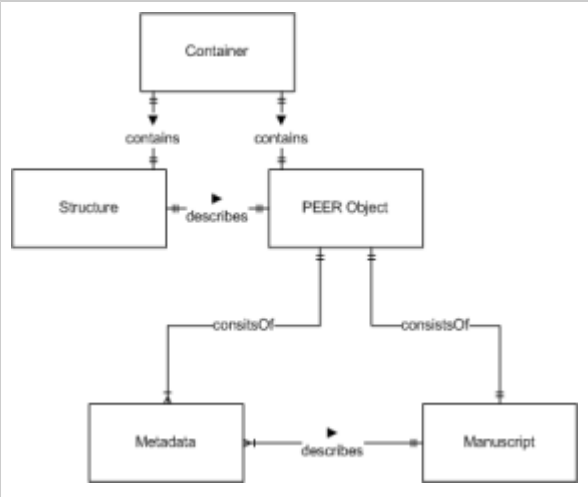

Figure 23: PEER Object model ERD

Field Name	Semantics	Syntax
Title	Article Title	
Creator	Corresponding author's name	Last name, first name
AuthorEmail	Corresponding author's e-mail address	
Description	Abstract	
Date	Date of publication	ISO 8601:2004 ; yyyy-MM-dd
Identifier	DOI of published article	
Coverage	Geographic location of the contributing Author	ISO 3166-1-A2
Journal	Journal title	
Affiliation	multi-tier organisation list	Country, Organization, Laboratory
ISSN		
Volume		
Issue		
Page		
Type	Semantic type of the publication	info:eu-repo/semantics/article info:eu-repo/semantics/acceptedVersion defaults to article.
Subject	Subject headings; Scientific classification (defaults to what is provided in the general STM Journal table See Appendix A.)	
Language	Language of the publication	ISO 639-3 (defaults to 'eng')
Embargo	Embargo Period (defaults to what is provided in the general STM Journal table[5])	

*Table 5: PEER information model*For deposit the PEER object will be packaged into a container. The OAIS reference model specifies the Submission Information Package (SIP) as a specialised Information Package (IP) - which is used by the KB in the eDepot - for submission purposes (see Figure 24).

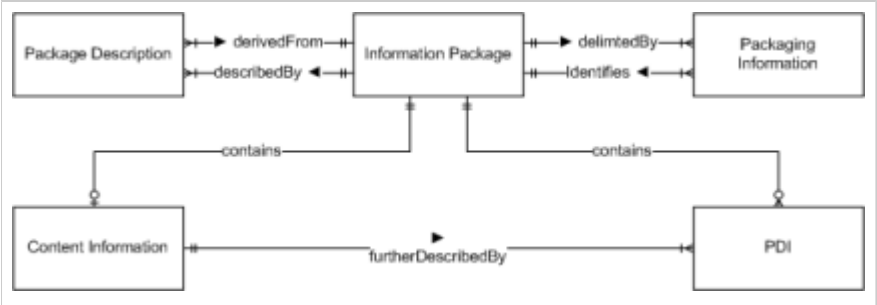

Figure 24: OAIS Information Package ERD

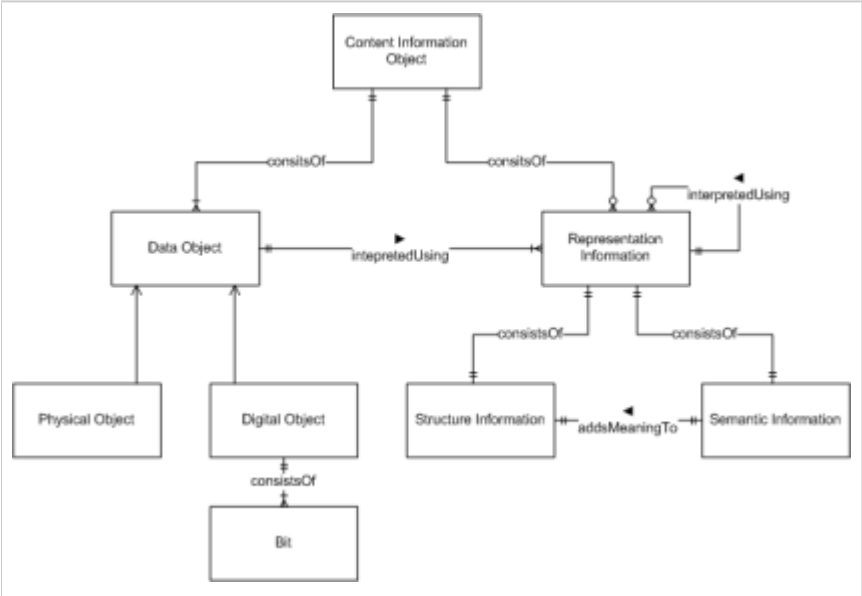

Figure 25: OAIS Content Information Object ERD

An IP consists of content information that is being described by Package Description Information (PDI).
The content information object is defined as a data object (i.e. PDF file) interpreted using representation information (i.e. mime-type, encoding version, etc.) (see Figure 25). Note the structure information being a part of the representation information.
The PDI (see Figure 26) contains Reference Information (i.e. bibliographic descriptions and persistent identifiers), Provenance Information (i.e. information about the conversion process), Context Information (i.e. reference to the research project a publication is based on) and Fixity Information (i.e. a checksum).

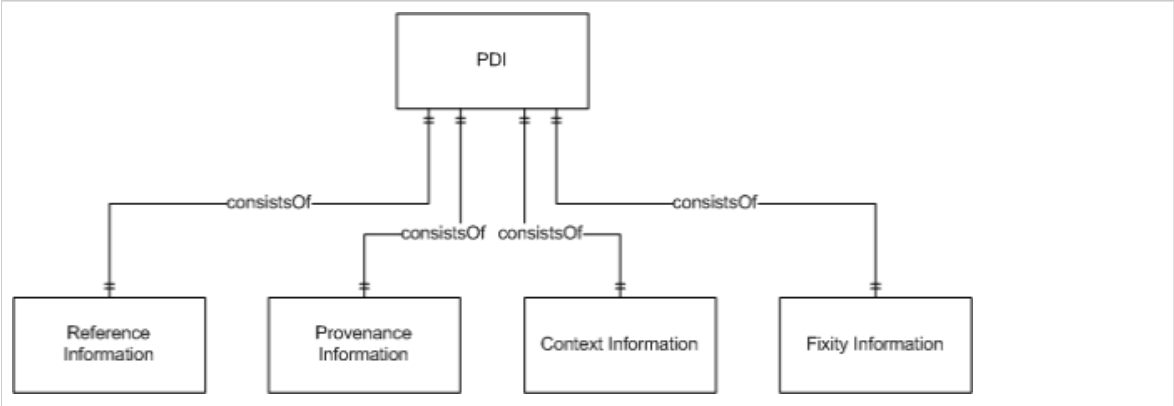

Figure 26: OAIS Package Description Information ERD

The PEER information model can be mapped onto the OAIS Reference Model as depicted in Figure 27. Here the structure object containing the structural information is being added to the PEER object model.

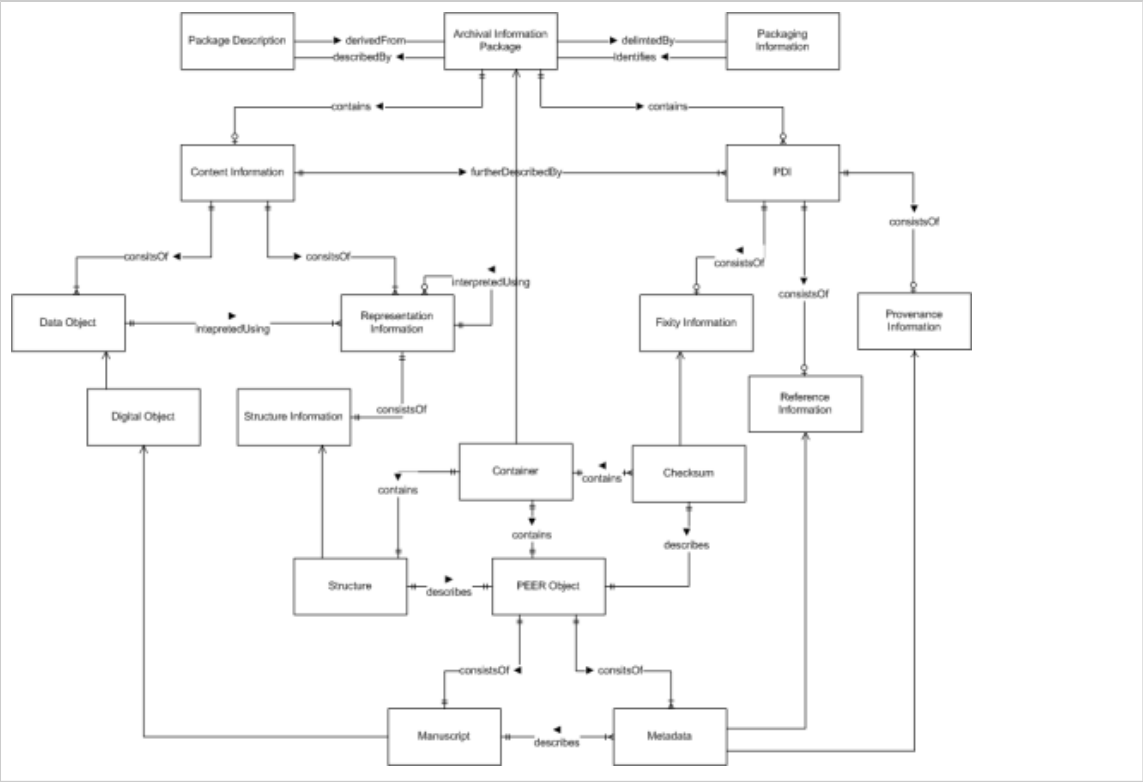

Figure 27: OAIS Reference Model-PEER Information Mapping

Figure 28 depicts a technical mapping of the PEER object model. The structure is expressed using the ORE abstract data model which is serialised as an Atom feed. This Atom feed is to be contained in an XML file. The metadata is serialised in TEI, again contained in an XML file. The manuscript is encoding in PDF/A, contained in a PDF-file. The XML file containing the ORE Atom feed, the XML file containing the TEI document and the PDF file containing the manuscript are then being packaged in a compliant ZIP-file. Upon deposit using SWORD, the ZIP-file is being placed into the body of the HTTP POST request. The Header contains an MD5 checksum and MAY contain authorisation information (see Figure 29).
The HTTP POST request is then being sent to the SWORD Interface Service as described in paragraph 1.2.

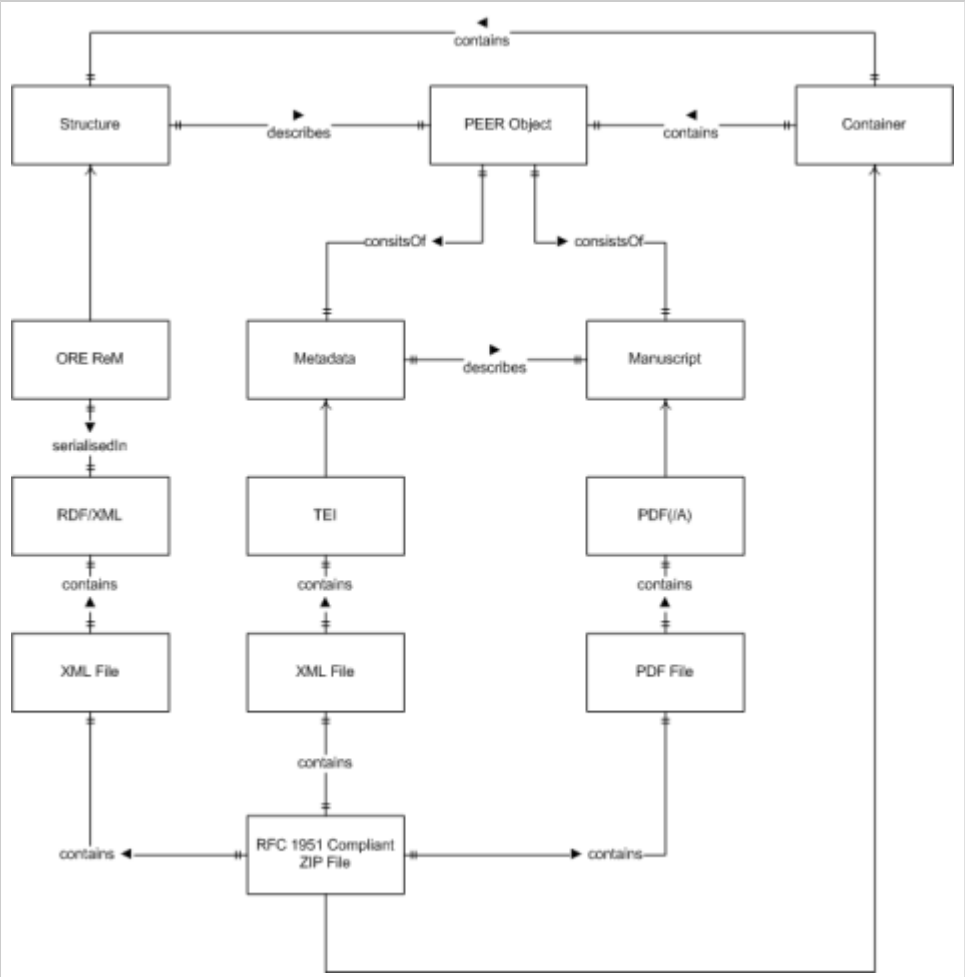

Figure 28: Technical Mapping of the PEER model

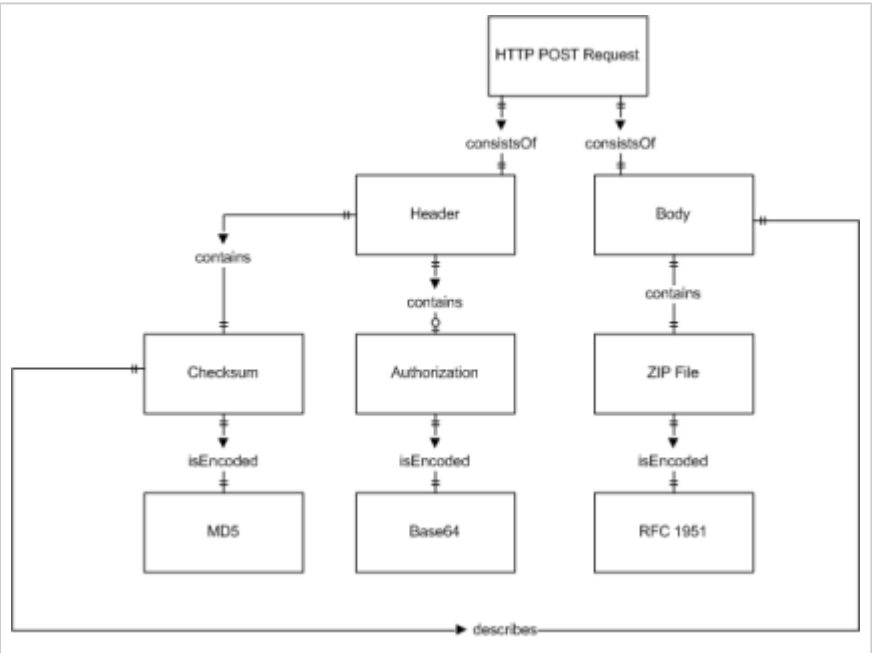

Figure 29: HTTP Mapping of the Technical Model

5 Implications on Repository level

5.1 Overview of the technical process

Generally speaking, the technical process of deposit can be broken in sequential order to the serialisation and deposit request sub-processes on the client side (i.e. the PEER Depot) and the de-serialisation, response and store sub-processes on the repository side (i.e. publicly available repository).

5.1.1 Serialisation - Client side

The serialisation processes involve the serialisation of the metadata from an internal storage to a specific (agreed upon standard) metadata field set and structure (i.e. DC) and the packaging of the metadata and object file(s) into a content package (i.e. MPEG-21 DIDL XML containers or compliant ZIP archives) which MAY include adding a manifest describing contents and their correlation (i.e. the relation between an XML file containing the descriptive metadata of a full-text publication and a PDF-file containing the actual full-text publication) to a bit stream.

5.1.2 Deposit Request - Client side

The deposit request process includes a client posting the data to a service (i.e. the HTTP POST request in SWORD) and the server receiving the data and placing it into a temporary storage (either in memory or on disk).

5.1.3 De-serialisation - Repository side

In the de-serialisation process the receiving server tries to interpret (decode) the bit stream again, in essence validating the contents. This MAY include the unpacking (when using ZIP archives) or decoding (when using XML containers) of the bit stream to be able to interpret the individual contents. It MAY also include the mapping or crosswalking of the metadata structure to an internal (proprietary) metadata field set and/or structure. This process MAY not necessarily be taking place in the actual interface service; it MAY include the sending of the bit stream to an internal storage service which then indicates a success or failure to the deposit service.

5.1.4 Response - Repository side

After the contents are being de-serialised successfully and the server confirms the contents of the received bitstream, the server MUST reply its status to the client (when using SWORD this means reporting the appropriate HTTP status code and correct Atom/SWORD response document). If the deserialisation process has failed for whatever reason, the server SHOULD reject the deposit request to indicate an unsuccessful deposit to the client, or accept the deposit with an HTTP 4xx status code and appropriate exception message indicating a partial successful deposit.

5.1.5 Store - Repository side

The final step of the deposit process includes the storage of the received (meta)data into the internal (meta)data store. This part of the process is implementation specific and considered outside the scope of this document.

5.2 Functional Requirements

A repository implementing a SWORD interface service in the PEER context MUST be able to:

- authenticate a user
- receive, process and respond to an HTTP POST request as specified in this document
- interpret and store a PEER Submission Information Package as specified by the PEER project

5.3 Implementation Steps

The implementation steps can be broken down into the implementation and exposure of the web service to the outside world and interface with the repository on the inside.
Depending on specific needs, an implementer of the SWORD profile may either choose to make an implementation by using one of the repository specific implementations available for DSpace, ePrints and Fedora on Sourceforge SWORD Project, *SourceForge.net: SWORD - Project Web Hosting - Open Source Software*, viewed on 25 March 2009, http://sword-app.sourceforge.net/http://sword-app.sourceforge.net/ or to write a custom SWORD server implementation (optionally by using the generic Java library also available from Sourceforge).
For the repository specific option please refer to the documentation provided with the designated packages.
The second option either involves writing a service from scratch or use the source code available from the SWORD Java library. This library contains ready to implement code for writing servers and clients.
In addition to creating the web service which behaves according to the guidelines specified in this document, special attention should be paid to the creation of crosswalk rules to map the expression(s) of the PEER SIP to the internal repository data structure and semantics.

5.4 Common implementation faults

In the initial testing phase, a number of common implementation faults have been identified. This paragraph will provide a brief overview of these findings and provide guidelines in order to avoid these mishaps.

5.4.1 Mandatory fields for Atom Entry

A number of issues have been identified with regard to the contents of the Atom Entry within the service response documents.
RFC 5023 (Atom Publishing Protocol) states:
*"Implementers are asked to note that [RFC4287] specifies that Atom Entries MUST contain an atom:summary element. Thus, upon successful creation of a Media Link Entry, a server MAY choose to populate the **atom:summary** element (as well as any other mandatory elements such as **atom:id**, **atom:author**, and **atom:title**) with content derived from the POSTed entity or from any other source. A server might not allow a client to modify the server-selected values for these elements."*

5.4.2 Field Semantics

All contents of the Atom Entry document are to be interpreted as within the SWORD context. For example, atom:author states the author of the transaction (i.e. authenticated user), not that of the contents being transported (i.e. author of the publication).

5.4.3 Atom:Id

The contents of the atom:id field MUST be encoded using IRI (International Resource Identifier) as defined in RFC3987.
Valid example:
<atom:id>http://hdl.handle.net/2437.2/20</atom:id>
Invalid example:
<atom:id>1234</atom:id>

6 Locations

When the server has processed the submission information package containing the fulltext and metadata files, a number of locations may be identified:

1. Location of the Media Link Entry
2. Location of the original submission information package
3. Location of the fulltext
4. Location of the metadata

...
SWORD APP Profile v1.3 states:
"The Location element of the HTTP header response MUST contain the URI of the Media Link Entry, as defined in ATOMPUB. The Media Link E ntry URI MUST dereference, and MUST contain an atom:content element with a src attribute containing a _URI."_Example:

```
HTTP/1.1 201 Created
Date: Mon, 18 August 2008 14:27:11 GMT
Content-Length: nnn
Content-Type: application/atom+xml; charset="utf-8"
Location: *http://www.myrepository.org/geo/atom/my_deposit.atom*


<entry ...>
  <title>My Deposit</title>
  <id>info:something:1</id>
  <updated>2008-08-18T14:27:08Z</updated>
  <author><name>jbloggs</name></author>
  <summary type="text">A summary</summary>
  ...
  <content type="application/zip"     src="http://www.myrepository.ac.uk/geo/deposit1.zip"/>
  <sword:packaging>http://purl.org/net/sword-types/tei/peer</sword:packaging>
  <link rel="edit"
    href="http://www.myrepository.org/geo/atom/my_deposit.atom" />
</entry>
```

The Server MUST use the correct MIME-type reference in the type attribute of the atom:content element in the Media Link Entry.
The Media Link Entry MUST contain an atom:element with a src attribute containing the location of the fulltext. This is used in the feedback e-mail to the author.
The Media Link Entry MAY contain an atom:element with a src attribute containing the location of the original submission information package, the raw metadata, ORE-ReM, jump-off page, etc.