



Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

BibSword

Implementation of SWORD client in Invenio for the
automated submission of digital objects to arXiv



Bachelor Thesis

CERN - IT

Barras Mathieu

From 2010-06-01 to 2010-09-11



Table of contents

| | |
|---------------------------------------|-----------|
| 1 Introduction..... | 1 |
| 1.1 Stakeholders..... | 1 |
| 1.2 Milestones..... | 2 |
| 1.3 Context..... | 2 |
| 1.3.1 CERN..... | 2 |
| 1.3.2 CDS and Invenio..... | 3 |
| 1.3.3 WebSubmit..... | 3 |
| 1.3.4 arXiv..... | 4 |
| 1.3.5 SWORD..... | 4 |
| 1.4 Document structure..... | 4 |
| 2 Project..... | 5 |
| 2.1 Objectives..... | 6 |
| 2.2 Activities..... | 7 |
| 2.3 Synthesis..... | 8 |
| 3 Analyse..... | 9 |
| 3.1 Protocol..... | 9 |
| 3.1.1 Atom format..... | 9 |
| 3.1.2 Atom Publishing Protocol..... | 10 |
| 3.1.3 SWORD..... | 13 |
| 3.2 arXiv SWORD Interface..... | 16 |
| 3.2.1 Authentication..... | 17 |
| 3.2.2 Service document..... | 17 |
| 3.2.3 Resource deposit..... | 20 |
| 3.2.4 Resource-tracking..... | 21 |
| 3.2.5 Errors..... | 22 |
| 3.3 Synthesis..... | 23 |
| 4 Design..... | 24 |
| 5 Data format..... | 25 |
| 5.1 Invenio..... | 25 |
| 5.1.1 MARCXML..... | 26 |
| 5.2 arXiv..... | 27 |
| 5.2.1 Mandatory fields..... | 28 |
| 5.2.2 Optional metadata..... | 28 |
| 5.3 SWORD matching format..... | 29 |
| 5.3.1 MARC information..... | 29 |
| 6 BibSword..... | 30 |
| 6.1 Prerequisite..... | 30 |

| | |
|---|-----------|
| 6.1.1 Authentication..... | 30 |
| 6.1.2 Acknowledgement..... | 30 |
| 6.2 Integrity..... | 31 |
| 6.2.1 swrREMOTESERVER..... | 32 |
| 6.2.2 swrDATA..... | 32 |
| 6.3 Overview..... | 34 |
| 6.4 CLI..... | 35 |
| 6.5 API..... | 35 |
| 6.5.1 list-remote-servers..... | 36 |
| 6.5.2 list_remote_server_informations..... | 37 |
| 6.5.3 list_collections_from_server..... | 38 |
| 6.5.4 list_collection_informations..... | 39 |
| 6.5.5 list_mandated_categories_from_collection..... | 40 |
| 6.5.6 list_optional_categories_from_collection..... | 41 |
| 6.5.7 list_submitted_resources..... | 42 |
| 6.5.8 get_marcxml_from_record..... | 43 |
| 6.5.9 get_media_from_marcxml..... | 44 |
| 6.5.10 compress_media_file..... | 45 |
| 6.5.11 deposit_media..... | 46 |
| 6.5.12 format_metadata..... | 47 |
| 6.5.13 submit_metadata..... | 48 |
| 6.5.14 perform_submission_process..... | 49 |
| 6.6 Workflow..... | 50 |
| 6.6.1 Select destination..... | 50 |
| 6.6.2 Deposit media..... | 51 |
| 6.6.3 Submit metadata..... | 52 |
| 6.7 Class diagram..... | 53 |
| 6.7.1 Class creation..... | 53 |
| 7 Web client..... | 54 |
| 7.1 Use cases..... | 54 |
| 7.1.1 Actors description..... | 54 |
| 7.1.2 Cases description..... | 55 |
| 7.2 GUI model..... | 56 |
| 7.2.1 Select remote server..... | 56 |
| 7.2.2 Select server's collection..... | 57 |
| 7.2.3 Select categories..... | 58 |
| 7.2.4 Check metadata..... | 59 |
| 7.2.5 List submission..... | 61 |
| 7.3 Synthesis..... | 61 |
| 8 Implementation & Test..... | 62 |
| 8.1 Data formatting..... | 62 |
| 8.1.1 BibSword Implementation..... | 63 |
| 8.1.2 arXiv implementation..... | 63 |
| 8.2 BibSword..... | 63 |
| 8.2.1 Database table implementation..... | 64 |
| 8.2.2 MARC file insertion..... | 65 |
| 8.2.3 HTTP Authentication..... | 66 |

| | |
|---|-----------|
| 8.2.4 Generation of HTTP request..... | 67 |
| 8.3 Web client..... | 68 |
| 8.3.1 Page generation..... | 68 |
| 8.3.2 Keeping state..... | 69 |
| 8.4 Tests..... | 70 |
| 8.4.1 Unit tests..... | 70 |
| 8.4.2 Regression tests..... | 71 |
| 8.4.3 Web testing..... | 71 |
| 8.5 Installation..... | 72 |
| 8.5.1 Inegration..... | 72 |
| 9 Sustainable development..... | 73 |
| 9.1 IT as sustainable development tool..... | 73 |
| 9.2 Invenio perspective..... | 73 |
| 9.2.1 BibSword and the sustainable development..... | 74 |
| 10 Conclusion..... | 75 |
| A1 : Glossary..... | 76 |
| A2 : Bibliography..... | 77 |
| A3 : List of figures..... | 78 |

1 Introduction

This document presents the results of my work at CERN for my bachelor thesis. The project proposed by CERN is the Implementation of a SWORD client in Invenio for the automated submission of digital objects to arXiv.

Invenio is the underlying open source software used by the CERN Document Server (CDS) service, where my training took place.

CDS and arXiv are digital repositories of physics documents mostly. Invenio offers many features of a Digital Library and is developed and maintained by the CDS service. arXiv is maintained by the Cornell University in Ithaca, New York. It does not offers as many features as Invenio but most of High Energy Physic papers are deposited on it.

Since recently, arXiv offers a new submission interface implementing SWORD.

SWORD is a brand new protocol that defines a simple way to deposit digital document on web repositories.

Many documents submitted to Invenio at CERN are also deposited on arXiv by authors. For this reason, it is relevant for Invenio to offer an automated option "Forward to arXiv" to its users.

The aim of this project is to analyse, design and implements a library oriented SWORD client module (BibSword). This module will contain an end user interface and it will be integrated into the Invenio submission process.

1.1 Stakeholders

Here are listed the stakeholders in the project supervision. As this project isn't done at school, there are more supervisors than in a normal project. The next table introduces each person, his activity and his function in the project.

| Name | Occupation | Function in the project |
|----------------------|---------------------------------|---|
| Supervisor | | |
| Mr Jean-Yves Le Meur | CDS section leader (IT) at CERN | The on-site supervisor of the project . He is the contact person for all questions about the specifications of the project. A meeting is organised with on a weekly basis to evaluate the status of the project and if necessary find solutions to possible issues. |
| Professor | | |
| Dr Omar Abou Khaled | Professor at the EIA-FR | The professors are the persons who asses the project in every aspect: the management, the technical ease the documentation and the communication. They have to be told of the progress of the project and of possible problems that may occur. |
| Dr Elena Mugellini | Professor at the EIA-FR | |

| External Expert | | |
|------------------------|----------------------|---|
| Mr Giovanni Celato | Professor at the EMF | The external expert judges the presentation of the work. One or two meetings are planned during the project to expose the progress of the work. The expert can, if necessary, require some adaptations. |

1.2 Milestones

The milestones are the main deadlines in the project. The following table illustrates every milestones and their deadlines.

| Milestones | Deadlines |
|----------------------------------|--|
| Validation of the specifications | Tuesday, the 8th of June |
| Meeting with the external expert | Friday, the 11th of June |
| Rendering of the bachelor thesis | Friday, the 16th of July at 17p.m. |
| End of the internship at CERN | Tuesday, the 31th of August |
| Oral defence | From Monday 6th to Wednesday 8th of September |
| Rendering of the poster | Thursday, the 9th of September |
| Open house days | Friday 10th and Saturday 11th of September |
| End of the bachelor | Saturday, the 11th of September |

1.3 Context

1.3.1 CERN

The European Organisation for Nuclear Research (CERN) is the world's largest particles physics laboratory. It is located at the north west of Geneva, on the Franco-Swiss border.

The CERN was founded in 1954 by 12 European countries including Switzerland. Currently, it is the workplace of about 2600 full-time employees and more than 7000 scientist and engineers coming from all over the world.

The main goal of the CERN is to provide all the tools needed by the scientists to study what matter is made of. The most known of these is the Large Hadrons Collider (LHC), the world's largest particles accelerator in the world (circumference of 28 km).

The CERN is also the birthplace of the World Wide Web. The aim of this project initiated by Berners-Lee in 1989 was to facilitate the sharing of the results and the analyses of the scientists. It was offered by CERN to the world in 1993. [1]

1.3.2 CDS and Invenio

The CERN Document Server (CDS) is part of the IT department. Its main role is to provide tools to store and share every document used and produced by the researchers. It is, to some extent, an extension of the original idea of Tim Berners-Lee.

Invenio is an integrated digital library system. It is composed of applications which provide the framework and tools for building and managing an autonomous digital library. The software is already available as free software, licensed under a GNU GPL (General Public Licence).

The technologies offered by Invenio cover many features of a library. The application is mainly accessible through a web interface and provides the following features:

- A navigable collection tree allowing an easy classification of the documents.
- A powerful search engine with “Google like” performances.
- A flexible indexation of the document with metadata using the MARC library standard (and its XML derivation MARCXML).
- A customizable user interface.

Invenio is designed for middle to large databases. For example the CERN document server running Invenio contains about 1'000'000 articles organized in more than 500 collections.

Invenio may be too advanced for small digital libraries with limited needs. Indeed, to get started with it, it requires a correct knowledge of Linux and programmers must be familiar with web technologies.

Invenio is mostly written in Python. This language is used by many companies and schools. Like Java, it is an interpreted language that makes it a bit slower to execute than C or C++. In the other hand, the development in Python is quite easy and fast. [2]

1.3.3 WebSubmit

Invenio is composed of several inter-connected modules connected to each other. The most important module for this project is the one named “WebSubmit”.

WebSubmit is a comprehensive submission system allowing authorized users to submit individual documents into the system. The submission system uses a flow-control mechanism that allows to define approval steps by authorized units. In total there are several different exploitable submission schemas available, including an automated full text document conversion from various textual and image formats. This module can also take advantage of information extraction functionality, focusing on bibliographic entities such as references, authors, keywords or other implicit metadata. [3]

In short it is the interface that provides the capabilities for users to submit documents on Invenio.

1.3.4 arXiv

arXiv is a platform hosted by the Cornell University Library. It is a repository providing access to scientific digital documents, like CDS does. It offers less functionalities than Invenio, but covers a large number of disciplines.

In physics as well as in mathematics or computer science, many scientists are depositing their pre-published papers (preprint) on arXiv. arXiv contains today more than half a million of preprint and about 5000 are added every month. [4]

The scientists who put their work on-line in the CERN institutional repository (CDS) using Invenio often want to also publish it on arXiv.

To facilitate the deposit of preprints into arXiv, a new interface dedicated to the automation of document deposit has recently been implemented. This new interface implements the SWORD protocol.

1.3.5 SWORD

The Simple WebService Offering Repository Deposit (SWORD) is a lightweight protocol used for depositing digital documents on internet repositories. It is the key protocol that is to be used for Invenio based repositories to exchange data with the arXiv server [5], and also with any other SWORD-compliant service.

As it is relative young (2008), it is not much spread over yet. However it can be expected that it will become very successful thanks to its simplicity and its efficiency.

1.4 Document structure

This document is structured in order to help readers following the evolution of the project. The main parts are:

- **Project:** it is the description of the features of the module developing in this project.
- **Analysis:** it is a summary of every studies made for the good understanding of the protocols used in the project.
- **Conception:** it offers an overview of the different elements composing the BibSword module and how they are tied together.
- **BibSword:** it contains the description of the application's business logic.
- **Web-client:** it shows the design of the web client. It is the main user interface of the system.
- **Implementation & Tests:** it contains justification of the technical decisions and the test suite strategies.
- **Sustainable development :** it is an analysis of the impact of IT in general, CDS Invenio and the module BibSword on the environment.
- **Conclusion :** it summarizes the project and analyzes its result.
- **Annexes:** Glossary, Bibliography, List of the figures, Planning, Minutes and Logbook

2 Project

The aim of this project is to improve the digital document submission process on Invenio by adding an automated « Forward to arXiv » function.

A digital document in Invenio is usually separated in two parts : the metadata and the associated files called full text.

arXiv used the same way to stores it digital document. The difference is that its metadata are not exactly the same as on Invenio. One of the issue is then to match Invenio metadata with arXiv metadata.

Invenio and arXiv organize their digital documents in collections and categories. Again, those collections and categories are not the same. Another issue of this project will be to find a way to indicate to arXiv the chosen collection.

An important point of the Forward function is the redundancy control. Before a submission to arXiv, the function has to check that the document is not already stored in it.

The SWORD protocol is not only implemented by arXiv. It is then interesting to implements a SWORD client that is not dedicated to arXiv but who support forwarding of document to other SWORD remote servers.

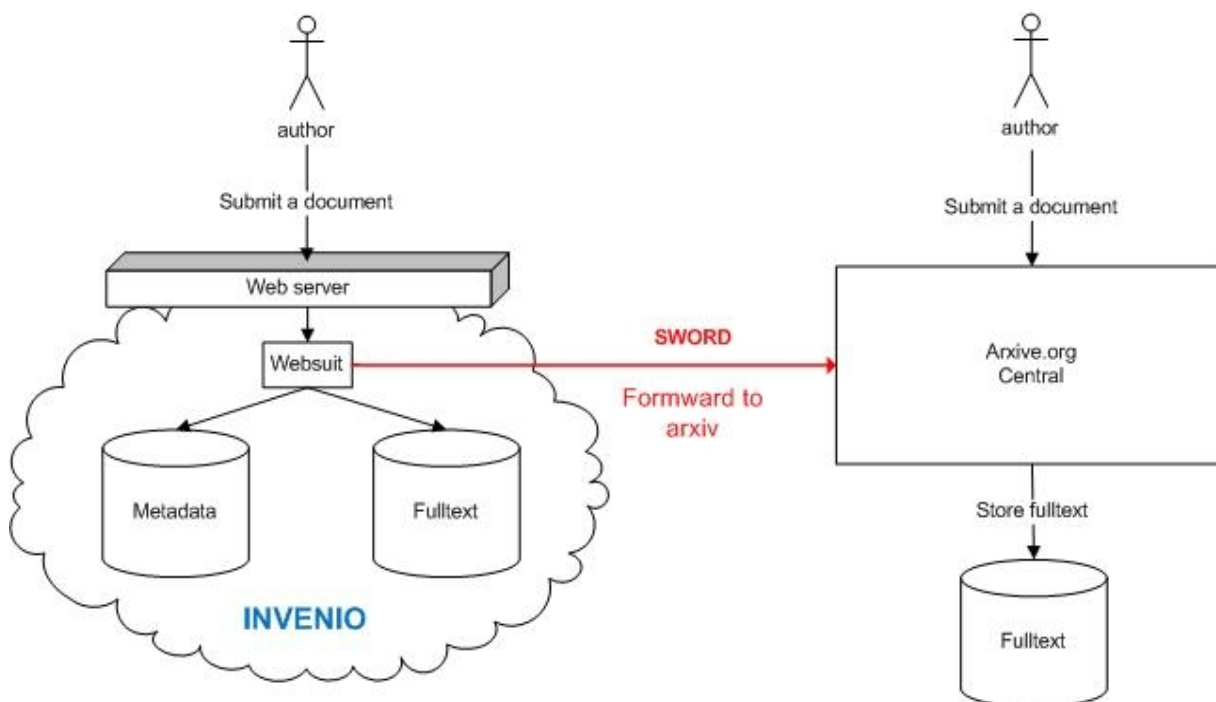


Figure 1: Schema of the project

The improvements offer by this project are the following :

- No need for the users to resubmit digital document to arXiv
- Ensure that material also exists on arXiv
- Add a SWORD client feature to Invenio

2.1 Objectives

The main requirement of the project are the following:

- Design of a SWORD compatible format
- Conception of a SWORD client API for Invenio
- Implementation of a SWORD Web client for arXiv
- Integration of the client into Invenio

Design of a SWORD compatible format

The first part of the project is dedicated to the study of the SWORD document format standard. This format is used in the arXiv deposit API. The goal is to allow Invenio to format a document into a format based on SWORD which fits with the arXiv deposit API. Even if some format already implemented in Invenio are close to the SWORD format, in particular the XML Dublin Core, it is necessary to do some adaptation to make it compatible.

Conception of a SWORD client API for Invenio

arXiv is not the only one system that implements a SWORD server. The main added value of this project is the conception of a SWORD client API. This API is called BibSword according to the standard of Invenio nomenclature. BibSword stands for Bibliographic and SWORD is the name of the deposit protocol. This API offers tool to developers to implement client that may deposit resources not only on arXiv but also on any other SWORD server. This way of processing allows programmers to use the SWORD client with a few configuration parameters.

Implementation of a SWORD Web client for arXiv

This part of the project involved the implementation of a web client that uses the BibSword API. This client must allow users to submit resource using a browser. As specified by the BibSword API it will allow to submit resources to any SWORD compatible server. In this project, arXiv will be the only server that will be reached by this interface. The reason is both because it is the aim of the project and for a question of time limitation. Anyway the implementation of one interface should be enough to demonstrate the proper functioning of the BibSword API.

Integration of the client into Invenio

Once both the client and the API are tested separately and together, they are integrated into Invenio as an independent module. This module must be reachable from the WebSubmit interface to allow users to forward their work to arXiv during the submission on Invenio. It can also be reached by the standard search interface of Invenio to allow users to submit some resources that have previously been submitted to the digital library running the Invenio system.

2.2 Activities

The Bachelor Thesis covers 7 weeks. The activities table gives an overview of each task and its approximative duration.

| Objective | Activity | Duration |
|---|--|----------------|
| Study and Apprenticeship | CDS Invenio & Python | 1 week |
| | Output Format in Invenio | 1.5 days |
| | Module WebSubmit | 1.5 days |
| | arXiv submission protocol and API | 1.5 days |
| | | 2 weeks |
| Design of a SWORD compatible format | Design of a SWORD compatible format for Invenio | 2 days |
| | Implementations of a formatting function | 2 days |
| | Implementation of a test suit for the function | 1 days |
| | | 1 weeks |
| BibSword conception | Conception of the BibSword API | 2 days |
| | Implementation of the BibSword API | 4 days |
| | Implementation of a Command Line Client (CLI) | 2 days |
| | Implementation of a test suit the API | 2 days |
| | | 2 weeks |
| Implementation of the Web-client for arXiv | Model of the web-client user interface | 1 days |
| | Implementation of the Web-client | 3 days |
| | Implementation of a test suit for the web-client | 1 days |
| | | 1 weeks |
| Integration into Invenio | Single function test and validation | 1 day |
| | Validation of the regression test | 1 day |
| | Integration to the CERN's Invenio instance | 1 day |
| | Validation of the integration | 2 days |
| | | 1 weeks |
| | | 7 week |

The above table does not consider the time taken to write the thesis.

However the time taken by these task is about 20 to 30 % of the work. For this reason the duration of each tasks is arbitrarily overestimated.

2.3 Synthesis

This project is part of a global system on which many people are working. One of the constraints is then that the BibSword module must be integrated with the existing project without bringing any down side, such as performance degradation.

The produced code must respect the coding guidelines imposed by the software. The developer documentation of Invenio[6] specifies:

- **The coding style** : for example the variables and constants name, the comments of the functions and the code compliance with the standard style.
- **The test suit strategies** : specifies how and what to test: the core of the functions, the function as a black box and the regression test that ensure that the new function integrate with the software.

It is one of the most important constraint of the project to follow this specification to allow easier future development.

At the end of the project, the result must be that the new module is fully integrated into the CERN Invenio instance. The end users have to be able to forward their work to arXiv in the most intuitive way. Obviously this forward function must not affect the proper functioning of arXiv.

During the project, some activity might be modified and even added depending of the progress of the work. Every change has to be agreed with the supervisor, Mr Le Meur. All changes will be transmitted to every stakeholder as they occur and reported to the final report, the logbook and the planning.

3 Analyse

The use of SWORD requires a good understanding of some protocols and document formats. The studies of these techniques are documented in the section 3.1.

arXiv implements a SWORD server. This server offers all the features specified by SWORD. arXiv adds some specific features that are not included in SWORD. The section 3.2 treats of the arXiv SWORD specific interface and describes in details the operations of the arXiv SWORD server.

3.1 Protocol

One of the protocols used by arXiv to interact with external tools over the internet is SWORD. The aim of this chapter is to understand how this protocol works, which services it provides and how arXiv implements its own customized version of SWORD.

The important topics to learn in order to understand the implementation of SWORD by arXiv are the following:

- Atom format : The basic XML format used by SWORD to share informations between systems
- Atom Publishing Protocol (APP) : A protocol oriented document sharing over the internet
- SWORD : A protocol oriented document deposit over the internet

The aim of this analysis is not to go into each of the details of the protocol specifications, but to a basic knowledge of the services that will be used to communicate with the arXiv SWORD interface.

3.1.1 Atom format

Atom [7] is a XML-based format. Its aim is to inform a system where a digital document is located on an Internet repository based on its Unique Resource Location (URL). It provides general informations about the digital document such as title, type, author and its URL.

Atom describes lists of related elements called “**feed**”. Feeds are composed of items known as “**entries**” which describe metadata and location of the web resource. The entries have some mandatory items. For example every entry must have a title and an URL.

The interest of Atom is that it defines a standard way to gives the location of a resource on the Internet.

To define that a XML file is an Atom, the root of the file has to declare the atom namespace:

```
<?XML version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
```

Atom has two kind of documents:

- The Atom Feed Document containing some informations about the feed and many entries. Its root is `<atom:feed>`
- Atom Entry Document containing only one entry. Its root is `<atom:entry>`

In this project the Atom Entry Document will be the most interesting because it is the format used by SWORD for all exchange of interactions during a resource submission process.

The code bellow shows an example of a minimal atom entry document. (Minimal meaning that it contains only the required fields)

```
<entry>
  <title>Atom-Powered Robots Run Amok</title>
  <link href="http://example.org/2003/12/13/atom03"/>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <summary>Some text.</summary>
</entry>
```

The field “link” is very important as it contains the **URL** of the resource described by the entry.

The media type (MIME-Type) defining an atom document is “*application/atom+xml*”. In addition, the MIME-Type must contains the type of the atom file that is sent. An Atom Entry Document is specified by adding *type=entry* separated from the MIME-Type with a semi-colon (;)

The aim of the MIME-Type is to specify the nature of the resource sent over the internet.

The HTTP response shown bellow contains an Atom document as it is specified in the Content-Type field.

```
HTTP/1.1 201 Created
Date: Tue, 29 Apr 2008 18:35:46 GMT
Location: https://arxiv.org/sword-app/getid/app/.....
Content-Type: application/atom+xml;type=entry
```

3.1.2 Atom Publishing Protocol

The Atom Publishing Protocol (APP), also known as AtomPub, is an application-level protocol for publishing and editing Web resources using HTTP and XML. This protocol provides facilities for the management of these resources:

- **Collections:** Set of resources which can be retrieved in whole or in part.
- **Services:** Allows to discover the available collections.
- **Edition:** Allows the creation, modification and suppression of resources.

AtomPub uses Atom Entry Document format as basis for its resource description. It extends the atom format by importing some fields. Those fields are defined in the namespace available at the following URL:

```
<?XML version="1.0" encoding="utf-8"?>
  <feed xmlns="http://www.w3.org/2007/app">
```

Collections and resources

When a resource is created, it is assigned to a collection and receives an URL. The resource is then considered as a *Member* (also named *Member Resource*) of the collection. Two kinds of members are distinguished:

- *Entry Resources* that are Atom Entry Document and contains the meta data and the full text resource.
- *Media Resources* that have other representation than Atom Document Entry, for example a PDF, an JPG picture, a TeX file, ... and need to be described by an Atom Entry Document containing the meta data.

To allows Media Resource to be described, they are associated to Media Link Entry that contains the meta data describing them [Figure 2].

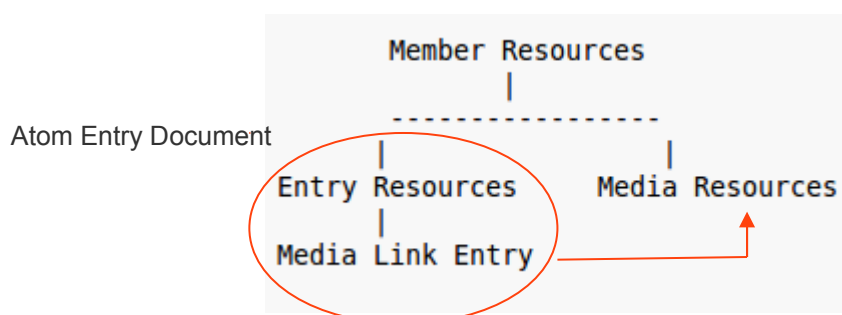


Figure 2: Collection's structure

Collections are sets of resources that group the same topic (Physics, Mathematics, Computer Science, ...). They are represented as Atom feeds. A collection may content many resources but also subset of collections called categories.

A collection atom feed define the root URL where the members are stored. The only thing that differentiate collection atom feed to resource atom feed is the fact that it is listed by a service document.

Service document

A service document defines the group of collection available on a server. These groups are called workspaces.

The service document allows to retrieve a collection in the case a user or a service wants to create a resource [Figure 3].

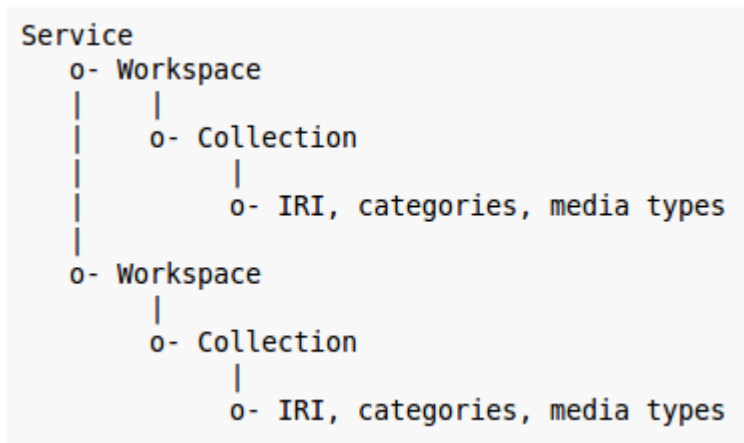


Figure 3: Service document structure

In addition it can indicate which type of media are accepted by a collection and the categories it offers.

Interactions

To interact with a server implementing APP, one use simple HTTP request GET, POST, PUT, DELETE. For some action like creating a resource (using POST) the HTTP request is followed by an XML entry that should fits with the Atom feed format. When an answer is sent back to the client, the response is often followed by an Atom feed.

The following illustration [Figure 4] offers an overview of the actions allowed by APP.

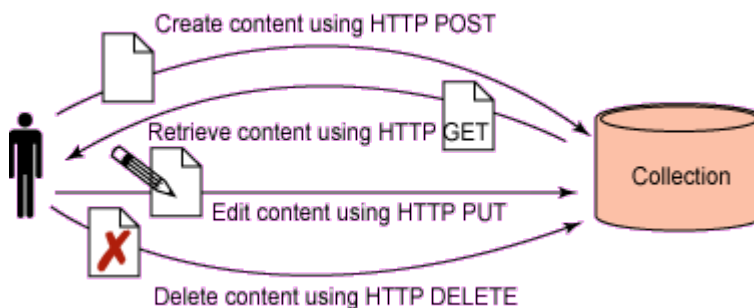


Figure 4: APP Interactions

The action are then:

Create : the URL of a valid collection has to be setted in the POST request

Retrieve : the metadata helps to retrieve the URL that is then requested with a GET

Edit : after having retrieved and modify the resource, a PUT allows to update it

Delete : a DELETE request with the appropriate URL allows to delete a resource.

3.1.3 SWORD

SWORD is a lightweight protocol for depositing content from one location to another. It stands for *Simple Web-service Offering Repository Deposit* and is a profile of the Atom Publishing Protocol (known as APP or AtomPub). [8]

SWORD was developed by the Joint Information Systems Committee (JISC) [9]. its aim was to offer a simple and standard way for the deposit of digital document into a repository on the Internet.

SWORD implements only the CREATE action defined by AtomPub. For this action, it uses the same specifications than AtomPub such as collections, service document, atom based XML messages, ... The interest of SWORD is that it provides some more interesting options for an automatic deposit.

This chapter focuses on the additional feature SWORD provide over AtomPub.

Namespace

As AtomPub, SWORD meta data structure is based on Atom feed format. The field that extends this format are defined in the SWORD namespaces at:

```
<?xml version="1.0" encoding="utf-8"?>
  <feed xmlns:sword="http://purl.org/net/sword/">
```

Package support

SWORD allows the deposit of compound resources. Compounds resources are usually archives in the form of a tarball or a zip file. As AtomPub doesn't support this kind of format, SWORD defines the `<sword:packageSupport>` node that indicates which type of resource is accepted by a collection. In addition, this tag can have an attribute "q" that specifies the preference rate (float value between 0 and 1) of the type of deposit. By default, the "q" value is 1.

```
<sword:acceptPackaging q="1.0">
  http://purl.org/net/sword-types/METSDSpaceSIP
</sword:acceptPackaging>
<sword:acceptPackaging q="0.8">
  http://purl.org/net/sword-types/bagit
</sword:acceptPackaging>
```

Mediated deposit

The mediated deposit function allows user to deposit a document owned by another user. To complete this task, SWORD provides a field X-On-Behalf-Of that can be added in HTTP request headers.

In practice, this function allows an authenticated software to automatically deposit a digital document without needing to ask the owner to gives his credentials. It implies that the accounts that are able to use the SWORD interface are "Trusted" by the SWORD server, because they can submit any document to it.

A condition to use this function is that the SWORD server implementation allows mediation for the collection. To be enable, the node `<sword:mediation>` must be setted to true:

```
<sword:mediation>true</sword:mediation>
```

When a digital document is deposit by an user acting on behalf of the real owner, the user becomes the author of the document in the resource meta data. The real owner is still represented as the first contributor of the resource.

Example

In the example below, an application deposits a digital document on behalf of John Doe.

Firstly, the client sends an HTTP request POST containing a zipped resource to the collection : "geography-collection".

```
POST /app/geography-collection HTTP/1.1
Host: www.myrepository.ac.uk
Content-Type: application/zip
User-Agent: RepoGateway/1.2
Authorization: Basic [digested auth information for 'Invenio']
Content-Length: nnn
X-On-Behalf-Of: jdoe
[zipped data]
```

The server answers with code 201 which mean that the submission has been accepted. The response header contains the type of the content of the response (application/atom+xml) and the final URI of the submitted file.

```
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.ac.uk/geography\_collection/atom/my\_deposit.atom

<?xml version="1.0"?>
  <entry xmlns="http://www.w3.org/2005/Atom"
        xmlns:sword="http://purl.org/net/sword/">
    <title>My Deposit</title>
    <author><name>invenio</name></author>
    <contributor><name>jdoe</name></contributor>

    <summary type="text">A summary</summary>
    <!-- In this case, the package has been placed on the workbench of the
On-Behalf-Of user -->
    <content type="text/html"
      src="http://www.myrepository.ac.uk/fdibner/workbench/my_deposit"/>
    <link rel="edit-media"
      href="http://www.myrepository.ac.uk/geography/my_deposit.zip"/>
    <link rel="edit"
      href="http://www.myrepository.ac.uk/geography-
collection/atom/my_deposit.atom" />

    <sword:userAgent>RepoGateway/1.2</sword:userAgent>
    <generator uri="http://www.myrepository.ac.uk/engine" version="1.0"/>
    <sword:treatment>Treatment description</sword:treatment>

  </entry>
```

In this way it is possible to retrieve the owner and the submitter of the digital document.

Developer tools

The SWORD standard describes some tools helping developers to implement SWORD clients. Those tools are the following:

- **No-op** (dry run): allows to test the client performing submission without to permanently store the digital document. The response contains the same data including possible errors as if it would be really created.
- **Verbose** : when this mode is activated, the server provides more detailed output for the action performed on it.
- **Client and server identity** : allows to check the identity of the client and the server.

Those tools have to be activated in the HTTP request:

```
POST /app/geography-collection HTTP/1.1
Host: www.myrepository.ac.uk
Content-Type: application/zip
User-Agent: RepoGateway/1.1 PythonLibHttp2/2.5
Authorization: Basic [digested auth information for 'invenio']
Content-Length: nnn
X-No-Op: true
X-Verbose: true
[zipped data]
```

The result appears then in the returned Atom Entry Document:

```
...
<sword:noOp>true</sword:noOp>
<sword:verboseDescription>
  Does collection exist? True.
  User authenticates? True.
  User: invenio
  User has rights to collection? True.
</sword:verboseDescription>
...
<sword:userAgent>RepoGateway/1.1 PythonLibHttp2/2.5</sword:userAgent>
<generator uri="http://www.myrepository.ac.uk/sword-plugin"
version="1.0"/>
...
```

3.2 arXiv SWORD Interface

arXiv implements a deposit interface based on the SWORD protocol [10]. The SWORD interface is accessible through standard HTTP request [Figure 5]. This implementation offers some additional options to the SWORD specification, in particular, the possibility of modifying a resource that has already been deposited.

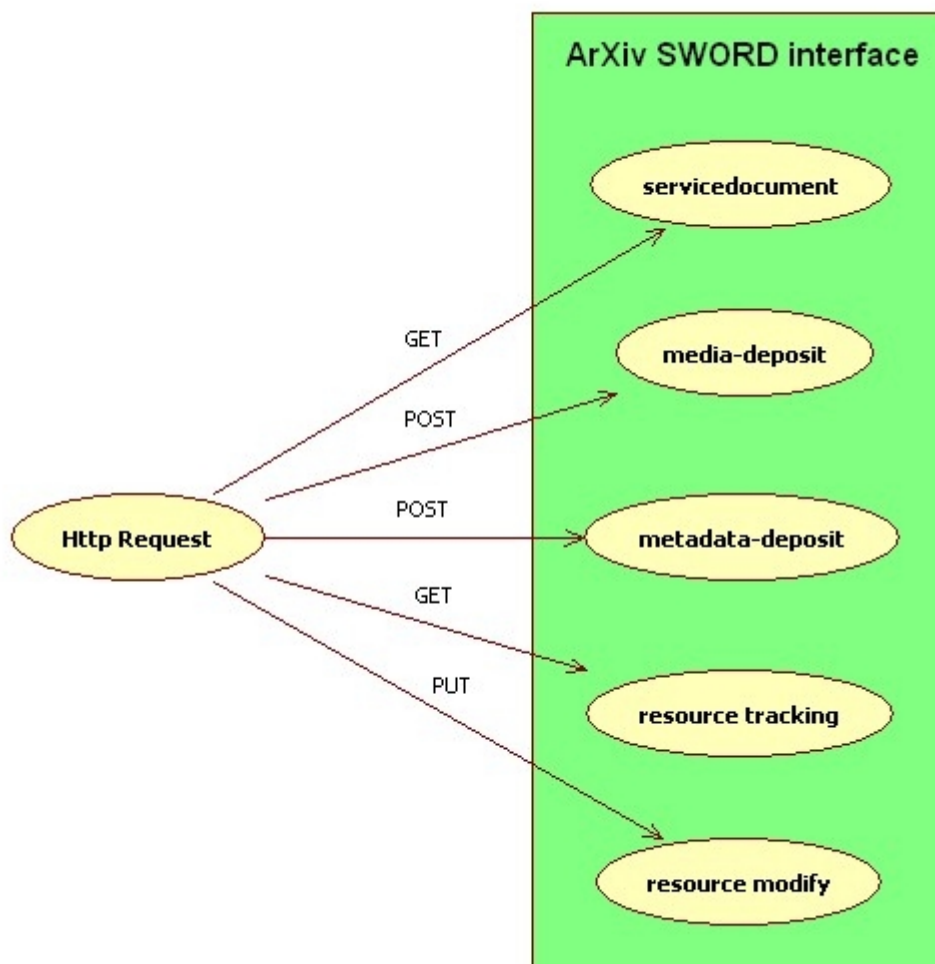


Figure 5: arXiv SWORD interface

All access to a SWORD action on arXiv needs an authentication. Every action returns a standard HTTP response with a standard status code:

200 : The action has been accepted

400 : Bad Request.

An XML atom entry comes with the response. This file contains some additional informations about the request.

3.2.1 Authentication

For obvious security reasons, arXiv SWORD interface checks authentication informations for every HTTP request. In a standard client-server application, the user should give a user name and a password to be able to enter the system.

As the aim of the currently developed function is to simplify the arXiv submission process, we cannot ask users for their arXiv credential.

To go around this problem, SWORD implemented a simple solution by adding an option called “X-On-Behalf-Of” to the header of the request. This option, followed by a name and an email address allows to tell arXiv that the request is not made by the user that is authenticated but by the person specified after the “X-On-Behalf-Of” header.

3.2.2 Service document

The service document lists the available collections and categories of arXiv. To get these important informations, one must execute a GET HTTP Request on the service document of arXiv. The response contains an XML atom entry containing information shown in [Figure 6 and Figure 7]:

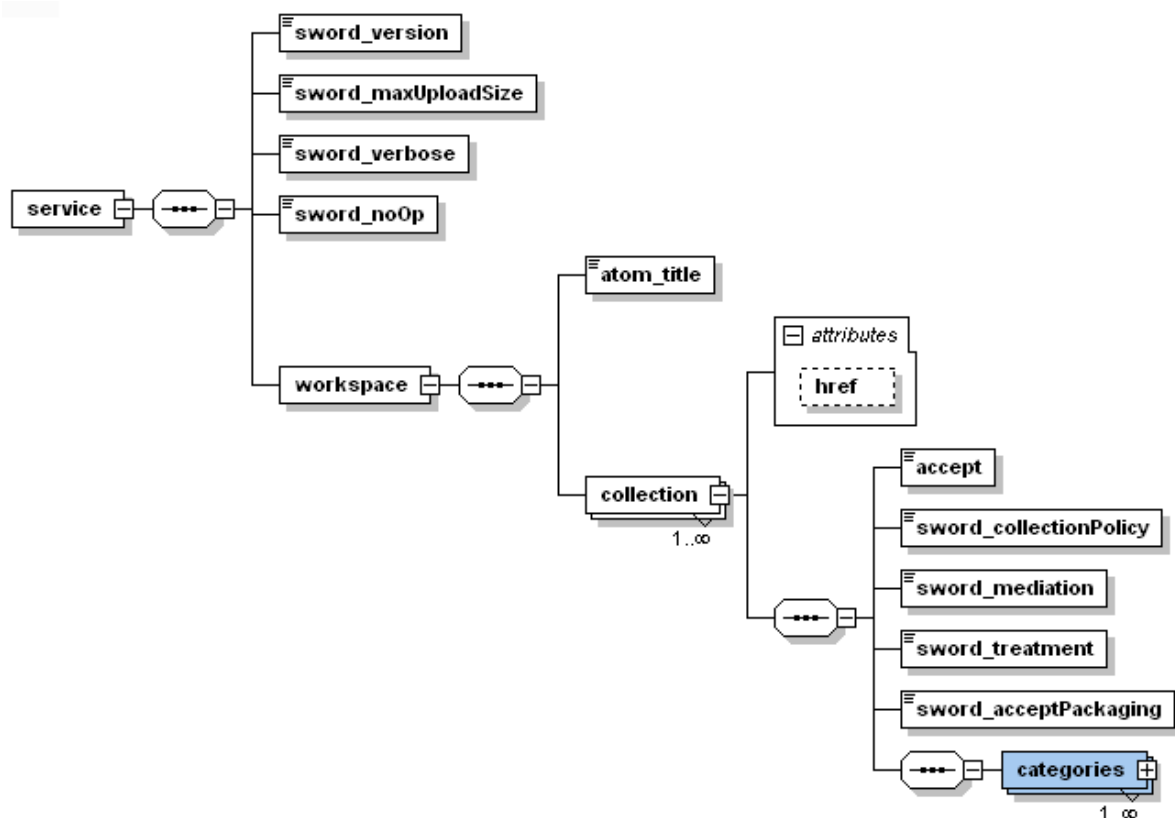


Figure 6: service document schema 1/2

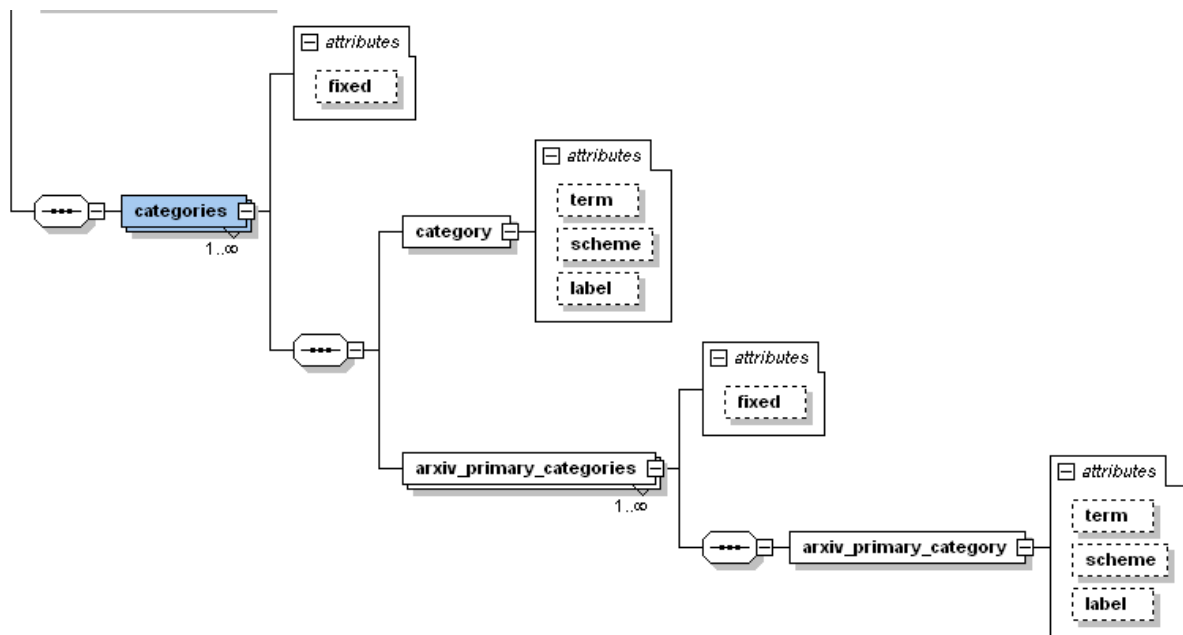


Figure 7: service document schema 2/2

The atom entry contains standard AtomPub informations like :

- **Workspace** : There is conceptually only one workspace containing every collections. It is the root of the service document. In fact, there is one workspace per authenticated user. Those are used to store resources that are waiting for approval of arXiv. The user workspaces contains the collections in which the user is allowed to deposit.
- **Collections** : The collections are the main repository directory. It is the root URL where the documents are stored. Each collection contains many categories.
- **Categories** : The categories are used to help retrieving resources. It is used to classify document in a collections.

Those information are common to every repository implementing the AtomPub protocol. As this atom actually contains arXiv SWORD protocol informations, it imports some other namespaces:

- **atom** : <http://www.w3c.org/2005/atom>
- **sword** : <http://purl.org/net/sword>
- **arXiv** : <http://arxiv.org/schema/atom>

Atom is only used to give a readable title to the namespace and the collections.

SWORD is used for two kind of informations :

- informations that are relevant for all collections of the repository:
 - `<sword:version>`: version of the SWORD server
 - `<sword:maxUploadSize>` : size of individual file accepted by the server (if bigger, send separated files)
 - `<sword:verbose>` : specify that the server implements a verbose mode

- `<sword:noOp>` : specify that the server implements a simulation mode
- informations that are proper to one collection:
 - `<sword:collectionPolicy>` : collection human readable description
 - `<sword:mediation>`: server allows X-ON-BEHALF-OF option or not
 - `<sword:treatment>`: specify how the server check the submission
 - `<sword:acceptingPackage>` : type and preference of deposition format

arXiv import his own namespace to specify the primary categories of each collection:

- `<arxiv:primary_category>` : allows the repository to separate standard categories that are multiple and optional for mandatory category that is unique.

The Figure 8 is an extract of the XML AtomPub file returned by the arXiv service document:

```
<service xmlns="http://www.w3.org/2007/app"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:sword="http://purl.org/net/sword/"
  xmlns:arxiv="http://arxiv.org/schemas/atom">

  <sword:version>1.3</sword:version>
  <sword:maxUploadSize>10000</sword:maxUploadSize>
  <sword:verbose>true</sword:verbose>
  <sword:noOp>true</sword:noOp>
  <workspace>
    <atom:title>arXiv</atom:title>

    <collection href="https://arxiv.org/sword-app/physics-collection">
      <atom:title>The Physics archive</atom:title>

      <accept>application/atom+xml;type=entry</accept>
      <accept>application/zip</accept>
      <accept>text/xml</accept>
      <accept>image/gif</accept>
      ...

      <sword:collectionPolicy>Open Access</sword:collectionPolicy>
      <sword:mediation>true</sword:mediation>
      <sword:treatment>will be posted pending moderator approval</sword:treatment>
      <sword:acceptPackaging>http://purl.org/net/sword-types/bagit</sword:acceptPackaging>

      <categories fixed="yes">

        <category xmlns="http://www.w3.org/2005/Atom"
          term="http://arxiv.org/terms/arXiv/astro-ph.CO"
          scheme="http://arxiv.org/terms/arXiv/"
          label="Physics - Cosmology and Extragalactic Astrophysics"/>
        <category xmlns="http://www.w3.org/2005/Atom"
          term="http://arxiv.org/terms/arXiv/astro-ph.EP"
          scheme="http://arxiv.org/terms/arXiv/"
          label="Physics - Earth and Planetary Astrophysics"/>
        ...

      </categories>

    </collection>
  </workspace>
</service>
```

Fi

Figure 8: example of service document response

3.2.3 Resource deposit

The resource deposit process is made in two steps:

1. Media deposit
2. Metadata submit

The *media deposit* allows to send media to the remote server using HTTP POST request containing at least the content type and the user credentials. This request is made to the URL of the remote server's collection:

```
POST https://arxiv.org/sword-app/cs-collection
Host: arxiv.org
User-Agent: arXiv SWORD demo 1.1
Content-Type: image/jpeg
Authorization: Basic .....=

data stream
....
```

The result of the deposit is an XML atom entry containing the link to the deposited media URL:

```
<content type="image/jpeg" src="https://arxiv.org/sword-app/edit/12334"/>
```

If the resource contains multiple media such as document and images for example, this process must be done for every media. There will be then one link per deposited media.

Once the media deposit is done, the metadata has to be sent and linked to the media. This process is called *metadata submit*. The content of a XML atom entry for metadata file is fully described in the chapter 5.2 : on page 27.

The submission of the metadata is made through a HTTP POST request exactly like the media deposit. It has to be addressed to the same collection's URL:

```
POST https://arxiv.org/sword-app/physics-collection HTTP/1.1
Host: arxiv.org
User-Agent: arXiv SWORD demo 1.1
Content-Type: application/atom+xml;type=entry
Authorization: Basic .....=
```

In the example above, the content-type is "application/atom+xml;type=entry". This header is required to tell the remote server that the file is a metadata and not a media.

To match with the posted media, the metadata file must contain a field <link> that contains the URL's reference to the deposited media:

```
<link href="https://arxiv.org/sword-app/edit/12334" type="application/pdf"
rel="related"/>
```

The response of the server contains then at least three important <link> elements:

```
<link rel="edit-media" href="https://arxiv.org/sword-app/edit/12334"/>
<link rel="edit" href="https://arxiv.org/sword-app/edit/12334.atom"/>
<link rel="alternate" href="http://arxiv.org/resolve/app/12334"/>
```

rel = "edit-media" : the URL of the media (could be multiple)

rel = "edit" : the URL of the metadata

rel = "alternate" : the URL where to get the status of the submission

3.2.4 Resource-tracking

To help SWORD client follow the status of their submission, arXiv offers a service called "Resource-tracking". By addressing a HTTP GET request to the "alternate" link (see chapter 3.2.3 :), an atom entry will be sent back containing following nodes:

```
<?xml version="1.0" encoding="UTF-8"?>
<deposit>
  <tracking_id>http://arxiv.org/resolve/app/12334</tracking_id>
  <status>submitted</status>
</deposit>
```

The status are :

- **submitted** - The SWORD submission is in the normal arXiv workflow and is queued to be announced according to the usual arXiv announcement schedule.
- **published** - The SWORD submission has been accepted and published by arXiv. The response will also include the final arXiv identifier in the `<arxiv_id>` element.
- **on hold** - The SWORD submission is in arXiv's workflow but was identified by arXiv administrators or moderators as needing further attention.
- **incomplete** - This status is not expected to be used for SWORD submissions. The submission is in process but not yet submitted and queued.
- **unknown** - The tracking URI is not know. More information may be given in an `<error>` element.

3.2.5 Errors

In case of errors (Status code 400) the XML atom response contains two additional fields:

- **<summary>** : a human readable description of the error
- **<atom:errorcode>** : the numerical code of the error

The numerical error codes are powers of 2 (for convenient bit masking, multiple error indication, etc).

Example

This example shows the result of a submission of a document to an unknown collection:

```
HTTP/1.1 400 Bad Request
Date: Thu, 08 May 2008 21:51:32 GMT
Server: Apache
Content-Type: application/atom+xml;type=entry
```

And the atom result:

```
<?xml version="1.0" encoding="utf-8"?>
<sword:error xmlns="http://www.w3.org/2005/Atom"
  xmlns:sword="http://purl.org/net/sword/"
  xmlns:arxiv="http://arxiv.org/schemas/atom"
  href="http://purl.org/net/sword/error/ErrorContent">
  <author>
    <name>SWORD@arXiv</name>
  </author>
  <title>ERROR</title>
  <id>info:arxiv/B48008EB-F5BF-3827-8ABA-8AB5F04EAAAA</id>
  <updated>2008-05-08T21:45:44Z</updated>
  <source>
    <generator uri="http://arxiv.org/sword-app/"
version="0.9">SWORD@arXiv.org</generator>
  </source>
  <sword:treatment>processing failed</sword:treatment>
  <sword:packaging>http://purl.org/net/sword-types/bagit</sword:packaging>
  <sword:userAgent>arXiv SWORD demo 1.1</sword:userAgent>
  <link rel="alternate" href="http://arxiv.org/help" type="text/html"/>
  <arxiv:errorcode>16</arxiv:errorcode>
  <summary>invalid collection: foobar</summary>
</sword:error>
```

In case of several errors, the errorcode is the addition of every error code and the summary field is repeated.

3.3 Synthesis

To synthesize the content of this analyse, the following chapter offers a list of important point with their relevant informations:

- **Protocols** : SWORD is based on AtomPub. It limit the possibility to the deposit action but increase the functionalities of it in compare with AtomPub. arXiv is implementing it's own SWORD server and follows every point of the SWORD specification. However, it add some possibility that are not in the SWORD specification, for example the possibility of modifying submitted resources.
- **Messages** : Every messages sent between SWORD client and server are based on the Atom Entry format. The specific fields added by SWORD or arXiv are imported from their own namespaces.
- **Communications** : Every communication between SWORD client and server are using HTTP request. An authentication is required but it is not the user but the application that connect itself through a specific SWORD account. To know who is doing the submission, the tag "X-On-Behalf-Off" followed by the submitter name and email is added to the HTTP header.

As SWORD is a protocol that is not only used by arXiv (a current project at CERN is to implement a SWORD SERVER for Invenio) it is very interesting to offers a global module that allows client to interact with any SWORD remote server.

The design of this project will be done in this perspective.

4 Design

The aim of this chapter is to show the global structure of the application. The operating schema [Figure 9] shows the main element composing and used by the system.

The description of each elements are useful to understand the next chapters of this document:

BibSword : BibSword is a module that contains all the business logic of the SWORD client. It contains the functions grouped by the following categories:

- **Formatting** : functions that allow to parse XML atom entries or retrieve node's values. It format knew XML atom entries. (for example to generate metadata files)
- **HTTP request** : set of functions that allows to interact with remote SWORD server using HTTP GET and POST requests. (for example to get the servicedocument or to submit a media)
- **db query** : contains several function that SELECT, INSERT or UPDATE data into the Invenio database. (for example to keep a trace of the submitted resources to avoid double submission)

BibSword API : The BibSword API is an interface that exposes the functions that an external application (such as a web client) can use. It offers all the functions needed to implement a SWORD client.

WebClient : The web client uses the BibSword API to get information about the remote server collections and categories, to deposit resources and to know the state of the submissions. It displays these information in a convenient way for the user doing the submission.

These component are explained in details in the chapter : and :

5 Data format

The submission of digital document through the SWORD arXiv API need to be associated with a meta data atom entry document. This meta data file has to be valid against the arXiv file format.

The aim of this chapter is to analyse the format provided by Invenio to find a way to designing a suitable format for the arXiv submission.

To start with this study, it is important to analyse the standard way metadata is represented in the library domain. This standard is called MARC and is also the basic representation of Invenio metadata.

Then a more comfortable way to use MARC metadata is to transform it in XML-based documents. This is also a format used in CDS Invenio.

After the analyse of the Invenio side metadata format, a analyse of the arXiv submission format is done.

Finally, all the informations documented in this chapter drive to the matching of metadata of Invenio and metadata of arXiv.

5.1 Invenio

Invenio stores the metadata in a format called MARC. MARC is a standard metadata format used by the libraries since the late 60's. These metadata can be stored in simple text files.

To easily understand how it works, the Figure 10 shows some lines of such a document.

```
000000094 100_ $$aPeriwal, V$$uPrinceton University
000000094 245_ $$aMatrices on a point as the theory of everything
000000094 260_ $$c1997
000000094 269_ $$aPrinceton, NJ$$bPrinceton Univ. Joseph-Henry Lab. Phys.$$c14 Nov 1996
000000094 300_ $$a5 p
000000094 520_ $$aIt is shown that the world-line can be eliminated in the matrix quantum
```

Figure

9: MARC example

In the example above, three columns are distinguished. These are described in the Figure 11

000000094 100_ \$\$aPeriwal, V\$\$uPrinceton University

Figure 10: MARC format

- Reference to the resource (000000094)
- Type of the metadata (Periwal)
- Value of the metadata (Princeton University)

Some type of metadata have many subtype.

The Figure 12 resume the most relevant type of Invenio.

| METADATA CONCEPT | PROPOSED MARC 21 REPRESENTATION |
|-----------------------------|---|
| ----- | ----- |
| Abstract | 520 \$a |
| Author, first | 100 \$a |
| Author(s), additional | 700 \$a |
| Collection identifier | 980 \$a |
| Email | 8560 \$f |
| Imprint | 260 \$a,b,c; 300 \$a |
| Keywords | 6531 \$a |
| Language | 041 \$a |
| OAI identifier | 909C0 \$o |
| Publication info | 909C4 \$* [many subfields] |
| References | 999C5 \$* [many subfields] |
| Primary report number | 037 \$a [unique throughout the system!] |
| Additional report number(s) | 088 \$a |
| Series | 490 \$a,v |
| Subject | 65017 \$a |
| Title | 245 \$a |
| URL (e.g. to fulltext) | 8564 \$u, \$z |

Figure

11: Main MARC fields

The MARC format is not convenient to use for computers. To improve that, Invenio has developed a module that translate MARC file in a XML format.

5.1.1 MARCXML

The MARCXML format is much easier to process than text MARC, because it is in XML. The metadata type are still id number, as shown in the code below but the format is much more convenient and “machine readable”.

```
<collection>
  <record>
    <controlfield tag="001">1181477</controlfield>
    <controlfield tag="003">SzGeCERN</controlfield>
    <controlfield tag="005">20100608135444.0</controlfield>
    <datafield tag="035" ind1=" " ind2=" ">
      <subfield code="a">Inspire</subfield>
    </datafield>
    <datafield tag="037" ind1=" " ind2=" ">
      <subfield code="a">CERN-IT-Note-2009-019</subfield>
    </datafield>
    <datafield tag="041" ind1=" " ind2=" ">
      <subfield code="a">eng</subfield>
    </datafield>
    <datafield tag="100" ind1=" " ind2=" ">
      <subfield code="a">Ivanov, R</subfield>
      <subfield code="u">CERN</subfield>
    </datafield>
    <datafield tag="245" ind1=" " ind2=" ">
      <subfield code="a">
        INSPIRE: a new scientific information system for HEP
      </subfield>
    </datafield>
  ...

```

As the generation of the MARCXML format is implemented by Invenio, it is obvious that it will be used as metadata source format to the generation of the arXiv format.

5.2 arXiv

arXiv defines a range of metadata that helps managing and retrieving stored documents. The example bellow [Figure 13] shows a full example of a metadata file that is recognized by arXiv in its SWORD submission interface.

```
<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Superconformal Symmetry in Linear Sigma Model on Supermanifolds</title>
  <id>E102241E-1BA8-11D0-81A6-69C93A33DA4A</id>
  <updated>2008-05-06T20:13:23Z</updated>
  <author>
    <name>A. Editor</name>
    <email>editor@example.com</email>
  </author>
  <contributor>
    <name>Shigenori Seki</name>
    <email>Shigenori Seki &lt;seki@...phys.kyoto-u.ac.jp>></email>
  </contributor>
  <contributor>
    <name>Katsuyuki Sugiyama</name>
  </contributor>
  <contributor>
    <name>Tatsuya Tokunaga</name>
    <arxiv:affiliation xmlns:arxiv="http://arxiv.org/schemas/atom">Kyoto Univ.</arxiv:affiliation>
  </contributor>
  <content type="xhtml">
    <div xmlns="http://www.w3.org/1999/xhtml">SWORD/APP arXiv submission wrapper</div>
  </content>
  <summary>We consider a gauged linear sigma model in two dimensions
  with Grassmann odd chiral superfields. We investigate the Konishi anomaly of
  this model and find out the condition for realization of superconformal
  symmetry on the world-sheet. When this condition is satisfied, the theory is
  expected to flow into conformal theory in the infrared limit. We construct
  superconformal currents explicitly and study some properties of this
  world-sheet theory from the point of view of conformal field theories.</summary>
  <category term="http://arxiv.org/terms/arXiv/hep-th"
    scheme="http://arxiv.org/terms/arXiv/"
    label="High Energy Physics - Theory"/>
  <arxiv:primary_category xmlns:arxiv="http://arxiv.org/schemas/atom"
    scheme="http://arxiv.org/terms/arXiv/"
    label="High Energy Physics - Theory"
    term="http://arxiv.org/terms/arXiv/hep-th"/>
  <arxiv:comment xmlns:arxiv="http://arxiv.org/schemas/atom">24 pages, 2 figures</arxiv:comment>
  <arxiv:journal_ref xmlns:arxiv="http://arxiv.org/schemas/atom">Nucl.Phys. B753 (2006) 295-312</arxiv:journal_ref>
  <arxiv:report_no xmlns:arxiv="http://arxiv.org/schemas/atom">KUNS-2018</arxiv:report_no>
  <arxiv:report_no xmlns:arxiv="http://arxiv.org/schemas/atom">YITP-06-19</arxiv:report_no>
  <arxiv:doi xmlns:arxiv="http://arxiv.org/schemas/atom">10.1016/j.nuclphysb.2006.07.013</arxiv:doi>
  <link href="https://arxiv.org/sword-app/edit/08050001" type="application/zip" rel="related"/>
</entry>
```

Figure 12: submission metadata file to arXiv

arXiv distinguishes two kinds of metadata : The mandatory one and the optional one. In addition some fields of the metadata can be repeated. The multiples entries are written with a small stars (*) according to the standard XML representation.

5.2.1 Mandatory fields

The mandatory elements are those that must be present in the XML atom entry submitted to arXiv.

- **<title>** : The title will be checked against other recently arXiv-ed papers to avoid inadvertent double submissions.
- **<author>** : The author element is used for the authenticated user, i.e. the person who initiates the submission. It has nothing to do with the author of the submitted paper.
- **<contributor>*** : The repeatable contributor element is used to specify the individual authors of the material being deposited to arXiv. At least one /contributor/email node must be present in order to inform arXiv of the email address of the primary contact author. If multiple /contributor/email nodes are found, the first will be used. Optionally the primary contact author's (name and) email address can also be specified in the **X-On-Behalf-Of** HTTP header extension.
- **<summary>** : The summary must be at least 20 characters long
- **<arxiv:primary_category ...>** : The *arxiv:primary_category* must be selected from the list of available choices for the collection, as given in the servicedocument
- **<link rel=related ...>*** : The *link(s)* of relation "*related*" must refer to the previously deposited material.

If one of those node is missing or is not well-formed, the submission will not be considered and the SWORD server will answer with the error « 400 : Bad request »

5.2.2 Optional metadata

- **<category>*** : Category element(s) can be used to provide secondary classification(s), selected from those listed in the servicedocument, e.g. :
 - *High Energy Physics - Theory*
- **<arxiv:comment>** : Indicate number of pages and number of figures containing by the submission file, e.g. :
 - *24 pages, 2 figures*
- **<arxiv:journal_ref>*** : This field is only for a full bibliographic reference if the article has already appeared in a journal or a proceedings. Indicate the volume number, year, and page number, e.g. :
 - *J.Hasty Results 1 (2008) 1-9*
- **<arxiv:doi>** : This field is only for a DOI (Digital Object Identifier) that resolves (links) to another version of the article, in a journal for example. DOIs have the form
 - *10.1016/S0550-3213(01)00405-9*

- **<arxiv:report_no>*** : Put in the institution's locally assigned publication number, e.g. :
 - *KUNS-2018*
- **<arxiv:affiliation>*** : Put the affiliation of the author in this tag, e.g. :
 - *Kyoto Univ.*

If it is specified, this node must be child of the contributor's node.

5.3 SWORD matching format

To allow the SWORD client to submit correct and well-formed metadata to arXiv, the metadata must be taken from Invenio and formatted in the arXiv XML atom entry. Most of the metadata are found in the XML Marc file but there will also be some of them taken from other sources:

- **author** : taken from authentication values
- **primary categories** : taken from arXiv servicedocument
- **link** : find in the media deposit xml atom response
- **categories** : taken from arXiv servicedocument

All other informations are stored in the MARC file.

5.3.1 MARC information

As already seen in the chapter 5.1, the MARC file contains informations referenced by a tag number. The table below shows the mapping between MARC entries and arXiv metadata node.

| Comment | arXiv (xml Node) | | Invenio (MARC) | |
|------------------------|------------------|---------------------|----------------|------------|
| | Parent node | Node | Tag number | code value |
| Document title | <entry> | <title> | 245 | a |
| ID of the resource | <entry> | <id> | 37 | a |
| Author of the document | <contributor> | <name> | 100 | a |
| Contributor | <contributor> | <name> | 700 | a |
| Abstract | <entry> | <summary> | 520 | a |
| Nb page, pictures, ... | <entry> | <arxiv:comment> | 500 | a |
| Journal reference | <entry> | <arxiv:journal_ref> | 650 | a |
| Digital Object Id | <entry> | <arxiv:doi> | 773 | a |
| Report number | <entry> | <arxiv:report_no> | 88 | a |
| Author affiliation | <contributor> | <arxiv:affiliation> | 100, 700 | u |

6 BibSword

BibSword is an Invenio module that implements every function of the communication for a SWORD client to a SWORD server. It is the main part of this project. It offers an interface (the BibSword API) for a client application to abstract every specificity of a SWORD communication.

The BibSword API (Application Programmers Interface) is an interface between the Invenio local client and any remote server implementing a SWORD interface. The aim of the API is to facilitate any SWORD-client implementation by standardizing:

- the use cases allowed by the SWORD protocol
- the format of the parameters and the return values.
- the workflow of a SWORD submission process

This chapter describes the design and the implementation of the BibSword API. It contains the following sections:

- The overview of the API shows every action that a client can perform.
- The operation describes in details each functionalities and their implementations. An example is given for each use case.
- The workflow offers a view of an API standard usage.
- The class diagram presents the way the API has been implemented

Some use cases of the API might seem to be too simple are too much decomposed. The purpose of this decomposition in small units is both to facilitate the development and secondly to provide flexibility in programming SWORD-client.

6.1 Prerequisite

To ensure the good work of the use of the BibSword API, some remarks have to be considered.

6.1.1 Authentication

When user wants to submit a digital document, he first has to get it on the Invenio database. For this action, he should be authenticated by the Invenio system.

However the BibSword API does not specify any authentication to access the resources managed by Invenio.

The responsibility to check access right is therefore left to the programmer implementing the client.

6.1.2 Acknowledgement

Once a submission is done, the SWORD system sends an acknowledgement email to the submitter. For this reason, the client implementation using the BibSword API must be able to provide the email address of the submitter.

6.2 Integrity

Integrity of data means that data have to be correctly and completely submitted on the remote repository. It also involves that data already sent must not be submitted again to avoid duplicate resource on a remote server.

To ensure data integrity several controls and operations before and after a resource submission have to be done:

- Before the submission:
 - Is the resource already stored on the remote server?
 - Has the resource already been submitted from Invenio to the remote server?
 - Are formats of the media and metadata correct and well-formed for the remote server collection?
- After the submission occurred:
 - Has any error happened during the submission process?
 - Has the remote server accepted the resource?

To allow the control of the submission, some data has to be serialized in the system so the programs can retrieve the state of a submission. To achieve this, three techniques have been designed:

- **Creation of two tables in the Invenio database [Figure 14]:**
 - **swrREMOTESERVER:** contains server information such as authentication information, URL, servicedocument, ...
 - **swrDATA:** contains submission's informations such as username / email of the submitter, data, status, ...

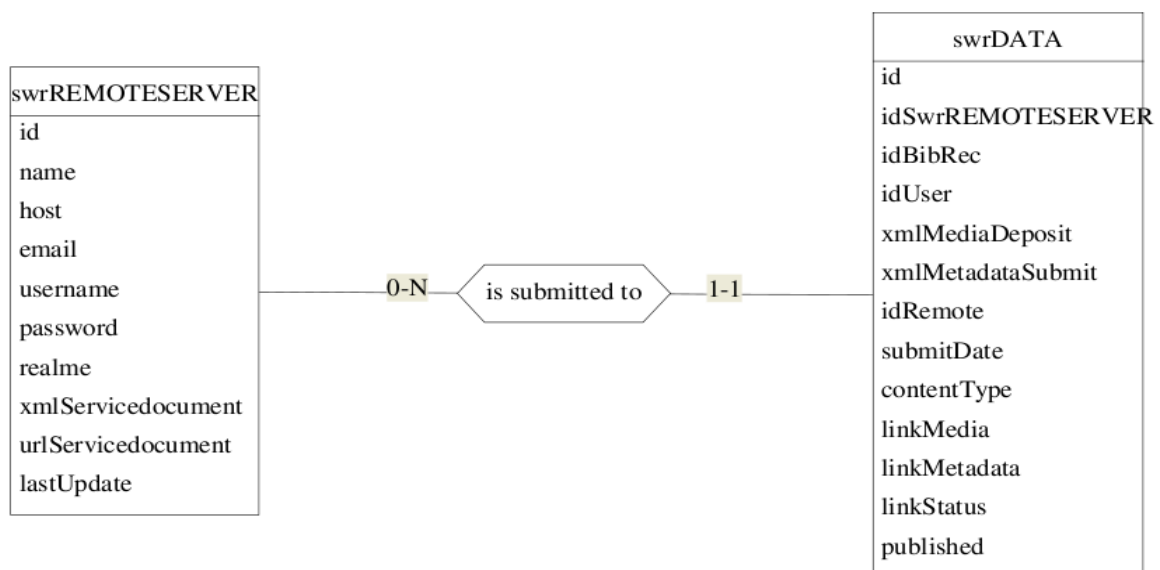


Figure 13: database model

- **Insertion of a fields in the MARC file:** this field is used to inform the librarian that the current record has been sent to a SWORD remote server. It contains the id given by the remote server. It is also saved in the `idRemote` field of the `swrDATA` table.
- **Implementation of a “check state daemon”:** a daemon is a task that is automatically launch at a given time periodically. In BibSword a daemon is used to check once a day the state of every “Submitted” resource to know if they have been accepted by the remote server.

6.2.1 swrREMOTESERVER

The SWORD remote server contains information needed to connect to the server, the entire service document and information to reload if needed.

- **id:** unique number that allows to identify the server.
- **name:** the name of the server displayed to the user. For example : *arXiv*.
- **host:** the host name of the server is only a reference to the root host site. For example : *http://arxiv.org*.
- **username:** name used to authenticate the application to the remote server. For example : *CDS_Invenio*.
- **password:** password used for the authentication. For example *sword_invenio*.
- **email:** the email address where to sent the acknowledgement of a submission.
- **realm:** root URL for the space accessed by the authentication on the server.
- **urlServicedocument:** address of the servicedocument location.
- **xmlServicedocument:** entire servicedocument containing server's informations, collections, categories.
- **lastUpdate:** timestamps of the last update of the servicedocument. Usually used to know if the servicedocument needs to be reloaded from its URL.

6.2.2 swrDATA

The SWORD Data table contains information about a single submission.

- **id:** unique number that identifies the submission. It is also used in the MARC file to retrieve an already submitted resource.
- **idSwrREMOTESERVER:** foreign key that identifies the server where the submission has been made.
- **idBibRec:** foreign key that identifies the record on the Invenio system. Used to retrieve the metadata file.
- **IdUser:** Invenio user that started the submission.
- **xmlMediaDeposit:** response from the server after the deposit of media. Can be an acknowledgement as well as a SWORD error message.

- ***xmlMetadataSubmit***: response returned by the server after the submission of metadata. Can be an acknowledgement but also a SWORD error message.
- ***idRemote*** : identifier given by the remote server for the submitted resource.
- ***submitDate*** : Date when the submission has been initialized.
- ***content-type*** : Type of the submitted media, e.g. : pdf, latex, gif, ...
- ***linkMedia*** : url of the remote location for the submission.
- ***linkMetadata*** : url of the remote metadata file for the submission.
- ***linkStatus*** : url where to GET the status of the submission.
- ***published***: boolean indicating if the submission is published on the repository.

6.3 Overview

The overview of the BibSword API is represented by an use case diagram [Figure 15]. Every case is an action offers from the API to the clients. For readability reasons, the client is not tied to every cases. The cases on which the client is tied are only those required for a submission.

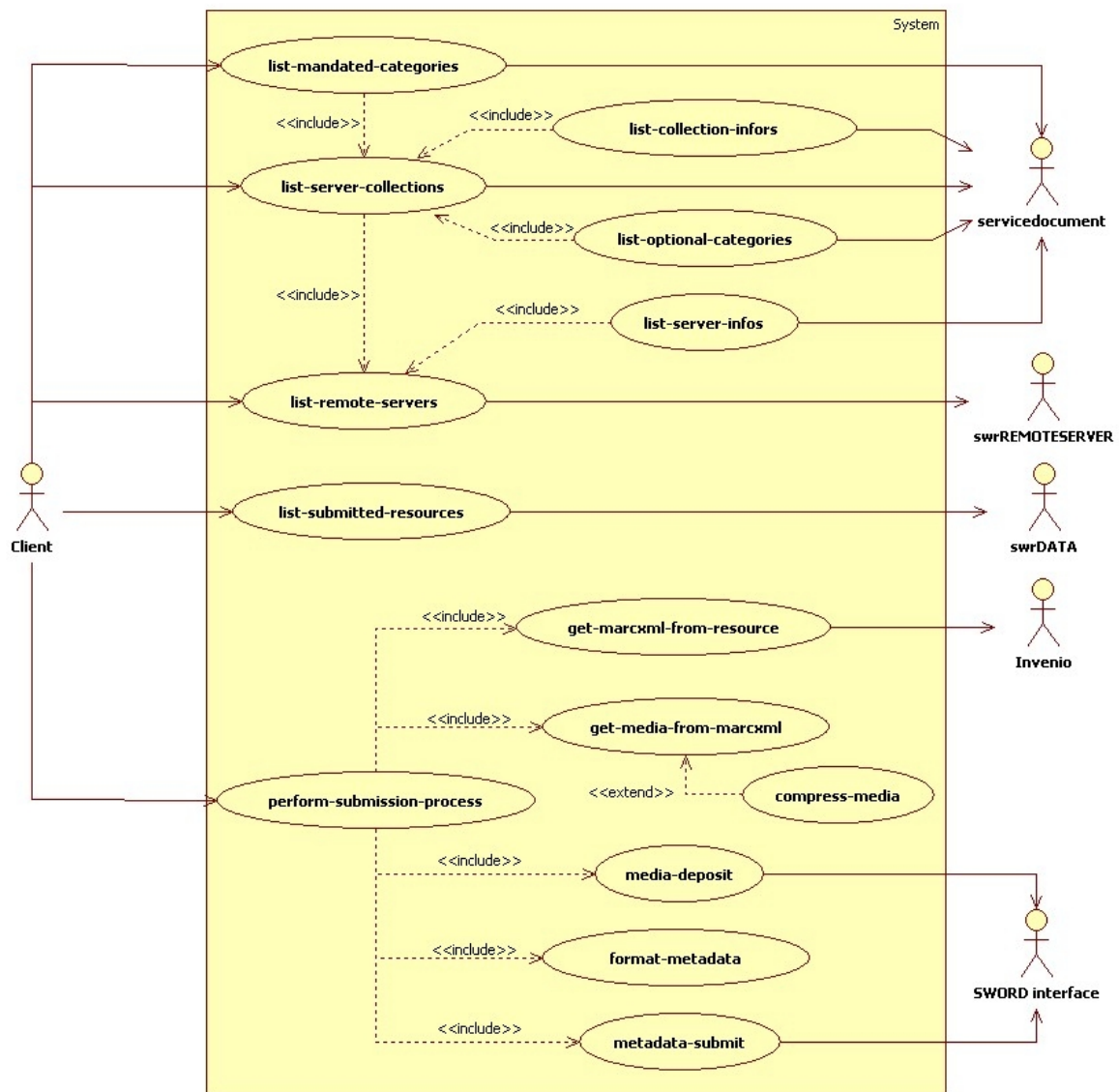


Figure 14: Overview of the API

To improve the comprehension of the API the external system are also represented on the diagram.

Some of internal actions such as SWORD server authentication or service document retrieving are not represented on overview because it would make it too complicated. However, they are described in details in the operation chapter.

6.4 CLI

CLI stands for Command Line Interface. This is a client that implements every single functions of the API. This client was used to test and gives examples of the good operations of each elements of the SWORD client API. The CLI offers a user manual [Figure 16] by giving the option “-h”:

```
Usage: ./bibsword [options] <webdocname>

*****
                        OPTIONS
*****
-h, --help           : Print this help.
-s, --simulation     : Proceed in a simulations mode

*****
                        HELPERS
*****
-r, --list-remote-servers : List all available remote server
-i, --list-server-info --server-id : Display SWORD informations about server
-c, --list-collections --server-id : List collections for the specified server
-p, --list-primary-categories --server-id --collection_id : List mandated categories
-o, --list-optional-categories --server-id --collection_id : List secondary categories
-v, --list-submission [--server-id --id_record] : List submission entry in swrDATA

*****
                        OPERATIONS
*****
-m, --get-marcxml-from-recv --recv : Display the MARCXML file for the given record
-e, --get-media-resource [--marcxml-file]--recv] : Display type and url of the media
-z, --compress-media-file [--marcxml-file]--recv] : Display the zipped size archive
-d, --deposit-media --server-id --collection_id --media : deposit media in collection
-f, --format-metadata --server-id --metadata --marcxml : format metadata for the server
-l, --submit-metadata --server-id --collection-id --metadata : submit metadata to server
-a, --proceed-submission --server-id --recv --metadata : do the entire process of deposit
```

Figure

15: CLI user manual

6.5 API

The API is composed of all operations offers to the any SWORD client. Each operations is described in details in the sub chapters of the API. They all contains the same points:

- Syntax: the signature of the function.
- Description: small description of the operation and the utility of the function.
- Operations: sequence diagram that shows operations of the function in details.
- Parameters: type and format of the value argument given to the function.
- Return value: type and format of the value returned by the function
- Example: standard use of the function and its result

Those informations should allow developers to understand how to call the BibSword API to implement a SWORD client.

6.5.1 list-remote-servers

Syntax : *list_remote_servers()*

Description : Get the list of remote servers implemented by the Invenio SWORD API

Operations: [Figure 17 : diagram sequence: list_remote_servers]

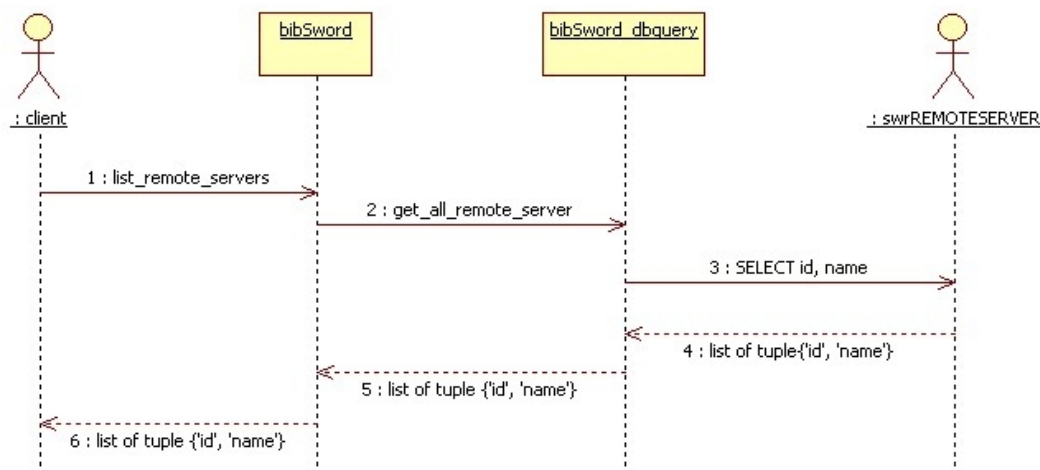


Figure 16: diagram sequence: list_remote_servers

Parameters : None

Return value : list of tuples [{ 'id' , 'name' }]

Example:

```
./bibsword -r
1 : sword_test ( SWORD.org )
2 : arXiv ( arxiv.org )
```

6.5.2 list_remote_server_informations

Syntax : *list_remote_server_informations(id_server)*

Description : List all informations about the server's options such as SWORD version, maximum upload size [Kb], acceptance of simulation mode, ...

Operations : [Figure 18 : diagram sequence : list_remote_server_informations]

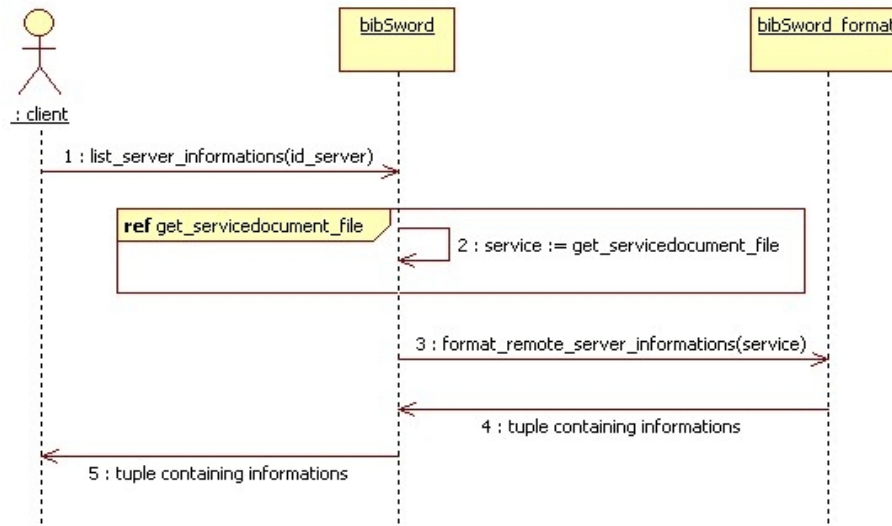


Figure 17: diagram sequence : list_remote_server_informations

Parameters : *id_server* (int) : identifier of the server in the table swrREMOTESERVER

Return : information's tuple : { 'version', 'maxUploadSize', 'verbose', 'noOp' }

Errors : Server id must be an id of a swrREMOTESERVER table

Example :

```

./bibsword -i --server-id 3
SWORD version : 1.3
Maximal upload size [Kb] : 10000
Implements verbose mode : true
Implements simulation mode : true
  
```

6.5.3 list_collections_from_server

Syntax : *list_collections_from_server(id_server)*

Description : List all the collections found in the servicedocument of the given server.

Operations : [Figure 19 : sequence diagram : list-collections-from-server]

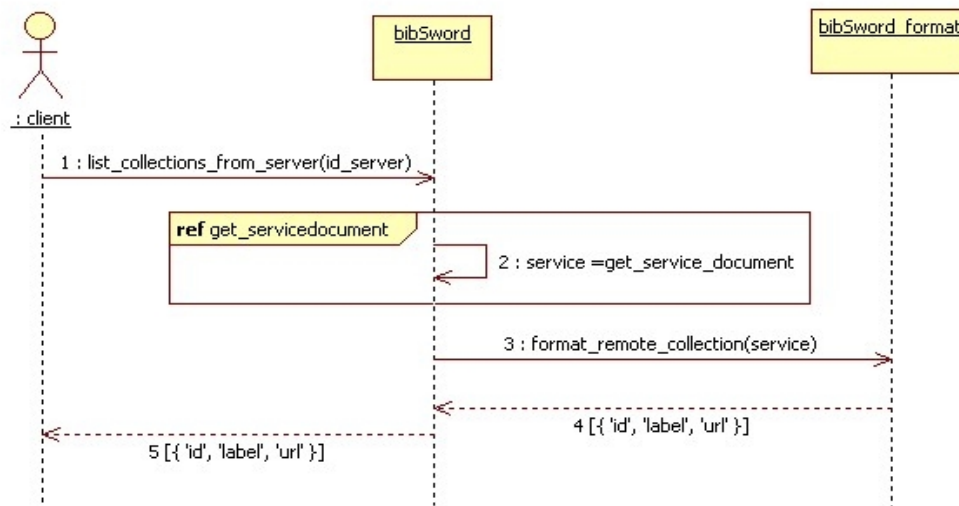


Figure 18: sequence diagram : list-collections-from-server

Parameters : *id_server* (int) : identifier of the server in the table swrREMOTESERVER

Return : list of tuples [{ 'id', 'label', 'url' }]

Example :

```

./bibsword -c --server-id 3
0 : The Physics archive - https://arxiv.org/sword-app/physics-collection
1 : The Quantitative Biology archive - https://arxiv.org/sword-app/q-bio-collection
2 : The Statistics archive - https://arxiv.org/sword-app/stat-collection
3 : The Computer Science archive - https://arxiv.org/sword-app/cs-collection
4 : The Quantitative Finance archive - https://arxiv.org/sword-app/q-fin-collection
5 : The Mathematics archive - https://arxiv.org/sword-app/math-collection
6 : The Nonlinear Sciences archive - https://arxiv.org/sword-app/nlin-collection
  
```

6.5.4 list_collection_informations

Syntax : *list_collection_informations(id_server, id_collection)*

Description : List all information concerning the collection such as the list of accepted type of media, if the collection allows mediation, if the collection accept packaging, ...

Operations : [Figure 20 : sequence diagram : list_collection_informations]

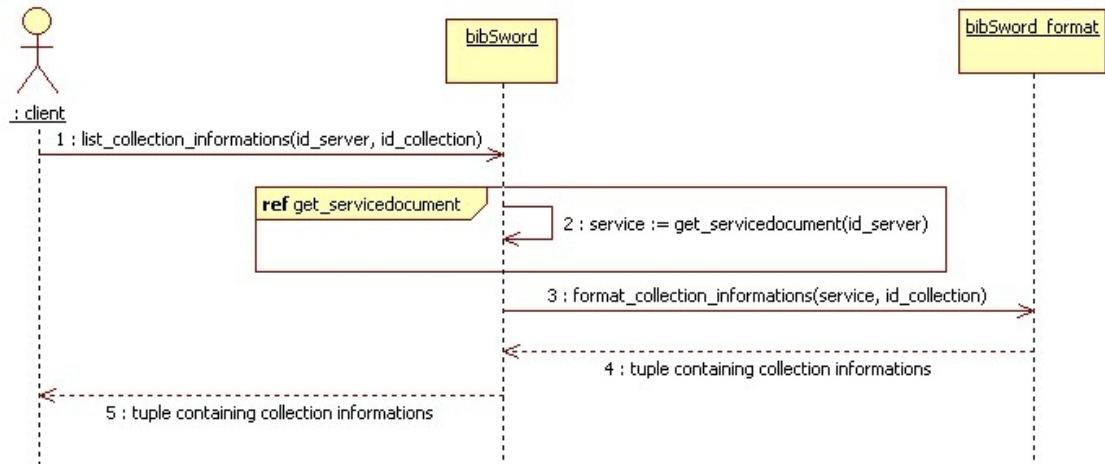


Figure 19: sequence diagram : list_collection_informations

Parameters : *id_server* (int) : identifier of the server in the table swrREMOTESERVER
id_collection : id of the collection found in collection listing

Return : information's tuple : {[accept], 'collectionPolicy', 'mediation', 'treatment', 'acceptPackaging'}

Example :

```
./bibsword -n --server-id 3 --collection-id 1
```

Accepted media types:

```
- application/atom+xml;type=entry
- application/zip
- application/xml
- application/pdf
- application/postscript
- application/vnd.openxmlformats-officedocument.wordprocessingml.document
- text/xml
- image/jpeg
- image/jpg
- image/png
- image/gif
```

collection policy : Open Access

mediation allowed : true

treatment mode : will be posted pending moderator approval

location of accept packaging list : <http://purl.org/net/sword-types/bagit>

6.5.5 list_mandated_categories_from_collection

Syntax : *list_mandated_categories_from_collection(id_server, id_collection)*

Description : The primary categories are the mandated categories for a collection. In some SWORD implementation, they are not used

Operations : [Figure 21 : sequence diagram : list_mandated_categories_from_collection]

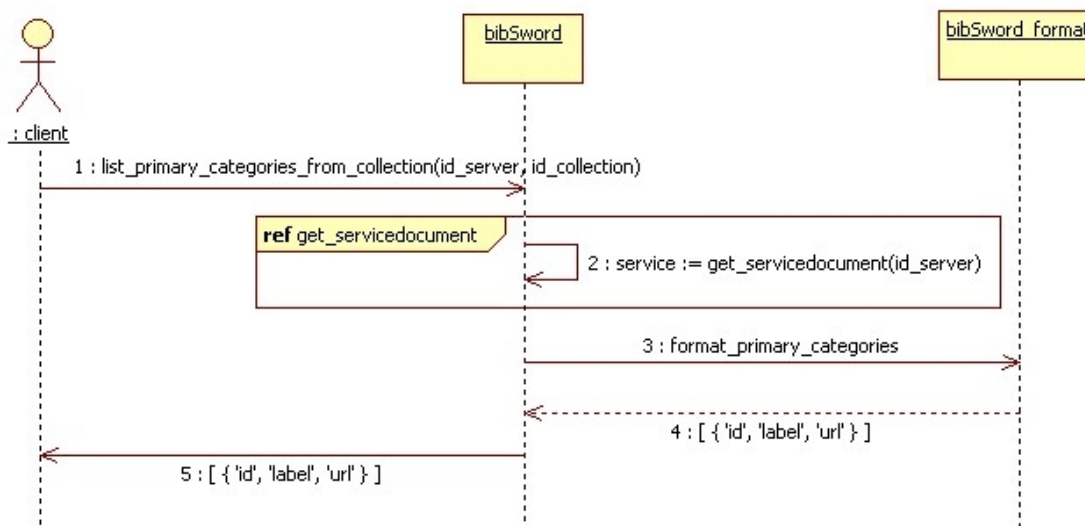


Figure 20: sequence diagram : list_mandated_categories_from_collection

Parameters : *id_server* (int) : identifier of the server in the table swrREMOTESERVER
id_collection : id of the collection found in collection listing

Return : list of mandated category's tuple [{ 'id', 'label', 'url' }]

Example :

```

./bibsword -p --server-id 3 --collection-id 1
0 : Quantitative Biology - Biomolecules - http://arxiv.org/terms/arXiv/q-bio.BM
1 : Quantitative Biology - Cell Behavior - http://arxiv.org/terms/arXiv/q-bio.CB
2 : Quantitative Biology - Genomics - http://arxiv.org/terms/arXiv/q-bio.GN
3 : Quantitative Biology - Molecular Networks -
http://arxiv.org/terms/arXiv/q-bio.MN
4 : Quantitative Biology - Neurons and Cognition -
http://arxiv.org/terms/arXiv/q-bio.NC
5 : Quantitative Biology - Other Quantitative Biology -
http://arxiv.org/terms/arXiv/q-bio.OT
6 : Quantitative Biology - Populations and Evolution -
http://arxiv.org/terms/arXiv/q-bio.PE
7 : Quantitative Biology - Quantitative Methods -
http://arxiv.org/terms/arXiv/q-bio.QM
8 : Quantitative Biology - Subcellular Processes -
http://arxiv.org/terms/arXiv/q-bio.SC
  
```

6.5.6 list_optional_categories_from_collection

Syntax : *list_optional_categories_from_collection(id_server, id_collection)*

Description : The optional categories are only used as search option to retrieve the resource.

Operations : [Figure 22 : sequence diagram : list_optional_categories_from_collection]

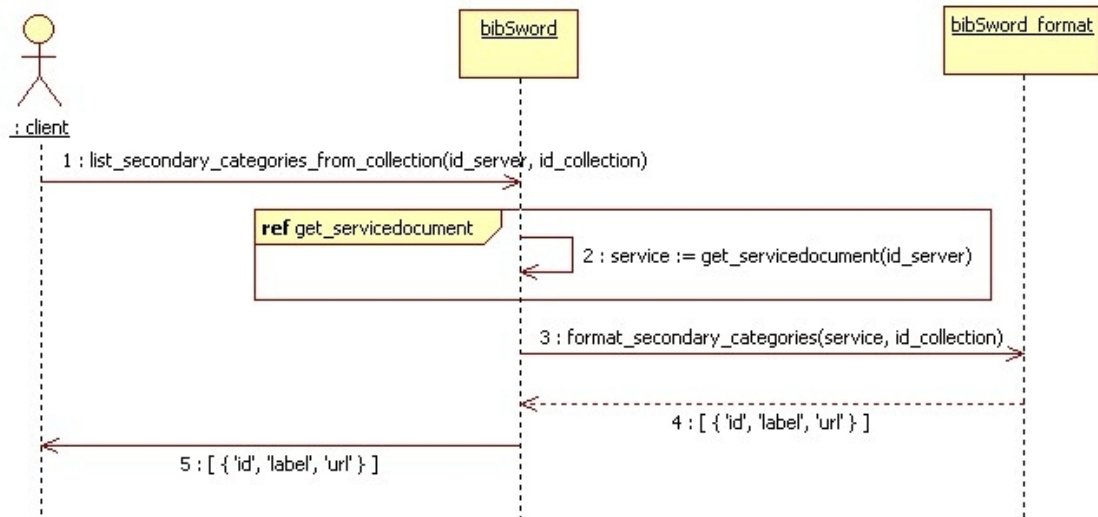


Figure 21: sequence diagram : list_optional_categories_from_collection

Parameters : *id_server* (int) : identifier of the server in the table swrREMOTESERVER
id_collection : id of the collection found in collection listing

Return : list of optional category's tuple [{ 'id', 'label', 'url' }]

Example :

```

./bibsword -o --server-id 3 --collection-id 1
0 : Physics - Cosmology and Extragalactic Astrophysics -
http://arxiv.org/terms/arXiv/astro-ph.CO
1 : Physics - Earth and Planetary Astrophysics -
http://arxiv.org/terms/arXiv/astro-ph.EP
2 : Physics - Galaxy Astrophysics - http://arxiv.org/terms/arXiv/astro-
ph.GA
3 : Physics - High Energy Astrophysical Phenomena -
http://arxiv.org/terms/arXiv/astro-ph.HE
4 : Physics - Instrumentation and Methods for Astrophysics -
http://arxiv.org/terms/arXiv/astro-ph.IM
5 : Physics - Solar and Stellar Astrophysics -
http://arxiv.org/terms/arXiv/astro-ph.SR
6 : Physics - Disordered Systems and Neural Networks -
http://arxiv.org/terms/arXiv/cond-mat.dis-nn
... (more than 100)
  
```

6.5.7 list_submitted_resources

Syntax : *list_submitted_resources(id_server=0, id_record=)*

Description : List the swrDATA table informations such as submitter, date of submission, link to the resource and status of the submission. It is possible to limit the amount of result by specifying a remote server, the id of the bibRecord or both

Operations : [Figure 23 : sequence_diagram : list_submitted_resources]

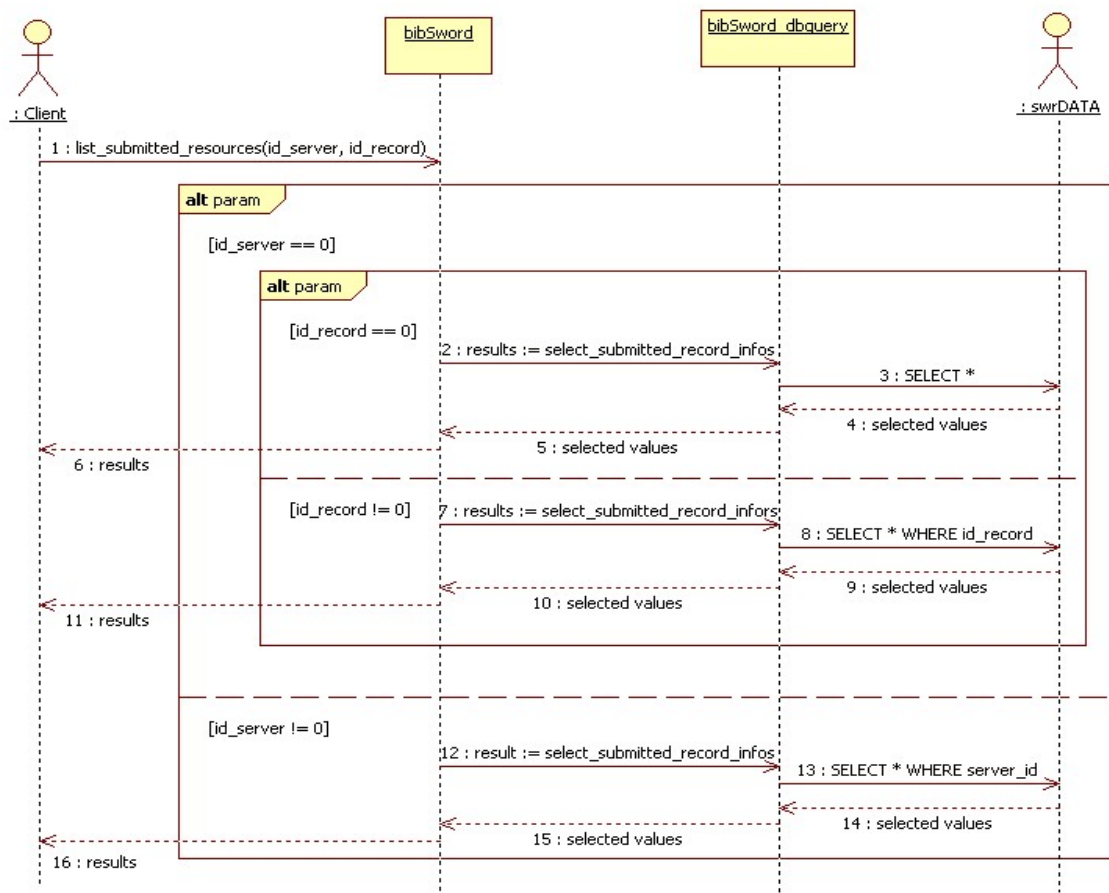


Figure 22: sequence_diagram : list_submitted_resources

Parameters : *id_server* : id of the remote server where to list the submissions, if not specified, list all servers submissions

id_record : unique name of the record in the Invenio MARC file

Return : list of submission's tuple [{ 'id', 'links', 'type', 'submitter', 'date', 'status' }]

Example :

```

./bibsword -v
submission id : 1
remote server id : 3
submitter id : 1
local record id : 34038281
remote record id : info:arxiv/app/10070221 ...
  
```

6.5.8 get_marxml_from_record

Syntax : get_marxml_of_record(recid)

Description : Return a string containing the metadata in the format of a marxml file. The marxml is retrieved by using the given record id

Operations : [Figure 24 : sequence diagram : get_marxml_from_record]

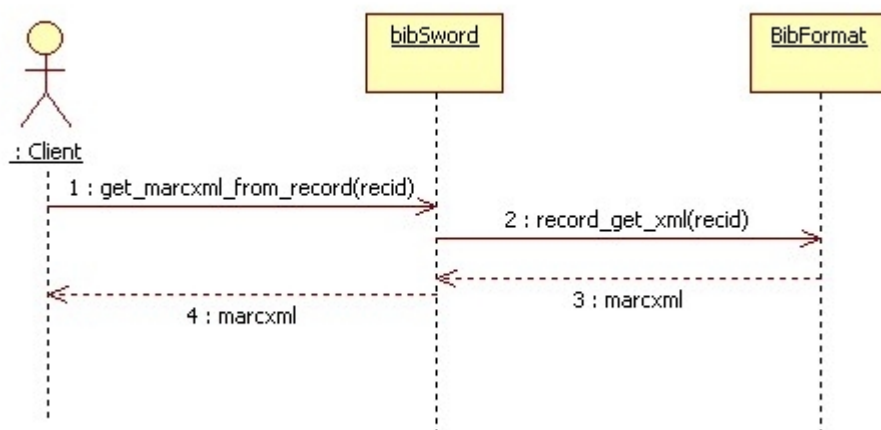


Figure 23: sequence diagram : get_marxml_from_record

Parameters : recid : id of the record to be retrieve on the database

Return : string containing the marxml file of the record

Example :

```

./bibsword -m --recid 97
<record>
  <controlfield tag="001">97</controlfield>
  <controlfield tag="003">SzGeCERN</controlfield>
  <controlfield tag="005">20060914104330.0</controlfield>
  <datafield tag="035" ind1=" " ind2=" ">
    <subfield code="9">INIS</subfield>
    <subfield code="a">34038281</subfield>
  </datafield>
  <datafield tag="035" ind1=" " ind2=" ">
    <subfield code="9">UNCOVER</subfield>
    <subfield code="a">251,129,189,013</subfield>
  </datafield>
  <datafield tag="041" ind1=" " ind2=" ">
    <subfield code="a">eng</subfield>
  </datafield>
  <datafield tag="088" ind1=" " ind2=" ">
    <subfield code="9">SCAN-0005061</subfield>
  </datafield>
  ...
  
```


6.5.9 get_media_from_marcxml

Syntax : `get_media_from_marcxml(id_server, marcxml)`

Description : Parse the marcxml file to retrieve the link toward the media. Get every media through its URL and set each of them and their type in a list of tuple.

Operations : [Figure 25 : sequence diagram : `get_media_from_marcxml`]

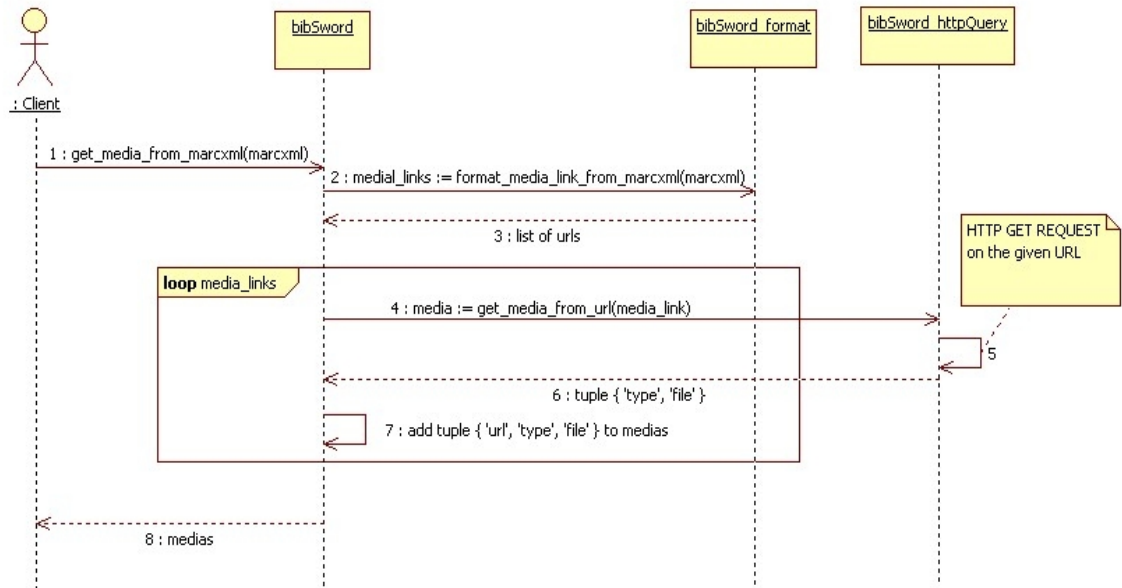


Figure 24: sequence diagram : `get_media_from_marcxml`

Parameters (marcxml) : string containing a marcxml file

Return : list of tuples : [{ 'type', 'file' }]

Example :

```

./bibsword -e --recid 96
media_link = http://localhost/record/96/files/0002060.ps.gz
media_type = application/postscript
media_link = http://localhost/record/96/files/0002060.ps.gz
media_type = application/postscript
(followed by the 2 files not displayed)
  
```

6.5.10 compress_media_file

Syntax : compress_media_file(media_file_list)

Description : Compress each file of the given list in a single zipped archive and return this archive in a new media file list containing only one tuple

Operations : [Figure 26 : sequence diagram : compress_media_file]

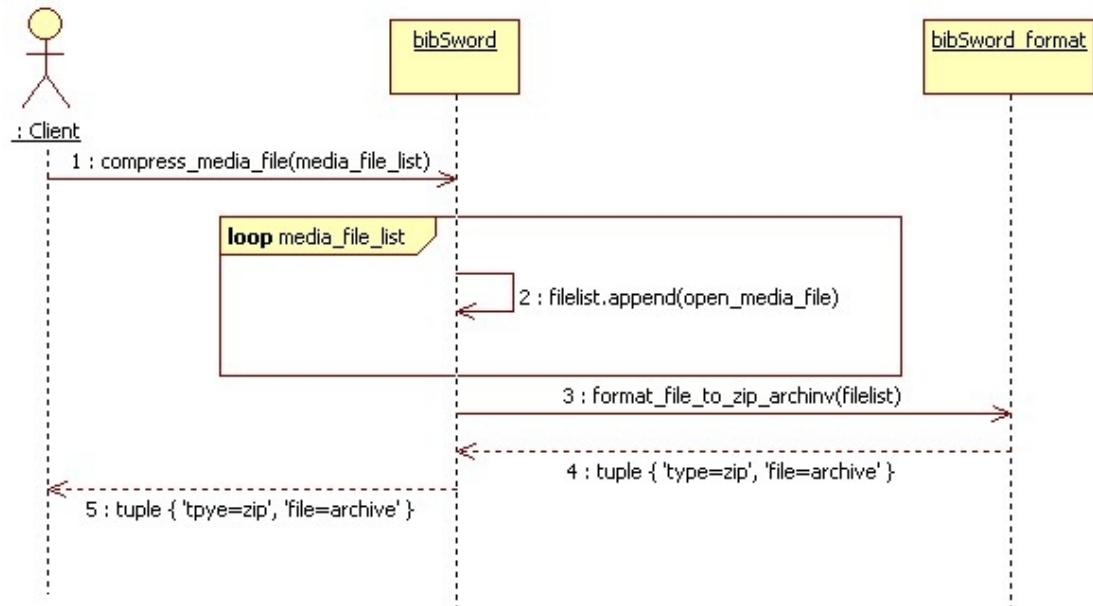


Figure 25: sequence diagram : compress_media_file

Parameters (media_file_list) : list of tuple { 'type', 'file' }

Return : list containing only one tuple { 'type=zip', 'file=archive' }

Example :

```

./bibsword -z --recid 96
media_link = http://localhost/record/96/files/0002060.ps.gz
media_type = application/zip
(followed by the zip archive containing 2 zipped files)
  
```

6.5.11 deposit_media

Syntax : *deposit_media(server_id, media_file_list, deposit_url, username="", email=")*

Description : Deposit all media containing in the given list in the deposit_url (usually the url of the selected collection. A user name and password could be selected if the submission is made 'on behalf of' an author

Operations : [Figure 27 : sequence diagram : desposit_media]

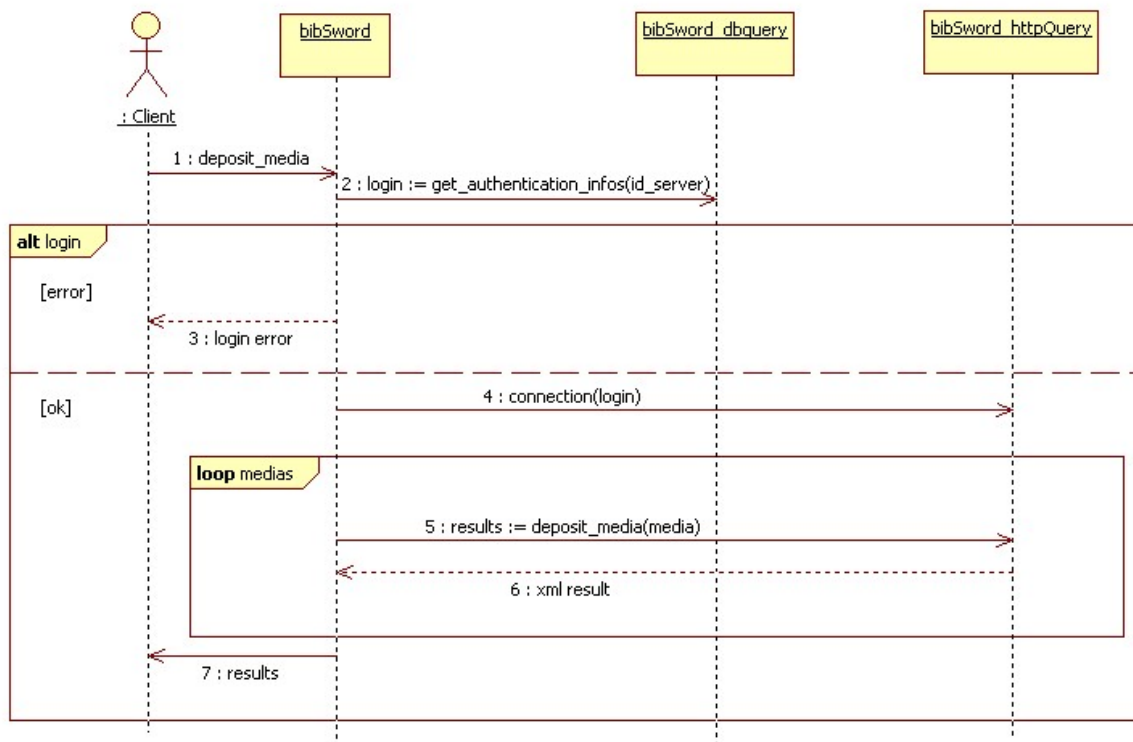


Figure 26: sequence diagram : desposit_media

Parameters (media_file_list) : list of tuple [{ 'type', 'file' }]

(deposit_url) : url of the deposition on the internet

(username) : name of the user depositing 'on behalf of' an author

(email) : allow user to get an acknowledgement of the deposit

Return : list of xml result file (could be sword error xml file)

Example :

```

./bibsword -d --server-id 3 --collection-id 3 --media test.pdf
media successful deposited !
media_link = https://arxiv.org/sword-app/edit/10070020
  
```

6.5.12 format_metadata

Syntax : `format_metadata(server_id, marcxml, deposit_results, user_info, metadata={})`

Description : Format an xml atom entry containing the metadata for the submission and the list of url where the media have been deposited.

Operations : There is only one call to the format metadata function to achieve this task.

Parameters (id_server) : identifier of the server in the table swrREMOTESERVER

(marcxml) : file containing metadata from Invenio

(deposit_result_list) : list of obtained response during deposition

(metedata) : optionally give other metadata that those from marcxml

Return : xml atom entry containing formatted metadata and links

Example :

```
./bibsword -f --server-id 3 --marcxml marcxml_test.xml
<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://www.w3.org/2005/Atom"
xmlns:arxiv="http://arxiv.org/schemas/atom">
<title>Superconformal Symmetry in Linear ... Supermanifolds</title>
<id>E102241E-1BA8-11DD-81A6-69C93A33DA4A</id>
<updated>2008-05-06T20:13:23Z</updated>
<author>
<name>CDS_Invenio</name>
<email>Matthieu.Barras@cern.ch</email>
</author>
<contributor>
<name>Mathieu Barras</name>
<email>mathieu.barras@edu.hefr.ch</email>
</contributor>
<content type="xhtml">
<div xmlns="http://www.w3.org/1999/xhtml">SWORD app submission
wrapper</div>
</content>
<summary>We consider a gauged linear ... theories.</summary>
<category term="http://arxiv.org/terms/arXiv/hep-th"
scheme="http://arxiv.org/terms/arXiv/"
label="High Energy Physics - Theory"/>
<arxiv:primary_category scheme="http://arxiv.org/terms/arXiv/"
label="High Energy Physics - Theory"
term="http://arxiv.org/terms/arXiv/hep-th"/>
<arxiv:comment>24 pages, 2 figures</arxiv:comment>
<arxiv:journal_ref>Nucl.Phys. B753 (2006) 295-312</arxiv:journal_ref>
<arxiv:report_no>KUNS-2018</arxiv:report_no>
<arxiv:report_no>YITP-06-19</arxiv:report_no>
<arxiv:doi>10.1016/j.nuclphysb.2006.07.013</arxiv:doi>
<link href="https://arxiv.org/sword-app/edit/10063064"
type="application/zip"
rel="related"/>
</entry>
```

6.5.13 submit_metadata

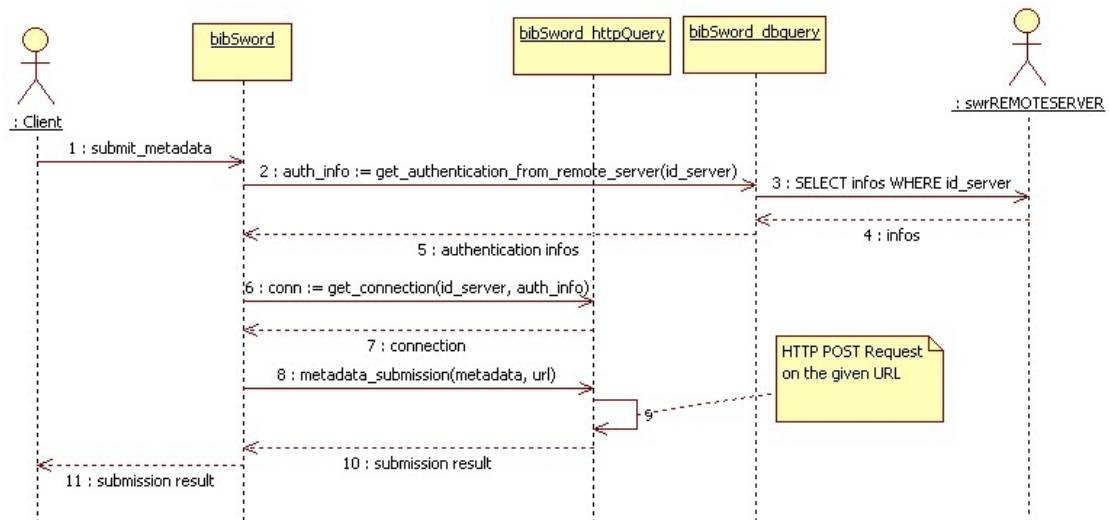
Syntax : *submit_metadata(metadata, deposit_url, username="", email=")*

Description :

Operation : [Figure 28 : sequence diagram : submit_metadata]

Description : Submit the given metadata xml entry to the deposit_url. A username and an email address might be used to priced on behalf of the real author of the document.

Operations : [Figure 28 : sequence diagram : submit_metadata]



Parameters (metadata) : xml atom entry containing every metadata and links

(deposit_url) : url of the deposition (usually a collection' url)

(username) : name of the user depositing 'on behalf of' an author

(email) : allow user to get an acknowledgement of the deposit

Return : xml atom entry containing submission acknowledgement or error

Example :

```

./bibsword -l --server-id 3 --collection-id 3 --metadata metatest.xml
metadata successful deposited !
metadata_link = https://arxiv.org/sword-app/edit/10070020.atom
related_link = https://arxiv.org/sword-app/resolve/app/10070020
  
```

6.5.14 perform_submission_process

Syntax : `perform_submission_process(server_id, user_id, metadata, collection, metadata="")`

Description : This function is an abstraction of the 2 steps submission process. It submit the media to a collection, format the metadata and submit them to the same collection. In case of error in one of the 3 operations, it stops the process and send an error message back. In addition, this unction insert informations in the swrDATA and MARC to avoid sending a record twice in the same remote server

Operations : [Figure 29 : sequence diagram : perform_submission_process]

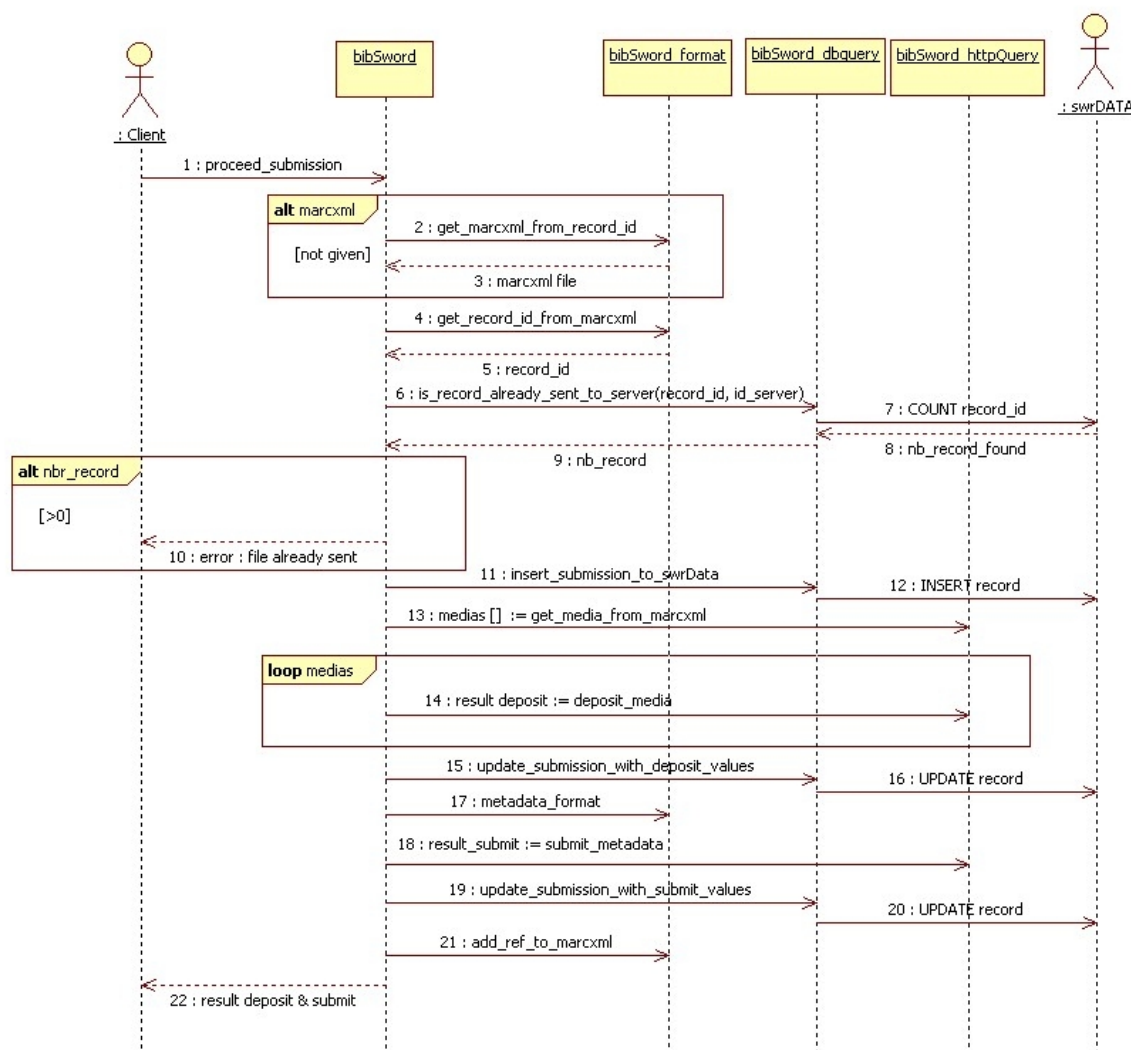


Figure 28: sequence diagram : perform_submission_process

Parameters (metadata) : xml atom entry containing every metadata and links

(deposit_url) : url of the deposition (usually a collection' url)

(username) : name of the user depositing 'on behalf of' an author

(email) : allow user to get an acknowledgement of the deposit

Return : xml atom entry containing deposit acknowledgement and submission acknowledgement

Example : see examples of deposit, format and submit process.

6.6 Workflow

The workflow of the forward resource process can easily be split in three phases:

- The selection of the destination (server, collection and categories)
- The deposit of the media (the full text documents)
- The submission of the metadata (corresponding to the deposited media)

The first step is the only one that needs user intervention. In fact, the selection of the collection and the categories in the server is not done automatically.

6.6.1 Select destination

The select destination workflow needs between three and four operations by the user [Figure 30 : Workflow : Select destination]:

1. Remote server selection
2. Server's collection selection
3. Collection's primary category selection
4. Secondary categories selection (optional)

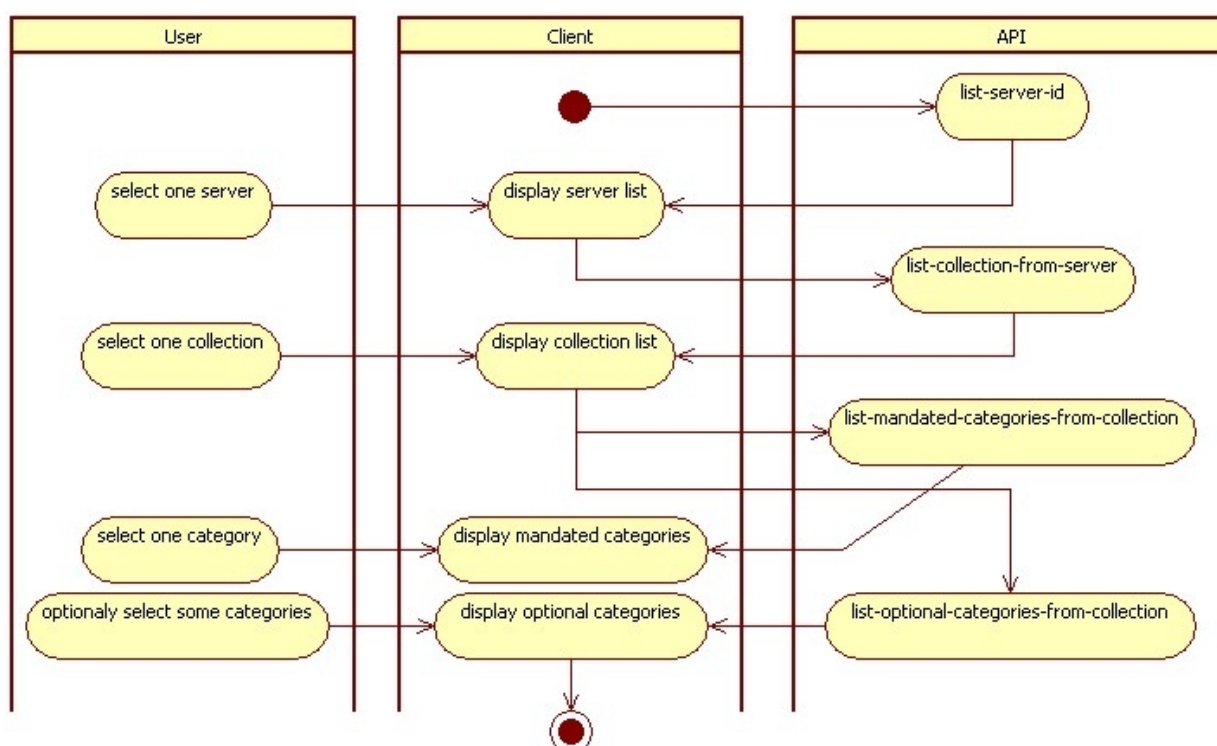


Figure 29: Workflow : Select destination

By getting the server and collections, the system can also provide server information.

6.6.2 Deposit media

The deposit media workflow [Figure 31 : Workflow : deposit media] used destination information to send given resource media to the remote server. The record id is known by BibSword according to the action the user was doing before choosing “Forward to SWORD” option. Therefore it is outside of the BibSword workflow.

For example, if the user was submitting a document to Invenio, the metadata and the media of this document are known by the system before the beginning of the BibSword workflow.

In a typical submission workflow on Invenio, the initial gathering of data and creation of the record is handled by the Invenio submission system. It is only after the creation of this record that BibSword controls the forwarding of the digital document to arXiv.

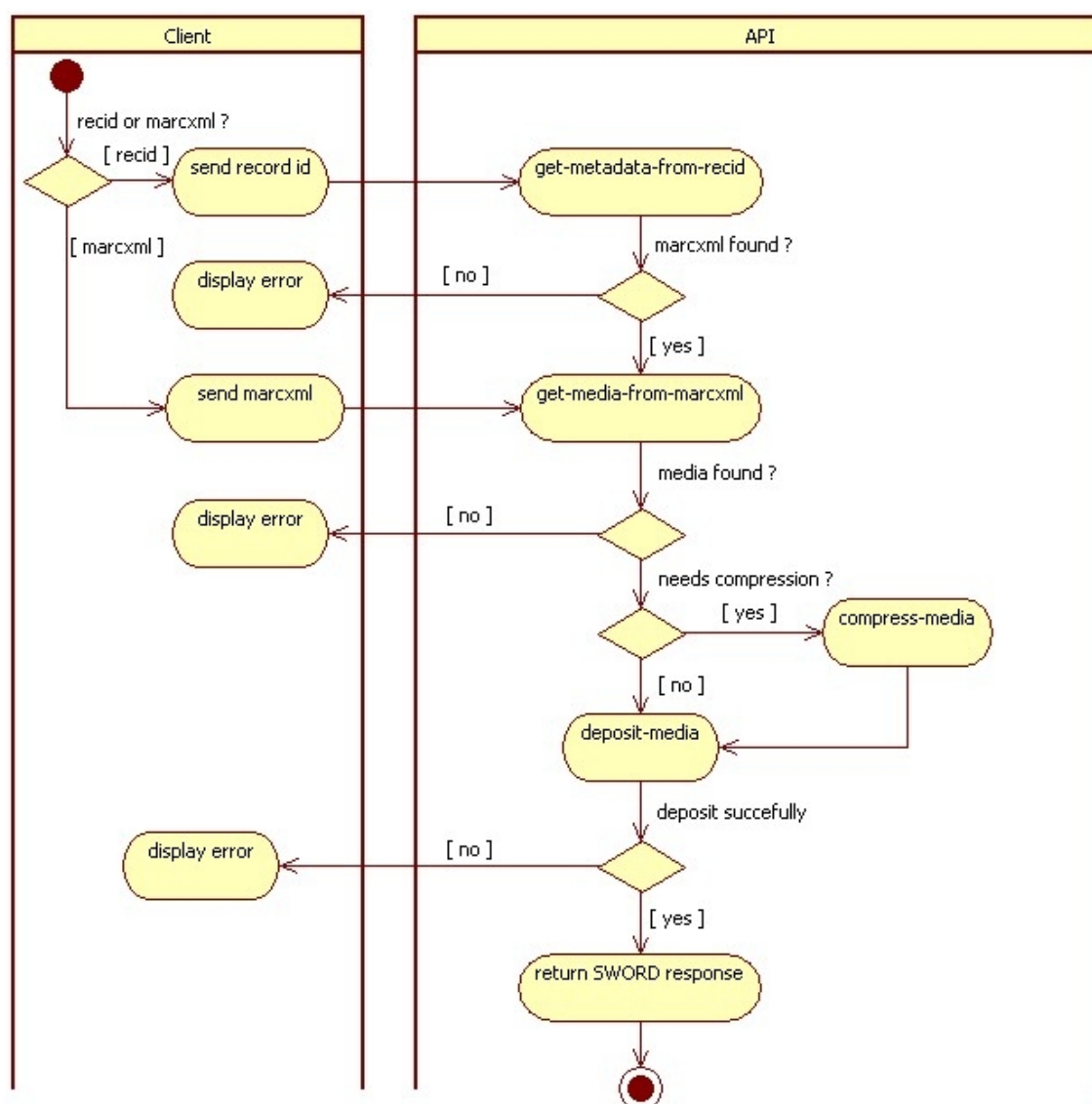


Figure 30: Workflow : deposit media

6.6.3 Submit metadata

Submission metadata [Figure 32 : Workflow : submit metadata] needs information from the media deposit to work properly. These information is the URL link(s) to the deposited media. The submission is then made to the same collection's url than the media.

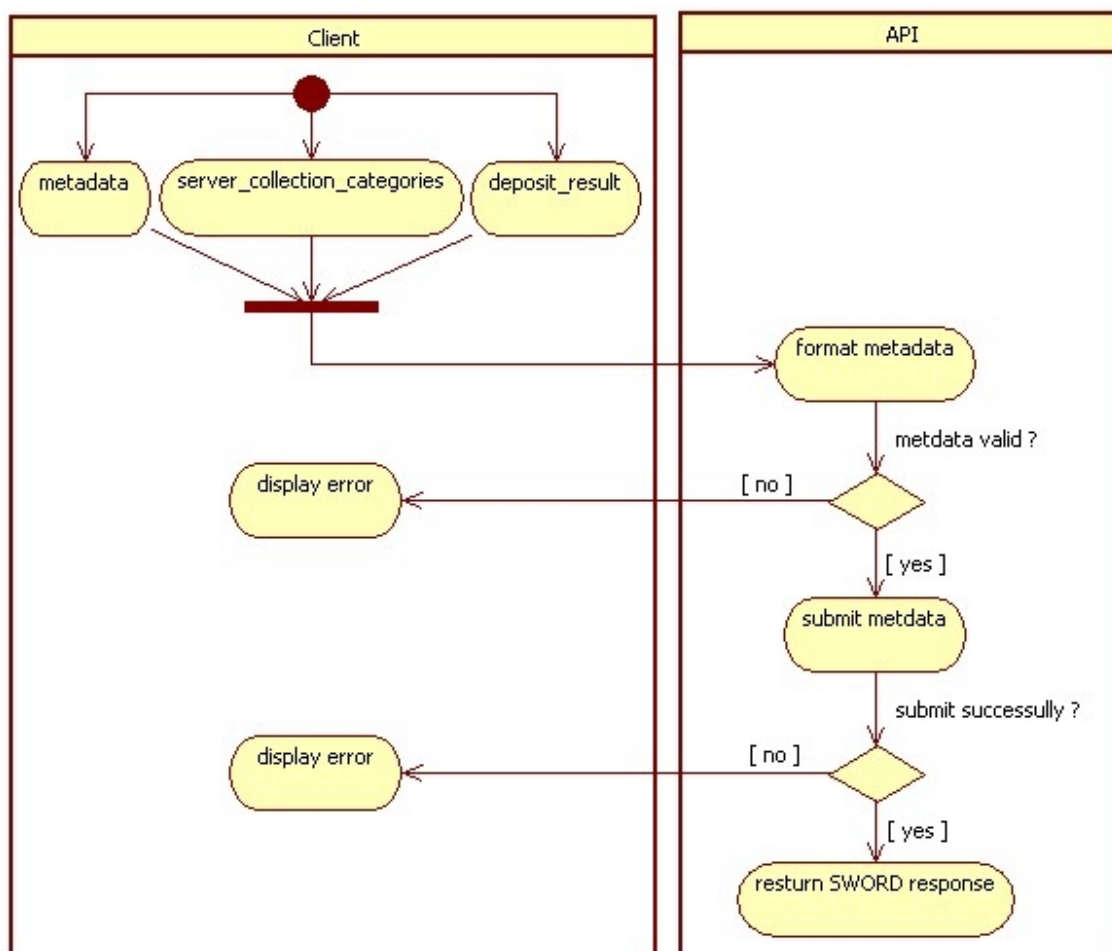


Figure 31: Workflow : submit metadata

6.7 Class diagram

The Figure 33 shows the class diagram of the BibSword module. Python is a functional and object oriented language. Invenio coding guidelines recommends to use object-oriented patterns only when necessary.

Conceptually, each element shows below is a class, event though the implementation uses more a “functional” approach to, du to the guidelines.

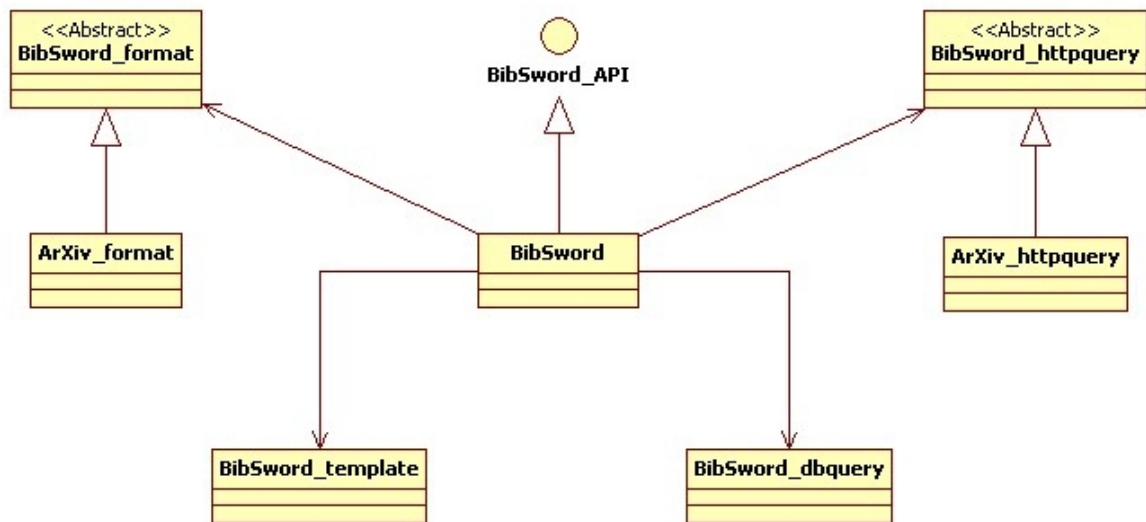


Figure 32: BibSword class diagram

6.7.1 Class creation

The action of creation and class calling is simple:

Depending on the selected remote server, a child class of **BibSword_httpquery** and **BibSword_fomrat** is created. For example, in this project the remote server is arXiv. When a submission is initialized, a function called `get_instance()` contains in the Abstract classes (**BibSword_format** and **BibSword_httpquery**) return the class dedicated to format and communicate with the remote server.

This technical is close to an Abstract factory pattern, but as Python allows to declare function outside of an object, no Class “Factory Method” are need for the object creation. They are replaced by the `get_instance()` functions.

7 Web client

The web client is using the BibSword API to connect to a SWORD server. As the goal of this project is to forward digital documents to arXiv, all examples of communication are to the arXiv SWORD server.

7.1 Use cases

The use case diagram [Figure 34] gives a global idea of the functionalities offered by the function “Push to arXiv”. It shows the elements (actors) that interact with the system, the use cases and there dependencies (include or extends) and the links between use cases and actors.

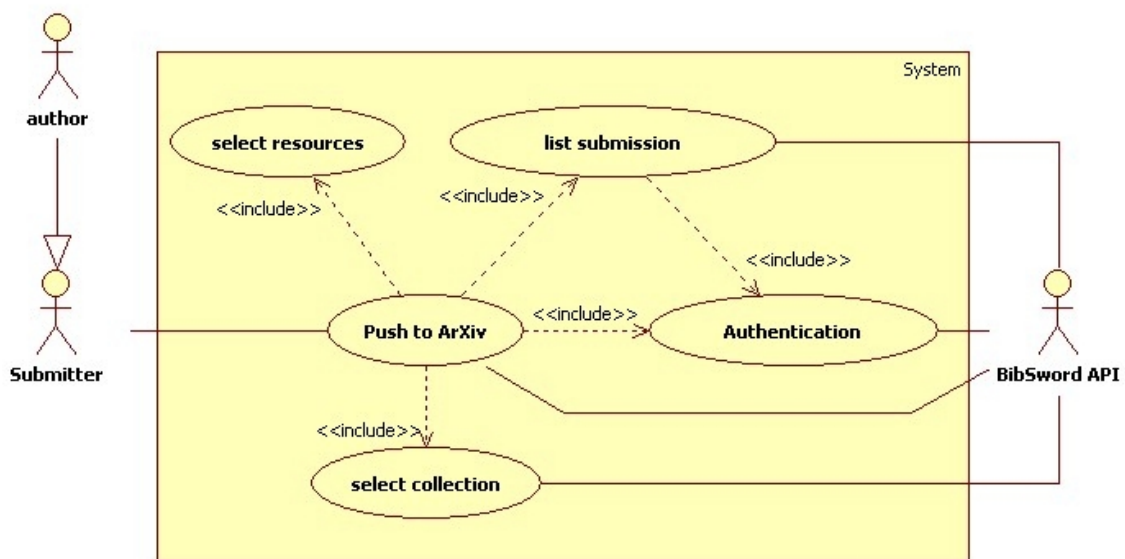


Figure 33: Use cases

As shown on this diagram, the submitter can do only one action which is to push a resource to arXiv. This function includes lots of operations and can be done in different ways depending on the context.

7.1.1 Actors description

The actors are all the elements that are outside the described system and interact with it.. For example, the BibSword API is an external element of the client.

Submitter : The submitter is the person that triggers the submission of resources to arXiv. A good example is the assistant of a physics laboratory that has to deposit some resources to Invenio and to forward them it to arXiv.

Author : The authors are the persons who write documents such as laboratory reports or technical notes. They can take the role of the submitter to submit their work to arXiv. A submission can be done without any interaction of the author by using the “on behalf of” option of arXiv

BibSword API : The BibSword API hides all the business logic part of a communication between the client and the final remote repository.

7.1.2 Cases description

The use cases basically are actions that users can execute on the system. They are also needed here to show actions that are included in the main function. The “Push to arXiv” function must be as much automatised as possible, so that lots of tasks avoid any interaction with the users.

Push to arXiv: The case “Push to arXiv” is the main part of the function. It allows user to deposit any resource from Invenio to arXiv. To achieve this task, the client must:

- Know one or many resources given by the submitter
- Ask for the collection where to put these resources
- Check if the resources has not already been submitted

The client could push resources to other remote server than arXiv using the BibSword API but arXiv is the only remote server that is currently implemented in BibSword.

Select resources: The resources (id or metadata file) to submit are given by the submitter to the web client. These resources are stored on the internal database of Invenio that contains the media and the metadata.

Select destination: The destination is the “place” where the submitter wants to deposit the resources. It is composed of the remote server, the collection URL, the primary and the optional categories. The BibSword API is able to provide the full list of information. Each collection also inform the user of the type of media it supports. For example, some collections do not accept PDF or ZIP files.

Authentication : The client is responsible to check that the user is allowed to submit the resources it gives to BibSword.

List submission : This use case is used by the system to know the state of a resource that might already been submitted. It is useful to know if it is:

- Submitted : waiting the approval of arXiv mandatory
- Published : accepted by the arXiv mandatory
- Unreachable : suppressed by the arXiv mandatory

7.2 GUI model

The GUI (Graphical User Interface) offers to the submitter a convenient way to go through each steps of a SWORD document submission. It check if all required data have been input by the users and if they are well-formed. In addition, lists, check boxes and radio button are used as offend as possible to minimize user errors.

7.2.1 Select remote server

When the submitter reaches the BibSword interface, he has to select where he wants to push its document.

The user must select it in a drop down list containing each remote servers [Figure 35 : GUI model : remote server selection].

Because this project is the first implementation of a SWORD client, the only choice is arXiv.

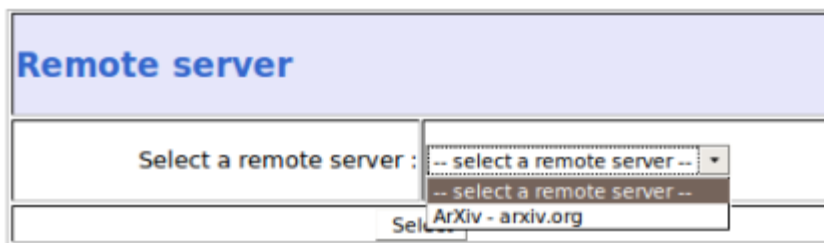


Figure 34: GUI model : remote server selection

If the user click the select button without having chosen a remote server, an error message will be printed [Figure 36 : GUI Model : remote server selected error]

The following error was found in the formular :

- No remote server was selected

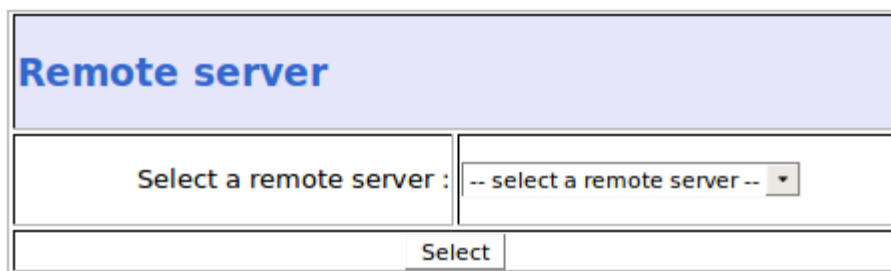


Figure 35: GUI Model : remote server selected error

The informations concerning the remote server are directly found in the swrREMPTESESERVER data table.

7.2.2 Select server's collection

Once the server arXiv has been selected, the submitter accede to the "Push to arXiv" web form [Figure 37].

Informations concerning the arXiv remote servers are displayed. The button "Modify server" allows the user to select an other one if he has been mistaken.

The collections are display in a drop down list that contains the label of each arXiv's collections.

Once selected, the user can go further by pushing the "Select" button.

Push to ArXiv

Figure 36: GUI Model : Push to arXiv - select collection

If the submitter has not selected any collection, an error message appear [Figure 38] and the same page displays.

Push to ArXiv

The following error was found in the formular :

- No collection was selected

Figure 37: GUI Model : Push to arXiv - select collection (error)

7.2.3 Select categories

The last destination information that the user has to specify is the mandated category. The categories are display in a drop down list [Figure 39]. He also can choose many optional categories by doing Ctrl+click on the optional categories list.

Figure 38: GUI Model : Push to arXiv - select categories

The selection's category page also display informations about the remote server and the selected collection [Figure 40].

The buttons “Modify server” and “Modify collection” allow user to modify their precedent selections.

If the user submit this form without having chosen a primary category, the error display on the Figure 40 will be displayed and the same page reload.

Push to ArXiv

The following error was found in the formular :

- No primary category selected

Figure 39: GUI Model : Push to arXiv - select categories (infos & errors)

7.2.4 Check metadata

Once every destination's informations are known by the client, it displays several informations to inform the submitter about the submission he is doing.

First of all, the list of collections and destination (with URL's) is displayed [Figure 41]

| Destination | |
|---|---|
| ArXiv | Collection : The Physics archive (https://arxiv.org/sword-app/physics-collection) |
| | Primary category : Physics - Galaxy Astrophysics (http://arxiv.org/terms/arXiv/astro-ph.GA) |
| | Category : Physics - Galaxy Astrophysics (http://arxiv.org/terms/arXiv/astro-ph.GA) |
| | Category : Physics - Solar and Stellar Astrophysics (http://arxiv.org/terms/arXiv/astro-ph.SR) |
| <input type="button" value="Modify destination"/> | |

Figure 40: GUI Model : Push to arXiv - List of selected destination

Then, information about the person doing the submission are display in input text field. That allows the user to change the email destination for the arXiv deposit confirmation [Figure 42].

| Submitter | |
|-----------|--|
| Name : | <input type="text" value="mbarras"/> |
| Email : | <input type="text" value="mathieu.barras@cem.ch"/> |

Figure 41: GUI Model : Push to arXiv - Submitter informations

The media box contains the list of full text document composing the resource to submit, their size and their type. The submitter has the possibility to compress it in a zip archive [Figure 43].

| Media | | |
|---|------------------------|---------------|
| URL : http://localhost/record/97/files/convert_SCAN-0005061.pdf | Type : application/pdf | Size : 379492 |
| Compress media in a zip archiv ? : <input type="radio"/> Yes <input checked="" type="radio"/> No | | |

Figure 42: GUI Model : Push to arXiv - Media informations

The metadata table contains the list of metadata related to the resource to submit [Figure 44]. The user can modify them but some controls are checked to ensure the content is correct.

| Metadata | |
|------------|--|
| Id* : | 002471378CER |
| Title* : | Development of photon beam diagnostics for VUV radiation from a SASE FEL |
| Summary* : | <p>For the proof-of-principle experiment of self-amplified spontaneous emission (SASE) at short wavelengths on the VUV FEL at DESY a multi-faceted photon beam diagnostics experiment has been developed employing new detection concepts to measure all SASE specific properties on a single pulse basis. The present setup includes instrumentation for the measurement of the energy and the angular and spectral distribution of individual photon pulses. Different types of photon detectors such as <u>PSi-photodiodes</u> and fast thermoelectric detectors based on <u>YBaCuO-films</u> are used to cover</p> |
| Author* : | Treusch, R |

The fields having a * are mandatory

Figure 43: GUI Model : Push to arXiv - Metadata informations

To submit the resource, the user has to submit the form. Every state are then automatically proceeds. If an error appears during the submission, the corresponding error message will be displayed on screen.

If everything goes fine, the new entry of the swrDATA is displayed as shown in the Figure ...

If any required field is not entered or not well-formed, a list of error is displayed at the beginning of the web page and the resource is not submitted [Figure 45].

| Push to ArXiv |
|---|
| <p>Following errors were found in the formular :</p> <ul style="list-style-type: none"> • Id is missing • summary must have at least 25 characters • No author specified |

Figure

44: GUI Model : Push to arXiv - Metadata informations (error)

The Figure 44 and 45 do not contains every fields. There are just the most important.

7.2.5 List submission

When an administrator wants to check the status of the submissions (as well as directly after a submission), the list of all submitted resources with useful information is displayed [Figure 46]

| ArXiv submission list | |
|-----------------------|---|
| 34038281 | Remote id : info:arxiv/app/10070221 |
| Status : submitted | Submission date : 2010-07-06T13:20:24Z |
| | Submitter : M. Barras |
| | Media link : https://arxiv.org/sword-app/edit/10070221 |
| | Metadata link : https://arxiv.org/sword-app/edit/10070221.atom |
| | Status link : http://arxiv.org/resolve/app/10070221 |

Figure 45: GUI Model : List submission (simple element)

These information indicate that the submission has been correctly done because it contains the link to the media, the metadata and the status of the submission.

Of course, on the example given in figure 46, there is only one resource but in case of many resources, the table would contains one of these boxes for each submitter resources.

7.3 Synthesis

This GUI has several purposes:

- To offer a simple way to submit documents
- To show SWORD interface possibility to the final user
- To proceed with fully integration of the “Push to arXiv” function in Invenio

Even if this interface seems to be quite basic, it could become even more easier and faster to use by:

- Preselecting remote server. For example by putting “Push to arXiv” (or any other remote server) button.
- Preselect the collection. In the case of the CERN instance of Invenio, 99.9% of the submissions are made to “Physics-collection”.
- Limit amount of categories
- Forbid user to modify any of the metadata or submission information fields

For these reasons it is likely that some fields and even some forms will completely seem different in the final integrated version of the “Push To arXiv” function.

8 Implementation & Test

This chapter of the documentation treats of the purely technical aspects of the project. After having studied what to use (Analyse) and what will be made (Conception), this part is about how to implement each part of the project :

- Data formatting : explain the technique used to generate XML atom entry files
- BibSword : shows how the persistence of data is implemented between HTML pages.
- Web client : explain how information are passing between page and how pages are generated
- Test : shows the test procedure that have been built in accordance with the Invenio standard
- Installation : expose the techniques used to include the installation of BibSword as part of the Invenio distribution

All these sections are just showing the techniques used with some example. Of course, some points are not treated because they are considered not very relevant for the understanding of the project implementation.

8.1 Data formatting

One of the biggest part of the project was to design a file format that was conforming to the XML Atom entry format used by SWORD for the metadata file. The generation of such a format must consider two aspects:

- The field format and occurrence (optionally, simple, multiple) has to be checked
- The implementation must be as flexible as possible, it means the insertion, modification or suppression of any field must be fast and easy.

After having analysed this format and the way Invenio was implementing similar tasks, I proposed two alternatives:

- To use a XSL template that would generate a XML code using a XSL processor
- To generate the XML file by programming in the Python code

The following table resume how the decision was made:

| Criterion | Weight | Results | |
|----------------------|--------|---------------|----------------|
| | | XSL processor | by programming |
| Speed of execution | 3 | 5 | 3 |
| Speed of development | 4 | 3 | 5 |
| Flexibility | 6 | 4 | 4 |
| Invenio standard | 5 | 2 | 5 |
| Total | | 63 | 78 |

The result of this short study shows that the programming solution is a bit preferred as the XSL one.

The main reason is actually that after some discussion with CDS collaborators, they told me that most of peoples working on Invenio was using the programming solution and therefore would prefer this solution.

8.1.1 BibSword Implementation

The BibSword_format class contains a function that allows to format a SWORD standard XML atom file. This main function is using lots of other functions that are each dedicated to one node of the final file.

The code of the main function look like shown in Figure

```
<entry xmlns="http://www.w3.org/2005/Atom"
xmlns:arxiv="http://arxiv.org/schemas/atom">
%(title)s
%(id)s
%(updated)s
%(author_name)s
%(author_email)s
%(contributors)
%(summary)s
%(categories)s
</entry>
```

Each line composed with a %(...)s is actually a call to a function and is replaced with the return of the call to it. This function check the content of the argument given and display one or more node. In case of error, a special tag is return and the generation process is stopped.

This way offers a big flexibility because when a node do not receive values, the function that is called only return an empty string and nothing is displayed.

8.1.2 arXiv implementation

The arXiv_format class inherit from BibSword_format. It means that it used exactly the same method to create the XML atom entry nodes. In addition, this class contains every method that allows to generate arXiv specific nodes (those having the <arxiv:...> extension.

8.2 BibSword

The implementation of the BibSword module is the biggest implementation part. Indeed, it contains every business logic but also some important point such as:

- Database table implementation
- MARC file field insertion
- HTTP authentication
- Generation of the HTTP Request

The implementation technique of those points is documented in the next sub chapter.

8.2.1 Database table implementation

As shown in the BibSword conception, two table were necessary to serialized relevant data about the remote servers and the resources submission.

Invenio used MYSQL to stores this kind of informations. As my tables are just used for the BibSword module, there where no need to study the rest of the database model.

The implementation of the two tables used in BibSword are the following:

- **swrREMOTESERVER**

```
CREATE TABLE IF NOT EXISTS swrREMOTESERVER (  
  id int(15) unsigned NOT NULL auto_increment,  
  name varchar(50) unique NOT NULL,  
  host varchar(50) NOT NULL,  
  username varchar(50) NOT NULL,  
  password varchar(50) NOT NULL,  
  email varchar(50) NOT NULL,  
  realme varchar(50) NOT NULL,  
  urlServicedocument varchar(80) NOT NULL,  
  xmlServicedocument longblob,  
  lastUpdate int(15) unsigned NOT NULL,  
  PRIMARY KEY (id)  
) TYPE=MyISAM;
```

- **swrDATA**

```
CREATE TABLE IF NOT EXISTS swrDATA (  
  id int(15) unsigned NOT NULL auto_increment,  
  idSwrREMOTESERVER int(15) NOT NULL,  
  idBibRec varchar(50) NOT NULL,  
  idUser int(15),  
  xmlMediaDeposit longblob,  
  xmlMetadataSubmit longblob,  
  idRemote varchar(50),  
  submitDate varchar(100),  
  contentTypes varchar(150),  
  linkMedias varchar(250),  
  linkMetadata varchar(150),  
  linkStatus varchar(150),  
  published boolean default 0,  
  PRIMARY KEY (id)  
) TYPE=MyISAM;
```

This code are set in the tabcreate.sql file situated in the installation repository. This way, when Invenio is installed, the tables are created as any other one.

8.2.2 MARC file insertion

Once a document has been submitted, a field is inserted into the MARC file. To be able to write a tag in a MARC file, a tool named BibUpload is provided in the standard Invenio distribution.

This task involves to format a XML node conforms to the standard of MARC and to add it using BibUpload. The following code implements these actions:

```
def update_marxml_with_remote_id(remote_id):
    """
    Write a new entry in the given marc file. This entry is the remote record
    id given by the server where the submission has been done
    @param (marxml) : the xml file corresponding to the marc file
    @param (remote_id) : the string containing the id to add to the marc file
    return : boolean true if update done, false if problems
    """

    # concatenation of the string to append to the marc file
    node = '<datafield tag="%s" ind1=" " ind2=" ">' % (tag)
    node += '<subfield code="a">%s</subfield>' % (remote_id)
    node += '</datafield>'

    # creation of the tmp file containing the xml node to append
    filename = CFG_TMPDIR + '/sword_append_remote_id.xml'
    f = open(filename, 'w')
    f.write(node)
    f.close()

    # insert a task in bibschedul to add the node in the marc file
    return task_low_level_submission(bibupload, 'BibSword', '-a', filename)
```

The method `task_low_level_submission()` is part of the BibUpload module that is part of the standard Invenio distribution.

8.2.3 HTTP Authentication

Each HTTP request must be authenticated with adequate credentials. To avoid to always send these credentials over the network and then to improve efficiency and security, Python offers tool to initialize a connection and keep always the same user password that are encrypted before being sent over the network.

To improve the flexibility and the simplicity of the code, the class `BibSword_httpquery` initialized the connexion in it constructor, as shown in the code below:

```
def __init__(self, authentication_infos):  
    '''This method the constructor of the class, it initialise the  
    connection using a passord. That allows users to connect with  
    auto-authentication.  
    @param (self)      : reference to the current instance of the class  
    @param (realme)    : name of the realme provide by this object  
    @param (username)  : name of an arxiv known user  
    @param (password)  : password of the known user  
    ...  
  
    #password manager with default realm to avoid looking for it  
    passman = urllib2.HTTPPasswordMgrWithDefaultRealm()  
  
    passman.add_password(authentication_infos['realme'],  
                        authentication_infos['hostname'],  
                        authentication_infos['username'],  
                        authentication_infos['password'])  
    #create an authentificaiton handler  
    authhandler = urllib2.HTTPBasicAuthHandler(passman)  
  
    h=urllib2.HTTPHandler(debuglevel=1)  
  
    opener = urllib2.build_opener(authhandler, h)  
    # insalling : every call to opener will use the same user/pass  
    urllib2.install_opener(opener)
```

The steps of this method are:

1. creation of a password manager object that takes every required authentication information (kept in the `swrREMOTESERVER` table) given in parameters.
2. creation of an authentication handler that will be used by every HTTP request. Thanks to that handler, HTTP requests do not need to specify authentication any more
3. creation and installation of the opener object. This object specifies that every HTTP request has to use the handler. The installation of the handler makes it able to live during the entire life of the object.

8.2.4 Generation of HTTP request

To understand how HTTP requests work in Python, the best way is to analyse a complete example:

```
def metadata_submission(self, deposit_url, metadata, onbehalf=''):
    """
    This method send the metadata to ArXiv, then return the answere
    @param (metadata) : xml file to submit to ArXiv
    @param (onbehalf) : specify the persone (and email) to informe of the
                        publication
    """

    #prepare the header of the request
    headers = {}
    headers['Host'] = 'arxiv.org'
    headers['User-Agent'] = CFG_DEFAULT_USER_AGENT
    headers['Content-Type'] = 'application/atom+xml;type=entry'
    #if on behalf, add to the header
    if onbehalf != '':
        headers['X-On-Behalf-Of'] = onbehalf

    headers['X-No-Op'] = 'True'
    headers['X-verbose'] = 'True'

    #format the request
    r = urllib2.Request(deposit_url, metadata, headers)

    #launch request
    try:
        response = urllib2.urlopen(r).read()
    except :
        return ''

    return response
```

This method does four different things:

1. Creation of a HTTP header that contains:
 - The host : only used for information
 - The User-Agent : used to specify the software doing the request (Invenio SWORD-client)
 - The content-type : required to indicate the MIME-Type of the data sent
 - The X-On-Behalf-Of: if inserted, indicate the SWORD-server that it is not the author that is doing the submission
 - X-No-Op and X-verbose : indicate that the action is sent in simulation mode.
2. Creation of the request with the destination URL, the DATA and the Header
3. Launching of the request
4. Fetching and reading of the response

8.3 Web client

There are two distinct challenges in the implementation of the Web-client:

- Generating the body of the pages
- Keeping state between pages

8.3.1 Page generation

The generation of HTML code is done in `BibSword_template`. Every page is implemented in the same way that for XML atom file generation (see chapter 8.1).

All dynamic elements of the page are replaced with keys. When the Python code is generating the html page, it parses the code and replaces the key with given values. For some of them, typically for values that are not always displayed such as errors, it calls specific methods. These methods format and return the part of dynamic code that must be displayed.

For example, to display the first page of the client (Select remote server) the code is the following:

```
body = '''
<form method="post" enctype="multipart/form-data" onsubmit="return
tester();" accept-charset="UTF-8">
  <input type="hidden" name="status" value="select_server"/>
  %(error_message)s
  <table border="1" valign="top" width="%(table_width)s">
    <tr> <td align="left" colspan="2" bgcolor="#e6e6fa">
      <h2>Remote server</h2></td>
    </tr>
    <tr> <td align="right" width="%(row_width)s">
      <p>Select a remote server : </p>
    </td>
    <td align="left" width="%(row_width)s">
      <select name="id_remote_server" size="1">
        <option value="0">-- select a remote server --</option>
        %(remote_server)s
      </select>
    </td>
    </tr>
    <tr>
      <td colspan="2" align="center">
        <input type="submit" value="Select" name="submit"/>
      </td>
    </tr>
  </table>
</form>''' % {
  'error_message' : self.display_error_message_row(error_messages),
  'table_width'   : '100%',
  'row_width'     : '50%',
  'remote_server' : self.fill_dropdown_remote_servers(remote_servers)
}
return body
```

In the example above, we clearly see that the tags “error_messages” and “remote_server” are replaced by the return value of the called method.

For example if there is no error messages, the method will just return an empty string.

Dynamic elements are always generated server-side. I decided not to use it in this project for two reasons:

1. JavaScript can be disabled on client Browser.
2. The server side generation must be implemented in any cases according to Invenio standards. Implementing JavaScript would takes twice the time.

8.3.2 Keeping state

To keep the state of a page, that is to keep information selected by user between two web pages, Three techniques exist:

1. Using the session
2. Using cookies
3. Using form

In Invenio, the use of session to store informations is very restricted. The only things that are able to access the session are for example: user login .

Cookies are not really used to keep informations between two web pages but more between session, that is to remember a user between each visit of the website. In addition this techniques imply that the user has the cookies activated in its browser, which is not always the case.

The only solution to keep the state of the pages is then to use hidden input. This techniques consist in putting additional informations in forms that are transmitted with the page and then can be reused in the next page.

The following example shows how the selected remote server name is sent from one page to another.

```
<form method="post" enctype="multipart/form-data" onsubmit="return
tester();"
  accept-charset="UTF-8">
  <input type="hidden" name="status" value="select_collection"/>
  <input type="hidden" name="id_remote_server" value="%(id_server)s"/>
```

The value of the selected remote server is then sent with the form like any other input values, but it is not displayed.

8.4 Tests

According to the Invenio standard development process, the test suite strategy is divided in three components:

1. unit tests
2. regression tests
3. web tests

This chapter offers only a short insight of the test implementation followed by some examples made in this project. For more information about the Invenio testing philosophy, refer to the CDS Invenio test documentation [11].

8.4.1 Unit tests

The aim of unit tests is to check that returned value from sets of functions are conform with what is expected according the given parameters.

It is not necessary to test every single function but the relevant one must ensure to be correct even for unusual request.

The Python library *unittest* is used in Invenio for writing test cases. The following example shows the syntax and the use of unit test in Python:

```
import unittest

class TestBibSwordFormatXmlAtom(unittest.TestCase):
    """Test for washing of search query parameters."""

    def test_format_id_node(self):
        """BibSword format id node test"""

        id = '<id>1</id>'
        self.assertEqual('', bibSword_format.frmt_optional_resource_id(''))
        self.assertEqual(id, bibSword_format.frmt_optional_resource_id(1))
        self.assertEqual(id, bibSword_format.frmt_optional_resource_id(2))
```

The result of this test is the following:

```
bibSword - test format resource id '' ... ok
bibSword - test format resource id 1 ... ok
bibSword - test format resource id 2 ... fail
```

To improve the use of test cases, Python allows to implements test suite that are function that launch every test in one time. It works as follow:

```
TEST_SUITE = make_test_suite(TestBibSwordXmlAtom,
                              TestBibSwordHTTPRequest,
                              ...)
```

This case, when a user call the global testcase, every test are done.

8.4.2 Regression tests

Regression tests are used to check the proper functioning functionalities of the application. They are typically used to prevent the repartition of bug when a new functionality is added.

Regression test try to catch issues due to the context in which the application is running. For example :

- Bugs coming from wrong data from the database
- Wrong parameters given by user
- Lost of connection (may typically happened in the “Push to arXiv” function if the remote server goes down)

The regression test must be written with consideration of the data that they need. For example, regression tests are always run on the fresh Demo database containing sample data that is installed during Invenio installation.

In BibSword, the regression test are written for every actions that involves user interactions such as:

- Check the values entered in a form
- Check the validity of the given record

But also SWORD remote server interaction such as:

- Correct format response from a correct deposit
- Conformity of returned errors of wrong data submission
- Correct error message after a wrong authentication

Finally, database interaction are tested to know if tables fields have been changed by some other developers or applications.

The gain of implementing regression test is huge because in case of errors, they would find exactly what's wrong and which part of the application is affected even if it is in a module that seems to have nothing to do with the currently developed.

8.4.3 Web testing

Web testing is the last required test that Invenio requires to validate a module. The aim of such tests is to test web application as a black box testing.

Even if some the regression test already make those kind of controls, they are useful to check the application using real Browser.

It is also the only way to test JavaScript in real client side.

The way proposed by Invenio to write web test cases might be to use Selenium IDE test recording capabilities: you browse in Firefox as a normal user would, and you add simple tests for the presence of expected text on the pages as you go. Selenium IDE can record and store tests for later replaying.

8.5 Installation

BibSword is fully integrated in Invenio. It means that there is no need to install it. It automatically installed during the Invenio standard installation.

The installation documentation of Invenio is available on the Invenio developer documentation [1]

8.5.1 Ingegration

The integration of BibSword in Invenio has been made like any other standard module: using Makefile.

The integration of BibSword was made in three steps:

1. The subdirectories have to be declared in a makefile called "Makefile.am" located at the root of the module:

```
mathieu@mathieu-laptop:/$ cat Invenio/modules/bibsword/Makefile.am
SUBDIRS = bin doc lib
CLEANFILES = *~
mathieu@mathieu-laptop:/$
```

2. Each subdirectories have to contains a makefile called "Makefile.am" that declare the source code files:

```
mathieu@mathieu-laptop:/$ cat /modules/bibsword/lib/Makefile.am
pylibdir = $(libdir)/python/invenio
pylib_DATA = bibSword_config.py \
              bibSword.py \
              bibSword_tests.py \
              bibSword_dblayer.py \
              bibSword_regression_tests.py \
              bibSword_templates.py \
              bibSword_webinterface.py \
EXTRA_DIST = $(pylib_DATA)
CLEANFILES = *~ *.tmp *.pyc
```

3. The directory of the module has to be listed in the main makefile called "Makefile.am" located at the root of the module:

```
mathieu@mathieu-laptop:/$ cat /modules/Makefile.am
SUBDIRS = bibcatalog bibcheck bibcirculation bibclassify bibconvert
bibedit bibsword ...
CLEANFILES = *~
```

Once every makefiles has correctly configured, a program called autotools will generate all the Makefiles. Those makefiles will be called during Invenio installation and the BibSword module, including the Web-client will be properly installed.

9 Sustainable development

This section covers the analysis of my project and its perspective in terms of sustainable development.

First of all, let's see the definition of « Sustainable development » ?

sustainable development is to design systems and processes that meet the needs of present without compromising future needs

Even if IT is a small contributor to the global warming, with only 2% of the global emission of greenhouse gases [12]. It can reduce its consumption of electricity of 1 to 4 times by doing simple things such as putting the computer down at night or just enabling standby mode.

The interest of improving IT environmental impact is that it is getting more and more ubiquitous in industrial, administrative and private domains. Knowing this, it is easy to realize that even a small reduction of energy consumption of each computer system can have a really significant impact on global CO₂ emissions in the future.

9.1 IT as sustainable development tool

Beside being an energy consumer, IT is above all a tool that can help other domains to reduce their need of energy by improving the efficiency of their system. For example by :

- Dematerializing the informations
- Increasing efficiency of motors or electricity networks
- Reduce the needs of people to travel, e.g. by offering video-conference tool

In addition, the use of internet and email allow people to share documents more efficiently without the need of print them and neither of using conveyance to transmit them.

9.2 Invenio perspective

From the perspective of sustainable development, Invenio, as a digital library, is a typical example of a tool reducing considerably the amount of paper used for the publication of reports, theses and articles.

Before the implementation of such a system at CERN, every document had to be printed and sent by post or other intermediate every time a scientist needed it for its studies and researches.

Knowing that there are more than a million of records in Invenio, the amount of paper might represent many tonnes of woods and probably the same weight of CO₂ emission for routing each of them to their reader.

Another aspect of sustainable development allowed by Invenio is, as this is a public library, that an instance of the system could be set up for the storage of reports, analysis and articles that treating of sustainable development. The tools offered by Invenio would undoubtedly be a great help to share informations of how to improve

the way peoples and companies can manage to reduce their impact on the environment. Currently though, there is no such instance of the system.

Finally, Invenio do not require lots of calculation power. It is optimized to be installed on a single server and still offering good performances. In compare with lots of other systems including some digital libraries, that needs to have a whole computer farm using lots of power for running machines and air-conditioning, Invenio is a proper system that respect the environment.

9.2.1 BibSword and the sustainable development

The points in this section is to analyse the impact of the BibSword module on the environment. There are two different type of impact:

1. The impact of the use of the BibSword module on the environment
2. The impact of the result of the use of BibSword on the environment

Impact of use:

BibSword is a module of Invenio. It add no resources requirement that would consume electricity or material. In that way, it has neither a good nor a bad impact on the environment.

Result of use:

BibSword only has indirect benefits to sustainable development:

- The SWORD protocol offers a standard and simple way to forward document to a remote place. The implementation of such a protocol in a world wide used software like Invenio is a good promotion and can encourage peoples to use it and so, stopping sending media by other more pollutant way.
- The BibSword API suits not only to arXiv. As it is designed to easily and rapidly implements client for other remote server supporting SWORD, it can help people to change their submission process.

As a conclusion, even though BibSword is not a project related to the sustainable development, there are still some points that are linked to it.

10 Conclusion

In this report, we have focused on the development of a new module, BibSword that allows submitters of document to forward their work to a remote repository implementing the SWORD protocol: arXiv.

The specification of the project was done in the chapter 2. It includes the detail of the objectives (section 2.1) and the estimated duration of each activities (section 2.2)

The chapter 3 was about the studies and the analysis made before the beginning of the conception. In fact, as BibSword uses the SWORD protocol that is completely new for me, but also a bend new protocol, some protocol specifications had to be explain. The section 3.1 was dedicated to that. The second part of the analyse is about the implementation of the SWORD server-side by arXiv. The section 3.2 offers a good idea of how arXiv furnishes functionalities to deposit document using SWORD.

The conception of the project is composed of different chapter. The chapter 4 offers a global idea of the element composing the BibSword module and how they are working together. The chapter 5 is about the metadata used between Invenio and arXiv and the way they are formatted. The chapter 6 is the main part of the conception because it documents the design of BibSword and the API offers by it to implements SWORD-client on Invenio. Finally, the chapter 7 is the model of conception for the SWORD-client implemented in this project.

The last part of the project development, the chapter 8, treat of the Implementation of the functionalities and the tests. They contents some examples of techniques but also choices that have been made and the reason of each implementation decisions taken. The last part of this chapter, section 8.5 shows how BibSword is integrated in Invenio and thus, do not require any specific installation in addition to the normal CDS Invenio one.

Following points can summarize the result of the project:

- The functionalities described in the project specification (chapter 2) are fully respected and works properly
- The functions implementation have been tested and validate. They fits the Invenio requirement.
- The Web-client is working and suits to the client need.

In my point of view, this project was very interesting in many points.

First of all, working at CERN is an unique experience that gave me lots of satisfactions. The CDS team was very nice and everybody was motivated by his work but also by giving something new to Invenio.

Concerning the project, the design of an API using HTTP communication over the Internet was technically a good challenge for me. It was not easy at all but the result satisfies my a lot.

Finally, the fact that my work will be included in one of the better digital library in the world is a great motivation for the following of my studies and future work.

A1 : Glossary

| | |
|--------|---|
| Atom: | standard XML document format used by every protocols in the project |
| arXiv: | repository of digital document for scientific report. It is hosted by the Cornell University of Ithaca, New York, USA |
| SWORD: | Protocol used in this project to deposit digital document on repository over the Internet |

A2 : Bibliography

[1] : CDS team, "Invenio developer documentation", 2005,
<http://cdsware.cern.ch/invenio/documentation.html>

[arXiv SWORD/APP API] L. Bettini, "arXiv.org SWORD/APP Deposit API User's Manual", http://arxiv.org/help/submit_sword

[Atom] M. Nottingham, and R. Sayre, "The Atom Syndication Format", [RFC 4287](http://www.ietf.org/rfc/rfc4287.txt), December 2005. <http://www.ietf.org/rfc/rfc4287.txt> (see also non-normative html version at <http://APP.org/rfc4287.html>)

[AtomPub] Gregario, J. and B. de hOra, "The Atom Publishing Protocol", [RFC 5023](http://www.ietf.org/rfc/rfc5023.txt), October 2007. <http://www.ietf.org/rfc/rfc5023.txt> (see also non-normative html version at <http://bitworking.org/projects/atom/rfc5023.html>)

[JISC] "JISC official web site", <http://www.jisc.ac.uk/>

[MARC] CDS team, "HOWTO MARC Your Bibliographic Data" <http://cdsweb.cern.ch/help/admin/howto-marc>, 03 April 2008

[MARCXML] ndmso@log.org, "MARC 21 XML Schema Official Website", <http://www.loc.gov/standards/marcxml/>, 13 March 2009

[Python]

[SWORD profile] J. Allinson, L. Carr, J. Downing, D. F. Flanders, S. Francois, R. Jones, S. Lewis, M. Morrey, G. Robson, N. Taylor, "SWORD AtomPub Profile version 1.3", <http://www.swordapp.org/docs/sword-profile-1.3.html>, 22 February 2008

[SWORD] "the SWORD web site", <http://www.swordapp.org/>

A3 : List of figures

| | |
|---|----|
| Figure 1: Schema of the project..... | 5 |
| Figure 2: Collection's structure..... | 11 |
| Figure 3: Service document structure..... | 12 |
| Figure 4: APP Interactions..... | 12 |
| Figure 5: arXiv SWORD interface..... | 16 |
| Figure 6: service document schema 1/2..... | 17 |
| Figure 7: service document schema 2/2..... | 18 |
| Figure 8: example of service document response..... | 19 |
| Figure 9: MARC example..... | 25 |
| Figure 10: MARC format | 25 |
| Figure 11: Main MARC fields..... | 26 |
| Figure 12: submission metadata file to arXiv..... | 27 |
| Figure 13: database model..... | 31 |
| Figure 14: Overview of the API..... | 34 |
| Figure 15: CLI user manual..... | 35 |
| Figure 16: diagram sequence: list_remote_servers..... | 36 |
| Figure 17: diagram sequence : list_remote_server_informations..... | 37 |
| Figure 18: sequence diagram : list-collections-from-server..... | 38 |
| Figure 19: sequence diagram : list_collection_informations..... | 39 |
| Figure 20: sequence diagram : list_mandated_categories_from_collection..... | 40 |
| Figure 21: sequence diagram : list_optional_categories_from_collection..... | 41 |
| Figure 22: sequence diagram : list_submitted_resources..... | 42 |
| Figure 23: sequence diagram : get_marcxml_from_record..... | 43 |
| Figure 24: sequence diagram : get_media_from_marcxml..... | 44 |
| Figure 25: sequence diagram : compress_media_file..... | 45 |
| Figure 26: sequence diagram : desposit_media..... | 46 |
| Figure 27: sequence diagram : submit_metadata..... | 48 |
| Figure 28: sequence diagram : perform_submission_process..... | 49 |
| Figure 29: Workflow : Select destination..... | 50 |
| Figure 30: Workflow : deposit media..... | 51 |
| Figure 31: Workflow : submit metadata..... | 52 |
| Figure 32: BibSword class diagram..... | 53 |

| | |
|--|----|
| Figure 33: Use cases..... | 54 |
| Figure 34: GUI model : remote server selection..... | 56 |
| Figure 35: GUI Model : remote server selected error..... | 56 |
| Figure 36: GUI Model : Push to arXiv - select collection..... | 57 |
| Figure 37: GUI Model : Push to arXiv - select collection (error)..... | 57 |
| Figure 38: GUI Model : Push to arXiv - select categories..... | 58 |
| Figure 39: GUI Model : Push to arXiv - select categories (infos & errors)..... | 58 |
| Figure 40: GUI Model : Push to arXiv - List of selected destination..... | 59 |
| Figure 41: GUI Model : Push to arXiv - Submitter informations..... | 59 |
| Figure 42: GUI Model : Push to arXiv - Media informations..... | 59 |
| Figure 43: GUI Model : Push to arXiv - Metadata informations..... | 60 |
| Figure 44: GUI Model : Push to arXiv - Metadata informations (error)..... | 60 |
| Figure 45: GUI Model : List submission (simple element)..... | 61 |