

UNIVERSITÀ DEGLI STUDI DI MILANO – BICOCCA  
Facoltà di Scienze Matematiche, Fisiche e Naturali  
Dipartimento di Informatica Sistemistica e Comunicazione  
Corso di Laurea Magistrale in Informatica



**Authentication/authorization issues and fulltext document  
migration for the CERN Document Server**

Relatore: Prof. Gianpiero CATTANEO  
Correlatore: Dr. Fabio FARINA  
Controrelatore: Dr. Tibor ŠIMKO

Tesi di Laurea di: *Samuele KAPLUN*  
Matricola: 047947

Anno Accademico 2006–2007



# Summary

This is a report on the work carried out at the Department of Information Technology of the European Organization for Nuclear Research (CERN) under the CERN Technical Studentship Programme.

The project started on 16<sup>th</sup> January 2007. During this period I worked in the CERN Document Server (CDS) team of the User and Document Services (UDS) group as a fulltime developer on the software *CDS Invenio*.

The present report describes the subject, designed ideas and solutions, developed code and principal results.

Chapter 1 is an introductory chapter describing the CDS Invenio software and its use at CERN.

Chapter 2 presents the state of the art of the authentication system of CDS Invenio. It was the first project of the present work to enhance it by developing a new plugin infrastructure to support multiple external login authentication systems, importing external user details and group membership information.

Chapter 3 describes the role-based access authorization system of CDS Invenio and the design and implementation of FireRole, a new language to exploit user details and group membership in order to enhance the access system allowing complex user- and group-based authorization configurations.

Chapter 4 presents the second project of the present work, where a new tool for the fulltext storage subsystem of CDS Invenio was developed. The tool is responsible for migration of about 500,000 fulltext scientific papers managed by the CERN Document Server from a legacy system to the new architecture, recreating necessary data structures, collecting missing information about obsolete revisions through heuristics and importing file download statistics.

Finally the conclusion briefly summarizes the main results of this work.



---

# Contents

Summary	iii
<b>I Introduction to CDS Invenio</b>	<b>1</b>
1 Introduction	3
1.1 CERN	3
1.2 CDS Invenio	4
1.2.1 Key Features	5
1.2.2 User Classes	9
1.2.3 Modules Overview	9
<b>II Authentication and Authorization issues</b>	<b>13</b>
2 Authentication and eGroups	15
2.1 Introduction	15
2.2 Initial situation	16
2.2.1 Identifying users	17
2.3 Authentication at CERN	18
2.4 The improvements	19
2.4.1 How to handle external user details	19
2.4.2 How to handle external user group memberships	20
2.4.3 The new authentication plugin infrastructure	21
2.4.4 Results	22

---

2.4.5	Importing external groups . . . . .	23
2.5	Testing . . . . .	24
2.5.1	Issues . . . . .	24
2.6	Improvements to local password management . . . . .	27
2.7	Local passwords encryption . . . . .	28
2.7.1	Password resetting . . . . .	29
2.7.2	Mail cookies . . . . .	29
2.7.3	Administrative tasks . . . . .	30
2.8	SSO . . . . .	30
2.8.1	A brief overview on how Shibboleth works . . . . .	31
3	Firewall-like role definition for the access control module . . . . .	35
3.1	Introduction . . . . .	35
3.2	The idea . . . . .	36
3.3	The access control RBAC core workflow . . . . .	36
3.4	Proposed extension . . . . .	37
3.5	Users' details . . . . .	38
3.6	The FireRole language . . . . .	39
3.6.1	Syntax . . . . .	40
3.6.2	Semantic . . . . .	40
3.7	Data flows . . . . .	41
3.8	Tricks to improve performances . . . . .	42
3.8.1	Caching tools . . . . .	43
3.9	Tests . . . . .	44
3.10	Code Testing . . . . .	45
3.11	Temporary Roles . . . . .	45
<b>III</b>	<b>Fulltext document migration</b> . . . . .	<b>47</b>
4	Migration of fulltext documents with s2d . . . . .	49
4.1	Introduction . . . . .	49

---

4.2	The legacy fulltext architecture at the CERN Document Server . . . . .	51
4.2.1	Metadata . . . . .	51
4.3	The current CDS Invenio fulltext data structure . . . . .	52
4.3.1	Current database table structure to store Fulltext Document Information	52
4.3.2	Current filesystem structure to store Fulltext Document Information .	53
4.4	The S2D tool . . . . .	54
4.5	Detailed preliminary analysis . . . . .	54
4.5.1	What should be migrated . . . . .	55
4.5.2	The SetLink URL . . . . .	55
4.5.3	Remote fulltext file revisions . . . . .	56
4.5.4	Database data to be recreated . . . . .	56
4.5.5	Metadata handling . . . . .	57
4.5.6	Importing statistics . . . . .	57
4.5.7	Guidelines followed in writing the tool . . . . .	57
4.6	The S2D algorithm . . . . .	57
4.6.1	The input . . . . .	58
4.6.2	The output . . . . .	58
4.6.3	Undo possibility . . . . .	59
4.6.4	Migrating statistics . . . . .	60
4.7	Test and results . . . . .	60
4.8	Indexing issues . . . . .	60
4.8.1	IntBitSet . . . . .	62
4.8.2	IntBitSet benchmarks . . . . .	63
4.8.3	IntBitSet testsuite . . . . .	63
A	Mail cookies . . . . .	67
A.1	The implementation . . . . .	67
A.2	Developer interface . . . . .	68
A.3	Cookie content issues . . . . .	68

B	Minor activities	69
B.1	BibSched log viewer . . . . .	69
B.2	BibSched job pruner . . . . .	69
B.3	Refactoring of BibTasks . . . . .	70
B.4	BibSched and Invenio Garbage Collector . . . . .	70
	Conclusions	73
	Acknowledgements	81



---

## List of Figures

1.1	The CERN Document Server homepage. . . . .	4
1.2	CDS Navigable document taxonomy. . . . .	5
1.3	CDS powerful search engine. . . . .	6
1.4	Excerpt from an example record's MARC21 representation . . . . .	7
1.5	Excerpt from an example record's MARCXML representation . . . . .	7
1.6	CDS Multiple output formats. . . . .	8
1.7	CDS Personalization and Collaborative Features. . . . .	8
1.8	CDS modules relationship and user interaction. . . . .	12
2.1	CDS Invenio authentication system prior of the project start. . . . .	17
2.2	Imported External Settings screenshot. . . . .	23
2.3	The author's current configure CDS Invenio settings. . . . .	23
2.4	The External Groups panel . . . . .	24
2.5	External groups to share baskets. . . . .	25
2.6	External groups as receivers of web messages. . . . .	26
2.7	A snapshot of the current dual login at CERN. . . . .	27
2.8	The new password reset panel. . . . .	28
2.9	The sequence diagram of resetting a lost local password . . . . .	30
2.10	The possible SSO data flow (first part). . . . .	32
2.11	The possible SSO data flow (second part). . . . .	33
3.1	Flow of data within <code>acc_authorize_action()</code> . . . . .	37
3.2	A FireRole definition example. . . . .	40
3.3	A draft for the FireRole grammar . . . . .	40

3.4	FireRole administrator interface. . . . .	42
3.5	<code>run_sql()</code> usage to query for the name of a user given his email. . . . .	44
3.6	<code>run_sql_cached()</code> usage to query for the name of a user given his email. . . . .	44
4.1	A legacy page for downloading the fulltext file of a document. . . . .	50
4.4	How locally-hosted fulltext document could be represented in MARC . . . . .	52
4.5	ER diagram of records – fulltext documents files. . . . .	53
4.2	The current Data Flow at CERN . . . . .	64
4.3	The Data Flow of a clean installation . . . . .	65
4.6	An example of running the IntBitSet testsuite. . . . .	66
B.1	The BibSched log viewer. . . . .	70

## Part I

# Introduction to CDS Invenio



# Chapter 1

## Introduction

### 1.1 CERN

*The European Organization for Nuclear Research (CERN)* is the world largest particle physics centre (see [CER04]). Physicists at CERN explore what matter is made of and what forces hold it together. This organization provides them the necessary tools for their research. These are mainly particle accelerators able to bring particles to almost the speed of light, and detectors to make the particles visible.

Founded in 1954 by 12 countries, CERN has grown to the present 20 member states. Some 6500 visiting scientists, half of the world particle physicists, come to CERN for their research. They represent 500 universities and over 80 nationalities ([Com07]).

Since its creation, CERN has made many important discoveries for which scientists have received prestigious awards, including Nobel prizes.

While a fluctuating, international labour force is may create some collaborative friction, it is also believed to stimulate thought exchange thanks to many different cultural backgrounds.

In March 1989, then CERN-employee Tim Berners-Lee, recognized in [BLCGP92] the organization's need for more efficient knowledge-transferring, mainly caused by the migrating and wide-spread users community, and proposed the World Wide Web (see [Ben95]) as a measure to prevent the problem of "losing information at CERN".

Today's surplus of information has put search engines, and search companies like WWW-specific Google, into premium business segments. Though providing relevant documents at impressive speeds is an interesting computer science feat and indeed an essential artifact of the modern world, institutions like CERN need more specific technologies when dealing with huge sets of scientific material. It is not enough to be able to locate information quickly. The need is to have the infrastructure to maintain and foster a community where information not only is readily available, but wherein knowledge is updated, scrutinized, shared, passed on, and always kept securely stored for future community generations.

Moreover, scientific material in the form of articles and papers can be managed in the form structured information, which allow to build a real digital library, where information can be retrieved in a deterministic way,

## 1.2 CDS Invenio



Figure 1.1: The CERN Document Server homepage as can be seen by visiting <http://cdsweb.cern.ch>.

CERN Document Server Invenio (CDS Invenio) is a complete solution for running and managing integrated digital libraries. It is made up by a suite of interconnected modules, providing the framework for running an electronic preprint server, an efficient search engine, and an OAI-compliant<sup>1</sup> document repository [PBG<sup>+</sup>05].

Being developed at CERN, the main incentives to produce and maintain CDS Invenio, as honored by the CDS Consortium<sup>2</sup>, can be described as a refinement of the very incentives that pursued Berners-Lee to create the embryo of the WWW in the first place, namely the open dissemination of CERN's need for producing, storing, and sharing scientific results.

CDS Invenio is freely available, and licensed under the GNU General Public License. It is, as of September 2007, currently deployed at 20 major institutions around the

<sup>1</sup> *Open Access Information* (see [VBLMS02]), an open standard for harvesting and exchanging data between storage systems, <http://www.openarchives.org>

<sup>2</sup> <http://cdsware.cern.ch>

world [cds07a]. Technology-wise, CDS Invenio runs on GNU/Unix platforms, backed by a MySQL database server and an Apache/Python web application server. Python is the main programming language, though some *ad hoc* modules are written in Common Lisp [PBG<sup>+</sup>05] and as we will explain later in C.

CERN’s live installation, called “CERN Document Server”<sup>3</sup> (whose homepage can be seen in Figure 1.1 on the facing page), currently hosts around 1 million records and 500,000 fulltext documents, serving 20,000 monthly users issuing around 8,000 queries per day.

## 1.2.1 Key Features

**Navigable Document Taxonomy** Indexed documents are organized into a basic collection tree (see Figure 1.2), in which each node can implement different user privileges, be managed by different curators (see section 1.2.2 on page 9), and display custom informative presentations.



Figure 1.2: Navigable document taxonomy provided by the CERN Document Server.

**Powerful Search Engine** Provides “Google-like” speed for searches in document repositories of up to 1.5 million records [cds07b], with results optionally clustered according to the taxonomy (see Figure 1.3 on the next page).

CDS Invenio implements the Standard Boolean Model of Information Retrieval (see [BYRN99]). A suite of ranking algorithms exists that can be applied on top the retrieved results thus allowing for improvement in relevance. One of the plugin implements

<sup>3</sup>Publicly available at <http://cdsweb.cern.ch>

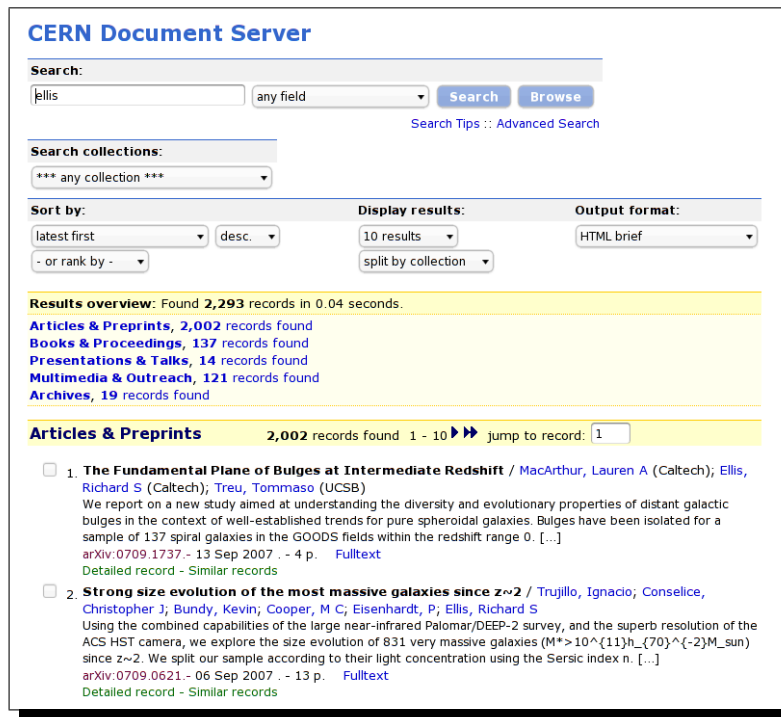


Figure 1.3: The powerful search engine provided by the CERN Document Server. Results are clustered according to the taxonomy.

the Standard Vector Model of Information Retrieval (see [SWY75])<sup>4</sup>. Stemming (see [Fra92]) is optionally applied, by means of the Porter’s suite of algorithms (see [Por])<sup>5</sup>.

**Flexible Data Acquisition and Output Formats** During the process of inserting information about a document into the repository, all its structured information, which include the *title*, *author*, *abstract*, *year* and so on (in what follows we will call this set “the document metadata”) is stored in the format MARC21, a standard for bibliographic data [MAR]. A MARC-flavor was chosen because it constitutes a mature *de facto* standard in the library world, and is considered flexible enough to cope with all kinds of bibliographic information [CDS07c]. Thus, CDS Invenio aligns well with traditional librarian tasks, while securing long-term usefulness without relying on future changes of internal structures.

As obvious from Figure 1.4 on the facing page the markup is neither straight-forward to parse, nor very readable to humans. Hence, whenever bibliographic metadata is to be worked with externally, CDS Invenio uses MARCXML<sup>6</sup>, exemplified in Figure 1.5,

<sup>4</sup>At the CERN installation of CDS Invenio, this kind of ranking is not enabled by default since physicist prepare to know what are the latest scientific paper.

<sup>5</sup>As a side project the author have added support for the latest Porter algorithms in the form of the Snowball project (see [Por06])

<sup>6</sup>MARCXML is an XML-schema based on the MARC21 bibliographic standard [MAR]. It was proposed and developed by the US Library of Congress [?] in order to ease interaction with MARC data,



```

000999421 088__ $$aCERN-BROCHURE-2006-003-ENG
000999421 246__ $$afaq$$bfrequently asked questions
000999421 260__ $$aGeneva$$bCERN$$c2006
000999421 300__ $$a36 p
000999421 65017 $$2SzGeCERN$$aInformation Transfer
000999421 690C_ $$aREPORT
000999421 710__ $$gCERN Communication Group

```

Figure 1.4: Excerpt from an example record’s MARC21 representation

```

<datafield tag="100" ind1=" " ind2=" ">
  <subfield code="a">Ellis, Nicolas</subfield>
  <subfield code="u">CERN</subfield>
</datafield>
<datafield tag="260" ind1=" " ind2=" ">
  <subfield code="c">2006</subfield>
</datafield>
<datafield tag="690" ind1="C" ind2=" ">
  <subfield code="a">ARTICLE</subfield>
</datafield>

```

Figure 1.5: Excerpt from an example record’s MARCXML representation

instead.

As visible in these examples, MARC standards make use of an extensive number of *datafields*, identified by a number ranging from 010 to 999 (number ranging from 000 to 009 are reserved as control field), accompanied by any number of *subfields* indicating a bibliographic value of predetermined type. An exhaustive listing of MARC keys, and their types of values, is available in [Bet00].

If metadata about a document being inserted into the repository happens to be accompanied by the fulltext file representing the document, this additional file is run through an automatic citations extractor. The results are then added to the metadata and made searchable.

On request mode, bibliographic records can then flexibly formatted in a wide range of ways (see Figure 1.6 on the next page), including HTML, Bib<sub>TEX</sub><sup>7</sup>, MARCXML, and Dublin Core<sup>8</sup>, the format used by OAI-PMH (see below).

**Personalization and Collaborative Features** Users can manage personal collections, referred to as “baskets”, of interesting documents, marking them as either private, public

a format that was essentially developed during 1960s to work with punch-cards, in order to cope with modern computing technologies such as XML.

<sup>7</sup>Manages citations in L<sup>A</sup>T<sub>E</sub>X typesetting.

<sup>8</sup>Standard for on-line resource descriptions [Dia05].

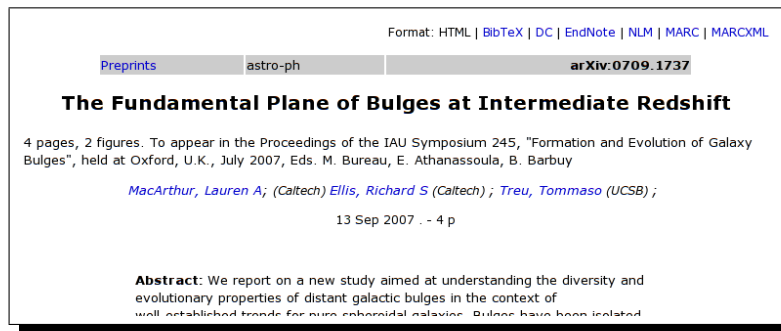


Figure 1.6: Multiple output formats link to export a record are displayed in the upper-left corner.

or shared with a user's group. Users can also set automated email notification called alerts about new results found by configurable queries, using the full search capabilities as described earlier in this section. Finally, users can review and rate documents as well as discuss them with other users, through a commenting system. (see Figure 1.7)

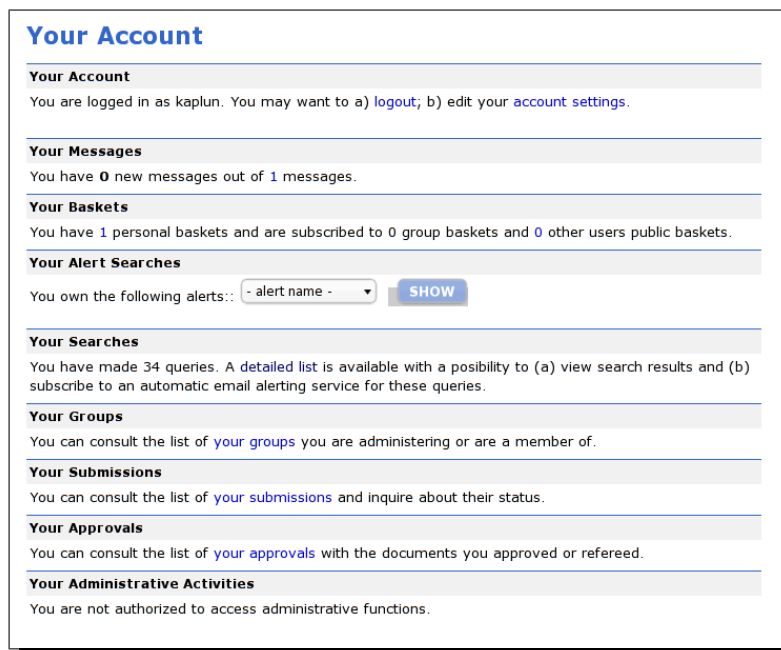


Figure 1.7: The summary of a CDS user account listing all the personalization and collaborative features.

**Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)** A protocol defined by the Open Archives Initiative (OAI), OAI-PMH is used to facilitate metadata exchange among various data providers.

OAI-PMH is based on a client-server architecture, in which "harvesters" request information on updated records from "repositories". Requests for data can be based on timestamp

ranges, and can be restricted to named sets as defined by the provider.

CDS Invenio supports both exporting and importing metadata using OAI-PMH, hence acting both as a harvester and a repository.

Though a converter module CDS Invenio is also able to exchange metadata with non OAI-PMH compliant digital libraries and data repositories.

### 1.2.2 User Classes

CDS Invenio supports several kinds of user classes, the following being mandatory, but not necessarily made up by separate individuals. Along with each user class is a brief description of their most common digital library use-case scenarios.

**Regular Users** Regular users are visitors to the digital library with no particular responsibilities. They search for documents, browse the taxonomy, write reviews, submit comments, and extract bibliographic data.

**Authors** Authors create and submit documents stored in the digital library. Common actions are submitting and updating documents, but also monitoring of document statistics etc.

**Curators** A curator is a content gate-keeper for a collection in the taxonomy. The curator is assumed to have specific and recognized knowledge in the collection's domain, and has been assigned to oversee quality and factual correctness by approving or rejecting document submissions.

**Librarians** A librarian is responsible to assess the quality of metadata, and to insert metadata for books and documents provided at a library.

**System Administrators** Responsible for administration of the servers on which CDS Invenio runs. Administrators are interested in keeping a functional system, by properly configuring the numerous provided features, monitoring the health of service, updating the installation, managing authorities and other users issues.

### 1.2.3 Modules Overview

As explained above, CDS Invenio is a modularized software. What follows is a list of its modules, each accompanied by a brief explanation, while in Figure 1.8 on page 12 can be seen the relationship between all the modules and the various kind of users.

**BibClassify** Used to automatically classify documents according to keyword taxonomies and thesauri.

**BibConvert** Used to configure OAI metadata harvester for eventual periodical batch upload of data. For example, you can define from where to harvest, with what periodicity, how to transform data before uploading them into CDS Invenio, etc. Also, used to define your OAI sets in case you want to export your own data.

**BibEdit** Used to directly manipulate bibliographic data, edit a single record, do global replacements, and other cataloguing tasks.

**BibFormat** Used to specify how the bibliographic data is presented to the end user in the search interface. You can decide that titles should be presented in bold font, that for each author an automatic link to author's home page should be created according to some receipt, etc.

**BibHarvest** Used to configure OAI metadata harvester for eventual periodical batch upload of data. For example, you can define from where to harvest, with what periodicity, how to transform data before uploading them into CDS Invenio, etc. Also, used to define your OAI sets in case you want to export your own data.

**BibIndex** Used to configure "word files", i.e. to define which bibliographic fields are indexed into which word indexes. The word indexes are then used by the search interface. For example, you can define that the logical author index is constructed from physical 100 \$a and 700 \$a bibliographic tags, you can force reindexing of the fulltext index, etc.

**BibMatch** Tools for matching XML MARC files against the repository content. Useful when importing third-party metadata files.

**BibRank** Used to configure various ranking methods to be used by the search engine. You can rebalance existing ranking sets, create new ranking methods, etc.

**BibSched** Used to inspect bibliographic task queue, to postpone or reschedule jobs, to make priorities, to run periodical tasks, etc.

**BibUpload** Used to configure eventual local special operations to be done on the data being uploaded.

**ElmSubmit** Used to configure the submission of documents by electronic mail.

**MiscUtil** Miscellaneous core libraries useful across different modules .

**WebAccess** Used to define who has got access or admin rights on various CDS Invenio modules. For example, you can define that John is the bibliographic data manager, that Jim can modify the search interface pages, that Jill is the submission approval editor, etc.

**WebAlert** Used to inspect and manipulate user alerts set up on the system, to run the alert engine, etc.

**WebBasket** Used to inspect and manipulate user baskets set up on the system, to make them public/private, etc.

**WebComment** Used to manipulate readers comments and reviews, see which ones were reported as abuse/spam, delete them, etc.

**WebHelp** Used to handle admin guides and documentations.

**WebMessage** Used to configure the messaging system.

**WebSearch** Used to configure the search interface for various metadata collections. You can define new collections and organize them in the tree, you can define various portalboxes that would appear on the screen, you can define search options and search fields to present, etc.

**WebSession** Used to inspect the status of guest sessions and to expire them; the status and details on registered users, etc.

**WebStat** Used to configure the usage statistics reporting system.

**WebStyle** Used to customize default CDS Invenio page style and the CSS style sheet.

**Websubmit** Used to configure the submit interface and logic for various document types. For example, you can define which metadata fields should be submitted for various doctypes, what to do with the inputted values before uploading, possible peer review and approval strategy, etc.

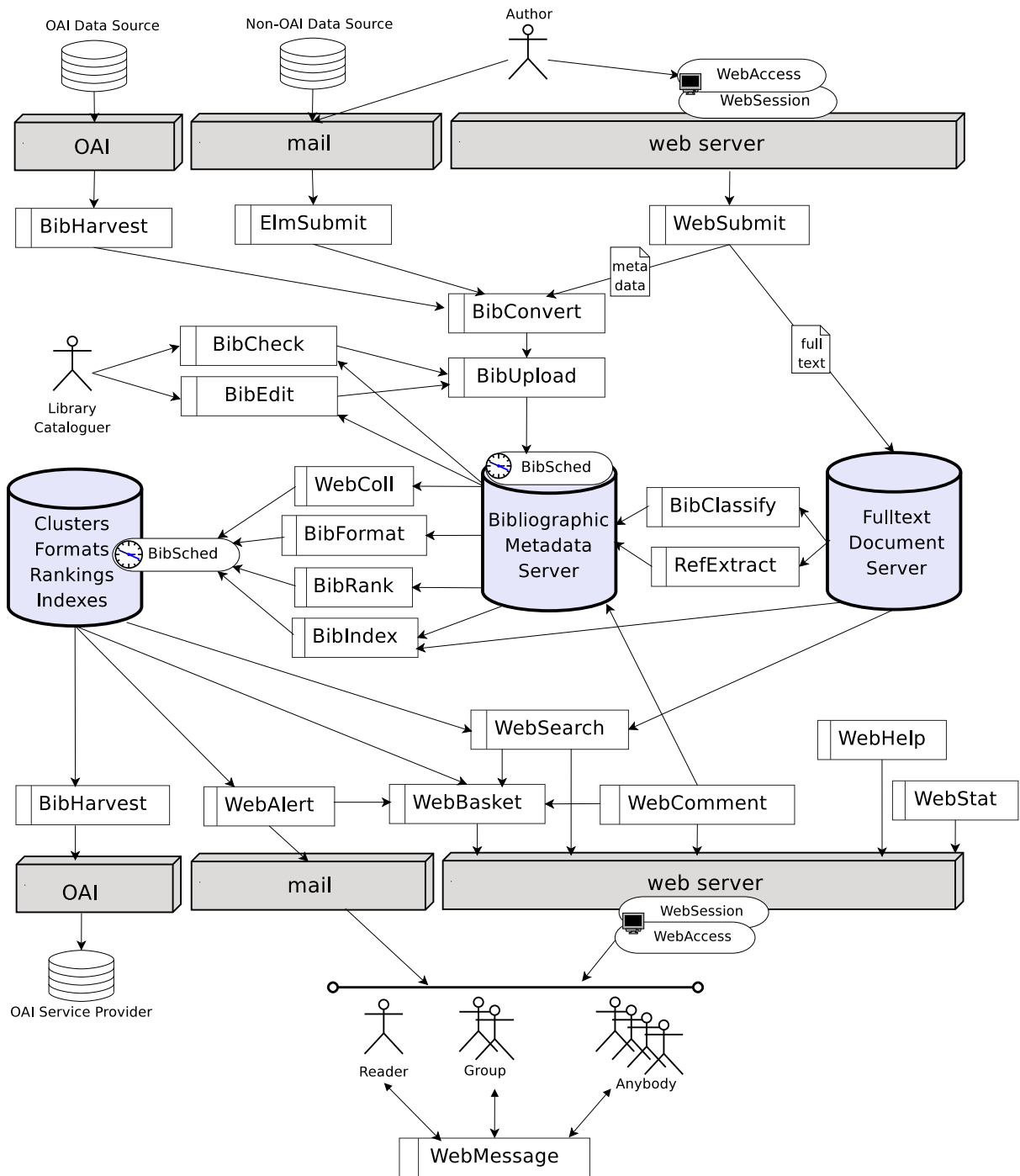


Figure 1.8: CDS modules relationship and user interaction.

## Part II

# Authentication and Authorization issues





## Chapter 2

# Authentication and eGroups

### 2.1 Introduction

Almost every software built to interact with an arbitrary number of users needs to authenticate them. Authentication is the action through which a user of the software prove to the system her<sup>1</sup> identity.

The classical way to get authenticated is to provide a username associated with a secret password, known only to the particular user.

An *authenticated user*, depending on the software, may have more privileges and possibilities in the system than a *guest* one. In CDS Invenio, an authenticated user gains persistent baskets in which she can save a document of interest, can receive email notification about new interesting documents, can submit publications and material and can review others' material. She can send messages to other users, can post comments and reviews about documents she is interested in and can administrate the system, supposing she has granted the proper rights,.

An authenticated user may be, in the future, profiled in order to improve her search experience, with more suited and personalized results. CDS Invenio may, on the contrary, improve the relations among documents given the information provided by the user either passively (through gathering search statistics) or actively (through comments...).

Moreover, a user, both in social real life and in the virtual world provided by community driven software, can belong to a set of different groups together with other users, based on some characteristics. Group support is a recent addition to CDS Invenio and is a feature that is exploited more and more. A group connects users with the same interests and for this purpose is used, in CDS Invenio, to have common shared baskets, to send messages to all the people in a group etc.

---

<sup>1</sup>For typographical reasons we have decided to use the pronouns she and her when referring to a generic user.

It was the subject of the present work to enhance the CDS Invenio authentication framework, especially by better supporting external authentication systems, lightening the job of writing new code to interface CDS Invenio to them, exploiting potential externally-provided user details and group memberships.

In this chapter will be presented the initial status of the art and the improvement designed and implemented by the author about the CDS Invenio authorization system.

In the next chapter will be then demonstrated how this new readily available information will be used to enhance the CDS Invenio authorization system.

## 2.2 Initial situation

CDS Invenio has, since longtime, a basic way to support user authentication, called **local account**.

A local account is a basic way to associate a user to a nickname, an email and a password in order to have the user recognized when this is a need.

A user is able to register herself providing something similar to an email, a nickname and an optional password. The registration process can be tuned in order to give to the administrator the ability to decide whether users can effectively register or whether the user registration requires explicit approval by the administrator.

In the original implementation, though, the email was not checked for real existence, and the password was stored in clear text in the database. That means that if the user lost the password she was able to recover it, by providing her email into a form. The password was then sent back in a clear text by email.

CDS Invenio provided natively the possibility to authenticate users through an **external authentication system**. The only difference from a local account was that the email/password correspondence was checked by an external entity, relieving CDS Invenio from locally storing the password.

At CERN, when this project started, the *Central Nice Authentication System* was providing password checking by means of a SOAP webservice.

The diagram 2.1 on the facing page explain the initial situation of the CDS Invenio login system at CERN.

To login into CDS Invenio the user has only to provide her CERN email or nickname and the so-called NICE password (associated with the CERN email and Windows Systems). CDS Invenio, then, calls the proper SOAP Webservice query providing this information and obtaining the details of the user when the provided data are correct. Prior to this project CDS Invenio considered the authentication result as a boolean value, disregarding all the returned user details. If user details were returned, the user nickname/email was trusted. After this event nothing was different between a user authenticated through the

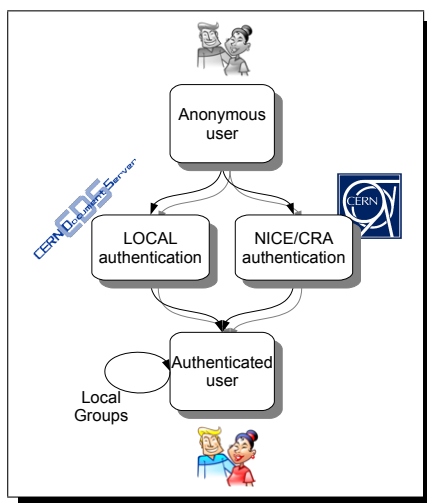


Figure 2.1: CDS Invenio authentication system prior of the project start. CERN Users were able to login either via local account or using their CERN Nice credentials but no added value was there in either choice.

local system and the user authenticated at CERN. The only advantage to use an external authentication system was that, since the user was already registered externally, there was no need to setup “yet another local account” for account service. Moreover the first time a user had used the external login system, her email was automatically imported and associated with a user identification number.

CDS Invenio could be configured to have multiple external authentication systems at the same time, along side the local one and could also be configured to rely only on one external authentication.

### 2.2.1 Identifying users

Every user who interact with either the web interface of CDS Invenio or its command line tools, is identified, at the database level, by a unique incremental integer number, called **user id**.

Prior to login, users are considered *guest*, thus receiving a user id which is assigned to them as long as their browser session lasts.

After logging in, if locally registered, users have also associated a **nickname** (which must be set at registration time and can not be changed anymore), an unique **email**<sup>2</sup>, a field to hold the password for the local account if any, and a generic container for different information called **settings**. As we will see this will be exploited to improve the integration between an account in CDS Invenio and the surrounding institution in which it is installed.

<sup>2</sup>the email is not checked for existence, right now, but is the author will to add this check in the near future.

Special mention is needed instead for users who are automatically registered on the first time they use their external authentication credentials. Their email is infact imported, while the nickname remains set to a NULL value until they first manually set it. After this operation, there is no difference between an externally authenticated user and a local one.

The user id is linked with many user-centric features, such as groups, comments, reviews, submission, rights and restrictions, etc.

## 2.3 Authentication at CERN

As a first step to work on the authentication issue the author has studied what was effectively provided by the CERN SOAP authentication webservice (also called NICE authentication). This webservice is made up by a set of simple functions that offer an interface to the central set of information stored at the CERN Human Resources Department. The functions are:

- **CCIDisNice(ccid)**: Verifies that the `ccid` (CERN user identification number) is associated to a NICE account.
- **GetGroupsForUser(Username)**: Returns a list containing all the group names the specified user is member of. `Username` is the NICE login nickname or NICE email.
- **UserIsMemberOfList(Username, ListName)**: Checks if user `Username` is member of the specified mailing-list. Mailing-list are a form of centrally managed group structure.
- **UserIsMemberOfGroup(Username, GroupName)**: Checks if user `Username` is member of the specified NICE Group.
- **GetUserInfo(Username, Password)**: Authenticates user `Username` from login and password. Login can be NICE email address or NICE login. Return users detail when user credentials are correct.
- **SearchGroups(pattern)**: Searches for a group, based on given `pattern`.
- **GetUserInfoEx(Username, Password, GroupName)**: Authenticates user `Username` from login and password. Login can be NICE email address or NICE login. Return users detail when user credentials are correct and user is member of `GroupName`
- **ListUsers(DisplayName)**: Searches users with given `DisplayName`. This can consist of firstname plus lastname, or email, and can contain wildcards.

To access these functions the system needs to establish a connection with a protected SOAP webservice provided by the NICE authentication system at CERN<sup>3</sup>. Since the service is implemented via the SOAP protocol, queries to the plugin are done through to *http get requests*, while answers are provided in *XML* format.

The first draft of the plug-in for exploiting the external authentication provided by CERN was using just the `GetUserInfo()` function in a boolean fashion not taking advantage of all the added value of the returned information and all the knowledge about what CERN groups (and mailing-lists) the user was member of.

## 2.4 The improvements

As a second step after the CERN NICE Authentication System as been studied, we have proceeded to the conception and development of a new Python class wrapping all the Webservice-provided functions to make further usage of all the extensive information provided by CERN NICE Authentication System.

Two directions have been undertaken in order to let CDS Invenio better cooperate with the external authentication system. The first has been to analyze CDS Invenio in order to find a place to plug the new gathered information in the handiest possible way. On the other side it emerged the need to rewrite and enhance the CDS Invenio authentication plugin infrastructure in order to abstract from the CERN NICE authentication system and gather similar information from other external authentication system.

### 2.4.1 How to handle external user details

Since the CERN SOAP webservice provides a function (`GetUserInfo()`) that, not only authenticates the user, but also gathers administrative data about a user (her telephone, home institution, building, department...), it would have been nice to import these information into CDS Invenio for future usage, and to allow other plugins developed for other institutions and technologies to import similar data. In CDS Invenio a very handy field in the users table represents all the settings of an authenticated user in the form of a serialized Python dictionary. The idea had been to add all the user info's provided by CERN NICE Authentication System to this dictionary as a set of readonly external settings.

New functions have been added by the author to the `WebSession` module of CDS Invenio to add, delete, merge and modify external user details.

Moreover, the author added to the plug-in interface (which is described in Section 2.4.3 on page 21 the method `fetch_user_preferences()`). If a plug-in implement this (optional)

---

<sup>3</sup>The NICE service is one of the authentication provider. As we will see in a section 2.8 on page 30, the ultimate dream at CERN about authentication is to have only a single source for all the authentication based on the Single Sign-On technology.

method, the list of details provided is merged at login time with local information and settings about the user. An algorithm was written in order to handle those external details in the form of a dictionary<sup>4</sup>. Those data are stored in the central database of CDS Invenio for whatever usage<sup>5</sup>.

## 2.4.2 How to handle external user group memberships

CERN NICE Authentication System provides the concept of groups. Every CERN user, once registered at the CERN Human Resources database, belong to a set of groups, based on her age, nationality, language, experiments, computing authorizations, job level, building office, etc. The affiliation to these groups is centrally managed with care, and would have been a great enhancement to exploit this knowledge within CDS Invenio.

In CDS Invenio the support for local groups was already there. Users were able to create new groups based on some interests, and to decide the policy for new memberships to already existing groups.

The author introduced, then, into CDS Invenio the concept of *external group*. A new policy was added to distinguish external groups from local ones. An external group can not be created, or deleted, and users can not subscribe or get unsubscribed through CDS Invenio. Instead, groups are automatically and gradually imported into the system, as soon as a user, logging in with an external account, is member of them. Thus external groups are mirrored into CDS Invenio in a lazy fashion.

If the external account plug-in exports the `fetch_user_groups_membership()` method (see Section 2.4.3 on the facing page), a very fast algorithm performs the following actions:

- For every group that is not already added to the system, it is added alongside its description;
- The system retrieves from the database the list of group members;
- The system also adds any new external members to local copy;
- The system removes any members no more reported by the plug-in.

In this way, CDS Invenio builds, as soon as users log in, a precise idea of the groups the users are members, or those that not, and which groups exist. The only shortcome of this procedure is that if the user changes its external group membership at some point in time, the system does not realize the change until she next logs in. This should not be a very important issue for the group membership practices and the security level requirements of CDS Invenio.

---

<sup>4</sup>A Pythonic structure made by unique keys and corresponding values.

<sup>5</sup>We will see particular usage scenarios in detail in the chapter 3 on page 35

If, anyway, administrators decide that they need an instantly accurate representation of what is happening outside Invenio with respect to group membership, then, if their external authentication system permits such a query, as will be listed in Section 2.4.3, they should implement into the plug-in the method `fetch_all_users_groups_membership()`. Such method should return all the existent groups, with all the emails of the user members of these groups. If such a method is implemented, a corresponding algorithm in CDS Invenio makes use of it. Given the returned map, it applies new memberships to all existent users, and remove obsoleted membership. At the moment of this writing, this algorithm is not yet callable by administrators of Invenio, but it can be easily added if need be.

### The algorithm

The algorithm iterates over all the external plugins and for each plugin looks for an implementation of the method `fetch_all_users_groups_membership()` (described in Section 2.4.3). If such a method exists, it is called, obtaining a mapping between all existing external groups and all the emails of external users belonging to each group. The algorithm retrieves the set of all the emails registered in CDS Invenio. This set is compared to the set of emails linked with each group. If the intersection between the two sets is empty, the linked group is removed from the list of groups to be synchronized. Having stripped the list of empty groups and non existing users from the whole list of groups, this set is checked against the set of groups already existing in the system. All the new groups are directly added with their membership information, all the no-more existing groups are directly removed with all their membership information. All the remaining groups are then checked for changes: for each group, all new users are added, all disappeared users are removed. At the end, a check for group description change is performed.

### 2.4.3 The new authentication plugin infrastructure

In order to integrate external user details and group, it has been the subject of the present work to rewrite the authentication plugin infrastructure of CDS Invenio.

Starting from a draft plugin to fully support the CERN NICE Authentication System, the plugin has been generalized and abstracted to make an abstract class which expose the following API:

- `__init__(self)`: a constructor
- `auth_user(self, username, password, req=None)`: a method to authenticate users, which return a boolean value, with true meaning user has been successfully authenticated.

- `user_exists(self, email, req=None)`: a method to check if a given email exists in the external system. This is an optional method. When the method exists CDS Invenio allows users who are registered with some authentication system to switch to this particular system.
- `fetch_user_groups_membership(self, username, password, req=None)`: should return a dictionary of groups and their descriptions, to which the user is externally subscribed. When such a method exists in the plug-in, CDS Invenio will be automatically able to import external group membership. Otherwise, no effort will be made in this respect.
- `fetch_user_nickname(self, username, password, req=None)`: to retrieve the nickname linked with a user (supposing the email was used for authenticating the user and supposing that the external authentication system holds the nickname information, too).
- `fetch_user_preferences(self, username, password=None, req=None)`: should return a dictionary of keys, and their value, with freeform information about the user. This, as we will see, will be useful for giving rights and restriction on user, with a fine-grain configuration.
- `fetch_all_users_groups_membership(self, req=None)`: if building such an algorithm is possible with the given external authentication system, then this method will be used in order to batch-resynchronize the groups memberships, in a fast-paced way.

The implemented interface is as much backwards-compatible as possible to allow for other CDS Invenio client installations to happily upgrade their code adding new features smoothly.

## 2.4.4 Results

### Importing user details and user settings

The most visible characteristic of the new Import User Details feature is a new panel in the CDS Invenio account page of a user authenticated by an external plug-in, as can be seen from Figure 2.2 on the next page.

Administrative details are visible only to the account owner.

As a proof of concept of the usage pattern of the user settings attribute, three new local settings were added by the author to test the storage of personal local settings alongside the external ones as can be seen in the screenshot in Figure 2.3 on the facing page.

These settings were the first personal settings stored into CDS Invenio to let the user customize the system. More are to come in the near future, that will take part in a



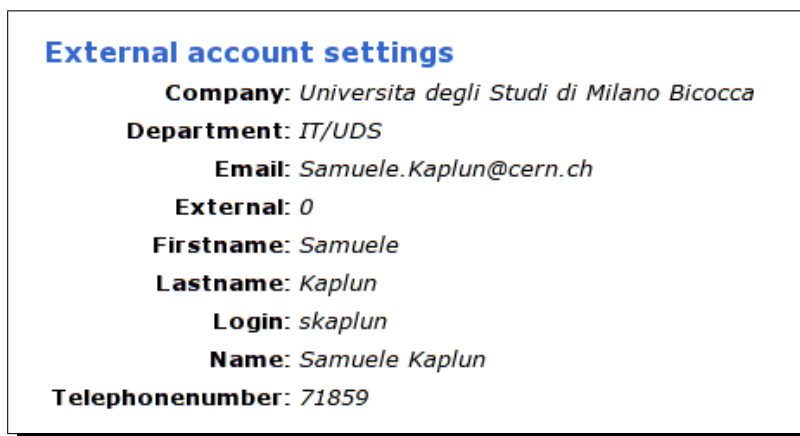


Figure 2.2: A screenshot of the author's (CERN) External Settings imported into CDS Invenio.

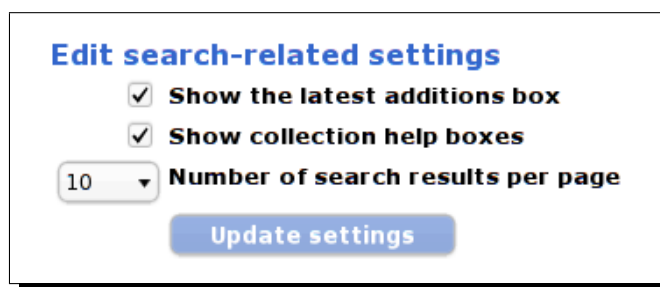


Figure 2.3: The author's current configure CDS Invenio settings.

complete user setting profile. Some already proposed settings are the interface language, the default ranking algorithm, the default search interface (either simple or advanced), and so on.

## 2.4.5 Importing external groups

As a result of the work on external group importing, external groups can now be used in every context where local groups were already used before:

- They can be enumerated and checked in the group page of CDS Invenio (see Figure 2.4 on the next page);
- They can be used to share CDS Invenio baskets<sup>6</sup> (see Figure 2.5 on page 25);
- They can be the receiver of CDS Invenio web messages (see Figure 2.6 on page 26)

---

<sup>6</sup>A metaphor representing a set of records documents that can be shared within multiple groups.

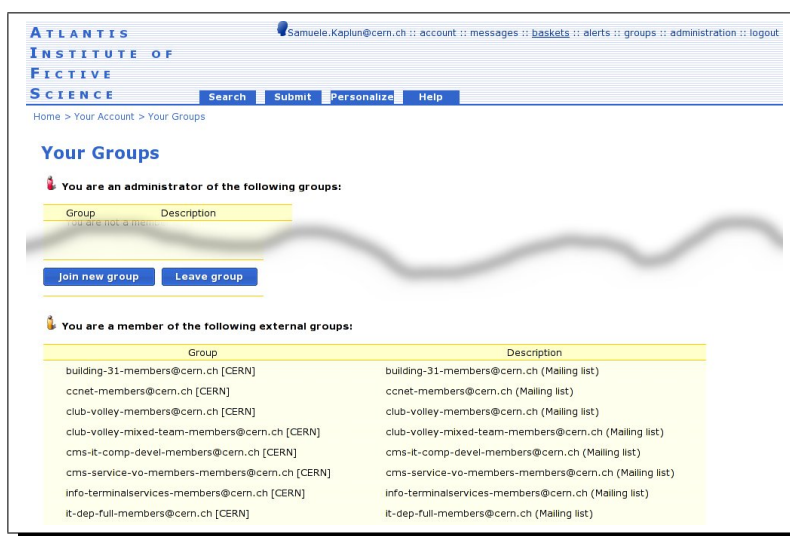


Figure 2.4: CERN groups the author is member of can be seen in the External Groups panel of CDS Invenio.

## 2.5 Testing

CDS Invenio comes with a huge testsuite that tests the whole set of its modules. A unit test for testing the CERN SOAP WebService has been implemented that helped in spotting many hidden bugs and is currently included in the CDS Invenio testsuite.

### 2.5.1 Issues

**How to identify a user.** Because of technical decisions and historical reasons, a user is univocally identified either by an *integer*, a *nickname* or an *email*. All the three data are used within CDS Invenio. Nickname exists in order to protect the privacy of a user whenever she wants to send a message, to sign a review or a comment or to be member of a group. Email should, in theory, allow for the verification of real existence of the user, i.e. to check if the user exists (and is not faked by a robot) by sending some cookie to her email address requiring to provide it back in order to activate the account, as is used by majority of mailing list servers. This feature does not exist yet in CDS Invenio but will be implemented in the near future. Additionally, emails are used to contact the user, either directly or by the automated feature of the system. Integers, as we can imagine, are internal identification very useful within database queries and algorithms.

Having three ways to identify a user can be problematic, especially when it comes with having more than one authentication system enabled.

**A dual login system.** Let's imagine a dual login system (internal and external). A user can be added to the database in two completely different ways. Either by registering a

The screenshot displays the 'Edit basket' interface for a user named 'Samuele.Kaplun@cern.ch'. The page title is 'ATLANTIS INSTITUTE OF FICTIVE SCIENCE'. The user is editing a basket named 'Funny stuff'. The interface is divided into three main sections:

- General settings:** Contains a text input for 'Basket's name' (value: 'Funny stuff'), a dropdown for 'Choose topic' (value: 'The more, the better!'), and a text input for 'or create a new one'.
- Manage group rights:** Shows a dropdown menu with 'club-volley-members@cern.ch [CERN]' selected and a 'View records' button. An 'Add group' button is also present.
- Manage global sharing rights:** Shows a dropdown menu with 'No rights' selected.

At the bottom of the form, there are three buttons: 'Save changes' (blue), 'Cancel' (yellow), and 'Delete basket' (yellow).

Figure 2.5: This screenshot shows how CERN/External groups can be used directly to share baskets.

local account<sup>7</sup> or by using the external account system (Like in picture 2.7 on page 27).

**Local login.** If the user decide for a local account, she will be asked for her bare data, an email and a nickname. An integer ID will be automatically assigned (in fact the user will never see this integer ID). Because of identity preservation reasons (e.g. signed users comments and reviews) the nickname is not changeable after confirming the registration. In fact the nicknames are used for all the signature and it would be a great issue for past comments, reviews and webmessage if the signature changes. As stated before, the email is not yet checked for real existence.

**External login.** The user can, otherwise, login using her external account data, providing credentials that are recognized by the external account as a username (either a real nickname or an email) and a password. If the external account system authenticates the user (that means it is able to return an email related to the user) the user is entered into the database, receiving automatically an integer, which will be the same for every future login. What happens for the nickname? If the external authentication system assigns to the user a value that can be considered a nickname and this value is retrieved by CDS

<sup>7</sup>Based on the system policy, the administrator could decide to have to explicitly approve the user registration request or to manually register each user. For the purpose of this use-case we can just consider the first policy of free registration.

ATLANTIS  
INSTITUTE OF  
FICTIVE  
SCIENCE

Samuele.Kaplun@cern.ch :: account :: messages :: baskets :: alerts :: groups :: administration :: logout

Search Submit Personalize Help

Home > Your Account > Your Messages > Write a message

### Write a message

To: Users   
Groups

Subject:

Message:

Send later? Day  Month  Year

Find users or groups:

Please select one or more:

Atlantis Institute of Fictive Science :: Search :: Submit :: Personalize :: Help  
Powered by CDS Invenio v0.92.1.20070320  
Maintained by cds.support@cern.ch  
Last updated: 14 Feb 2007 18:33

This site is also available in the following languages:  
Български Català Český Deutsch Ελληνικά English  
Español Français Hrvatski Italiano   
Norsk (Bokmål) Osetiá Dordomúče Durrupá Kuvanešu

Figure 2.6: This screenshot shows how CERN/External groups can be the receiver of web messages.

Invenio, then, according to a plug-in-level flag the external nickname can be imported directly as all the other registration data. Otherwise it will be assigned a null value and after the first login the user will manually choose a nickname that will remain for the future. A random local password is assigned at registration time for the login method switch possibility discussed in Paragraph 2.5.1 on the next page.

**From external to local.** At login time the user can choose from a drop down menu what login system she wants to use. This choice could be different from the authentication system used at the registration time. If she has registered externally we have decided, as a matter of security, to not let the user login locally unless she have previously voluntarily switched manually her login method after being logged in externally.

**From local to external.** On the contrary, since we trust the external authentication system and since it is usually richer in the information it can provide about the user, we let the user who was registered locally to choose the external login method at login time, provided she has used an email recognized by the external method when she have registered and that she fills the internal password in place of the external one. Note that if the user is recognized externally by a different email address than the used locally, then when she will use that email it will create a new independent account.



**Login**

If you already have an account, please login using the form below.  
If you don't own an account yet, please [register](#) an internal account.

**Login method:**  ▼

**Username:**

**Password:**

([Lost your password?](#))

**Note:** You can use your nickname or your email address to login.

Figure 2.7: A snapshot of the current dual login at CERN. The selection provides “CERN” and “Local” login methods.

**Stealing emails.** At the moment, a malicious user can register locally using a valid external email which is owned by somebody else who is not yet registered. The proper owner will not be able then to register locally since her email is already used. Anyway, since she knows the external password she can login externally without problems, then switch to local account, and reset the password. Anyway the problem of stealing emails will be soon addressed implementing the cookie for certifying emails during local registration process.

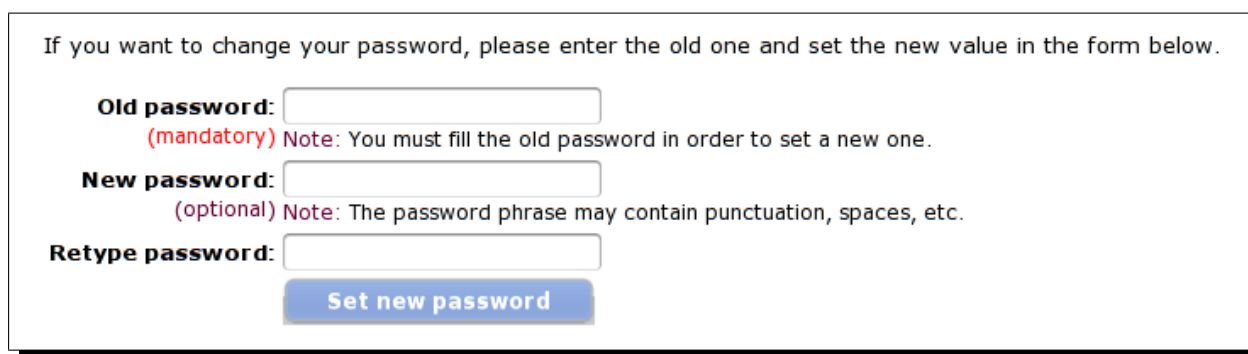
**Switching login methods.** The author had to address the issue of switching from local to external method and back after the login, too. Imagine a user who has registered locally with some general email address who want to switch to external. This was always possible with the initial implementation. But what if the user have used an email which is not recognized by the external system? If the user leaved her settings in this situation, she would never be able to login since she is not allowed to choose local method at login time, once she set external method in the configuration. It is therefore reasonable to limit the login method switch. This action would be allowed only if the external authentication plug-in provides a method that answers if the email is a valid and externally registered one. Only in this very case the user is allowed to make the switch by means of a drop down menu in his account preferences panel). If the user provides correct external authentication credentials, she should never be locked out the system.

## 2.6 Improvements to local password management

The original implementation of the configuration panel for setting the local password was, according to the author, was in need of restyling. In fact it had the following issues:

- local password was changeable even when the current login method was set to external;
- there was no easy means to avoid a random malicious guy passing by an abandoned logged in terminal to change the user password;

The first issues were easily addressable since they were simple bugs rather than architecture design flaw. To solve the third issue the author decided to stick with the usual tradition of requiring the old password to be entered in order to change it to a new one. In Figure 2.9 on page 30 you can see how the reset password dialog appears now.



If you want to change your password, please enter the old one and set the new value in the form below.

**Old password:**   
(mandatory) Note: You must fill the old password in order to set a new one.

**New password:**   
(optional) Note: The password phrase may contain punctuation, spaces, etc.

**Retype password:**

Figure 2.8: The new password reset panel.

## 2.7 Local passwords encryption

The initial implementation of local accounting stored users' passwords in clear text inside a field in the user database table. That was a security issue because administrator or whoever have access to the database would have been able to steal all of them. The system warned the user to not use valuable passwords<sup>8</sup>, but people are lazy and tend to use the same valuable passwords everywhere.

The author decided to encrypt local passwords via the **AES** encryption (see [DR02]) of the user's email address via the user's password. In this way, there is no advantage in being able to decrypt the password field in the database (the content would be obvious), and if two users, by chance, have chosen the same password, then they would be obfuscated by the fact they had been applied to two surely different (by definition) emails address (something similar to what happens in salted encryption in Unix systems, see [FK89]).

---

<sup>8</sup>Passwords are not even required to register a local account on CDS Invenio because historically the personal accounts were used only for non-sensitive exchange of information. However this has become a real problem with a recent growth of community features as reviewing and comments.

### 2.7.1 Password resetting

Applying encryption and decryption of passwords to every place inside CDS Invenio code was not the only change required to correctly implement password encryption. The fact that originally local passwords were stored in cleared form was taken as an assumption in different use-cases of CDS Invenio.

Since users may forget passwords, there must be a way to not render them unable to use the system. Either the system must give the user a way to retrieve the lost password or to reset it. Since the system originally stored passwords in clear format, there were no practical issues in providing the password back to the user, for example via email (the user is required to enter a valid email address of which she is the owner). Since now passwords are one-way encrypted and the system can only check whether a given string is the requested password without being able to explicitly decrypt it back, then a new solution had to be found. The solution was to send a cookie to the email address of the user which corresponds to a web URL to the system. If the user follows that URL she can reset her password by providing a new one. The cookie mechanism is described in detail in the next section.

### 2.7.2 Mail cookies

For the purpose of sending a cookie via email the author added a new generic library to CDS Invenio which is accounted for in Appendix A on page 67. He also refactored at the same time the email sending code within the system. As time passed, in fact, two implementations of code to send automated emails were developed within the system. They all addressed the same goal to send email providing a simple API.

Given the new tool of *mail cookies* and the new cleaned code for sending automated emails the author adapted an already existing web interface to retrieve lost password and used it to instead reset it. The new procedure, as depicted in Figure 2.9 on the following page consists of the following steps:

1. the user clicks on the reset password link and provides her email;
2. a request for resetting the local password for account linked with the given email address is created in the system;
3. a cookie URL is sent via email to the provided email address;
4. the user follows before a given expiration time the received cookie;
5. if the cookie is considered valid by the system, a form to set a new password is provided;
6. the user sets the new password.

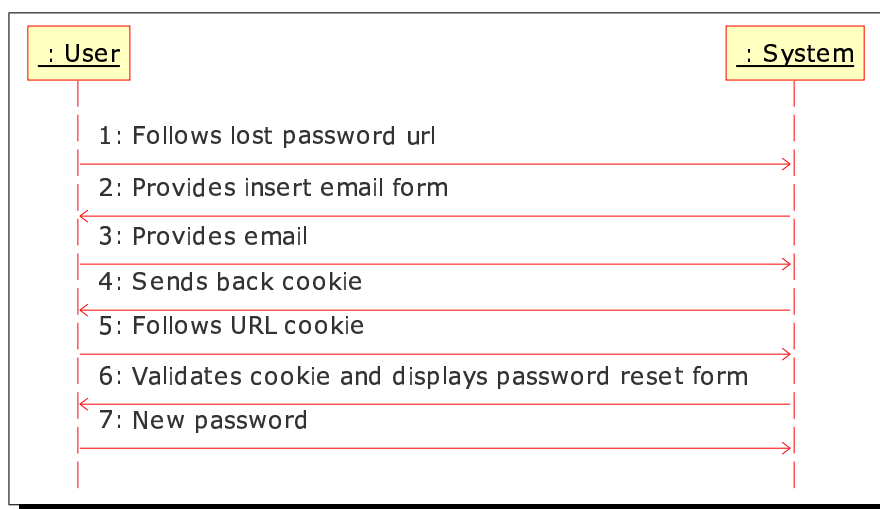


Figure 2.9: The sequence diagram of resetting a lost local password

The cookie is given a certain expiration-time so that a fund is avoided. Moreover, the cookie values are encrypted using database only information in order to prevent cookie stealing.

### 2.7.3 Administrative tasks

One of CDS Invenio goals is to be administered remotely through a web interface. Among the different administrator's activities is to fix troubles within users owned objects, such as web-messages and web-baskets. The current implementation relayed on having clear text local password. If the administrator has to fix somebody's object, once she has chosen an account the system presented the administrator an already filled login dialog with the user's nickname and password, so that by clicking on the login button the administrator can login as the desired user.

This behaviour is no more possible once passwords are one-way encrypted. A solution is being worked on.

## 2.8 SSO

While the author was working on implementing the above-mentioned plug-in structure for authentication, a new direction has been taken at CERN about authentication. Since CERN is a huge institution with many different services offered via web, as years passed every service has implemented its own kind of authentication. Some services are based on the central authentication source of information. Others have implemented proprietary authentication system with new account and passwords.



Many passwords to remember and different security policies have become an issue that can distract scientists in their relationships with web based services. The solution have emerged in a recent technology called “*Single Sign On*” (SSO).

The SSO framework consists of a set of software and protocols that centralize the handling of accounts among different webservice (and, in general, computer based services).

There exists different implementations and flavours. At CERN has been chosen an implementation that works across different *operating systems*, since both *GNU/Linux* and *Microsoft Windows* system are used, was chosen.

The Unix client part is implemented by the **Shibboleth** software, while **Microsoft ADFS** has been chosen for Windows architecture. Shibboleth is a daemon that must run on the system. There exists an Apache module that wraps it.

To integrate CDS Invenio with the new SSO technology, the author had to work on the system level configuration, by setting up the daemon, and a proper configuration of Apache, and then made Invenio application aware of Shibboleth.

### 2.8.1 A brief overview on how Shibboleth works

An Apache server, in general, answers every URL request by building a web page. An URL may not always correspond to a concrete file in the server filesystem. Shibboleth is hooked to a given particular URL. If a user of the server is induced to follow this URL (by means of a login link), the server behaviour is ruled by Shibboleth (see [EC02]). If this is the first time that the user logs into the SSO architecture, her browser is redirected to the central authentication server, which authenticate her provided correct authentication credentials (either in the form of a nickname/email and password or a browser encrypted certificate). The user is then redirected to a page inside the client application. The system now knows that the user has been authenticated, and who the user is, by a set of environment variables that are set, for our special case, inside the Apache system.

At CERN the variables set by SSO contain roughly the same set of information as that provided by the SOAP webservice discussed in the section 2.3 on page 18.

From the point of view of CDS Invenio authentication, user credentials can not be queried actively by some Python code. Data are instead found passively inside prefilled environment variable<sup>9</sup>. This is a very different approach from that expected by the plug-in infrastructure developed so far. Anyway, data provided by the SSO system at CERN were almost the same as those coming from the SOAP webservice. The author decided to integrate SSO into CDS Invenio via a fake plug-in which exported the same API returning the correct data, that are retrieved in an *ad-hoc* fashion, from the above environment variables. Since logging in and logging out of CDS Invenio have different meanings and

---

<sup>9</sup>Every rendered web page stems from a new Python process, which imply the possibility to gather new environment variable from the containing system.

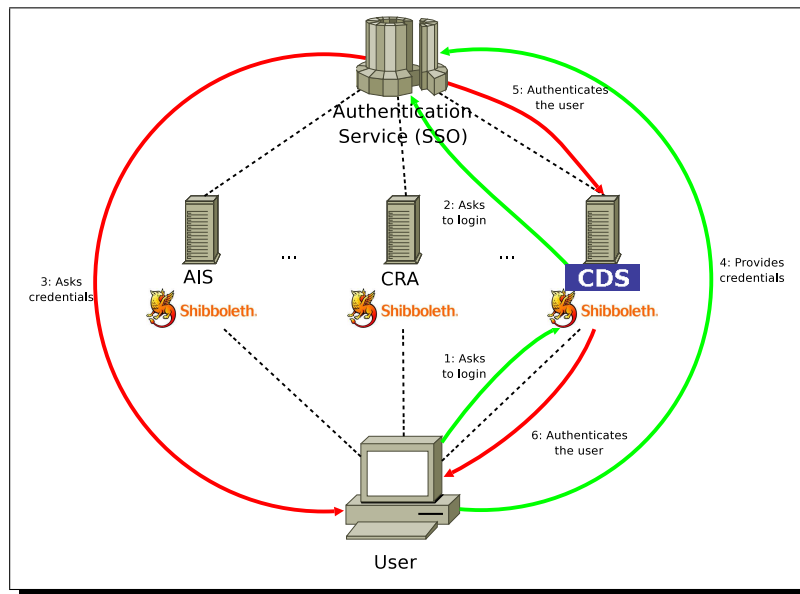


Figure 2.10: This graph show a possible SSO data flow. There a three web services (AIS, CRA and CDS are examples of existing CERN web services) which export authentication by means of the Shibboleth architecture. The authentication is centralized. If a user, who has never login through SSO in the current session, wants to login to CDS (1), her request is forwarded to the central Authentication Service (2), which in turn asks the user her credentials (3); once they are provided (4) they are forwarded to the initial web server (5), which finally authenticate the user (6).

behaviours when using SSO, the author had to add a special flag to switch the system from/into an SSO aware behaviour.

The SSO-based authentication plugin was fully developed and successfully tested and is ready for deployment.

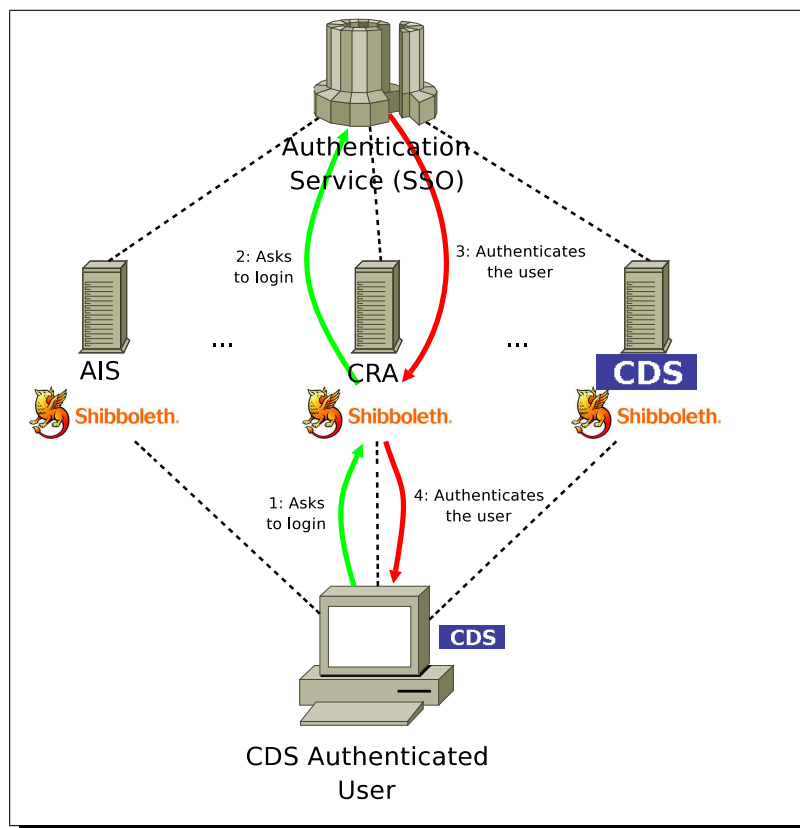


Figure 2.11: This graph show a possible new SSO data flow. Given the authenticated user of Figure 2.11 wants now to authenticate to another web service, say CRA (1), will have her request forwarded to the central Authentication Service (2); since now the user is known to own valid credentials which are stored in the current user session, the service directly authenticate the user, forwarding again her credential to the web server (3), which in the end authenticate the user (4).



## Chapter 3

# Firewall-like role definition for the access control module

### 3.1 Introduction

As a natural consequence of working on authentication the author then concentrated his efforts on authorization issues. Having integrated external groups into the user's information of which the system is now aware, it emerged a possibility to extend existing authorization system to exploit groups and later external user settings such as email, IP address, office building and so on.

CDS Invenio employs a common RBAC<sup>1</sup> system (see [FKC03]). This means that users can execute *actions* in the system based on their membership to proper *roles* which are explicitly linked by means of *authorizations* to actions at administration level. Actions in CDS Invenio have a fine-grained control by having *parameters*. These parameters are filled at authorization time by values taken from real-time situations, in authorization requests.

An example is submitting a paper for a given category, e.g. Computer Science Preprints. An action could be set as “submit”, with a parameter called “category” to which at authorization check time is given the value “Computer Science Preprints”, and this action could be linked to role “Submitters of Computer Science Preprints”. When a user is a member of such a role, she can reach the proper submitting place, following the right links, to submit a paper for this exact category.

Although this architecture may seem very adaptable, it has anyway some scalability issue. In particular to give authorization to users they must be explicitly linked to proper roles one by one.

The author worked on solving this issue by extending the RBAC architecture of CDS Invenio by a notion of grouped-users in a firewall-like setup.

---

<sup>1</sup>Role Based Access Control.

## 3.2 The idea

Even though the majority of scientific documents is public, the CERN Document Server hosts various restricted documents as drafts that has to be approved or rejected, documents private to some department, etc.

It was therefore necessary to conditionally restrict records and fulltext document for certain categories of users.

Before the project started the only provided way to restrict records viewing, was by means of *Apache passwords* checks and *Apache groups* memberships. In order to restrict collections of records, they were connected to an Apache groups by the administrator, then the administrator had to manually each user to proper group. Although this method fulfilled the exact need to restrict records, it was completely orthogonal to broad architecture of authentication and to the RBAC system. This method which could not exploit the features given by having groups (both local and external) nor the already existing RBAC framework.

Therefore, as a side issue of the work about fulltext migration about which the author will account in chapter 4, a natural consequence on the work done on integrating external groups knowledge, he started another work on a way to integrate this knowledge in order to restrict records and documents viewing based on users' memberships to local and external groups.

The first idea has been to create a small language defining which users are allowed to see which collections and to plug this language in place of Apache users password and group checks. It appeared that this was the exact meaning of a role within the RBAC framework, with an action representing an act of viewing a collection, and a parameter representing the given collection being viewed.

## 3.3 The access control RBAC core workflow

As described in chapter 2, section 2.2.1 on page 17, every user is identified within the system by a unique integer, **user id**. This happens for guest users, too. Whenever the user in her activities reaches a part of the system which is RBAC enabled, a call to the function `acc_authorize_action()` within the `access_control_engine` module of **WebAccess** is called, with the given *user id* and the proper action and parameters related to what the user is currently asking to do. This function is the core of the runtime implementation of the RBAC system. Roughly speaking, it checks which roles the user belongs to, and then checks if the action requested (with the given parameters) is allowed to be executed by at least one of these roles, as depicted in Figure 3.1 on the next page.

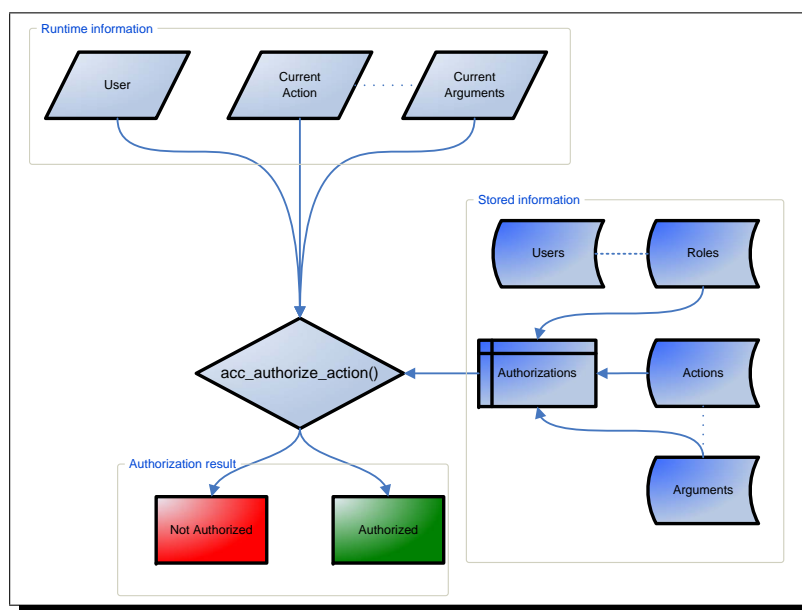


Figure 3.1: Flow of data within `acc_authorize_action()`

### 3.4 Proposed extension

From Figure 3.1 it can be seen that users are directly connected to roles. This means that an administrator has to manually add them one by one to corresponding roles, using the web administrative interface. But roles often represents sets of people with something in common, especially in the example given previously, of viewing a particular collection of documents. If it is possible to extract what the users who belong to a particular role have in common, given an enhanced description of those users (i.e. something more than an integer), then the administrator should be able to exploit this information implicitly to say who can belong to which role.

The extension of the current RBAC system in CDS Invenio, on which the author worked on, has been to add, alongside an explicit linkage between users and roles, and implicit one by means of a new simple language that has been developed and plugged into roles descriptions. This language allows the administrator to write an equivalent in mathematics of a **membership function** for a role.

A new language called **FireRole** (from the contraction of Firewall-Like Role Definition) has been designed which implements boolean functions with parameters ranging over various information on the users.

Given the FireRole language, roles are extended with optional definitions which state implicitly what users are linked to the defined role. The traditional way of explicitly linking users to roles via user ids takes precedence over the FireRole definition.

User information consists of any information which the system is able to gather about the user, e.g. her name, nickname, real name, email address, department, IP address,

browser configuration, internal/external group memberships, etc.

As an example, at CERN, physicists belong to various experiment to which related external groups are assigned. Given an experiment, there could be a collection of private documents and internal notes related to this experiment, which only entitled members are allowed to browse. Given the FireRole language, the administrator can now write down a simple expression to encompass the experiment community in one go. Connecting the defined role with a “view collection” action, everything is done, and no explicit user linking is required.

## 3.5 Users’ details

The FireRole language deals with users details. A user details is a piece of information connected to the user. The set of details makes up a Python dictionary of information that is an abstract data structure constituted by keys which identify values.

The list of user details in a generic installation of CDS Invenio, depends on various factors that can be run-time or configuration related. Some of the possible existing details are:

**uid** is the integer representing univocally a user. This value exists for the whole browsing session if the user is not logged in (i.e. is a *guest*), or for the whole life of the user’s account, if she is registered.

**nickname** is the nickname the user have chosen to univocally identify him/herself. It does not exist if the user is guest.

**email** is the email the user have provided.

**guest** evaluates to 1 if the user has not logged in, 0 otherwise;

**group** is a Pythonic list of strings; each string represent a group name to which the user belongs. Note that external imported groups are also listed here, if an external authentication plug-in is used and if it supports external groups.

**remote\_host** is the URL (or IP address if a URL does not exist) of the user’s computer from which she is browsing. If the user is using command line tools of CDS Invenio this detail does not exist.

**remote\_ip** is the IP address of the user’s computer from which she is browsing. If the user is using command line tools of CDS Invenio this detail does not exist.

**referer** is the URL of the web page from where the user came to the page that is being created. This detail exists only when using the web interface of CDS Invenio.

**uri** is the URL of the page that is being created. This detail exists only when using the web interface of CDS Invenio.



**agent** is the *user agent string* of the browser used by the user. This detail exists only when using the web interface of CDS Invenio and depends on the user's browser configuration.

**apache\_user** is supported for backward-compatibility reasons. The previous authentication method used solely for viewing restricted collections and based on Apache authentication has, in fact, been ported into the FireRole structure, allowing for a soft migration to the new authentication techniques, while at the same time giving more power to the Apache authentication since it can be used everywhere RBAC is applied.

**apache\_group** is a Pythonic list of string representing Apache groups the user belong to, with the same meaning of *group*.

**External user setting.** If an external authentication plug-in that let import external user's information is used, then, if the user log into the system using such a plug-in, all his information are imported too and can be used within FireRole. At CERN, currently does information are:

- full name
- last name
- first name
- department
- company
- office building
- telephone number
- login name
- CERN Computer Center identification number

These key are imported alongside the previously listed set of user details.

## 3.6 The FireRole language

Since the language to plug a membership functions into roles is a kind of filter with a two exit partitioning output, we decided to take inspiration from languages used in network firewalls and Apache configuration rights.

In figure 3.2 on the next page can be seen an example of FireRole definition.

```
ALLOW group "lhcb-members@cern", "Users PH-DT2"
ALLOW remote_ip "128.141.0.0/16"
```

Figure 3.2: An example of FireRole definition, for defining a possible LHCb experiment only role. The first row matches all the users that are members at least of the *lhcb-members* mailing list or of the *Users PH-DT2* group. The second row matches users who are browsing within the CERN network.

### 3.6.1 Syntax

The syntax is very simple, as you can see from the draft in Figure 3.3. An expression is a list of rows. Every row starts with either the reserved word **ALLOW** or **DENY** followed by either the word **ANY** or **ALL**, or the name of a user detail, optionally preceded by the reserved word **NOT**, followed by a *comma-separated list* of patterns. Each pattern could be, in turn, a *single* or *double-quoted free form string* or a *slash-quoted Python-recognized regular expression* (see [Fri02]).

```
expression ::= row(\n row)*
row ::= ("deny"|"allow")
      ("any"|"all"|["not"] feature patterns)
feature ::= (letter|_)(letter|digit|_)*
patterns ::= pattern(,pattern)*
pattern ::= freestring | regexp
letter ::= lowercase|uppercase
freestring ::= ".*" | '.*'
regexp ::= /.*/
lowercase ::= a...z
uppercase ::= A...Z
digit ::= 0...9
```

Figure 3.3: A draft for the FireRole grammar

Note that all the reserved words and symbols are *case-insensitive* and that anything that follow a hash mark sign (“#”) and is outside of a quoted string is considered a comment.

### 3.6.2 Semantic

Given a user with her details, each expression has the following meaning: the whole expression evaluates as the first row which match the user. This in turn evaluates **true** if the row starts with **ALLOW** and **false** if the row starts with **DENY**. A row match a user if one of the following is true:

- after `ALLOW/DENY` there is the reserved name `ANY` or `ALL` (both have the same meaning and exist to comply to English language, e.g. usually you would say “`DENY ALL`” and “`ALLOW ANY`”, but the system understands “`DENY ANY`” or “`ALLOW ALL`” in the same way);
- the reserved word `NOT` has *not* been used and *at least one pattern* matches the corresponding user feature;
- the reserved word `NOT` has been used and *no* patterns match the corresponding user feature.

If the required user detail is not known by the system the row is skipped. At the end of the list of rows there is an implicit row with value “`DENY ALL`”, which means that a user must be matched positively at least once to be assigned to a role.

Among all the user details listed above there are some special case that worth mentioning when dealing with the FireRole language:

- From the point of view of FireRole, `apache_group` and `group` behave in the same way. Since usually a user is part of more than one group, the administrator which creates a FireRole definition using (apache) groups can name the feature in plural form. This syntax is recognized and does not change any semantic. Since we required that, if the row is not using the “`not`” keyword, then the row matches whenever at least one of the listed patterns matches the feature. For groups, this must be extended to require that at least one pattern match at least one group. Similarly, if the row is using the “`not`” keyword, then it matches the user only if none of the patterns match none of the groups.
- The feature “`remote_ip`” has a third implicit way in which it could be matched. Alongside literal string and regular expression, there is the possibility to *network mask* expression (see [MR91]). A rule can be specified as in e.g.:

```
ALLOW remote_ip "192.168.0.0/16"
```

## 3.7 Data flows

*Firewall-like role definitions*, from the point of view of the administrator, are linked to a role. This definition can be entered through the administrative web interface of CDS Invenio as in Figure 3.4 on the next page.

Once the administrator has decided the correct definition, it is compiled into a performance-oriented format<sup>2</sup>.

<sup>2</sup>I.e. a compressed serialized Python structure.

**Modify Role**

**step 1 - give values to the requested fields and confirm to modify role**

role name

description

firewall like role definition [?]

Figure 3.4: Administrator interface to modify a role's name, description and its definition

From the run-time point of view, the previously described RBAC core function `acc_authorize_action()` has been extended in order to handle a set of user detail in addition to a traditional user integer identifier. In every place of CDS Invenio where a traditional call to `acc_authorize_action()` was used (i.e. where the code is RBAC aware), the developer must provide the result of the call to a new function: `collect_user_info()`. This is a function that do its best to gather as much information as possible about the user, with the limitations enumerated in section 3.5, and that produces a dictionary of user details.

The improved `acc_authorize_action()`, given the user's features, the required action and its parameters, first calls the traditional algorithm relying on the user's integer identifier. If this authorizes the user then the whole call authorizes the user, otherwise the algorithm iterates over all the roles linked with the action and its parameters, retrieving the FireRole definitions of each role, if it exists, and passing the user dictionary alongside the definition to a new core function called `acc_firerole_check_user()`, which in turn states if the user is matched by the definition. If at least one role, among those which have authorization to execute the requested action with the given parameters, has a definition which matches the user's features, then the user is authorized to perform the action.

## 3.8 Tricks to improve performances

A set of different tricks has been employed by the author to achieve good performance of the FireRole system:

1. The FireRole definition is compiled in an optimized format. Once a FireRole definition has been compiled there is no more need to modify it, so tuples, which are the

fastest data-structure usable in Python to store sequence of data, and are read-only, can be used without problems to represent FireRole symbols.

2. All the keywords are stored as booleans in the proper position of tuples.
3. Every feature string is stripped and lowercased at compile time.
4. All regular expressions are precompiled and the corresponding Python objects are serialized.
5. The same happens for all the IP masks.
6. A boolean is stored at the beginning of the structure to indicate whether the definition contains Apache passwords/groups. This is needed because to gather information about Apache passwords the server must actively send a request to the user browser, just before the web page building, and this must be done only for users who are going to actually use Apache passwords.
7. Usually, most of the SQL queries that make up the algorithm for checking authorization have static results that change only when the administrator changes FireRole definitions. That means that those queries can be cached independently of the user being authorized. This will be explained in more details in the next section.

### 3.8.1 Caching tools

In general, the CDS Invenio codebase tries to cache as much data as possible for the most part of data categories in order to deliver good performance.

The author worked on a small library for CDS Invenio to replace currently existing caching methods by a generic one. A technique of **memoization** was used.

#### **Memoization and `run_sql_cached()`**

*Memoization* is a technique to improve performance of repetitively used function by storing (with some criteria) function parameters alongside computed corresponding result. On the second call to the memoized-function with the same parameters, the stored pre-computed result is returned without having to repeat the expensive calculations. The author took inspiration from the Python recipe found in [MA02].

CDS Invenio has at its core a relational database for storing all the information. Interacting with it is often at the core of Invenio algorithms. To simplify writing code, one function has been developed — `run_sql()` — which accepts an SQL query and correctly applies its parameters as in the example in Figure 3.5 on the following page.

If the queries rarely returns different results over time, the memoization technique could be applied successfully. The author wrote a wrapper function called `run_sql_cached()`,

```
>>> run_sql('SELECT name FROM user WHERE email=%s',
...         ('Samuele.Kaplun@cern.ch', ))
(('kaplun',),)
```

Figure 3.5: `run_sql()` usage to query for the name of a user given his email.

that checks, given a cached query result, whether tables referenced by the query have not changed since the result have been cached (by saving last modification time of tables alongside queries and results) and directly returns the cached data. The new function call is similar to Figure 3.6.

```
>>> run_sql_cached('SELECT name FROM user
...               WHERE email=%s',
...               ('Samuele.Kaplun@cern.ch', ),
...               affected_tables=('user', ))
(('kaplun',),)
```

Figure 3.6: `run_sql_cached()` usage to query for the name of a user given his email.

To figure out whether to return a cached result or to compute a new one, the last result computation time is saved alongside the result, and it is compared with the last updated time of the involved SQL tables. If the table update times more recent, a new value is computed, otherwise the cached result is returned.

A maximum value of cached queries is configurable. Once this value is reached, the whole cache is erased. This appeared to be faster than to selectively remove expired results or to look for oldest queries.

## 3.9 Tests

Real-life tests of the conceived system have been performed. CDS Invenio is used, among other installations, at the *Ecole Polytechnique Fédérale de Lausanne* (EPFL). There are about 200 laboratories and for each one there is a collection of restricted documents administered only by the corresponding laboratory. EPFL has developed an external authentication plug-in that exploit the institutional central LDAP authentication service. This plug-in imports external groups, too. Every member of a laboratory is part of the corresponding laboratory group. Through scripting<sup>3</sup>, 200 roles have been created with a

<sup>3</sup>A new administrative feature has been developed by a colleague of the author, to import and export in a batch fashion, all the RBAC configuration, including FireRole definitions, in the form of XML configuration files.

FireRole definition similar to:

```
ALLOW group "Laboratory" xx"
```

The system reacts to authorization request in practically no human-noticeable time. Moreover the FireRole language simplified very considerably the maintenance of authorizations at EPFL.

The system might not scale well if lots of roles are authorized with the same association of action and parameters, and if the roles have very complex definition. But this is a very rare situation.

## 3.10 Code Testing

As mentioned, FireRole has been tested in a real environment at the EPFL CDS Invenio installation. Moreover the CDS Invenio testsuite as being integrated with FireRole unit-tests with the following current results:

```
firerole - compiling allow any role definitions ... ok
firerole - compiling complex role definitions ... ok
firerole - compiling deny any role definitions ... ok
firerole - compiling empty role definitions ... ok
firerole - compiling group field role definitions ... ok
firerole - compiling literal field role definitions ... ok
firerole - compiling literal list role definitions ... ok
firerole - compiling more rows role definitions ... ok
firerole - compiling not role definitions ... ok
firerole - compiling regexp field role definitions ... ok
firerole - compiling wrong role definitions ... ok
firerole - deserializing ... ok
firerole - firerole core testing empty matching ... ok
firerole - firerole core testing ip mask matching ... ok
firerole - firerole core testing literal email matching ... ok
firerole - firerole core testing literal group matching ... ok
firerole - firerole core testing non existant group matching ... ok
firerole - firerole core testing regexp email matching ... ok

-----
Ran 18 tests in 0.006s

OK
```

The language has been tested in its features and the emerged bugs had been fixed.

## 3.11 Temporary Roles

A second extension to the CDS Invenio RBAC architecture as been developed by the author.

The extensions exploit the new mail cookies technique (see appendix A on page 67 which has been developed to aid the user resetting the passwords, alongside the new encryption of local password (see chapter 2, section 2.7 on page 28). The new mail cookie system allows to send unique invitation to user, given their email address. These invitations let the receiver user to unlock special information on the basis of encrypted cookie contained in the mail. The library is generic enough to allow for any kind of information to be unlocked.

One information that could be stored is a temporary invitation to become temporary member of a particular role. Mail cookies have by definition an expiration time after which the provided URL is no more recognized as valid. Storing a rolename and membership expiration time as cookie-unlockable information is everything needed to accomplish this<sup>4</sup>.

A possible use of this new feature could be in a future peer-reviewing workflow. CDS Invenio has a peer-reviewing capability. After a new publication was submitted and is waiting for approval, the peer-reviewer are alerted by email to review the paper and either approve it for publication or reject it. With the mail cookie feature, the reviewer could receive an email featuring mail cookie request, with an URL that once followed give them a temporary role with rights to approve or reject the document.

Currently all the backend code for the temporary role via mail cookie technique is ready while the web-interface to call the proper algorithms and functions will be implemented in the near future.

---

<sup>4</sup>Only the database role table structure has to be enhanced with the expiration time field with as a default time infinite in the future.



## Part III

# Fulltext document migration



## Chapter 4

# Migration of fulltext documents with s2d

### 4.1 Introduction

The main unit of information within CDS Invenio is the **bibliographic record**. Every record is an aggregate of metadata details about a **document**, the fulltext of which can exist on paper or electronically in the form of a PDF file or of an image.

The details of a record contain a reference to the original document in the form of a URL.

**PHP era.** The predecessor of CDS Invenio submission module, called CDS Submit, was written in PHP. This module stored submitted document on a separate fulltext storage server. The record metadata then contained links to the document of the typical form:

```
http://{preprints|documents}.cern.ch/cgi-bin/SetLink?base=base&categ=categ&id=id
```

SetLink is a script that interprets the given parameter (base, category, document report number) in order to produce a page similar to Figure 4.1 on the following page.

The page produced by SetLink contains then URL to actually download real fulltext documents, undisclosed real filesystem paths.

SetLink born within the CDS Invenio effort as an abstraction from URL in order to hide backend storage method and provide consistent URL for documents which did not change during time (see [LMM00]). This technology is similar in idea as the *Digital Object Identifier (DOI)* standard (see [(ID06)]).

**Python era.** CDS Submit has then been integrated to CDS Invenio Suite developed in Python. The transition have required many different tools to be implemented to aid to migrate already stored data. This has been necessary for improvement and new features



Figure 4.1: A legacy page for downloading the fulltext file of a document.

that were meanwhile implemented into the CDS Invenio code. Because the migration from PHP to Python for the CDS Submit module required a lot of changes, the production machine of CDS at CERN is still running the old PHP CDS Submit code with all its dependency.

An overview of the actual data flow can be found in Figure 4.2 on page 64.

The PHP CDS Submit code interacts with the Pythonic BibUpload for when it comes to metadata (i.e. MARC), but use the legacy code to store fulltext document on its own. This means that from the Python point of view, fulltext are handled by a black-box, whose output is only the SetLink URL, which brings an already formatted page where real URL are presented. The Pythonic BibFormat must merge the legacy formatted page, retrieved from the SetLink URL with MARC metadata<sup>1</sup>.

Handling PHP legacy code as a blackbox, brings many shortcomings, where the worst is to not have any control on who download what, neither statistics nor restrictions via RBAC (only restrictions via Apache are possible).

On the other side the official (non CERN aware) CDS Invenio code is almost completely written in Python (some parts are in Lisp, though) and each module is tightly integrated with each other. The actual data-flow of CDS Invenio if it would be installed cleanly (without legacy PHP code) would be as in Figure 4.3 on page 65.

As can be seen the flow is much simpler. URL for fulltext documents are directly stored within the metadata and fulltext document can live alongside the CDS Installation, letting them to be much more controlled.

The purpose of the **s2d project** (SetLink to Document) has been update CDS Invenio installation at CERN from the legacy hybrid PHP-Python code to the current Pythonic-

<sup>1</sup>At present the formatting has changed and when a record is displayed in details, it appears an icon for the fulltext set2link URL, instead of merging the legacy page

only way, by

- migrating fulltext documents from a very old SunOS machine to the AFS distributed filesystem;
- rebuilding all the data-structures as expected by the Pythonic code;
- gathering as much information as possible in order to correctly fill database tables;
- feeding statistics from Apache logs on the old machine into statistics database table to be exploited by ranking algorithms;

## 4.2 The legacy fulltext architecture at the CERN Document Server

In this section will be explained the current CDS Invenio installation at CERN, which implements a legacy hybrid cooperation between old PHP and new Pythonic code.

### 4.2.1 Metadata

A document record is stored, within CDS Invenio, in the form of aggregate metadata that describe a document. The format in which they are expressed is the MARC standard. The real fulltext document when it is available in electronic format, can then be referenced in two different ways:

- an external URL; used when the document is hosted by some external web host;
- an internal URL; a URL pointing to the system itself;

When the document is referenced, a URL is stored in the 8564 tag of the MARC metadata. The information is structured using a subfield \$u for indicating the URL itself and an optional subfield \$y (or \$z according to the configuration of CDS Invenio), where a descriptive string of the URL (and hence of the referenced document) is stored.

URL of document hosted by the legacy hybrid CDS Invenio installation at CERN are in the form of *SetLink* URL. This means that the pointed document is in fact an indirect HTML page that display in a fancy way what formats do exist for the real document (for example .PDF and .PS or .DOC) and whether to create optional new format or to extract pictures from the document. As explained above, those HTML pages are handled by the legacy PHP code on a separate machine.

**Issues.** Having fulltext document files hosted on a separate legacy machine have rendered the communication between the real CERN Document Server and the file storage machine hard, thus preventing a natural interaction between the two machines. There is infact no direct way to know how many times a fulltext file has been downloaded, since the legacy PHP system did not support this feature. The only source of statistics is in the form of Apache log files. Moreover indexing the content of the fulltext files is not feasible since the document resides on a remote server, and there is no clear way to discover when a file has remotely changed and needs to be reindexed.

### 4.3 The current CDS Invenio fulltext data structure

It is still in the process to be confirmed how exactly local fulltext document are to be referenced by the MARC metadata, but the most probable emerging format is that a 8564 tag will exist for every single fulltext document format file. Supposing the record has an integer identification number 12345, there is a file name called 123doc existing in PDF and PS format, and the server URL is `http://cdsweb.cern.ch`, then the MARC metadata for the record will contains tags as in Figure 4.4.

```
000012345 8564_ $$uhttp://cdsweb.cern.ch/record/12345/files/123doc.pdf
000012345 8564_ $$uhttp://cdsweb.cern.ch/record/12345/files/123doc.ps
```

Figure 4.4: How locally-hosted fulltext document could be represented in MARC

Having fulltext document filenames stored in the MARC metadata allows the system to export this information to external harvesters which import data about scientific material hosted within CDS Invenio. Moreover fulltext document filename are also stored in the local filesystem in a structured way that allow for easy retrieval and streaming, and are referenced in special database tables, that will be analysed in the following section.

#### 4.3.1 Current database table structure to store Fulltext Document Information

Figure 4.5 on the facing page represent the ER diagram of the database tables used to store information about records and their relationships with locally-hosted fulltext document files.

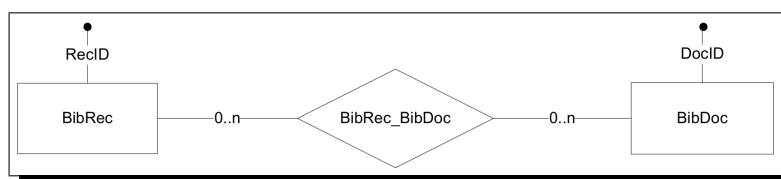


Figure 4.5: ER diagram of the database tables used to store information about records and their relationships with locally-hosted fulltext document files.

Each record in CDS Invenio is stored as an *ntuple* in the **bibrec** table, and is identified by a unique record identification number (**recid**). In the same way each locally-hosted fulltext document is represented as a *ntuple* in the **bibdoc** table, and is identified by a unique document identification number (**docid**). The **bibdoc** *ntuple* contains the filename of the document without extension and some other minor information. A **bibdoc** row represents a whole document regardless of the existing different formats and revisions. This information, as explained in the following section is structured in the filesystem hierarchy.

A relational intermediate table called **bibrec\_bibdoc** store the link between which record is connected to what fulltext document and what kind of relationship link a record to a document.

### 4.3.2 Current filesystem structure to store Fulltext Document Information

Fulltext files are stored on the filesystem with the following convention: under the `var/data/files` folder, within the CDS Invenio installation, there are progressive folders named `g0`, `g1`, `g2` and so on, each containing by default 5000 `docid` subfolders<sup>2</sup>. Each enumerated folder contains all the information referenced by the corresponding `docid`. Every folder will contain in turn all the various formats and revisions of the referenced document. All the files are named after the specified filename, followed by different extensions according to the corresponding format, and by “;*n*” where *n* is a progressive number which grows whenever a new revision of a file is added.

A hidden file called `.recid` contains then the `recid` of the record which is pointing to the given document.

**Benefits of the current CDS Invenio fulltext handling.** Having files stored locally allows the webserver to have direct control of the streaming process of these files to the outside world. That means, for example, that is possible to apply WebAccess restriction to file download and that every single download can be counted. Moreover, on Unix

<sup>2</sup>The `g0` folder contains folders named from 0 to 4999, the `g1` folder contains folders named from 5000 to 9999 and so on.

systems files which reside remotely can be referenced by means of local path name by mounting on the local directory tree a networked filesystem<sup>3</sup>.

Being able to know whenever a download happen, the CDS Invenio software can support fancy feature like statistics similar to “people who downloaded this article also downloaded this...”, and so on.

## 4.4 The S2D tool

It has been the subject of the present work to design and develop a tool to migrate locally hosted fulltext document files from the legacy hybrid structure to the current one. This tool has been called **S2D** after “SetLink to DocId” technique that it realizes.

The tool had to be designed with these characteristics in mind:

- **being reliable:** every single fulltext document is important within CDS Invenio, and even loosing a single document is not affordable;
- **being correct:** moving a document in the wrong place or creating broken references to it is like loosing the document;
- **being fast:** about half of the 1,000,000 records stored at CDS Invenio references a local fulltext document. Being every file of an average size of 100.000 bytes, the amount of stored information is indeed huge, hence the need to not add bottlenecks on top of the network migration speed;
- **being smart:** filesystem paths and metadata and database references to files have completely changed between the legacy and the current system; moreover revisions and formats have to be handled with care.

This tool would have run on a mirror of the CERN Digital Service server, working directly with the available CDS Invenio source code and database, and migrating files to an AFS space mounted on top of the `var/data/files` folder described in section 4.3.2 on the previous page

## 4.5 Detailed preliminary analysis

A detailed analysis as been undertaken in order to understand how the tool would have been best written to interact with the two system.

---

<sup>3</sup>For example CERN has some node shared in the global distributed AFS filesystem (see [HUC88] and [HKM<sup>+</sup>88]), and every server can take use of this space by mounting the AFS filesystem.



### 4.5.1 What should be migrated

The author had to define exactly what has to be moved. Indeed, among all the CERN Document Server hosted records, most of them refer to a fulltext document, but only half of them is actually referring a *locally-hosted* fulltext document by explicitly having a SetLink URL in the proper MARC tag of its metadata. So the author had to take care of scanning the whole set of records (or a given subset), and to consider only those records that had at least a SetLink URL among their metadata. Infact metadata of a record can contain more than one 8564 tag referring either to local and remote fulltext document link. Both kind of information had to be managed carefully.

### 4.5.2 The SetLink URL

Given the records with at least one SetLink URL among their metadata, real fulltext document files must be retrieved.

The SetLink URL referring a locally-hosted fulltext document file is infact pointing to an HTML page similar to what can be found in Figure 4.1 on page 50. The real file can then be retrieved by clicking on the correct link appearing in the page. In fact many URL appear on the page:

- the most part of them are of no use for the tool;
- someothers are indirectly referring the document by means of some script that will translate *on-the-fly* the real document to a different format letting then to stream the freshly created file;
- some are referring to sub-part of the real document, like images inside a scientific article (also by means of a script),
- some of them referer to real stored formats of the document.

For one document many different formats can be stored and it is important to migrate each one.

A way to discover what is the definition of an URL referring to a real stored file had to be found. Indeed it emerged that every URL of this kind was containing the word “archive” or “electronic” and not containing “cgi-bin” were exactly all and only the URL corresponding to stored fulltext files.

Parsing the provided HTML page referenced by the SetLink URL in a given record to find suitable URL of fulltext files appeared to be a simple task, thus to deep further into the fulltext storage legacy machine code and architecture was not needed.

### 4.5.3 Remote fulltext file revisions

One question arisen: where all the file retrievable by parsing HTML page the whole set of document to migrate? Infact the answer was no. The legacy storage machine was accessed by many people, and often by logging directly to Operating System without interacting with the provided web interface. Thus, as time had passed by, some of the remote fulltext file had been substituted by new revision, with an informal backup system. That means that the old revision of the file was actually saved in a file name named in similar way to the original and stored near the original but without a consistent policy. Some backup had been colled “file-backup.ext”, someother “file.ext-old”, some “file-dateofthebackup.ext” and so on.

Old file revisions are of less importance than final stored files, so it appeared important to find an heuristic to try to migrate as much old revision files as possible, without the requirement to migrate exactly all the existing old files.

### 4.5.4 Database data to be recreated

The database data to be recreated had been studied. It appeared that bibrec table had to remain untouched while, bibdoc table had to be filled with just-imported fulltext document information. Moreover bibrec\_bibdoc table had to be setup properly. Analysing this relation table, one field came up as very important, i.e. the document *type*. This field holds a freetext string that is filled by the very configurable WebSubmit module, in order to mean special document meaning. For article and preprints it appeared that this field was actually filled with only two values “Main” and “Additional”, meaning the the linked bibdoc was, respectively, the main document or an additional auxiliary file, say an image within a chapter or a single chapter within the whole document and so on.

How to recreate the additional/main information given the above described situation had to be found. The author had worked on an heuristic algorithm based on the whole set of document referenced by a SetLink URL and their extensions. Basically this had emerged:

- If only one file exists, then it is a main file;
- If more than one file exist, then those file with extension being PDF, PS or PPT and not containing a .FIG subextension (as in .fig.ps) are definitively considered as Main documents while the remaining are considered as Additional.
- If no file happen to be considered as main document, than the all the file are considered as Main one (usually this never happens).

Moreover, it has been decided that for semantic correctness a bibrec would have pointed to at most one Main bibdoc and as many Additional bibdoc as needed.

### 4.5.5 Metadata handling

MARC metadata of records involved into the migration had to be up updated accordingly. This means that in place of all the SetLink URL a set of URL pointing to the local machine and referring files in the proper way had to be put. This means that all non SetLink URL had to be preserved, too.

It has been decided to update the MARC metadata by means of exploitation of BibUpload tool. This take care of metadata by reading MARCXML files and, according to the chosen mode, can append or substitute the changed data.

### 4.5.6 Importing statistics

Current CDS Invenio software features support for download statistics. Every click on a locally-hosted fulltext file is stored in a table and analyzed by a background process, which take cares of producing nice statistics and potential improved ranking. Fulltext files migrated from the legacy installation, though, will appear as never being downloaded. Since this is not the truth a way to migrate statistics had to be found.

### 4.5.7 Guidelines followed in writing the tool

The author has decided to exploit as much as possible the already existing CDS Invenio tools and API in writing the S2D tool. The main reason for this decision has been to not add new sources of possible bugs in the migration process, by for example dealing directly with the database tables in a wrong way.

Since the process had to be reliable, the tool had to produce on purpose a huge quantity of debugging information, accounting for any decision it takes and any operations it carries out.

## 4.6 The S2D algorithm

Here follows the algorithm implemented by the S2D tool:

```
INPUT: a QUERY
```

```
EXECUTE the query on the machine
```

```
PREPARE MARCXML update file
```

```
FOREACH retrieved record:
```

```
    IF record reference a bibdoc:
```

```
        SKIP to the next record # it has already been migrated
```

```
RETRIEVE the MARCXML of the record
RETRIEVE the 8564 tags
FOREACH URL:
  IF SetLink:
    RETRIEVE pointed page
    RETRIEVE good URLs
    FOREACH good URLs:
      FIND whether it is Main or Additional
      FIND Old revision
      BUILD lists of Main/Additional/Old/New documents
    FOREACH lists:
      FILL database tables properly
      MOVE the file in the proper local filesystem position
      ANNOTATE move
    UPDATE MARCXML
  ELSE IF record has at least a SetLink URL:
    UPDATE MARCXML
  ELSE skip record
CONCLUDE MARCXML update file

OUTPUT:
  MARCXML update file
  MARCXML undo file
  statistics-about-the-migration file
  debbuging-information file
  undo information
  ALEPH compatible update file
  Apache redirection information file
  Python redirection information file
```

### 4.6.1 The input

The S2D tool takes as input a *query string* as it would be typed in the WebSearch interface of CDS Invenio. This allows to migrate documents one collection per run, which in turn corresponds to port one submission interface at a time, from the legacy system to the new. Moreover migration of record created or modified within a particular time interval is feasible, thus allowing to run the S2D tool only on documents which where not modified.

### 4.6.2 The output

The S2D tool, alongside the its migrating job, outputs many different files:

- a correct-mode MARCXML file, needed to substitute all the SetLink URL with the proper set of new URLs (it is the input of the BibUpload tool in “Correct-Mode”);
- a append-mode MARCXML file, used to append new URLs after existing SetLink URLs (in this way it is demanded to BibFormat tool to choose what URLs to display);
- a undo-mode MARCXML file, used to get back to previous status after having run BibUpload (see section 4.6.3);
- a statistic file, containing recids, docids, moved size and file name, in a format suitable to be the input of plotting tools;
- a debugging log file, with all the operations carried out in full details;
- an undo file, where are written in atomic form all the migrated recids (see section 4.6.3);
- a correct-mode Aleph file, needed to update URLs referenced by the external Aleph service offered by the CERN library;
- an Apache redirection file, to be plugged in the legacy machine as a configuration file, in order to have migrated SetLink URLs redirected to new proper URLs;
- a Python readable redirection file, containing a serialized dictionary easily loadable into memory by Python, which is used by the statistics migration tool (see section 4.6.4 on the next page);

All the output is produced as long as the process is running and all the files are properly closed at program termination, interruption or crash, thanks to proper handling functions installed.

### 4.6.3 Undo possibility

If something goes wrong, say there is an error in the code, or who is supervising the migration notice something broken, the migration can be interrupted, and an **undo** procedure can be performed.

The S2D tool is in fact accompanied by an undo tool that revert the migration. The undo tool job is infact simple: all it has to do is to remove all and only the bibdocs referenced by the bibrecs listed in the undo file. This is accomplished by exploiting properly the CDS Invenio API.

If BibUpload has already run, thus MARC metadata have been already modified, a corresponding MARCXML undo file can be fed to BibUpload in order to get back to the previous situation.

### 4.6.4 Migrating statistics

As introduced in section 4.5.6 on page 57 download statistics of fulltext file being migrated are very important data. As a second step on the S2D migration project the author worked on a tool to migrate statistics. This tool works next to S2D, exploiting its output.

Properly parsing Apache logs let extract very important statistical information:

- What file has been downloaded;
- What IP-address requested that file;
- When that file has been downloaded;

Given all the Apache logs of the legacy server up to a given day, the tool parse all the log looking for legacy fulltext URL (those URL are listed in the Python redirection file). For every encountered migrated URL it translates the URL to the new address, and retrieve thus the corresponding DocId. What miss is only to fill the proper ranking table with these information.

## 4.7 Test and results

The S2D tool has been run on a mirror of the CERN Document Server, setted up for this purpose. It has first been run on a 40,000-documents thesis collection, migrating documents to an AFS partition, locally mounted on the mirror machine. This first run on a small set helped in spotting bugs, and missing features. The Undo tool, indeed, has been designed after first trial and errors.

The tool has then been run to migrate the whole set of scientific papers and so-called “preprints”. More than 400,000 documents have been moved, and their statistics too. The migration lasted about 5 working day. Migration of the whole download statistics from 2003 ( 4 years, 1300 days) up to the migration day, took about 5 hours.

Since the AFS partition is a network filesystem, it can be mounted on many server at a time. Thus files moved during testing are already ready to be referenced by the official production machine, once the created database data and the corrected metadata are integrated into the production machine. This will happen in about a month since the defend of the present work.

## 4.8 Indexing issues

As a part of the migration process, other pieces of the whole CERN Document Server had been updated. The update of the MySQL server, though, brought some very important

speed issues. It appeared that upgrading database server from its 4.0 release up to the 4.1 changed the way it stored information on the table, particularly when it comes to store UTF-8 string (for some reference to the UTF-8 format, see [Yer98]). This in turn made the indexes table to occupy much more space than by using previous MySQL 4.0. For example the main forward index occupied now 20GBs to hold all the record hits. To work with such a huge index required much more computational time than with the previous MySQL release.

A way to shrink index table size had to be found.

The bottleneck was identified as being the data structure used to store the HitLists (i.e. the index which, given a word, holds the list of record containing that word). The data structure used was a Numeric Python array (for an introduction to Numeric Python, please see [San99]). The HitList is stored in the form of a *bitvector*, where putting bit number  $n$  to True meant saying that record with RecId  $n$  hold the given word. In fact Numeric Python implements an array of bytes to store bits. That means that indexes were 8 times more big than was needed.

A side work in the migration project, the author worked in designing and implementing a data structure to plug in place of Numeric Python bitvector array.

Before that Numeric Python had been chosen, a benchmarking tool had been used to test which data structure would have been the best, among alternatives existing at that time. Here are presented the results of running the benchmark on the machine used to type this thesis (an AMD Turion64 X2 CPU):

```
CDSware data structure testing in Python (n1=350000, n2=350000, nrec=800000)
=====
creating lists ..... 350000 + 350000 items created in 3.79 sec.
marshaling lists ..... 532616 + 532571 compressed chars marshaled in 1.22 sec.
demarshaling lists ..... 350000 + 350000 items demarshaled in 0.10 sec.
merging lists ..... 546965 items found in 0.46 sec.
intersecting lists ..... 153035 items found in 0.35 sec.
filling dicts ..... 350000 + 350000 items created in 0.42 sec.
marshaling dicts ..... 576491 + 576450 compressed chars marshaled in 0.78 sec.
demarshaling dicts ..... 350000 + 350000 items demarshaled in 0.39 sec.
merging dicts ..... 546965 items retained in 0.39 sec.
intersecting dicts ..... 153035 items retained in 0.31 sec.
filling sets ..... 350000 + 350000 items created in 0.43 sec.
marshaling sets ..... 532616 + 532571 compressed chars marshaled in 1.35 sec.
demarshaling sets ..... 350000 + 350000 items demarshaled in 1.54 sec.
merging sets ..... 546965 items retained in 2.79 sec.
intersecting sets ..... 153035 items retained in 0.36 sec.
filling builtin sets ..... 350000 + 350000 items created in 0.24 sec.
marshaling builtin sets ..... 532616 + 532571 compressed chars marshaled in 1.25 sec.
demarshaling builtin sets ..... 350000 + 350000 items demarshaled in 0.30 sec.
pickling builtin sets ..... 554960 + 555049 compressed chars pickled in 1.30 sec.
depickling builtin sets ..... 350000 + 350000 items depickled in 0.37 sec.
merging builtin sets ..... 546965 items retained in 0.10 sec.
intersecting builtin sets ..... 153035 items retained in 0.12 sec.
filling array ..... 350000 + 350000 items created in 2.92 sec.
marshaling array ..... 856409 + 856063 compressed chars marshaled in 2.50 sec.
demarshaling array ..... 350000 + 350000 items demarshaled in 0.32 sec.
filling b_from_a ..... 350000 + 350000 items created in 0.05 sec.
filling bitvs ..... 350000 + 350000 items created in 0.23 sec.
sizing bitvs ..... 350000 + 350000 items counted in 0.56 sec.
```

```

marshaling bitvs ..... 179865 + 179809 compressed chars marshaled in 1.09 sec.
demarshaling bitvs ..... 350000 + 350000 items demarshaled in 0.10 sec.
merging bitvs ..... 546965 items retained in 0.00 sec.
intersecting bitvs ..... 153035 items retained in 0.00 sec.

```

Two sets of random number were stored in the tested data-structure, and different set operations were performed, including serializing the structure to a string for storing into a database and deserializing it. As can be seen Numeric Python bitvectors (`bitvs`) happens to be the most performant structure among the benchmarked one.

### 4.8.1 IntBitSet

Using bitvectors appeared to be still the proper solution to holds HitLists. Infact, bitvectors could have been implemented in a way that exploited modern CPU features (the new server would been a 64bits machine, thus allowing for 64 parallel bit operations). Intersection, union and difference are all much used set operations when computing boolean-model search queries. Exploiting CPU lowlevel instruction would allow for set operations of time  $O(n/64)$ . Moreover since HitSet bitvectors are usually sparse, storing them in compressed format (e.g. by the pervasive ZIP algorithm, see [Sal04]), would have allowed for very low space occupation<sup>4</sup>.

The author studied a way to implement real Bit Vectors to store Integers, while exploiting modern CPU features and integrated with Python. The found solution consisted in implementing a Python extension by wrapping an *ad-hoc* bit vector library implemented in C (with code exploiting CPU shifts and register boolean operations).

**IntBitSet** has been implemented. This is a new CDS-Invenio module which usage is plugged in every previous HitSet (Numeric Python array) code. **IntBitSet** has been developed using *Pyrex* (see [Ewi]). Pyrex is metacompiler which takes code written in a hybrid language between Python and C (infact it is Python plus C types), and produce pure C code that compiles into a binary Python extension. Among Pyrex features is the complete support to wrap already existing C functions within Python functions. The author thus implemented all the algorithms in pure C and wrapped everything a Python extension.

The API exposed by the Python extension is a superset of the already existing Python `set` abstract data type, thus allowing for direct substitution in much of the existing CDS Invenio code (Numeric Python arrays expose infact a similar API). Moreover exposing the `set` API, would allow for clean code, given the maturity of this standard API. Differently from the Python set data type, **IntBitSet** can hold only integers, thus allowing for much improved optimization. Moreover the **IntBitSet** data type allows for storing **Infinite** set. This means that the “mathematical universe concept” can be expressed and many higher

<sup>4</sup>Indeed, the Numeric Python bitvector array was already zipped, but since it required unefficient serialization to transform it into a zippable string the used space in bytes was only half of the number of referenced RecIDs.



level algorithms can thus be written. `IntBitSet` can hold a dynamic number of integers, growing and shrinking as needed, with improved space efficiency.

## 4.8.2 `IntBitSet` benchmarks

As a result of code optimization and design design here follows `IntBitSet` performance given the same working condition as above:

```
filling intbitsets ..... 350000 + 350000 items created in 0.24 sec.
marshaling intbitsets ..... 99536 + 99547 compressed chars marshaled in 0.01 sec.
demarshaling intbitsets ..... 350000 + 350000 items demarshaled in 0.07 sec.
merging intbitsets ..... 546965 items retained in 0.00 sec.
intersecting intbitsets ..... 153035 items retained in 0.00 sec.
```

As can be seen `IntBitSets` requires almost no time for set operation, and almost 100 times less time for marshaling than Numeric Python arrays, by using half space of Numeric Arrays.

Plugging `IntBitSet` to the current `BibIndex` engine helped in reducing up to ten times the space used to store indexes, and reduced from twenty times and above (according to initial index size) the time to create from scratch the indexes.

## 4.8.3 `IntBitSet` testsuite

Given the mature API, implementing a self-contained data structure, realising a testsuite for the `IntBitSet` Python extension has been a straightforward procedure. We have initially tested mostly operations used in `Invenio`, proofing the code. Then we have extended the test suite to those set operations not needed by `CDS Invenio` such as the symmetric difference. All the API methods have been tested iterating over typical sets (1 element, many elements up to 64, many elements above 64), testing all the possible combination.

In Figure 4.6 an example of the results obtained by running the unit-tests is presented. As can be seen the implemented testsuite is broad and working (in this particular example it has even found an error). The testsuite helped a lot in finding many small bugs during developing. Moreover, dealing with a C extension, the code is exposed to hard-to-spot memory-handling bugs. Thanks to this testsuite and to the powerful **Valgrind** tool (see [NS03]), all the found bugs have been fixed, by running the testsuite in the virtual machine generated by `Valgrind`.

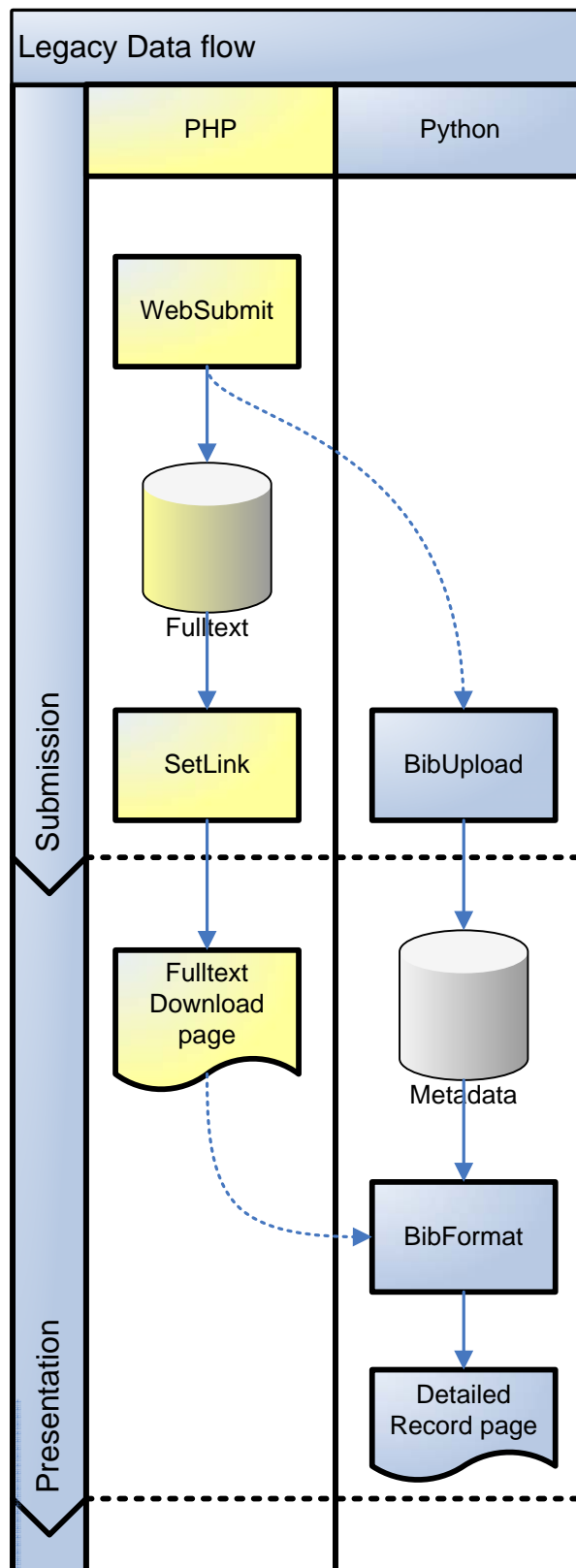


Figure 4.2: The current Data Flow at CERN

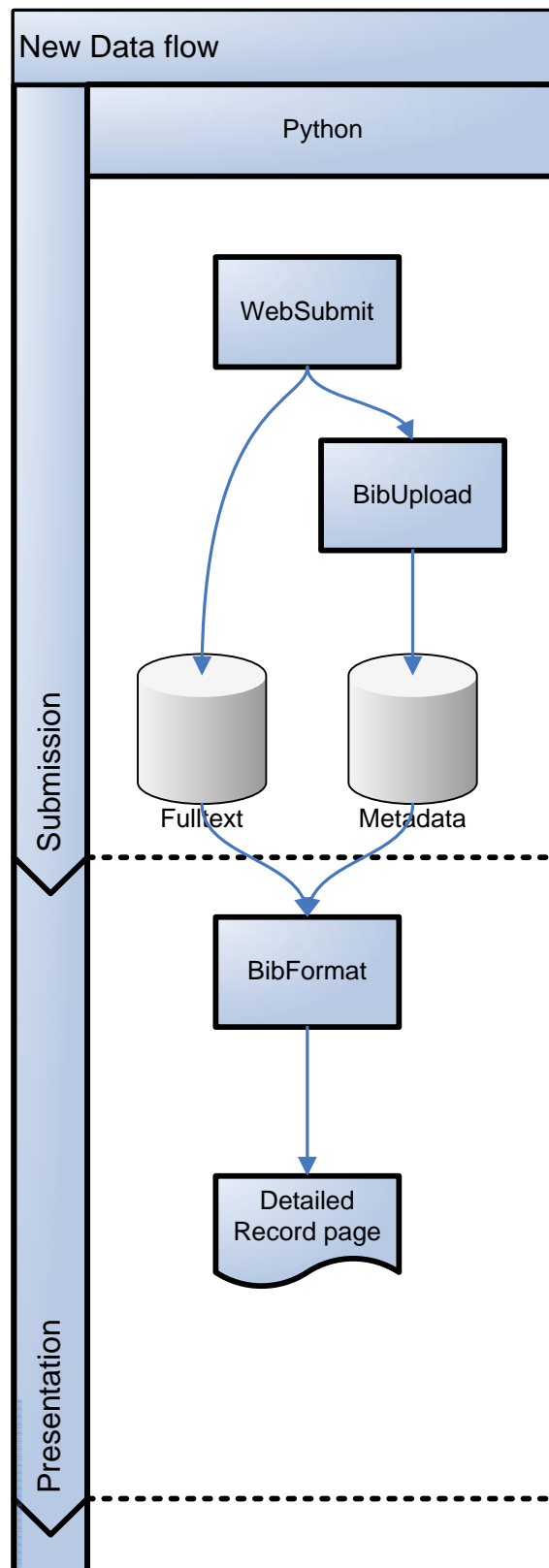


Figure 4.3: The Data Flow of a clean installation

```
intbitset - ascii bit dump ... ok
intbitset - list dump ... ok
intbitset - marshalling ... ok
intbitset - clearing ... ok
intbitset - set cloning ... ok
intbitset - (non infinite) set comparison ... ok
intbitset - set difference, normal set ... ok
intbitset - set difference, empty set ... ok
intbitset - set difference, empty set in place ... ok
intbitset - set difference, normal set in place ... ok
intbitset - set difference, infinite set ... ok
intbitset - set difference, infinite vs empty ... ok
intbitset - set difference, infinite vs empty in place ... ok
intbitset - set difference, infinite set in place ... ok
intbitset - set intersection, normal set ... ok
intbitset - set intersection, empty set ... ok
intbitset - set intersection, empty set in place ... ok
intbitset - set intersection, normal set in place ... ok
intbitset - set intersection, infinite set ... ok
intbitset - set intersection, infinite vs empty ... ok
intbitset - set intersection, infinite vs empty in place ... ok
intbitset - set intersection, infinite set in place ... ok
intbitset - Pythonic representation ... ok
intbitset - set symmetric difference, normal set ... ok
intbitset - set symmetric difference, empty set ... ok
intbitset - set symmetric difference, empty set in place ... ok
intbitset - set symmetric difference, normal set in place ... ok
intbitset - set symmetric difference, infinite set ... ok
intbitset - set symmetric difference, infinite vs empty ... ok
intbitset - set symmetric difference, infinite vs empty in place ... ok
intbitset - set symmetric difference, infinite set in place ... FAIL
intbitset - set union, normal set ... ok
intbitset - set union, empty set ... ok
intbitset - set union, empty set in place ... ok
intbitset - set union, normal set in placeusage to query for the name ... ok
intbitset - set union, infinite set ... ok
intbitset - set union, infinite vs empty ... ok
intbitset - set union, infinite vs empty in place ... ok
intbitset - set union, infinite set in place ... ok
intbitset - set update with signs ... ok
```

Figure 4.6: An example of running the IntBitSet testsuite. In this run, a bug has been found.

# Appendix A

## Mail cookies

“Mail cookies” is a new CDS Invenio feature developed by the author to facilitate and at the same time enhance some activities that involves users.

At the time of this writing an implementation for three different usage is ready:

- a password resetting facility, to let the user who have forgotten the password to set a new one (assuming an correct email has been provided to the system)
- an email checking facility, to let the system be sure that a given email is really owned by the user who has configured this email.
- a temporary role membership invitation system.

By now, only the first implementation has a corresponding web-interface that take advantage of it.

A mail cookie is in fact a short piece of information (i.e. a stream of at least 32 hexadecimal digits) sent via email in the form of a prebuilt url that, once followed by the user by opening it in a browser, it pass the cookie to the proper handling function that will use the given information, once proven valid, to unlock a row in the database that will correspond to the proper usage.

The hexadecimal digits are infact an encoded structure made up by the password that will unlock the given row, and the row id, in the cookies table.

### A.1 The implementation

The cookies table is a simple table with the following fields:

**data** where the encrypted data will be stored;

**expiration** the expiration time after which the cookie will be considered non-valid. This is also used by the cookie garbage-collector algorithm to periodically drop expired cookies;

**kind** to indicate what kind of handler is required to handle the encrypted data.

**onetime** a boolean flag to indicate whether the url provided by the cookie can be followed only once.

The data are encrypted via the password contained in the cookie. The data contains in turn a serialized python tuple which is constituted by the same element of the table (obviously excluding the data field itself), to check for correct decryption, plus all the parameters to be passed to the handling function corresponding to request service.

Since the password is put in the cookie itself, there's no need to store it elsewhere. Only the receiving user will know it and will be able to use it. That means that administrators are not able to know what is stored inside a cookie row, so for example, *role invitation could remain completely anonymous!*

## A.2 Developer interface

A generic function to generate a cookie has been provided. Given the necessary parameters it returns the cookie that corresponds to the just created cookie row.

Symmetrically, a function to check for a cookie and to unlock its content is provided.

Wrapping this pair of function there are three pair of other functions to implement each of the above enumerated services.

## A.3 Cookie content issues

Special care was used to decide on how the cookie had to be shaped. It had to store both the row where data to be unlocked are, and the password to unlock them. To accomplish this in an elegant way, we have finally decided to provide the user a string of anonymous hexadecimal digits. Looking at them nothing can be guessed about the content. Indeed they contain exactly everything that is needed. The password is a run-time random generated number, encoded in hexadecimal using the MD5 famous algorithm, to obtain a 32 bytes string. It is then broken in two and the hexadecimal encode of the row id is put between the two part, just to hide a bit the meaning of the whole string (if the row-id would have been put at the end of the string, the following generate cookie would have had the same last hexadecimal digits plus one).

# Appendix B

## Minor activities

Being a fulltime developer member of a team which not only develop and maintain an OpenSource project, but also offers a community service to the CERN community, there are several small works that need to be accomplished on a per-request basis. The author tapped into the TODO list of the CDS Invenio software to look for tasks still to be implemented.

### B.1 BibSched log viewer

Looking at how the CDS Invenio scheduler was implemented the author decided to implement the BibTasks Log Viewer feature of BibSched, CDS Invenio proprietary scheduler.

This feature was missing, and the administrator had to manually browse the filesystem looking through log files named after the corresponding BibTask number, for information and debugging about a given task.

The author wrote a curses-oriented file viewer that is activated upon a certain key press from the BibSched editor. (for more information about NCurses, please see [Goo07]). The feature is configurable in order to support the preferred Unix pager (e.g. *less*, or *more* or *most...*).

In Figure B.1 on the next page can be seen on the left a typical BibSched screen and on the right the current result of pressing L, thus displaying the log of the highlighted task, as it has been implemented by the author.

### B.2 BibSched job pruner

The author worked in implementing an other wished feature, namely the “P” key to purge from the list of BibTasks those in the *Done* status.

ID	PROC	USER	RUNTIME	SLEEP	STATUS	PROG
12	bibrank	admin	2007-09-22 21:55:42	1h	DONE	None
11	webcoll	admin	2007-09-22 21:55:31	1h	DONE	Part
9	bibreformat	admin	2007-09-22 21:55:21	1h	DONE	Petc
7	bibindex	admin	2007-09-22 21:55:10	1h	DONE	None
6	bibrank	admin	2007-08-20 09:47:14		DONE	rnk
5	webcoll	admin	2007-08-20 09:46:43		DONE	Part
4	bibupload	admin	2007-08-20 09:46:11		DONE	None
3	bibreformat	admin	2007-08-20 09:46:06		DONE	Form
2	bibindex	admin	2007-08-20 09:45:02		DONE	idx
1	bibupload	admin	2007-08-20 09:44:50		DONE	None

```

2007-06-05 11:39:26 -> Task #12 started.
2007-06-05 11:39:26 -> idxWORD03F contains 2257 words from 46 records
2007-06-05 11:39:26 -> idxWORD03F is in consistent state
2007-06-05 11:39:26 -> No new records added. idxWORD03F is up to date
2007-06-05 11:39:26 -> idxWORD03F contains 2257 words from 46 records
2007-06-05 11:39:26 -> idxWORD03F is in consistent state
2007-06-05 11:39:26 -> idxWORD04F contains 621 words from 70 records
2007-06-05 11:39:26 -> idxWORD04F is in consistent state
2007-06-05 11:39:26 -> No new records added. idxWORD04F is up to date
2007-06-05 11:39:26 -> idxWORD04F contains 621 words from 70 records
2007-06-05 11:39:26 -> idxWORD04F is in consistent state
2007-06-05 11:39:26 -> idxWORD02F contains 7 words from 76 records
2007-06-05 11:39:26 -> idxWORD02F is in consistent state
2007-06-05 11:39:26 -> No new records added. idxWORD02F is up to date
2007-06-05 11:39:26 -> idxWORD02F contains 7 words from 76 records
2007-06-05 11:39:26 -> idxWORD02F is in consistent state
2007-06-05 11:39:26 -> idxWORD09F contains 31223 words from 35 records
2007-06-05 11:39:26 -> idxWORD09F is in consistent state
2007-06-05 11:39:26 -> No new records added. idxWORD09F is up to date
2007-06-05 11:39:26 -> idxWORD09F contains 31223 words from 35 records
2007-06-05 11:39:26 -> idxWORD09F is in consistent state
2007-06-05 11:39:26 -> idxWORD01F contains 4136 words from 76 records
2007-06-05 11:39:26 -> idxWORD01F is in consistent state
2007-06-05 11:39:26 -> No new records added. idxWORD01F is up to date
More--(Zz)

```

Figure B.1: A typical BibSched screen (left), and the result of pressing the “L” key on the highlighted BibTask, thus opening its log (right).

## B.3 Refactoring of BibTasks

**BibSched.** The above small projects helped the author in discovering the scheduler of CDS Invenio. BibSched is a Python executable that can run as a Unix daemon and has also an Ncurses frontend to monitor running Bibliographic Tasks. The BibSched scheduler is implemented as a simple serial queue, to serialize the execution of the different BibTasks that form a CDS Invenio running environment. At any moment a monitored BibTask can be only in a given states, namely, waiting, running, done, error. Moreover BibTasks can be scheduled for repetition or run just only once.

Browsing the different CDS Invenio available BibTask code, the author noticed the behavior of copying and pasting the source code. Every BibTasks had been developed by forking from an already existing BibTask. Thus the author took care of merging all the common BibTasks code (mainly command line parameters handling, interaction with the BibSched scheduler and Unix signals handling) and to put everything under a new common class named **BibTask**. Every BibTask code had then been rewritten in order to derive from this very class. Moreover all the handling of command-line parameters had been centralized and standardized. This in turn made writing new BibTasks a very straightforward job. As a consequence the author worked in converting a plain CDS Invenio command-line tool into a new BibTask.

## B.4 BibSched and Invenio Garbage Collector

**SessionGC**, the CDS Invenio garbage collector, had been a plain CDS Invenio command-line tool. Its job is to clean expired user sessions and more in general to clean all expired or no more referenced rows in CDS Invenio database table. Documentation suggested the user to run this tool regularly or to run it through the Unix CRON job system. The author translated the tool, as a proof-of-concept of the newly refactored BibSched/BibTask system into a BibTask, thus empowering the tool of all the other BibTasks features. The



SessionGC tool can now be scheduled to be run within a given time frame, but, more important, its usage is scheduled by BibSched alongside other BibTasks, thus allowing for the correct serialization of SessionGC accesses to the database (and the filesystem).



# Conclusions

The thesis reports on the work carried out by Samuele Kaplun at the Department of Information Technology of the European Organization for Nuclear Research (CERN) under the CERN Technical Studentship Programme.

The author worked as a fulltime developer in the CERN Document Server team on its CDS Invenio Software package. Two main projects were presented, namely “authentication and authorization enhancements” and “fulltext document migration from the legacy CDS Invenio installation at CERN to the new system”.

The first project on the CDS authentication system presented the study of the system before the enhancements which were the subject of the present work, designing and then implementing a plugin architecture enhancements that support multiple external authentication systems, with feature like importing external user details and external user group membership information. The project has culminated in (i) a plugin architecture that fully exploited the CERN central authentication system by wrapping the provided SOAP webservice, and (ii) in a customization to fully support the Single Sign-On technology deployed at CERN through interfacing with the Shibboleth software on the Apache web servers, and (iii) in an externally developed plugin to exploit LDAP authentication.

As a side extension to the first project the author brought enhancements from the authentication system to the RBAC authorization architecture. A new (iv) firewall-like role definition language, called FireRole, has been designed and plugged to the RBAC system, in order to implicitly connect users to roles by exploiting user information such as their external details and group, thus helping the administration of CDS Invenio installation to better integrate the system to the surrounding social and institutional context.

Other minor tasks and developments have been undertaken by the author in order to solve real-life issue occurred over time in service like CDS Invenio, in the framework of authentication and authorization subjects, namely (v) the encryption of local password, (vi) the integration of command line tools with the external authentication plugin system and (vii) a realization of a library to support an authorization request and user invitation by means of small piece of cookie information sent by email.

The second project concerned the legacy fulltext file server production architecture at CERN, and the design and development of an automated migration tool, called “s2d” to move more than 500,000 fulltext documents (mainly scientific papers) from the legacy

system to the current CDS Invenio architecture. The tool had to retrieve documents from a legacy machine, parsing HTML pages in order to extract files, move them carefully, rebuild the metadata information by discovering multiple formats and revisions of the same document by means of heuristics. The tool also imported download statistics from Apache log files from the legacy system to the new database tables, e.g. to be able to rank search results by download statistics. The migration tool was written with reliability in mind, since losing scientific publications during document server migration cannot be naturally afforded. The tool has been successfully used to migrate (viii) the whole set of more than 400,000 Article and Preprints hosted by the CERN Document Server.

Having a huge subset of the fulltext document files locally referentiable by the CDS Invenio infrastructure rendered fulltext indexing feasible at CERN. However it emerged necessary to improve performance of the indexing engine. The author has worked on implementing (ix) a new fast and slim bit vector data container in the form of a Python C extension, to optimize speed and memory usage. By putting this extension on production, tables have typically shrunk up to ten times, and indexing speed has consequently improved typically twenty or even more times depending on the original index size.

Finally, a number of smaller tasks and improvements to CDS Invenio have been presented, namely (x) enhancements to the bibliographic task scheduler system, the (xi) garbage collector and the (xii) stemming library, carried out in parallel to the main project as a result of “real-life” necessities of a service-oriented system providing document services to an important and active scientific user community.

# Bibliography

- [Ben95] Ben Segal. A Short History of Internet Protocols at CERN. 1995.
- [Bet00] Betty Furrie. *Understanding MARC Bibliographic: Machine-Readable Cataloging*. Follett Software Company, 6 edition, 2000.
- [BLCGP92] Timothy J Berners-Lee, Robert Cailliau, J F Groff, and B Pollermann. World-wide webthe information universe. *Electron. Netw.*, 2:52–58, 1992.
- [BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern information retrieval*. ACM Press Books. Addison-Wesley, Reading, MA, 1999.
- [cds07a] Cds invenio demo, September 2007.
- [cds07b] Cds invenio features, September 2007.
- [CDS07c] CDS Software Consortium. Why to MARC at all?, 14 February 2007.
- [CER04] To the LHC and beyond, September 2004.
- [Com07] Communication Group. All about CERN... in 7 questions!, March 2007.
- [Dia05] Diane Hillmann. Using Dublin Core, 7 November 2005.
- [DR02] J. Daemen and V. Rijmen. *The Design of Rijndael: AES—the Advanced Encryption Standard*. Springer, 2002.
- [EC02] Marlena Erdos and Scott Cantor. Shibboleth-architecture draft v05. Technical report, “Internet2”, 2002.
- [Ewi] G. Ewing. Pyrex. A Language for Writing Python Extension Modules. *URL* <http://www.cosc.canterbury.ac.nz/greg/python/Pyrex>.
- [FK89] D.C. Feldmeier and P.R. Karn. UNIX Password Security-Ten Years Later. *CRYPTO Proceedings*, 1989.
- [FKC03] David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control*. Artech House, Norwood, Massachusetts, April 2003.

- [Fra92] WB Frakes. Stemming algorithms. *Information retrieval: data structures and algorithms table of contents*, pages 131–160, 1992.
- [Fri02] J.E.F. Friedl. *Mastering Regular Expressions*. O’Reilly, 2002.
- [Goo07] D. Gookin. *Programmer’s Guide to ncurses*. John Wiley & Sons, Inc. New York, NY, USA, 2007.
- [HKM<sup>+</sup>88] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):51–81, 1988.
- [HUC88] J.H. Howard, Carnegie-Mellon University, and Information Technology Center. *An Overview of the Andrew File System*. Carnegie Mellon University, Information Technology Center, 1988.
- [(ID06] International DOI Foundation (IDF). *The DOI? Handbook*, October 2006. Version 4.4.1.
- [LMM00] Jean Yves Le Meur and D McGlashan. Setlink the cern document server link manager. *High Energy Phys. Libr. Webzine*, 1(CERN-ETT-2000-001):1. 8 p, Jan 2000.
- [MA02] Alex Martelli and David Ascher, editors. *Python Cookbook*, chapter 17.7. O’Reilly Cookbook. O’Reilly, July 2002. Recipes from the Python Community.
- [MAR] MARC Standards.
- [MR91] K. McCloghrie and MT Rose. RFC1213: Management Information Base for Network Management of TCP/IP-based internets: MIB-II. *Internet RFCs*, 1991.
- [NS03] N. Nethercote and J. Seward. Valgrind: A program supervision framework. *Electronic Notes in Theoretical Computer Science*, 89(2):1–23, 2003.
- [PBG<sup>+</sup>05] Alberto Pepe, Thomas Baron, Maja Gracco, Jean Yves Le Meur, Nicholas Robinson, Tibor Simko, and Martin Vesely. Cern document server software: the integrated digital library. (CERN-OPEN-2005-018):6 p, Apr 2005. revised version submitted on 2006-02-01 17:03:23.
- [Por] M. Porter. The Porter Stemming Algorithm. *Accessible at <http://www.tartarus.org/martin/PorterStemmer>*.
- [Por06] M. Porter. Snowball: A language for stemming algorithms. *URL <http://snowball.tartarus.org/texts/introduction.html>. Visited May 4th, 2006*.

- 
- [Sal04] D. Salomon. *Data compression*. Springer New York, 2004.
- [San99] M.F. Sanner. Python: a programming language for software integration and development. *J. Mol. Graphics Mod*, 17:57–61, 1999.
- [SWY75] G. Salton, A. Wong, and CS Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.
- [VBLMS02] M Vesely, T Baron, Jean Yves Le Meur, and Tibor Simko. Creating open digital library using xmlimplementation of oai-pmh protocol at cern. (CERN-ETT-2002-003):7 p, Jul 2002.
- [Yer98] F. Yergeau. RFC2279: UTF-8, a transformation format of ISO 10646. *Internet RFCs*, 1998.

# Index

- acc\_authorize\_action, 36
- access\_control\_engine, 36
- Account,
  - Local, 16
- Apache
  - Groups, 36
  - Passwords, 36
- Authentication, 15
- Authentication system, 16
  - External,, 16
- bibdoc, 53
- bibrec, 53
- BibSched, 70
- BibTask
  - SessionGC, 70
- BibTasks, 70
- bit-vector, 62
- Caching tools, 43
- CDS
  - Consortium, 4
  - Invenio, 4
- CDS Submit, 50
- CERN, 3
  - NICE authentication, 19
  - SOAP Webservice, 18
- Data
  - Acquisition, 6
  - Output, 6
- DocId, 53
- Document
  - Taxonomy, 5
- Dual log-in, 16
- eGroups, 15
- Features,
  - Collaborative, 7
  - Personalization, 7
- FireRole, 35
  - Data flows, 41
  - Grammar, 40
  - Language, 37
  - Semantic, 40
  - Syntax, 40
- IntBitSet, 62
- Invenio, 4
- MARC21, 6
- MARCXML, 6
- Memoization, 43
- Metadata, 51
- Modules
  - Overview, 9
- MySQL, 60
- NCurses, 69
- Nickname, 17
- OAI, 8
- OAI-PMH, 8
- PHP, 49
- Pyrex, 62
- RBAC, 35
  - Actions, 35
  - Authorizations, 35
  - core workflow, 36
  - Roles, 35
- RecId, 53
- run\_sql, 43



- run\_sql\_cached, 43
- s2d, 49
- Search
  - Engine, 5
- SessionGC, 70
- set data structure, 62
- SetLink, 49
- Unix pager, 69
- User
  - Classes, 9
  - Details, 38
  - Email, 17
  - External settings, 39
  - Id, 17
  - Identification, 17
  - settings, 17
- User,
  - Authenticated, 15
  - Author, 9
  - Curator, 9
  - Librarian, 9
  - Regular, 9
  - System Administrator, 9
- UTF-8, 60
- WebAccess, 36
- ZIP algorithm, 62



# Acknowledgements

I would like to thank my supervisors in Italy, Prof. Gianpiero Cattaneo, Dr. Davide Ciucci and Dr. Fabio Farina, that followed me from the other side of the cable and guided me in writing this thesis.

I would like to thank Jean-Yves Le Meur and Tibor Simko for having welcomed me into the CDS Invenio gang here at CERN, and having guided me through. I've spent a great year here, you taught me a lot of stuff that matter! It has been a pleasure to work in your team! That's why I want also to thank each member I've been officemate of: Diane, Nick, Gabriel, Greg, Jerome, Ian, Marcus, Zibi, it has been great to work with you.

I'll try to thank again Prof. Luca Bernardinello (this time seeking to spell his name correctly!), for tolerating my computing "spetegüles" each time I came back to Italy and similarly I would like to thank my "computing 4<sup>th</sup>-floor clan", namely Scarch, Zia Alice, Trippo e Spiga, for always welcoming back and following me in this CERN adventure.

Thanks to Gabri and DvD for being great friends and perfect flatmates all this long year. Special mention to "Frappe", the landlord, for having welcomed me and my flatmates in the beautiful countryside of Thoiry, the perfect place to live when you need real relax while preparing exams or writing the thesis every weekend.

Thanks to Franco for the very big CERN binges with great Italian food, and all his rides!

I would like to thank my very family for, well, being so a *family*, namely, mamma, papà, Noe, Micka, Pivo, Marco, Monica, Martino, Stefano, Sophie, Rachele, Emanuele, Raffaele... Thank you for all the great enthusiasm you gave me! Thanks to the Storaci family, too, for having me welcomed every Italy return, with great meal and evenings!

Going back to the origin I have to thank Edo, just-daddy, my computing godfather, for having lent me his computing books and for my first great cellar computing sessions.

Especially, I would like to thank my fiancée, Bà, who has been so crazy and strong to carry this lazy computing guy in her stressed physicists world, always encouraging him to do his best, thus letting him discover a new broad fascinating world out of the cold cellar!