



SEVENTH FRAMEWORK PROGRAMME
FP7-ICT-2009-6

BlogForever
Grant agreement n°. 269963

D4.7: Final Weblog Digital Repository Component

Editor:	J. García Llopis, R. Jiménez Encinar
Revision:	First Version
Dissemination Level:	Public
Author(s):	J. García Llopis, R. Jiménez Encinar, Ş. Postacı, A. Çınar, H. Kalb, T. Šimko,
Due date of deliverable:	May 31, 2013
Actual submission date:	June 2, 2013
Start date of the project:	March 01, 2011
Duration:	30 months
Lead beneficiary name:	European Organization for Nuclear Research (CERN)

Abstract:

This report presents the implementation activities carried out for the development of the final BlogForever digital repository component. In this respect, it provides a complete description of the development performed according to the design done previously, served with a detailed list of implementation descriptions which explain in more detail what and how has been developed. Furthermore, this report outlines the adopted collaboration workflow together with the technologies utilised.

**Project co-funded by the European Commission within the Seventh
Framework Programme (2007-2013)**

The **BlogForever** Consortium consists of:

Aristotle University of Thessaloniki (AUTH)	Greece
European Organization for Nuclear Research (CERN)	Switzerland
University of Glasgow (UG)	UK
The University of Warwick (UW)	UK
University of London (UL)	UK
Technische Universitat Berlin (TUB)	Germany
Cyberwatcher	Norway
SRDC Yazilim Arastrirmave Gelistrirmeve Danismanlik Ticaret Limited Sirketi (SRDC)	Turkey
Tero Ltd (Tero)	Greece
Mokono GMBH	Germany
Phaistos SA (Phaistos)	Greece
Altec Software Development S.A. (Altec)	Greece

Revision History

Version	Description of Change	Author	Date
0.1	First partial draft	J. García Llopis, R. Jiménez Encinar	07/05/2013
0.2	First draft	J. García Llopis, R. Jiménez Encinar	24/05/2013
0.3	Updates for Chapter 2	J. García Llopis, R. Jiménez Encinar	28/05/2013
0.3	Updates for Chapter 1	Ş. Postacı, R. Jiménez Encinar	28/05/2013
0.4	Updates for Chapters 2 and 3	J. García Llopis, R. Jiménez Encinar	29/05/2013
0.5	Extra content for Chapter 3 and updates for Executive Summary and Introduction	J. García Llopis, R. Jiménez Encinar	29/05/2013
0.6	Updates in Chapter 3, conclusions and final review	J. García Llopis, R. Jiménez Encinar, T. Šimko	30/05/2013
0.7	Integration of implementation descriptions in Chapter 3	Ş. Postacı, A. Çınar	31/05/2013

Table of Contents

ExecutiveSummary	1
1 Introduction	2
1.1 Background	3
2 The Weblog Digital Repository Implementation	5
2.1 BlogForever and Invenio	5
2.2 Implementation	9
2.2.1 Overall architecture	9
2.2.2 Ingestion workflow	17
2.2.3 Metadata Structure	24
2.2.4 Scalability	27
2.2.5 Preservation features	28
2.2.6 User Interface	30
2.3 Development Cycle	36
2.4 Development Methodology	38
3 Implementation Descriptions	41
3.1 Features already in Invenio	41
3.2 List of implementation descriptions	42
3.2.1 Previously implemented features	43
3.2.2 Newly implemented features	44
3.2.3 Updated features	51
3.2.4 Features ported to the new interface	75

3.2.5	Postponed features	76
3.3	Features not retained	76
4	Conclusions	78
	References	79

List of Figures

1.1	BlogForever platform overview	2
2.1	Invenio repository overall architecture	6
2.2	BlogForever repository overall architecture	8
2.3	BlogForever bibformat elements	13
2.4	BibFormat template for a blog	14
2.5	BibFormat template for a blogpost	15
2.6	BibFormat template for a comment	16
2.7	MARC extraction	18
2.8	Cleaning HTML process	19
2.9	Record and submission ID addition	20
2.10	Attached files addition	21
2.11	Parent blog record ID addition	22
2.12	Parent blog visibility addition	23
2.13	Ingestion of original METS file in database	24
2.14	Shard manager components[1]	28
2.15	Shard manager ingestion workflow[1]	28
2.16	BlogForever main page at BF1	30
2.17	BlogForever blog record at BF3	30
2.18	Invenio main page	31
2.19	Invenio record	31
2.20	BlogForever main page at BF5	32
2.21	BlogForever search page at BF5	33
2.22	BlogForever blog record at BF5	34

2.23 BlogForever post record at BF5	35
2.24 BlogForever comment record at BF5	36
2.25 Development workflow	40

List of Tables

2.1	Blog record attributes - MARC tags mapping	25
2.2	Post and Page record shared attributes - MARC tags mapping	26
2.3	Post record extended attributes - MARC tags mapping	26
2.4	Page record extended attributes - MARC tags mapping	26
2.5	Comment record attributes - MARC tags mapping	27
3.1	Implementation Description: RF52	45
3.2	Implementation Description: RF65	47
3.3	Implementation Description: RF66	48
3.4	Implementation Description: RF72	51
3.5	Implementation Description: RF6	54
3.6	Implementation Description: RF1	54
3.7	Implementation Description: RF22	55
3.8	Implementation Description: RF48	57
3.9	Implementation Description: RF57-58-61	58
3.10	Implementation Description: RF64	59
3.11	Implementation Description: RF70	68
3.12	Implementation Description: RF71	73
3.13	Implementation Description: RF73	75

Executive Summary

This document presents the work conducted for the development of the final BlogForever Weblog Digital Repository Component since the initial prototype presented in *D4.5: Initial Weblog Digital Repository Prototype* [2].

The Weblog Digital Repository has been developed to implement aggregation, preservation, management and dissemination of blogs. Implementation activities of the digital repository component carried out within the scope of *Task 4.5: Implementation of the digital repository component* are described in Chapter 2 comparing the design depicted in *D4.4: Digital Repository Component Design* [3] and the final implementation. The methodology adopted for the development and assessment of the digital repository is also described in Chapter 2.

Chapter 3 lists the final implementation descriptions of the repository features. Conclusions are gathered in Chapter 4.

A demo of the latest weblog digital repository version is available at:
<https://blogforever.cern.ch>

Chapter 1

Introduction

The BlogForever project goal is to develop solutions for aggregating, preserving, managing and disseminating blogs. To achieve this goal, the BlogForever project aims to develop a software platform that enables real-time harvesting and preservation of blog entities to facilitate extensive search and exploration functionalities of the archived blogs.

The BlogForever platform consists of two main software components: the spider and the digital repository. The spider is responsible for crawling all the necessary blog data and characteristics designated for preservation while the repository is responsible for long term archiving, preservation and management of the blogs, as well as providing facilities for further analysis and reuse of the content.



Figure 1.1: BlogForever platform overview

This deliverable intends to describe the implementation process of the final digital repository component.

Finally note that, in the following text, the words *weblog* and *blog* will be interchangeably used to describe the same concept.

1.1 Background

The BlogForever Description of Work (DoW) describes the objectives of the digital repository component as “being responsible for weblog data preservation. The digital repository will ensure weblog proliferation, safeguard their integrity, authenticity and long-term accessibility over time, and allow for better sharing and re-using of contained knowledge” [4].

Developing such a comprehensive archiving system from scratch is rather difficult and time consuming, and therefore avoided since there are many open-source archiving solutions. In this respect, the archiving system selected as basis of the digital repository component was the Invenio¹ software suite developed at CERN², which is also one of the partners involved in the BlogForever project.

In order to define how Invenio ought to be extended and modified, a list of requirements gathered from DoW, a weblog survey and 26 semi-structured interviews, were presented in *D4.1: User Requirements and Platform Specifications* [5]. Based on these requirements, 89 repository features, to be built on top of Invenio, were defined in *D4.4: Digital Repository Component Design* [3] as part of the design of the repository, in order to develop a complete digital repository for blogs. These features follow the metadata structure and preservation recommendations previously given by WP2 (*Weblog Structure and Semantics* in *D2.2: Weblog Data Model* [6]) and WP3 (*The BlogForever Policies* in *D3.1: Preservation Strategy Report* [7]).

The main goal of *Task 4.5: Implementation of the digital repository component* is to modify, to extend and to customize the vanilla Invenio source code by implementing the set of 89 repository features defined, and hence, to fulfill the BlogForever repository requirements.

In order to accomplish this goal, an initial prototype of the repository was delivered in *D4.5: Initial weblog digital repository prototype* [2], being the final repository component presented in this deliverable, which extends the initial prototype with the implementation of not only new features, but also updates to existing ones.

The final digital repository component resulting of this deliverable will be integrated with the final spider component developed during *Task 4.3: Implementation of the weblog spider component* and presented in the deliverables *D4.3: Initial Weblog Spider Prototype* [8] and *D4.6: Final Weblog Spider Component* [9], in *Task 4.6: Integration and Standardization*.

Last but not least, implementation activities performed in WP4 are evaluated in WP5. To be more specific, implemented features are tested and validated during *Task 5.2: Implementation of the case studies* based on the 6 case studies designed in *D5.1: Design and Specification of Case Studies* [10]. Since the implementation phase adopts an agile approach, testing phase adopts as well principles of agile testing which require testing to be an integral part of the software development.

¹<https://invenio-software.org/>

²<http://www.cern.ch/>

Moreover, feedback retrieved from both, internal and external testers, is used to improve the initial prototype towards the final digital repository component.

Chapter 2

The Weblog Digital Repository Implementation

BlogForever deliverable D4.4[3] provides the specifications for the repository features, describing the functionality for each of them. These specifications are considered as points of reference during the implementation phase. The main aim of the implementation phase conducted during Task 4.5 is to develop these features according to their specifications to achieve a complete digital repository built on top of Invenio for blog preservation.

In order to accomplish this goal, an initial prototype of the repository was delivered in D4.5[2], being the final repository component presented in this deliverable.

In this Chapter, the importance of Invenio for the BlogForever project is presented, followed by the description of the implementation activities that have been carried out during the Task 4.5, comparing the results obtained with the design described in D4.4[3]. Also, the development life cycle as well as the methodology that has been followed to structure, plan, and control the process of developing are explained.

2.1 BlogForever and Invenio

Invenio was already introduced in D4.4[3] and D4.5[2]. It is the core of the BlogForever digital repository component. Rather than implementing a repository for blog preservation from scratch, which is quite time-consuming, the BlogForever project has enhanced Invenio's features and applied necessary modifications in order to answer blog needs and provide a comprehensive repository for blog preservation, management and dissemination.

In the deliverables D4.4 and D4.5, a diagram illustrating the relationship among the Invenio modules and how they are organized was presented. In this deliverable, we present an updated version of it in the Figure 2.1.

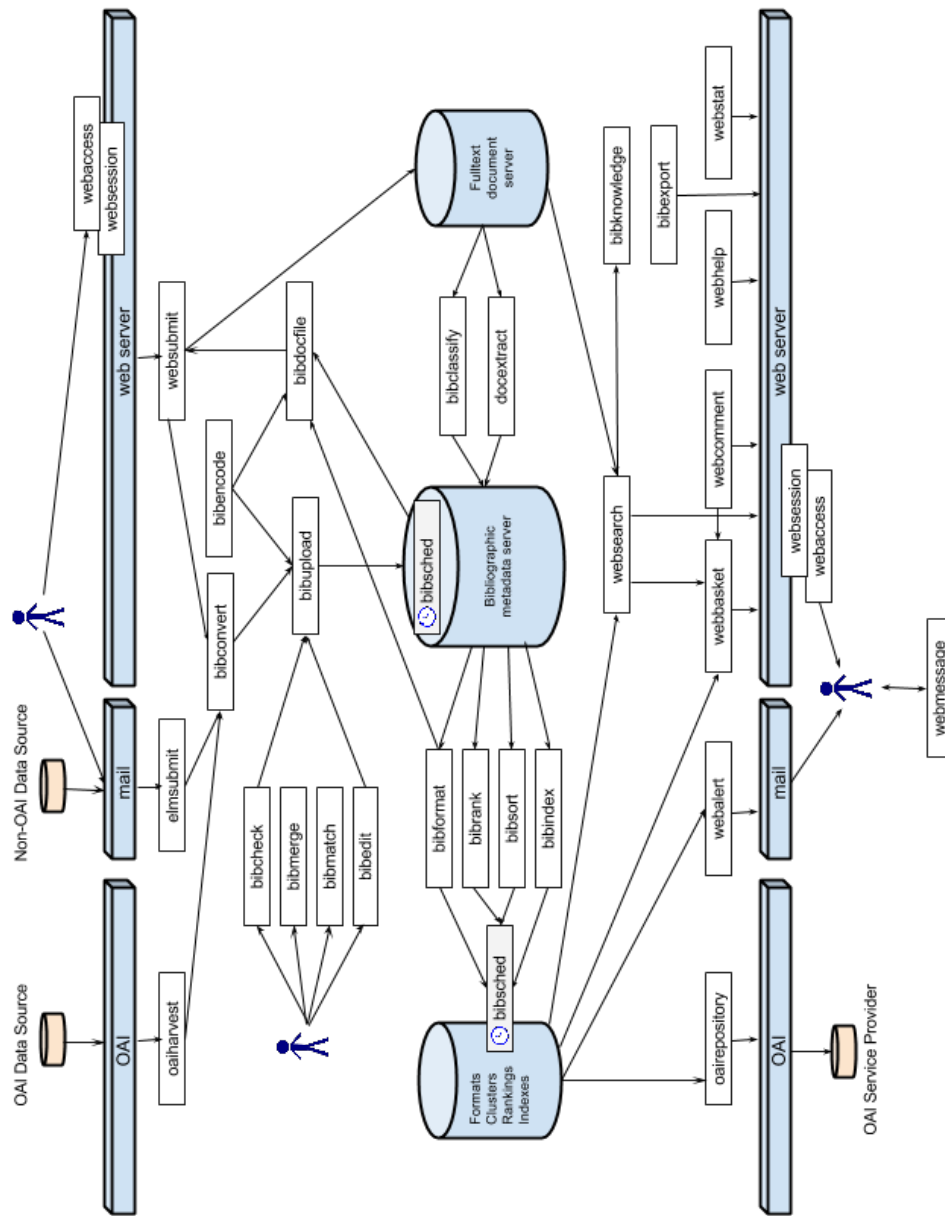


Figure 2.1: Invenio repository overall architecture

During the course of the project, Invenio reached the point where it was useful to adopt a new development framework based on new technologies in order to keep flexibility, manageability of growing number of modules, and speed-up prototyping and development. The BlogForever repository has been notably affected by this change since it was decided to follow the Invenio steps. This is explained in more detail in Section 2.3.

As the design document of the digital repository component D4.4[3] clearly states, 89 features were identified to fulfill the requirements for the final digital repository component. In general, these features describe how Invenio is extended and specify functionalities that enable an efficient storage for blog preservation and offer rich user interaction such as advanced weblog related information retrieval, managing a customizable dashboard and socializing by building a user community.

To be more specific, repository features denote the following extensions to Invenio:

- **Blog rendering** for representation of four main record types: Blog, Blog Post, Page and Comment
- **Blog metadata** to define blog specific properties and extend MARC¹ schema used in Invenio
- **Spider communication** for communication between the spider and the digital repository
- **BibIngest module** for ingestion of submitted material.
- **WebTag module** for enabling users to tag blogs
- **Spam filtering** for evaluation of the aggregated blog content
- **Social features** for dissemination of blog content in external social platforms and socialization of users in the platform
- New **export options** such as METS², PDF and JPEG
- **Billing system** for exploitation of added value services

The Figure 2.2 represents the overall architecture of the BlogForever repository built on top of the Invenio core. The extensions mentioned above are represented with green color. The description of the implementation activities that have been carried out in order to develop each of these new BlogForever specific elements, are described in depth in Section 2.2.

Implementation activities carried out within the scope of T4.5 and the methodology adopted for the development and evaluation of the digital repository component are explained in the following chapters.

¹<http://www.loc.gov/marc/>

²<http://www.loc.gov/standards/mets/>

2.2 Implementation

In this Chapter the design described in D4.4[3] is reviewed in comparison to the actual implementation of the repository. Each section in Chapter 4 of D4.4 will be discussed, explaining how the implementation matches the design or how and why the original design has been modified.

2.2.1 Overall architecture

The design of the overall architecture for the repository consists of taking Invenio as a starting point and add new modules and features on top of it. This design principle has been followed as can be seen in the following enumeration of the added modules and features. The mentioned feature specifications can be found at Chapter 5 of D4.4[3].

- **BibIngest module**

This module has been implemented according to the feature specifications *RF87-The archive transforms the databases for preservation purposes IPS received from the spider to AIPS* and *RF88-The archive stores the content of the AIPS in two different databases*. As stated in D4.4[3], this module is very important from the preservation point of view since this module is the responsible of storing the submitted material (ingestion packages) in its original format covering, at the same time, the requirements of the Open Archival Information System (OAIS) specification for the acquisition and storage of the Submission Information Package (SIP). In summary, this module holds the METS files submission and covers the needs of persistent storage of submitted material.

- **WebTag module**

This new module, still under development, is being implemented according to the feature specification *RF52-Users can tag archived records with personal tags*. This feature was defined in D4.4[3] as low priority feature (optional), reason why its development has been postponed until the rest of essential and conditional features have been implemented. The methodology followed to carry out the development and implement the different features will be explained in more detail in Section 2.4. So far, we have worked mostly in the back-end, creating 2 new tables in the database, (`tagTAG` and `tagTAGLOG`), as well as an API with functions to enable users to add new tags to blog records, to delete and to modify them and to get the tags set by an specific user (more details about this development will be given in Section 3.2).

- **Spam filtering**

This module, according to *RF41-The archive detects and eliminates spam content*, has been implemented using an external spam detection web service. The module populates a MARC tag indicating whether the record is considered spam or not.

It is not used before the ingestion of the content as mentioned in the design, but as a daemon. It runs periodically (configurable by the administrators) checking the records that do not have this MARC tag populated yet (presumably new records). It is implemented in a way that allows the ingestion process to use its functionality in case the checking needs to be performed before the ingestion, but it has been discarded because the webservice timeout could delay the process and make it too slow, and also because it is easier for the administrators to enable/disable it with the current daemon form.

- **Billing system**

According to the feature *RF70-The archive can provide services under some cost using a billing system*, a billing system has been implemented. While D4.4[3] mentioned disk space quota[3], the implementation does not offer this functionality. The amount of blog URLs sent by the users and the disk space they use will not be relevant for the repository. Instead, it is the feature specification where an accurate description of what has been implemented in the new `webpayment` functionality under the `WebAccess` module can be found.

This feature uses Invenio's roles and authorization system in conjunction with public/restricted collections to allow administrators to define restricted content that users can disclose by paying a fee. The cost and duration of the subscription can be configured by the administrators.

- **Spider-repository communication**

The spider-repository communication was designed in order to develop the communication between these two independent components in both directions, the repository fetches content from the spider (spider-repository) and the repository informs the spider about new submitted blogs and errors occurred during the content retrieval process (repository-spider).

The purpose of *T4.6:Integration and Standardization*, is to develop the first BlogForever prototype platform, ensuring interoperability and comprising the final weblog spider component and the final weblog digital repository component, fully functional and communicating optimally with each other.

At this stage, the communication in the spider-repository direction is fully working. For this purpose, a new tasklet called `bst_fetch_records_from_spider.py` has been implemented, which allows the repository to fetch content from the Spider by using its web service API methods. The steps that this tasklet follows to achieve its goal are:

- To establish a connection with the Spider using the API key provided, the corresponding web service url, and to set limits to the number of records downloaded.
- To download the list of Spider records crawled since the last execution of the tasklet.
- For each record, to download the METS file as well as all its attached files, and to store them in a temporary directory of the filesystem.
- For each record, to call `BibUpload`, which will finally process the record and store it into the repository.

- **Export options**

The BlogForever repository comes with new options to export records such as “Export to METS”, “Export to PDF” and “Export to JPEG”, as well as a new XML format used to create a network visualization of the content.

- **METS**: new output format used to export the metadata of a record or a group of records in METS XML. The corresponding METS of a record is retrieved from the mongoDB and displays it to the user. This export option has been implemented according to *RF59-Export data using XML (METS, MARC)*.
- **PDF and JPEG**: blog content can be downloaded in a human-friendly printable format, such as PDF and JPEG. This export option has been implemented according to *RF62-Export as PDF and JPEG*.
- **SRU**: it is a standard XML-focused search protocol for Internet search queries. Support for the SRU protocol has been added in the repository. This export option has been implemented according to *RF45-The archive is able to inter-operate with federated search engine dbwiz (SRU Server)*.
- **CMXXML**: this XML format has been implemented as a new output format that allows the repository to communicate with Commetrix³ and to create the network visualization requested by *RF65-The archive analyzes blog links and stores the connections between them separately*. The output XML can be used in other network-visualization tools. An example of this has been implemented in a Java application that is also delivered attached to the repository code. The demo-software *BF Network Generator* accepts both MARCXML and CMXXML and produces a preview of the network. This export option was not in the design, but has been added to the repository to profit from this software that provides the functionality needed.

- **Social features**

Some features classified as social features have been marked as “low priority” features, as explained in 3.3. This is the case of *RF50-The archive offers the option to disseminate newly archived content in external social platforms*.

RF3-“Share” option in “Your History” box lets the user send another user an activity he or she did. This functionality uses the WebMessage module to send the activities that the feature *RF2-“Your History” box as part of the user dashboard* has previously stored, like a search result, a record, a review, etc.

RF35-The archive displays other blogs that were viewed by people who also viewed the current blog suggests the user other records that may be similar to the record being displayed, according to other users’ previous readings.

These features enhance the existing Invenio social features like WebMessage that let users communicate with private messages; WebBasket that let users define group records creating their own collections and make them public for other users to subscribe; and WebComment that allows users to write reviews of the records and also rate them. This rating could be used to sort the search results.

³<http://www.commetrix.de/>

- **Blog rendering**

BibFormat is in charge of formatting the Invenio records that are displayed to users. It is called by the search engine when it has to format a record[11].

Different kind of formatting is needed depending on the type of record: blog, post, comment, page. In order to create the BlogForever related templates and elements, a new module called WebBlog has been created, which extends the bibformat Invenio templates and elements for this purpose.

Since the amount of records in the repository is potentially very high, the layout can not be specified for each of them. Instead, BibFormat uses a rule-based decision process to decide how to format a record. The workflow of BibFormat has 4 steps:

- **Step 1:** When Invenio has to display a record, it asks BibFormat to format the record with the given output format and language. The output format can be, for example, one of the XML formats explained above in the Export Options subsection, but typically it will be “HTML Detailed”. This means that somehow a user arrived on the page of the record and asked for a detailed view of the record.
- **Step 2:** The output format does not specify how to format the record, but contains a set of rules which define which format template must be used. In our case, this list of rules is:

```
tag 980.a:  
BLOG --- BlogHTML.tpl  
BLOGPOST --- PostHTML.tpl  
COMMENT --- CommentHTML.tpl  
PAGE --- PageHTML.tpl
```

The rules in the list are evaluated from top to bottom. Each rule defines a condition on a field of the record, and a format template to use to format the record if the condition matches. This is the output format for “HTML Detailed”.

If the field `980_a` of the record is equal to `BLOG`, then first rules matches, and the format template `BlogHTML.tpl` is used for formatting by BibFormat.

- **Step 3:** A format template is written using Jinja2 templates, which use at the same time some tags known as format elements, which are placeholders for the record values. These elements follow the naming convention of starting by `bfe_`. Different templates can share the same elements. The diagram in the Figure 2.3 shows the 3 main templates used for blog content and how they share the elements that have been created in order to define and customize the specific BlogForever templates.

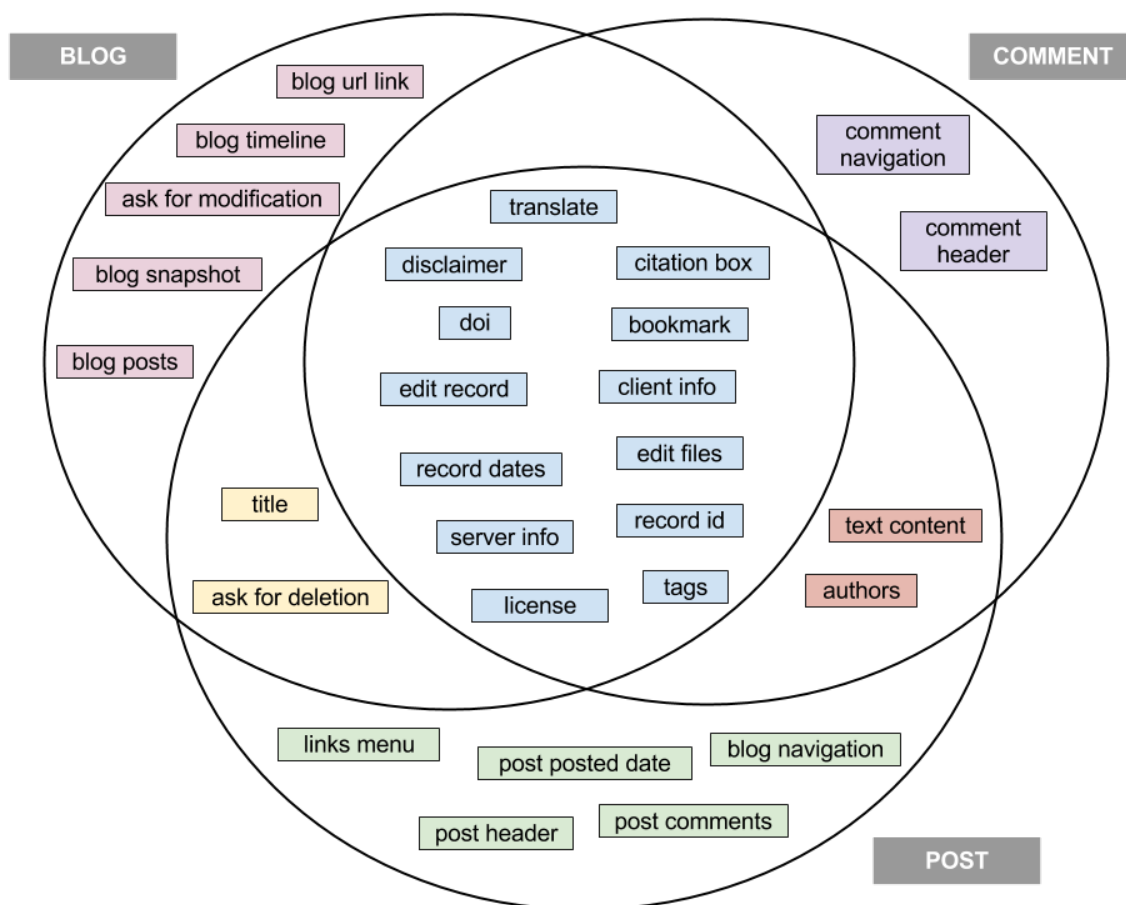


Figure 2.3: BlogForever bibformat elements

The elements that are common to the 3 main templates are explained below. Every element is presented together with the format element in which it has been implemented.

- * TRANSLATE(`bfe_translate.py`): displays the Google translator link which permits to the user translates the content of a record.
- * DISCLAIMER(`bfe_disclaimer.py`): returns the disclaimer notifying that what the user is seeing is just a copy of the original blog record.
- * RECORD_DATES(`bfe_record_dates.py`): displays the date in which the record was ingested into the repository. If the record is a `BLOGPOST`, this element also displays the date in which it was published.
- * LICENSE(`bfe_license.py`): displays the license name under the blog is licensed, as well as the url where more information about the license can be found, if available.
- * TAGS(`bfe_tags.py`): returns all the tags provided by the Spider for the corresponding record.
- * DOI(`bfe_doi.py`): returns an HTML link to the DOI.
- * CITATION_BOX(`bfe_citation_box.py`): displays the description of how users should cite any content of the archive.

- * EDIT_RECORD(`bfe_edit_record.py`): prints a link to BibEdit, if authorization is granted.
- * EDIT_FILES(`bfe_edit_files.py`): prints a link to simple file management interface (BibDocFile), if authorization is granted.
- * BOOKMARK(`bfe_bookmark.py`): returns a snippet of JavaScript needed for displaying a bookmark toolbar.
- * CLIENT_INFO(`bfe_client_info.py`): prints several client specific variables.
- * SERVER_INFO(`bfe_server_info.py`): prints several server specific variables.
- * RECORD_ID(`bfe_record_id.py`): prints the record identifier.

Figures 2.4, 2.5 and 2.6 show how the layout of a blog, blogpost and comment records respectively looks like. The screenshots have been annotated to highlight how the bibformat elements are used in the templates.

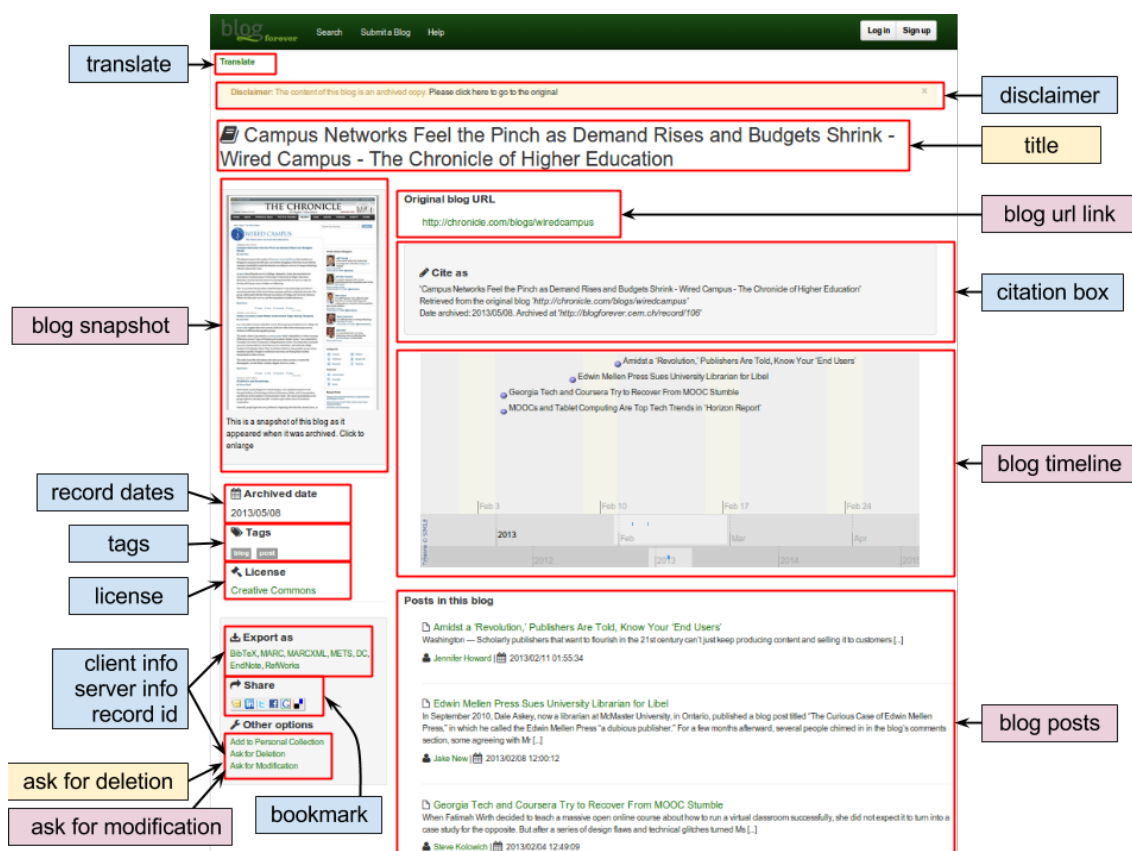


Figure 2.4: BibFormat template for a blog

The blog record specific elements are explained below. Every element is presented together with the format element in which it has been implemented.

- * TITLE(`bfe_title.py`): prints the title of the blog.
- * BLOG_URL_LINK(`bfe_blog_url_link.py`): displays the original blog link.

- * BLOG_SNAPSHOT(bfe_blog_snapshot.py): displays the snapshot of the blog.
- * BLOG_TIMELINE(bfe_blog_timeline.py): displays a timeline showing graphically all the posts of the corresponding blog in chronological order.
- * ASK_FOR_DELETION(bfe_ask_for_deletion.py): offers a link to delete the blog, redirecting the user directly to the blog submission interface.
- * ASK_FOR_MODIFICATION(bfe_ask_for_modification.py): offers a link to modify blog metadata, redirecting the user directly to the blog submission interface.
- * BLOG_POSTS(bfe_blog_posts.py): displays the posts of the blog sorted by posted date.

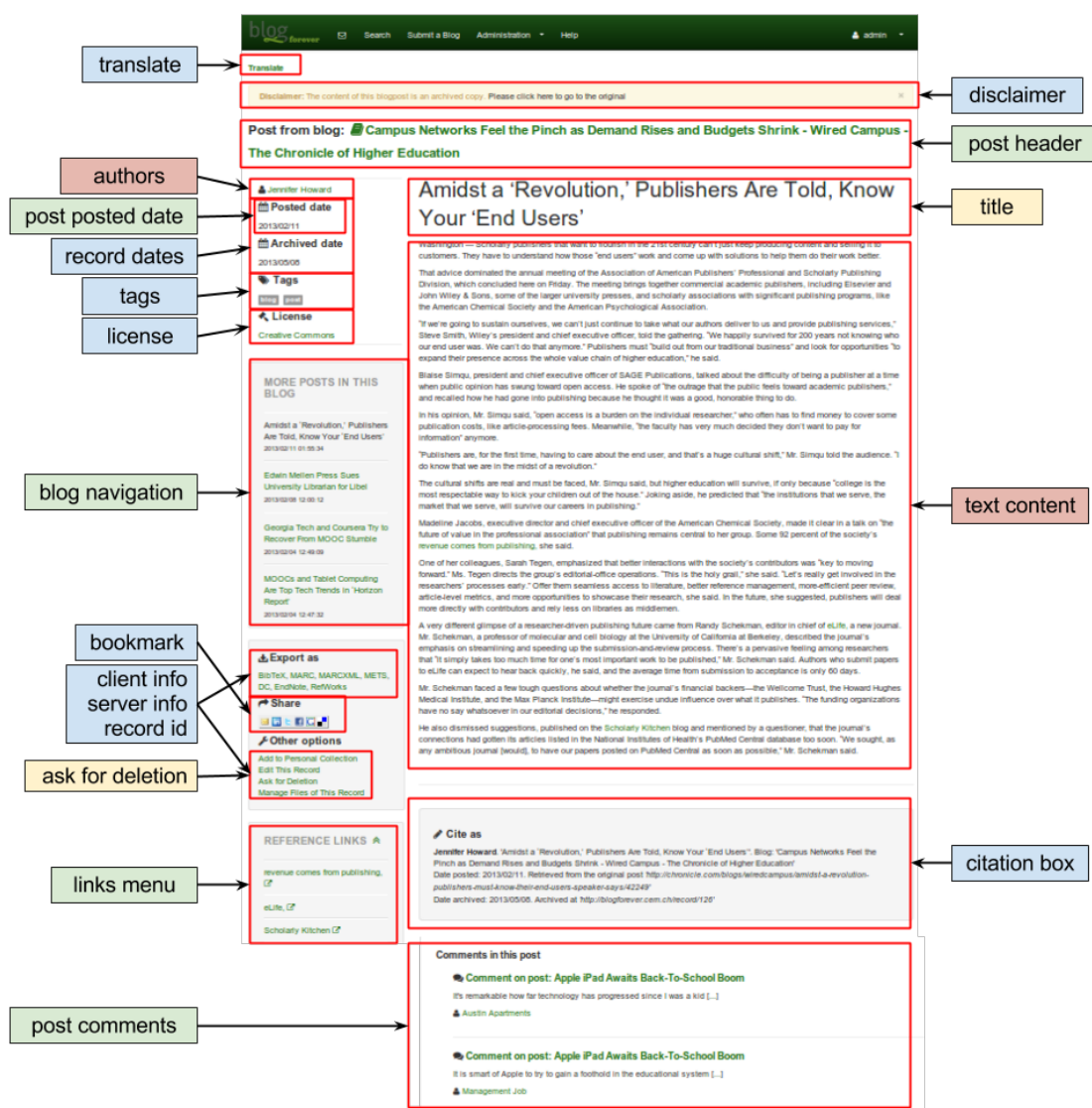


Figure 2.5: BibFormat template for a blogpost

The blogpost record specific elements are explained below. Every element is presented together with the format element in which it has been implemented.

- * `POST_HEADER(bfe_post_header.py)`: formats the header of the post showing the name of the blog it belongs to.
- * `TITLE(bfe_title.py)`: prints the title of the post.
- * `AUTHORS(bfe_authors.py)`: prints the author of the post.
- * `POST_POSTED_DATE(bfe_post_posted_date.py)`: formats and displays the date in which the post was posted.
- * `TEXT_CONTENT(bfe_text_content.py)`: prints the content of the post.
- * `BLOG_NAVIGATION(bfe_blog_navigation.py)`: creates a navigation menu showing all the other posts that belong to the same blog sorted by posted date.
- * `ASK_FOR_DELETION(bfe_ask_for_deletion.py)`: offers a link to delete the post, redirecting the user directly to the blog submission interface.
- * `LINKS_MENU(bfe_links_menu.py)`: returns a menu containing all the links used as references in the post.
- * `POST_COMMENTS(bfe_post_comments.py)`: displays the comments of the post.

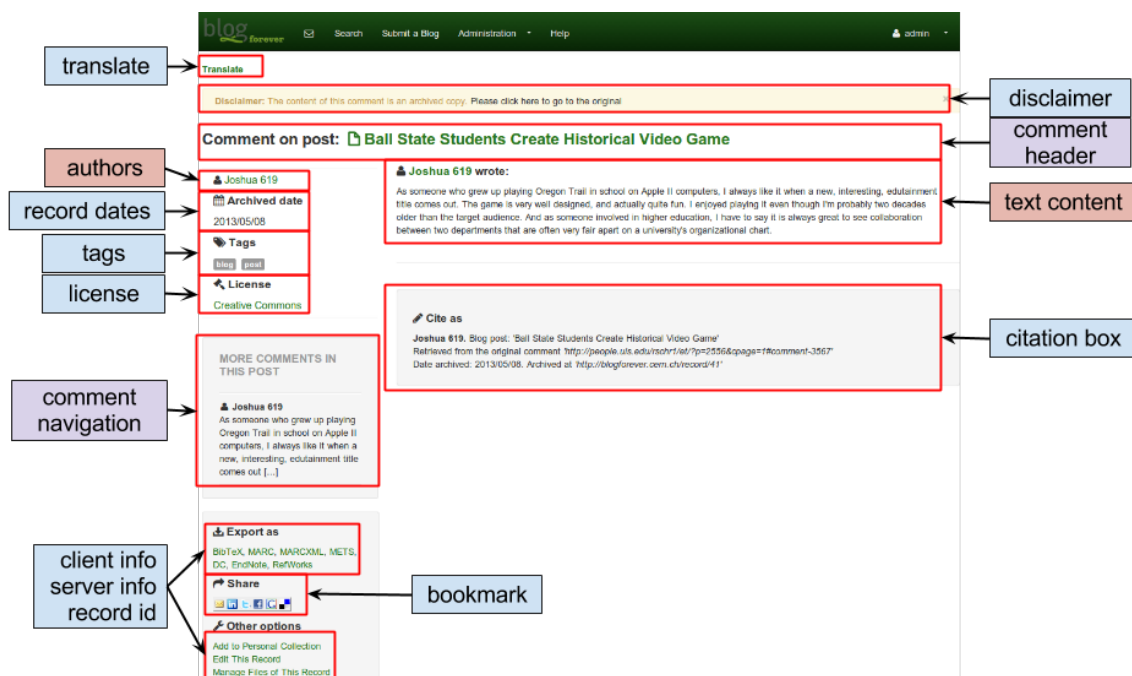


Figure 2.6: BibFormat template for a comment

The comment record specific elements are explained below. Every element is presented together with the format element in which it has been implemented.

- * `COMMENT_HEADER(bfe_comment_header.py)`: formats the header of the comment showing the name of the post it belongs to.
 - * `AUTHORS(bfe_authors.py)`: prints the author of the comment.
 - * `TEXT_CONTENT(bfe_text_content.py)`: prints the content of the comment.
 - * `COMMENT_NAVIGATION(bfe_comment_navigation.py)`: creates a navigation menu showing all the other comments that belong to the same post.
- **Step 4:** A format element is written in Python. It acts as a bridge between the record in the database and the format template. Each element outputs some text that is written in the template where it is called. Developers can add new elements by creating a new file, naming it with the name of element, and write a Python `format_element` function. Regular Python code can be used, including import of other modules.

In summary, BibFormat is called by specifying a record and an output format, which relies on different templates to do the formatting, and which themselves rely on different format elements.

2.2.2 Ingestion workflow

The ingestion is done in 2 phases. On a first phase, the content is fetched from the Spider component, while on the second phase it is inserted into the repository. This second phase has 3 steps: pre-processing, upload and post-processing.

The first phase consists of running the tasklet to fetch content from the Spider whose functionality was described above in Spider Communication subsection.

Regarding the second phase, the BibUpload task includes pre-processing and post-processing stages. The pre-processing prepares the record being inserted to be compatible with Invenio and enriches the metadata. This pre-processing has been implemented in the BibUpload plugin `bp_pre_ingestion.py`. The tasks performed at this stage are:

- Extracts the MARCXML from the METS received from the spider. This MARC will be used as the base to create an Invenio record in the repository. The original METS file will be later on attached to the record.



Figure 2.7: MARC extraction

- The HTML included in MARC as text content is cleaned for 2 reasons: to remove tags that might affect the repository rendering style and to scape the remaining tags so they do not affect the validity of the document when it is exported to MARCXML.



Figure 2.8: Cleaning HTML process

- The record ID and the submission ID are added in the corresponding MARC tags. First, the code checks whether the record already exists in the repository. In this case, the existing record ID will be added and the rest of the process will know that this is an update of an existing record. If it does not exist, a new record ID is created. The submission ID is the identifier used by the spider.



Figure 2.9: Record and submission ID addition

- The attached files are also included in the Invenio record. With this purpose, FFT⁴ tags are used to attach not just the files fetched from the spider (images, etc.) but also the original METS file, and the HTML and CSS files crawled from the original web page.

⁴<http://invenio-demo.cern.ch/help/admin/bibupload-admin-guide#3.6>



Figure 2.10: Attached files addition

- In the case of Posts and Comments, they have a blog parent record. The record ID of the parent blog is located and used to populate the corresponding MARC tag. This makes much simpler and quicker to perform usual tasks in the repository, like retrieving the list of all the Posts of a Blog.



Figure 2.11: Parent blog record ID addition

- At submission time, the user decides the visibility of the blog (public, restricted, private). It is at pre-ingestion time when Posts and Comments inherit the visibility of the parent blog.



Figure 2.12: Parent blog visibility addition

Once the record is ready, it is inserted into the repository databases by BibUpload. If this action ends in success, the post-processing plugin will be executed. It has been implemented in the BibUpload plugin `bp_post_ingestion.py`. The tasks performed at this stage are:

- Gets the original METS file as it comes from the Spider.
- Creates an XML tree from the METS file and extracts the record type (Blog, Post, Comment, Page).
- Identifies the Invenio record that corresponds to the METS file.
- The BibIngest module stores the original METS file in the submission database (mongoDB) using the record ID as identifier.

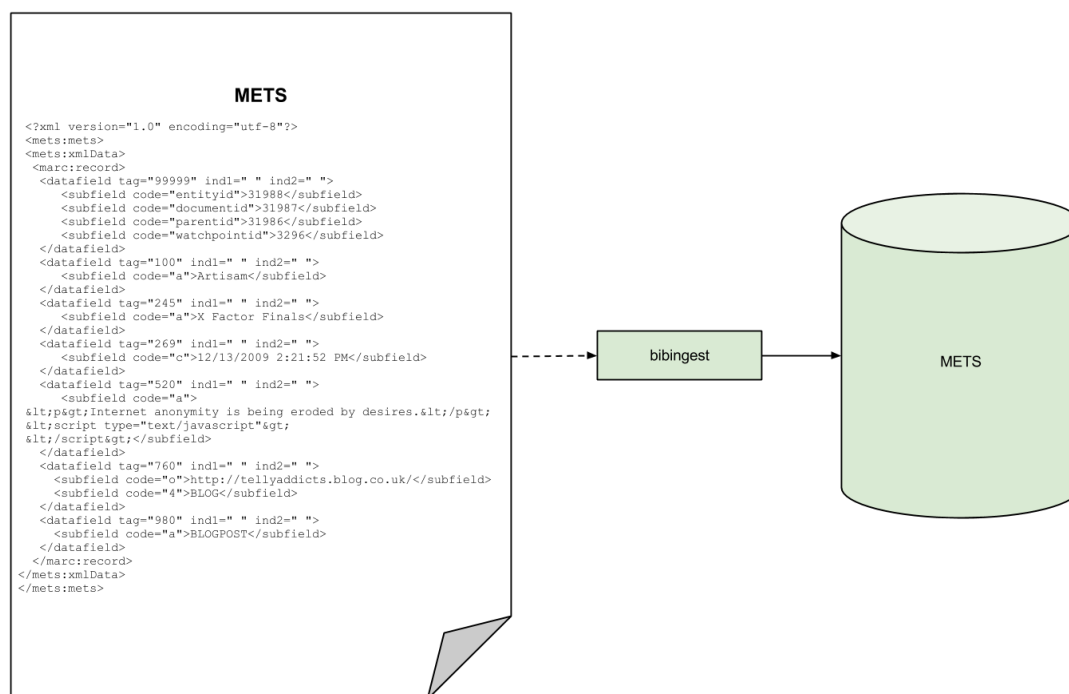


Figure 2.13: Ingestion of original METS file in database

More complete information about the communication with the Spider and error control will be included in *D4.8: Final BlogForever Platform*.

2.2.3 Metadata Structure

The metadata structure of BlogForever was explained in D4.4[3]. The implementation has followed the design regarding the record types and how they are linked, metadata schemes, the relationship between the METS document and the Invenio record, the storage of attached files, etc.

However, the MARC tags used to store the blogs' information have been extended for practical reasons during the implementation process. Tables 2.1 to 2.5 show the final design of the mapping between blog attributes and MARC tags, highlighting in bold the newly added or modified tags.

Record	Attribute/Metadata concept	MARC 21 representation
Blog	title	245 \$a
	subtitle	245 \$b
	URI	520 \$u
	aliases	100 \$g
	status_code	952 \$a
	language	041 \$a
	encoding	532
	sitemap_uri	520
	platform	781 \$a
	platform_version	781 \$b
	webmaster	955 \$a
	hosting_ip	956 \$a
	location_city	270 \$d
	location_country	270 \$b
	last_activity_date	954 \$a
	post_frequency	954 \$b
	update_frequency	954 \$c
	tags	653 \$1
	topic	654 \$a
	copyright	542
	ownership_rights	542
	distribution_rights	542
access_rights	542	
license	540 \$a \$u	

Table 2.1: Blog record attributes - MARC tags mapping

Record	Attribute/Metadata concept	MARC 21 representation
Entry (Post and Page)	title	245 \$a
	subtitle	245 \$b
	full_content	520 \$a
	full_content_format	520 \$b
	author	100 \$a
	URI	520 \$u
	tags	653 \$1
	topic	654 \$a
	reference_links	856 \$u \$y \$z
	aliases	100 \$g
	alt_identifier (UR)	0247 \$a
	date_created	269 \$c
	date_modified	260 \$m
	version	950 \$a
	status_code	952 \$a
	response_code	952 \$b
	geo_longitude	342 \$g
	geo_latitude	342 \$h
	access_restriction	506
	has_reply	788 \$a
last_reply_date	788 \$c	
num_of_replies	788 \$b	
child_of	760 \$o \$4 \$w	
license	540 \$a \$u	

Table 2.2: Post and Page record shared attributes - MARC tags mapping

Record	Attribute/Metadata concept	MARC 21 representation
Post	type	336
	posted_via	781 \$a
	previous_URI	780
	next_URI	785

Table 2.3: Post record extended attributes - MARC tags mapping

Record	Attribute/Metadata concept	MARC 21 representation
Page	template	962

Table 2.4: Page record extended attributes - MARC tags mapping

Record	Attribute/Metadata concept	MARC 21 representation
Comment	subject	245 \$a
	author	100 \$a
	full_content	520 \$a
	full_content_format	520 \$b
	URI	520 \$u
	tags	653 \$1
	topic	654 \$a
	status	952 \$a
	date_added	269 \$c
	date_modified	269 \$m
	addressed_to_URI	789 \$u
	geo_longitude	342 \$g
	geo_latitude	342 \$h
	has_reply	788 \$a
	num_replies	788 \$b
	is_child_of_post	773 \$o \$4 \$w
	is_child_of_comment	773 \$o \$4 \$w
is_child_of_blog	760 \$o \$4 \$w	
license	540 \$a \$u	

Table 2.5: Comment record attributes - MARC tags mapping

2.2.4 Scalability

Invenio infrastructure utilises load-balanced worker nodes for handling incoming search requests, thereby making search scalable. However, document ingestion is sequential due to the application level constraints. Moreover, current projects based on Invenio technology, like BlogForever, require an ability to handle up to 100 million documents. The Invenio collaboration developers have worked to improve Invenio scalability in order to cover these conditions. A more detailed description of the work done by the Invenio team can be found in the Master Thesis *Enhancing Scalability of Invenio*[1]. The main contributions of their work are:

- to development shard manager components to ease the addition of more machines on demand for document storage;
- to develop upload and search services that utilise shard manager component to speed up ingestion throughput at performant search times;
- to run experiments in order to analyse the scalability of the designed solution;
- to extend shard manager component allowing existing modules to take advantage of sharded architecture.

The Figure 2.14 shows an overall diagram of the sharded architecture, while Figure 2.15 represents a simple example of the ingestion process and how the records are split and dispatched.

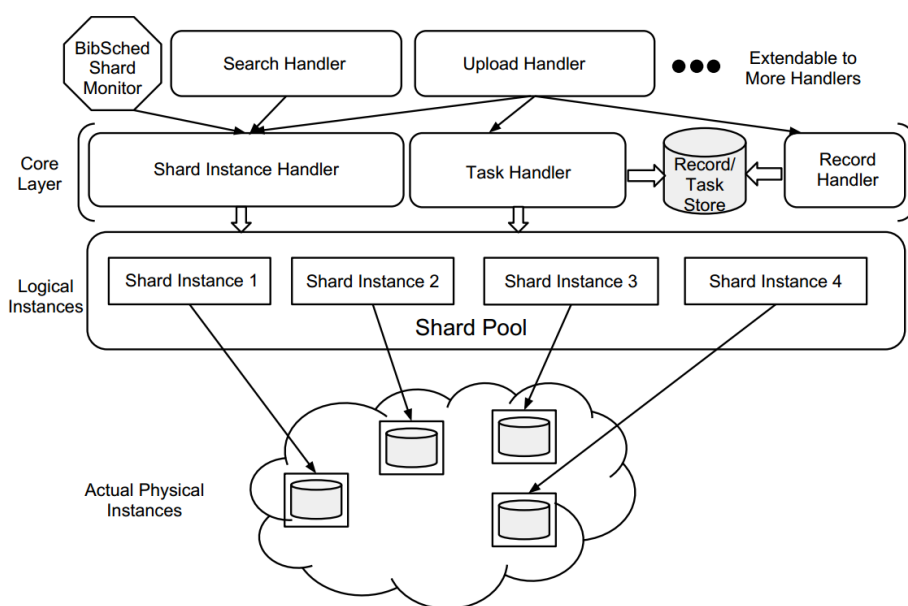


Figure 2.14: Shard manager components[1]

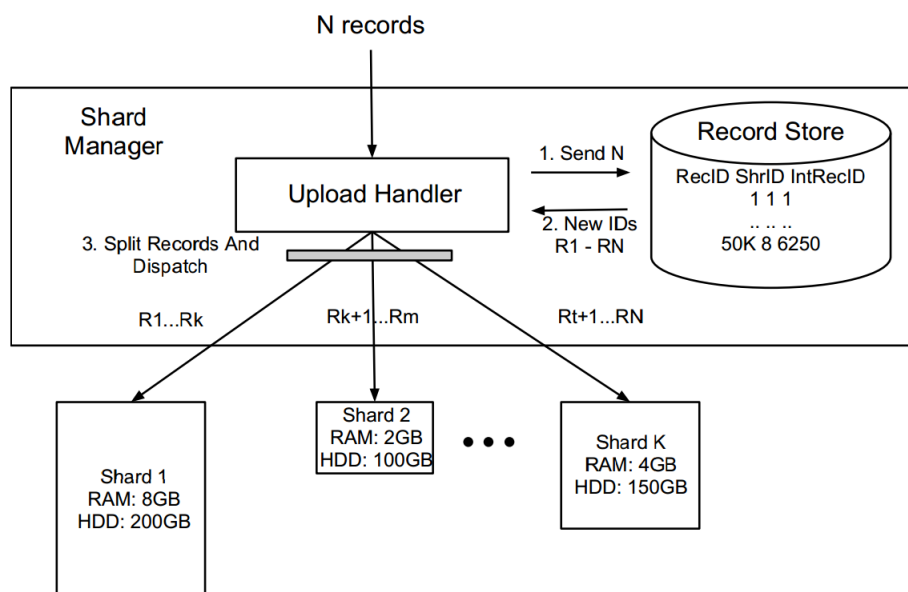


Figure 2.15: Shard manager ingestion workflow[1]

2.2.5 Preservation features

In D4.4[3] a list of features was introduced to satisfy the needs derived from the preservation recommendations coming from D3.1[7]. These recommendations have been followed in the implementation of the related features, according to the design

in D4.4's feature specifications[3] and some have already been documented in D4.5's implementation descriptions[2].

These features include *RF87-The archive transforms the SIPs received from the spider to AIPs* and *RF88-The archive stores the content of the AIPs in two different databases for preservation purposes*. However, the feedback received from WP5 testing and from the partners with more expertise in preservation issues showed that the design of these features was not enough to ensure that the repository is compatible with the Open Archival Information System (OAIS⁵) principles, especially regarding the Archival Information Packages (AIP). It was identified that even though the repository stores all the information needed in an AIP the information was stored in different places instead of being packed and stored together.

Therefore, the design of these preservation features has been modified. A new module called BibArchive will be implemented. This module will introduce a new database using mongoDB⁶, similar to the strategy used to store the METS documents coming from the spider. BibArchive will create an AIP for each record and will insert it in the preservation database. The package will be created using BagIt⁷. This package will include:

1. **Crawled XML:** It is stored unmodified checked using Message-Digest Algorithm 5 (MD5⁸) checksum to ensure that it has not been corrupted while downloading from the spider.
2. **Images and attachments:** They are stored in BibDocFile, unmodified, and checked using MD5.
3. **Checksum generation and validation for the above objects:** The METSXML coming from the spider includes MD5 checksums for each attached file, including images. First the METS file is checked, and then the checksum of the attached files is extracted and the files are checked. In any case, Invenio also creates a checksum for every attached file.
4. **Extracted technical metadata from the above objects:** Metadata extraction tools like Hachoir⁹ will be used to extract metadata from the attached files.

This modification of the design will be implemented during the next period and included in *D4.8: Final BlogForever Platform*.

⁵http://en.wikipedia.org/wiki/Open_Archival_Information_System

⁶<http://www.mongodb.org/>

⁷<https://confluence.ucop.edu/display/Curation/BagIt>

⁸<http://en.wikipedia.org/wiki/MD5>

⁹<https://pypi.python.org/pypi/hachoir-core>

2.2.6 User Interface

As will be explained in Section 2.3, the BlogForever repository started being developed on top of the Invenio *master* branch, which from the user interface point of view was offering an old-fashioned visual design and a markup structure that reduced the ability of customizing the interface. In the following Figures 2.16 2.17 some screenshots taken from the test servers BF1¹⁰ and BF3¹¹, both built on top of the Invenio *master* branch and used by WP5 to run their test cases, are offered.

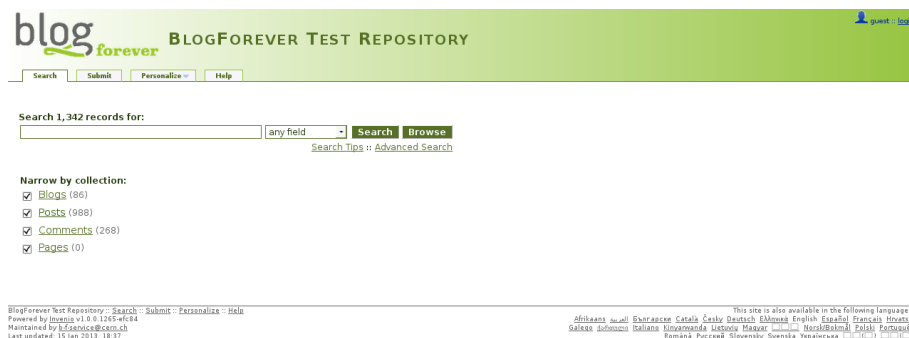


Figure 2.16: BlogForever main page at BF1

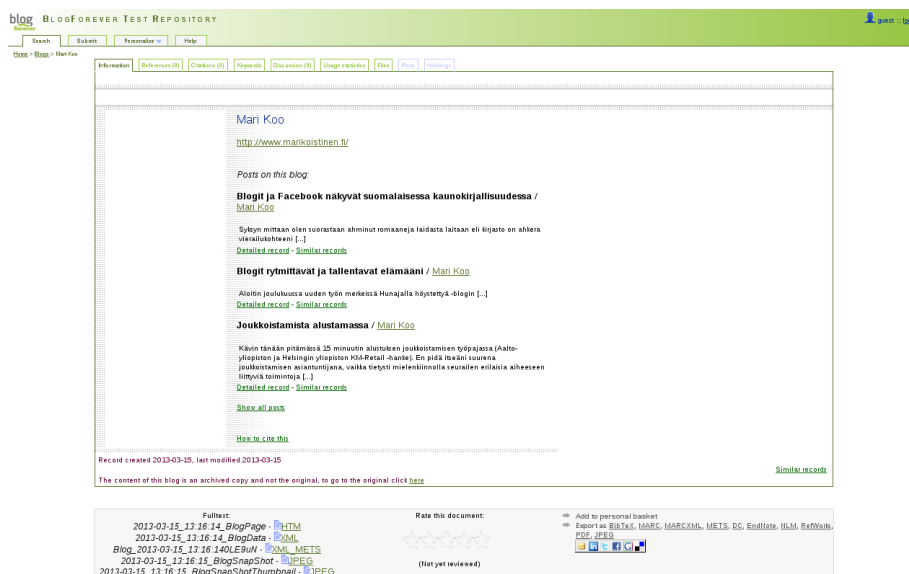


Figure 2.17: BlogForever blog record at BF3

At some point, Invenio started integrating new technologies in a new Invenio *next* branch such as Twitter Bootstrap and Jinja2 templates (as described in Section 2.3),

¹⁰<http://bf1.csd.auth.gr/>

¹¹<http://bf3.itc.auth.gr/>

providing a more modern appearance, as well as a more responsive and accessible design. The following Figures 2.18 2.19 present how Invenio looks like from the user interface point of view. This will help to the reader to appreciate the customization that has been done to get the new BlogForever repository user interface.

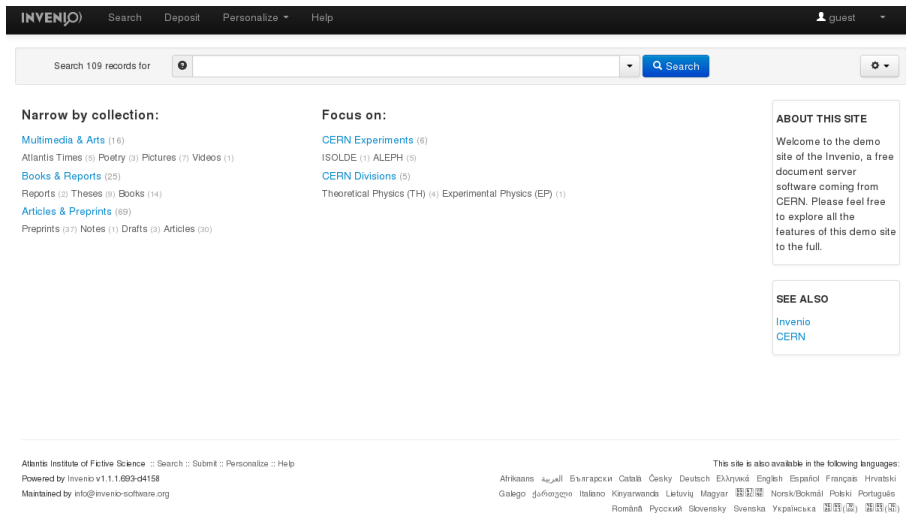


Figure 2.18: Invenio main page

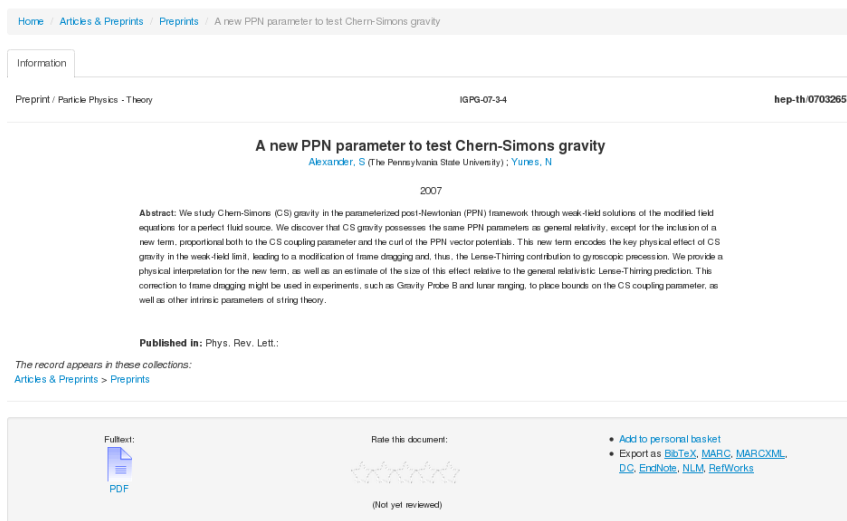


Figure 2.19: Invenio record

The Figures below show the current status of the BlogForever repository user interface. These screenshots have been taken from the test server BF5¹², which is built on top of the Invenio *next* branch and also used by WP5 to run their test cases.

¹²<http://bf5.itc.auth.gr/>



Figure 2.20: BlogForever main page at BF5

The screenshot shows the BlogForever search interface. At the top, there is a dark green navigation bar with the logo 'blog forever' and links for 'Search', 'Submit a Blog', 'Help', 'Log in', and 'Sign up'. Below this is a breadcrumb trail 'Home / Blogs'. A search bar contains the text 'Search 1245 records for of' and a 'Search' button. The results section is titled 'Blogs' and shows 'Showing records 1 to 10 out of 14 results.' There are two filter sections: 'Any Author' with a list of authors (publize70 (227), Borlotal (209), jaskaw (208), cjs92 (198), Simone1706 (187), More ...) and 'Any Year'. The main results list contains six entries, each with a checkbox, a thumbnail, a title, and a post count:

Checkbox	Thumbnail	Title	Posts
<input type="checkbox"/>		Deep inside of... moi ?!	1 Posts
<input type="checkbox"/>		Story of my Life	30 Posts
<input type="checkbox"/>		I am a Warrior of the light	30 Posts
<input type="checkbox"/>		The Other Side Of Southern Philippines: Mindanao	30 Posts
<input type="checkbox"/>		Dribblings of the Mind	30 Posts
<input type="checkbox"/>		Mixx of this Parish	30 Posts


Figure 2.21: BlogForever search page at BF5

Information Discussion Reviews Usage statistics Files Linkbacks

Translate

Disclaimer: The content of this blog is an archived copy. Please click here to go to the original

Rudybaer



This is a snapshot of the blog as it appeared when it was archived. Click to enlarge.

Archived date
2013/05/17

Tags
blog post

License
Creative Commons

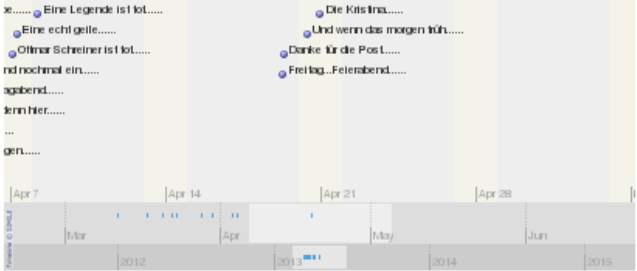
Export as
BibTeX, MARC, MARCXML, METS, DC, EndNote, RefWorks

Share

Other options
Add to Personal Collection
Ask for Deletion
Ask for Modification

Original blog URL
<http://rudybaer.blog.de/>

Cite as
Rudybaer
Retrieved from the original blog <http://rudybaer.blog.de/>
Date archived: 2013/05/17. Archived at <http://bib5.lit.auth.gr/record/28929>



Posts in this blog

Die Kristina.....
.....Will nach der Wahl nicht mehr ins Kabinett...!!!... Macht nichts...56 % der Deutschen würden sie eh nicht vermissen...Ich auch nicht..
rudybaer | 2013/04/21 01:54:07

Und wenn das morgen früh.....
.....auch so schön ist wie heute früh, dann wird morgen zum ersten mal in diesem Jahr auf dem Balkon gebrütselt...Jawohl... Es hätte heute zwar ruhig einen Tick wärmer sein können, aber ansonsten war es ein wunderschöner sonniger Samstag und ich habe es geschafft meinen Balkon herzurichten...Jetzt kann es Sommer werden... Ich hoffe ihr hatte/haht auch so einen schönen sonnigen Tag und wünsche euch noch ein schönes Wochenende...
rudybaer | 2013/04/20 11:50:00

Danke für die Post.....
.....es sind ja nur noch zwei Wochen...schön, ich treu mich auf dich, deine "HÖR AUF DEN HERZ TOUR 2013" und dein neues Album..
rudybaer | 2013/04/19 10:44:40

Show all posts

Figure 2.22: BlogForever blog record at BF5

Information
Discussion
Reviews
Usage statistics
Files
Linkbacks

[Translate](#)

Disclaimer: The content of this blogpost is an archived copy. Please click here to go to the original

Post from blog: **Rudybaer**

rudybaer

Posted date
2013/03/28

Archived date
2013/05/17

Tags
blog
post

License
Creative Commons

Die Wolfhasser haben wieder zugeschlagen.....

.....auch wenn sie es diesmal nicht unbedingt unmittelbar gewesen sind..



Deutschlands älteste freilebende Wölfin ist tot – Autopsie bringt Klarheit über Todesursache
Mücka/Rietschen. Deutschlands älteste freilebende Wölfin ist tot. Das „Einsauge“ genannte Tier des Nochtener Rudels wurde am Dienstag bei Mücka (Landkreis Görlitz) gefunden. Istle das Kontaktbüro Wolfregion Lausitz: Rietschen am Mittwoch mit. Bei der Autopsie im Institut für Zoo- und Wildtierforschung Berlin wurde festgestellt, dass die Wölfin an Bissverletzungen starb. Schrotkugeln und Metallpartikel wiesen darauf hin, dass die lahrende einjährige Wölfin in der Vergangenheit auch beschossen wurde. Das Kontaktbüro, das die Ausbreitung des Wolfes in Deutschland überwacht, nimmt an, dass „Einsauge“ bei Grenzstreifigkeiten benachbarter Wolfsfamilien zwischen die Fronten gerieten. Das 2000 oder 2001 im Muskauer Heide Rudel geborene Tier hatte 2005 mit einem nah verwandten Rüden das Nochtener Rudel begründet. Die Wölfin zog mindestens 42 Welpen auf. Nachkommen von ihr begründeten drei Rudel in Sachsen und eines in Niedersachsen. Sohn „Alan“ wanderte bis nach Weißrussland, ein 2006 geborener Enkel bis nach Dänemark. Er war dort der erste Wolf seit 200 Jahren.
Quelle: LVZ Online

MORE POSTS IN THIS BLOG

Die Kristina.....
2013/04/21 01:54:07

Und wenn das morgen küh.....
2013/04/20 11:50:00

Denke dir die Post.....
2013/04/19 10:44:40

Freitag...Feierabend.....
2013/04/19 09:46:21

Eine Legende ist tot.....
2013/04/08 08:27:10

Eine echt geile.....
2013/04/07 09:18:44

Export as

BibTeX, MARC, MARCXML, METS, DC, EndNote, RefWorks

Share

Other options

Add to Personal Collection
Ask for Deletion

Cite as

rudybaer. "Die Wolfhasser haben wieder zugeschlagen.....". Blog: Rudybaer
Date posted: 2013/03/28. Retrieved from the original post | <http://rudybaer.blog.de/2013/03/28/wolfhasser-zugeschlagen-15684789/>
Date archived: 2013/05/17. Archived at <http://dx.doi.org/10.1111/1471-6708.126974>

Figure 2.23: BlogForever post record at BF5

Information Discussion Reviews Usage statistics Files Linkbacks

Translate

Disclaimer: The content of this comment is an archived copy. Please click here to go to the original

Comment on post: **Apple iPad Awaits Back-To-School Boom**

Management Job
 Archived date
 2013/05/08
 Tags
 b1c2 post
 License
 Creative Commons

Management Job wrote:
 It is smart of Apple to try to gain a foothold in the educational system. Their competitors are making products nearly as good or just as good, but soon they will be doing it at half the cost of the iPad. There is a pretty good ASUS tablet on its way this spring that will retail around \$250. If Apple can entrench itself into the public school system, it will continue to have a large market to sell to. Not to mention those kids will grow up feeling comfortable using Apple products.

Cite as
 Management Job. Blog post: 'Apple iPad Awaits Back-To-School Boom'
 Retrieved from the original comment <http://people.us.edu/rschr1/el/?p=2760&page=1#comment-5188>
 Date archived: 2013/05/08. Archived at <http://blogforever.com.ch/record/63>

MORE COMMENTS IN THIS POST

Austin Apartments
 It's remarkable how far technology has progressed since I was a kid [...]

Management Job
 It is smart of Apple to try to gain a foothold in the educational system [...]

Export as
 BibTeX, MARC, MARCXML, METS, DC, EndNote, RefWorks

Share

Other options
 Add to Personal Collection

Figure 2.24: BlogForever comment record at BF5

If we compare these images with Figures 2.16 2.17, which show the old BlogForever user interface, the progress and improvement that the user interface has experienced can be appreciated.

2.3 Development Cycle

As was already mentioned in D4.5 (Section 2.2: Implementation), Git¹³ is the distributed revision control and Source Code Management (SCM) system used by Invenio developers for a collaborative development workflow, and hence, the tool that has been used to develop the BlogForever specific repository features.

The Git feature that really makes it stand apart from nearly every other SCM out there is its branching model. Git allows and encourages developers to have multiple local branches that can be entirely independent of each other.

The Invenio branches and repositories organization, as well as the collaboration model followed for development, was described in D4.5 (Section 2.2: Implementation).

¹³<http://git-scm.com/>

The prototype of the BlogForever repository (purpose of D4.5) was built on top of Invenio *master* branch, a stable branch where the new features are developed and where the new feature releases are made from, using technologies such as MySQL¹⁴, JavaScript¹⁵, AJAX¹⁶, JQuery4¹⁷ and HTML¹⁸ embedded within the Python¹⁹ code.

Just before starting the development of the BlogForever repository prototype, Invenio reached the point where it was useful to rewrite part of the code base using new software stack in order to keep flexibility, manageability of growing number of modules, and speed-up prototyping and development of new ones[12]. For this purpose, the Invenio team started building a new *next* branch on top of the existing *master* branch to adopt a new development framework based on new technologies such as SQLAlchemy²⁰, Jinja2²¹, Twitter Bootstrap²², WTForms²³, Flask²⁴ and Redis²⁵.

During the development of the BlogForever repository prototype, the idea of moving to the *next* branch was always in mind. In fact, to build the user interface of the prototype some of these new technologies were already introduced, such as Twitter Bootstrap and Jinja2 templates, as well as other existing resources such as Font Awesome²⁶ icons or Bootswatch²⁷ color templates. Finally, after studying the consequences of this step in terms of advantages and disadvantages, the final decision was to start taking advantage of all the work that the Invenio team was doing on the *next* branch and to bring the new technologies to the development of the BlogForever repository.

This change involved extra work to be done by the BlogForever repository developers since it meant to port already implemented repository features to the new technologies, as well as to rewrite the existing user interface templates by using the front-end related technologies mentioned before (new implementation descriptions have been written in Section 3.2 for those features which have been modified). An important change is for example that Flask uses a concept of blueprints for factorizing an application into a set of components that can be registered on an application at a URL prefix and/or subdomain even multiple times. Blueprints are preferred method used by Invenio developers to implement local custom modules allowing a clean separation of custom overlay from Invenio core. To follow this structure, new blueprints have been created within the BlogForever repository.

On the one hand, having taken this important step means an extra work, as was already mentioned above. In addition to this, the fact that the *next* branch is

¹⁴<http://www.mysql.com/>

¹⁵<http://www.w3schools.com/js/>

¹⁶<http://www.w3schools.com/ajax/>

¹⁷<http://jquery.com/>

¹⁸<http://www.w3schools.com/html/>

¹⁹<http://www.python.org/>

²⁰<http://www.sqlalchemy.org>

²¹<http://jinja.pocoo.org/>

²²<http://twitter.github.com/bootstrap/>

²³<http://wtforms.simplecodes.com/docs/1.0.4/>

²⁴<http://flask.pocoo.org/>

²⁵<http://redis.io/>

²⁶<http://fortawesome.github.io/Font-Awesome/>

²⁷<http://bootswatch.com/>

still in process of development, which means that currently the Invenio team is still adapting the existing Invenio modules, makes the BlogForever repository be continuously subjected to the adaptation and integration of the new changes.

On the other hand, there is a list of advantages that push these disadvantages into the background, and of course, helped us to make the final decision. The main advantages are explained by the features that the new technologies bring with them.

For example, SQLAlchemy, one of the most used Python SQL toolkits, comes with an optional Object Relation Mapper (ORM) component that provides the data mapper pattern, where classes can be mapped to the databases in open ended, multiple ways. It allows the object model and database schema to develop in a cleanly decoupled way from the beginning. The object models have been designed in a way permitting dynamic properties with strong checks ensuring data consistency.

Flask brings the blueprints as was described above, allowing also the easy way of enable and disable certain modules.

Jinja2 is a full-featured template engine with an expressive language that forces the strict separation between business logic and presentation. It allows creation of reusable building blocks across the whole system that reduces the need for repeating code. The code for the visual display of the repository can be maintained and edited separately to the back-end components. This allows faster and easier customization of the user interface and provides a more modular code framework that is easier to maintain.

Finally, Twitter Bootstrap contains other widely used interface elements and JavaScript based plugins to extend the functionality of existing interface elements. It is widely supported by a community of developers and designers which allows to take advantage of existing resources to develop very fast attractive and functional front-end features.

2.4 Development Methodology

In this section the methodology that has been followed to structure, plan, and control the process of developing is explained.

In D4.4 [3], three different levels were defined to group the list of repository resulting features:

- **High priority features:** implies that the software will not be acceptable unless these features are provided in an agreed manner.
- **Medium priority features:** implies that these are features that would enhance the software product, but would not make it unacceptable if they are absent.
- **Low priority features:** implies a class of functions that may or may not be worthwhile. This gives the supplier the opportunity to propose something that exceeds the Software Requirements Specification.

Therefore, the methodology followed to develop the repository features was carried out guided by the priorities defined. First of all the higher priority features are developed, to ensure that the time spent on them is enough, and the essential requirements are accomplished into the repository. Once the first iteration of the development of the essential features is finished, the ones with medium priority were attacked, leaving to the end those features defined as optional.

Once this is clear, the development process starts with the assignment of features among the partners involved in the development, being CERN and SRDC the main partners contributors. CERN, who is the responsible of coordinating the implementation stage, defined for this purpose a simple task tracking system in order to manage the tasks assigned to the different partners thought its life cycle. A series of status of features were defined by the following states:

- **ASSIGNED**: feature already assigned by CERN to the appropriate partner/developer.
- **IN_WORK**: partner/developer is working on the feature.
- **TO_REVIEW**: partner/developer finishes the implementation of the feature, pushes it to his/her public repository and the feature becomes ready to be reviewed by CERN.
- **REVIEWED**: CERN fetches the code of the feature from the corresponding public repository and reviews it. If there is any issue the feature can be marked as **IN_WORK** again to be solved by the corresponding partner/developer.
- **INTEGRATED**: CERN integrates the new feature into the main BlogForever branch and runs unit and regression tests to check its behavior out with the rest of components.
- **DEPLOYED**: once the feature is integrated, it is deployed on the corresponding test server to let WP5 partners to carry out their testing work.

The Figure 2.25 tries to represent the development workflow described above:

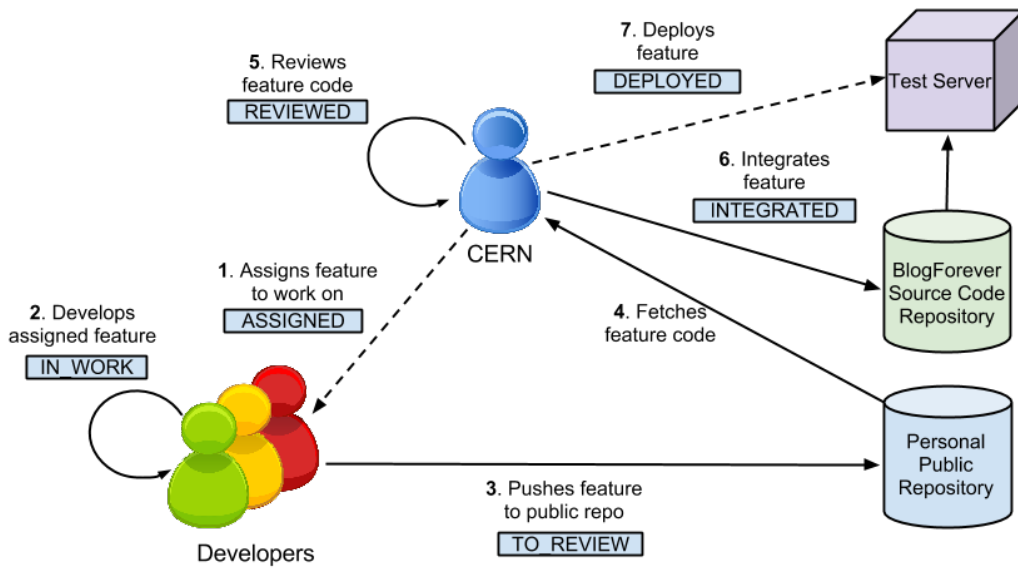


Figure 2.25: Development workflow

Chapter 3

Implementation Descriptions

One of the objectives of this deliverable is to describe the implementation activities of those repository features defined in D4.4 that were not included in D4.5, or those that went through any modification after delivering the prototype. In order to include the description of the features along with its implementation details, the same template that was created in D4.5[2] will be used.

Section 3.1 lists the features that Invenio already provides and fulfill our needs, Section 3.2 presents the implementation descriptions of those features that were implemented for the prototype, but have been ported to the new technologies running under next (as was already explained in Section 2.3) as well as those which have been implemented to complete the whole list of repository features, and Section 3.3 lists those repository features that were defined in D4.4 but finally has been decided not to implement them, as well as the reasons why this decision was taken.

3.1 Features already in Invenio

As mentioned earlier, Invenio is a comprehensive software for digital library management. Therefore, it already supports 34 of the repository features. However, some of these features needed to be configured in order to meet the requirements of the final BlogForever platform. These features are listed as follows:

- RF5 - The web interface provides harmonized access and ensures compatibility with major browsers
- RF7 - Export data using the OAI-PMH protocol
- RF8 - Export data using Dublin Core Schema
- RF10 - Archive user passwords are stored encrypted in the database
- RF11 - The archive web interface is available in many different languages
- RF13 - UTF is used as the default character encoding in the archive
- RF15 - Option to disseminate archive content in major social web platforms
- RF16 - The archive offers an RSS channel of its latest updates and/or users can receive notification when new content of their interest is added to the archive

- RF18 - The archive detects duplicated content and keeps only one copy
- RF19 - The archive can be indexed by external search engines
- RF20 - The archive's statistics are exported as CSV
- RF21 - The archive offers the option to login using SSO / LDAP
- RF27 - The archive displays a unique URL (DOI) for each record
- RF29 - The archive alerts the user when there are software updates
- RF33 - The archive can display only the very core information for each record
- RF37 - The archive restricts the access to its content to specific IP ranges
- RF38 - Users can communicate within the archive sharing and exchanging resources
- RF39 - Free open-source archive software
- RF43 - For each record the archive stores the search keywords used to find them
- RF44 - The archive enables pingback/trackback services
- RF51 - The archive is able to search within external sources external collections, hosted collections
- RF55 - The archive provides advanced APIs for use by developers to interact with the archive's content
- RF60 - The archive can export all its content, database entries and file system for migration
- RF69 - The archive facilitates searching by providing fuzzy indexing and stemming
- RF74 - The archive enables/disables certain functionalities based on the content rights
- RF77 - The archive provides a mobile version
- RF79 - The archive can handle a very large number of content and users
- RF80 - The archive provides mechanisms to control data redundancy
- RF81 - The archive is built based on a modular service-oriented architecture
- RF82 - The archive can be deployed using a range of different database server technologies
- RF83 - The archive provides multiple different views of the archive for each user
- RF84 - The archive offers a complete range of search options to the user
- RF85 - The archive provides support for OpenURL
- RF86 - The archive offers functions to edit metadata

3.2 List of implementation descriptions

In this Section the list of implementation descriptions of the finished repository features is presented. For the features developed to take part of the prototype

that have not went through any modification, just its name is provided (their corresponding implementation descriptions can be found in Chapter 3 of D4.5 [2]). For those that have been implemented in the second development phase, their implementation descriptions are presented, as well as for those which were implemented for the prototype but have been modified since then. The list of features that have just been improved from the user interface point of view is also presented, as well as a small list with those features whose development has been postponed until the third development phase.

3.2.1 Previously implemented features

The following list correspond to those repository features that were already implemented in the repository prototype. Their corresponding implementation descriptions can be found in Chapter 3 of D4.5 [2] since they have not been modified.

- RF2 - “Your History“ box as part of the user dashboard
- RF3 - “Share” option in “Your History” box
- RF12 - The archive can import METS
- RF26 - BlogUploader command line to upload, update and delete a list of blogs
- RF30 - Users are able to bookmark records, also using external bookmarking engines
- RF31 - The archive offers a complete blog submission interface to submit, modify and delete blogs/posts
- RF32 - Users are able to remove their personal data
- RF35 - The archive displays other blogs that were viewed by people who also viewed the current blog
- RF40 - The archive validates the content received from the spider
- RF41 - The archive detects and eliminates spam content
- RF45 - The archive is able to inter-operate with federated search engine dbwiz (SRU Server)
- RF53 - The archive respects content licenses and displays useful information about them
- RF54 - The archive keeps all the different versions of a record
- RF59 - Export data using XML (METS, MARC)
- RF62 - Export records as PDF and JPEG
- RF63 - The archive keeps snapshots of all the different designs of a blogs
- RF67 - The archive fetches and stores embedded content (introduced new clean process)
- RF88 - The archive stores the content of the AIPs in two different databases for preservation purposes

3.2.2 Newly implemented features

The following implementation descriptions belong to the features that have been implemented since D4.5 [2] submission.

Feature ID	RF52 (Repository Feature 52)
Name	Users can tag archived records with personal tags
Effort Spent	3 Weeks
Modules Affected / Created	WebTag

Description of the new feature

Users are able to assign tags to the records with personal words. With these tags, the user is able to organize the records in personal collections according to his/her preferences.

Users are able to add, to edit and to delete tags.

Implementation details

So far, only the back-end of this feature is ready. A new module called WebTag has been added which consists of:

- Two new tables in the database:
 - **tagTAG** : for every tag created by an specific user, this table stores the record ID, the language and the position in which the tag was created
 - **tagTAGLOG** : for every tag edited by an specific user, this table stores the new record ID, the new language, the new position in which the tag was edited, and the operation (*insert*, *update*, *delete*) that the user did.
- Three triggers on **tagTAGLOG** table: **insert_user_record_tag** , **update_user_record_tag** and **delete_user_record_tag**
- Nine public and three private functions which constitute the API of the module:
 - **rename_tag(user_id, original_tag, new_tag, language)** : changes one tag into another

<ul style="list-style-type: none"> – <code>user_record_tags_exist(user_id, record_id)</code> : checks if there are tags created by the user on the record – <code>get_user_tag_pairs(user_id)</code> : gets co-occurrent tags – <code>get_user_tagged_records(user_id, query, mode)</code> : gets records tagged by this user – <code>get_keywords_user_tags(user_id, search)</code> : gets keywords and user tags matching a search – <code>get_user_tags(user_id, search)</code> : gets tags with additional info – <code>get_user_record_tags(user_id, record_id)</code> : gets the tags for a user on a record – <code>delete_tags(user_id, record_id)</code> : delete the tags for a user on a record – <code>set_tags(user_id, record_id, tags, position, language)</code> : sets the tags for a user on a record – <code>_add_tag(user_id, record_id, tag, position, language)</code> : adds a single tag to a record for a user in the specified position – <code>_tag_valid(tag)</code> : checks if the tag is valid for insertion – <code>_tag_cleanup(tag)</code> : white space and character cleaning process <ul style="list-style-type: none"> • A configuration file called <code>webtag-config.py</code> containing the config variable <code>TAG_MAX_LENGTH</code> , which sets the maximum length for tags. 		
<table border="1"> <tr> <td>Implemented By</td> <td>CERN</td> </tr> </table>	Implemented By	CERN
Implemented By	CERN	

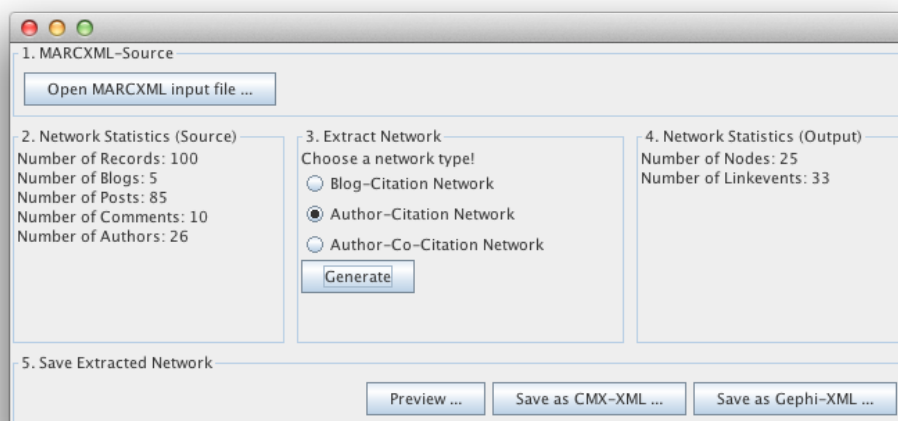
Table 3.1: Implementation Description: RF52

Feature ID	RF65 (Repository Feature 65)
Name	The archive analyzes blog links and stores the connections between them separately
Effort Spent	4 weeks
Modules Affected / Created	-
Description of the new feature	
<p>A network consists of nodes and links between the nodes. Three kinds of networks are extracted from the archived blogosphere:</p> <ul style="list-style-type: none"> • Blog-Citation network <ul style="list-style-type: none"> – Nodes: All Blog objects in the archived blogosphere. 	

- Links: A link is created when a hyperlink in a post references another post or a blog. The link goes from the parent blog of the sending post to the parent blog of the receiving post or the receiving blog.
- Author-Citation network
 - Nodes: All authors in the archived blogosphere.
 - Links: A link is created when a hyperlink in a post or in a comment references another post or comment. Sending and receiving posts or comments must have an author.
- Author-Co-Citation network
 - Nodes: All author objects in the archived blogosphere.
 - Links: A link is created if two or more posts or comments reference the same address through a hyperlink.

The Blog-Citation network and the Author-Citation network consider only links that direct to an object that is archived in the repository. The Author-Co-Citation network considers all kind of links (external and internal).

The implementation of the feature differs from the feature design: Links are not differentiated between Citations, BlogRoll, and Pingback/Trackback because these information can not be identified by the spider.



RF65 - GUI of the feature. The main steps in the network generating process are (a) to choose the input, (b) to choose the network type, and (c) to choose the output format and location

Implementation details

The feature is implemented as java application together with the repository feature RF72 which visualizes the extracted network.

Nodes are represented as objects that have an ID and a maximum of five details. Links are an aggregation of one or more Linkevents. A linkevent has one sender, one or more recipients, a timestamp, and a maximum of five details. The distinction of Linkevents with timestamps allows representing and analysing the network evolvement over time.

The application accepts as input a collection of blog, post, and comment objects in MARCXML from the repository. The objects in the collection are analyzed and the network extracted by the Java class *tub.BFMarcXMLReader* and stored in the Java objects

- *tub.dataElements.Network*
- *tub.dataElements.Node*
- *tub.dataElements.Linkevent*

Currently, the two output formats CMX-XML and GEXF are provided.

- With CMX-XML, the extracted network can be further analysed and explored with the Commetrix^a software. The Java class *tub.CMXMLWriter* transforms the extracted network into CMX-XML. The CMX-XML preserves the structure of link events, and, therefore, Commetrix facilitates dynamic and static analysis.
- GEFX can be analysed with Gephi^b, an open-source tool to visualize and analyse networks. The Java class *tub.GEXFWriter* transforms the extracted network into GEXF. Thereby, the linkevents has to be aggregated to edges due to the limitations of the GEXF format. Therefore, the Gephi tool facilitates only static analysis.

The Java class *tub.BFNetworkGenerate* contains the GUI for the Network Generator application.

^a<http://www.commetrix.de/>

^b<https://gephi.org/>

Implemented By	TUB
-----------------------	-----

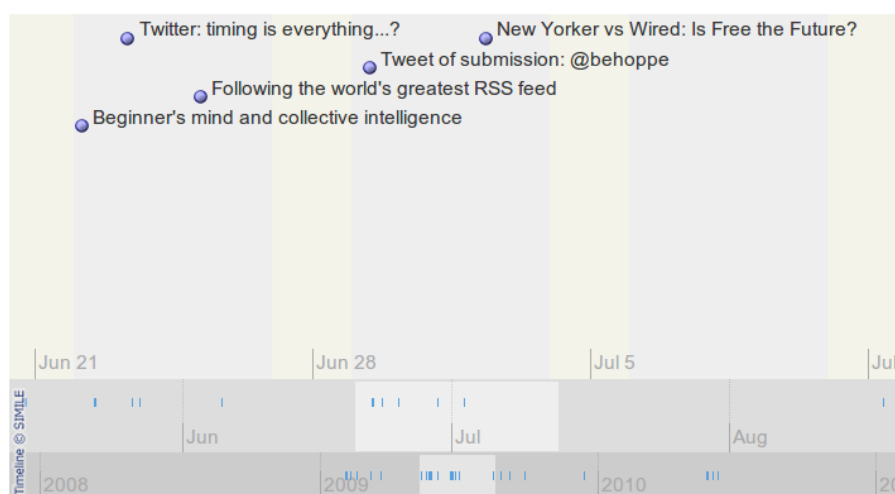
Table 3.2: Implementation Description: RF65

Feature ID	RF66 (Repository Feature 66)
Name	The archive provides a historical/chronological navigation
Effort Spent	2 Weeks
Modules Affected / Created	WebBlog, BibFormat

Description of the new feature

A new BibFormat element has been created and included in the BlogHTML template. This element displays a JavaScript slider showing the blog posts in a slider timeline. The events (post published) can be clicked and a short description of the record is displayed in a bubble, with a link to the detailed view of the record.

The slider is populated with data with an XML file created with the information of the blog as a hidden XML export option. It could easily be extended to include also in the XML file the comments of each post thus enriching the content of the timeline slider.



RF66 -

Chronological navigation of a blog

Implementation details

- Added a new install option to the Makefile options: (**install-jstimeline-plugin**) that installs the necessary JavaScript files. The code used is from the **simile-timeline.2.3.0** plugin.
- Added XML output format to feed the plugin. The format is called **xtl** and can be viewed adding **xtl** to the url in the detailed view of a blog.
- Added a BibFormat element called **bfe.blog.timeline.py** to be used in **BlogHTML.bft**

Implemented By	CERN
----------------	------

Table 3.3: Implementation Description: RF66

Feature ID	RF72 (Repository Feature 72)
Name	The archive provides a visualization of the blogs network structure
Effort Spent	4 weeks
Modules Affected / Created	-

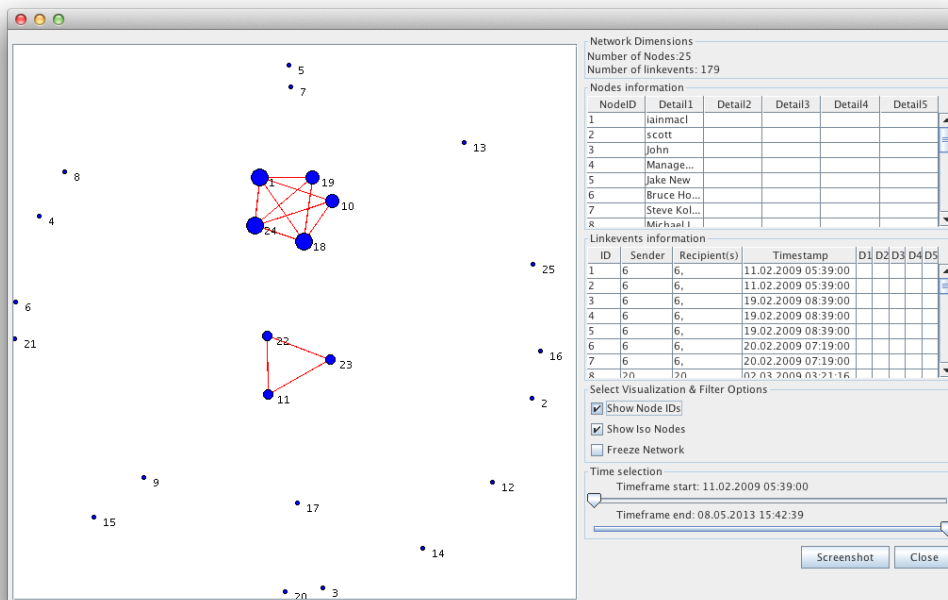
Description of the new feature

The exploration of a blog network needs a visual representation of the nodes and links. The feature provides a visualization with several options. Thus, a first exploration of the network can be conducted and the network adapted before it is further analysed with more sophisticated analysing software, e.g. Commetrix^a.

In the example figure below, an Author-Co-Citation network is shown. The network is visualized as blue nodes with red edges between them. The size of a node represents how many linkevent the node sends to another node. This means how often an author references something that somebody has referenced before. The nodes and linkevents of the feature are listed in the corresponding tables on the right side for a more detailed inspection. The visualization has the following options:

- Show Node IDs: Turns on/off that the node ID is shown as label for a node.
- Show Iso Nodes: Turns on/off that isolated nodes are shown. An isolated node is a node that does not have any connection with any other node.
- Freeze Network: The drawing algorithm for the network is dynamic and reacts on the re-positioning network nodes. The dynamic behaviour can be turned off so that the nodes of the network can be positioned manually without interfering of the drawing algorithm.
- Time selection: The time slot can be limited to examine the evolvement and state of the network at different points in time.

Furthermore, the feature provides the possibility to save the network visualization as a screenshot in png. Thus, the network can be used in reports or presentations.



RF72 - GUI of the feature

Implementation details

The feature is implemented as java application together with the repository feature RF65 which generates the network representation of the blogosphere.

Nodes are represented as objects that have an ID and a maximum of five details. Links are an aggregation of one or more Linkevents. A linkevent has one sender, one or more recipients, a timestamp, and a maximum of five details. The distinction of Linkevents with timestamps allows representing and analysing the network evolution over time.

The application uses the following Java classes to represent the network structure:

- *tub.dataElements.Network*
- *tub.dataElements.Node*
- *tub.dataElements.Linkevent*
- *tub.dataElements.Link*

The following classes represent the network graph that is used for the visualisation:

- *tub.aladin.graphNode*

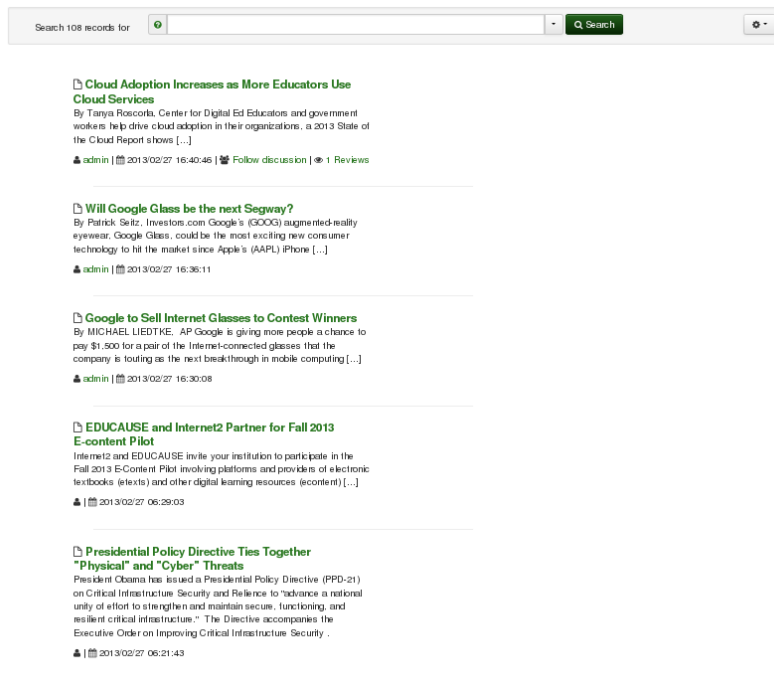
<ul style="list-style-type: none"> • <i>tub.aladin.graph</i> <p>The GUI of the network visualization is performed by the classes <i>tub.aladin.showGraphPanel</i> and <i>tub.aladin.showNetworkGraphFrame</i>.</p> <p>The class <i>tub.aladin.settings</i> contains the color configuration of the network visualization.</p> <p>^ahttp://www.commetrix.de/</p>		
<table border="1"> <tr> <td>Implemented By</td> <td>TUB</td> </tr> </table>	Implemented By	TUB
Implemented By	TUB	

Table 3.4: Implementation Description: RF72

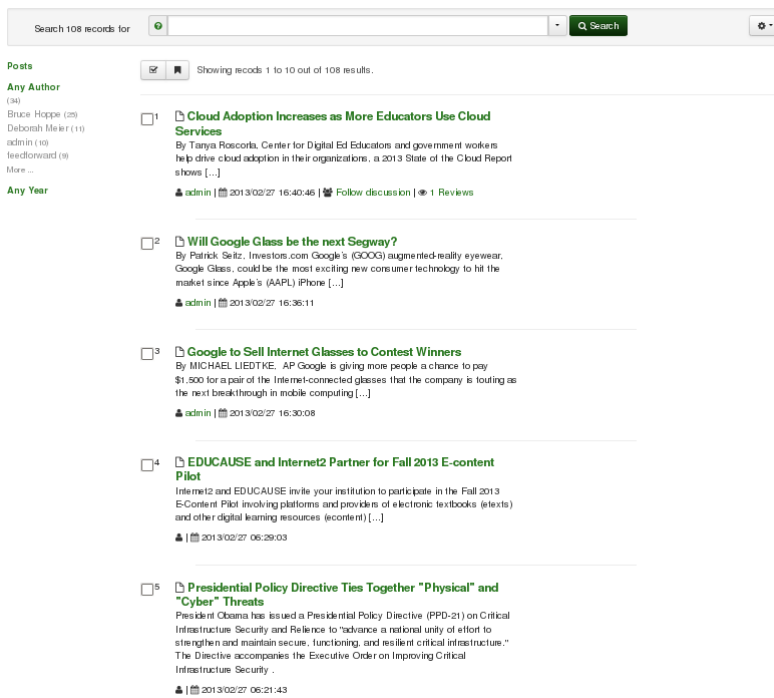
3.2.3 Updated features

The following implementation descriptions describe the features that were implemented for the repository prototype, but have been modified since then because, they have just been fixed or improved, or because they have been ported to the Invenio *next* branch. These implementation descriptions should be considered as an update to the ones that can be found in Chapter 3 of D4.5 [2].

Feature ID	RF6 (Repository Feature 6)
Name	Latest posts are displayed sorted by posted date
Effort Spent	1 Week
Modules Affected / Created	WebSearch, BibSort
Description of the new feature	
<p>Posts are always offered sorted chronologically by the date in which they were posted on their blog.</p> <ul style="list-style-type: none"> • Latest posts page: page displayed when the user clicks on Posts collection 	



- Search posts results: results displayed when the user search within the Posts' collection



- “More posts in this blog” menu: menu offered within the post template with the rest of posts which belong to the same blog

Posts in this blog

- Cloud Adoption Increases as More Educators Use Cloud Services
 By Tanya Roscorla, Center for Digital Ed Educators and government workers help drive cloud adoption in their organizations . a 2013 State of the Cloud Report shows [...]

admin | 2013/02/27 16:40:46 | Follow discussion | 1 Reviews
- Will Google Glass be the next Segway?
 By Patrick Seitz, Investors.com Google's (GOOG) augmented-reality eyewear, Google Glass, could be the most exciting new consumer technology to hit the market since Apple's (AAPL) iPhone [...]

admin | 2013/02/27 16:36:11
- Google to Sell Internet Glasses to Contest Winners
 By MICHAEL LIEDTKE, AP Google is giving more people a chance to pay \$1,500 for a pair of the Internet-connected glasses that the company is touting as the next breakthrough in mobile computing [...]

admin | 2013/02/27 16:30:08

Show all posts

- “Posts in this blog” section: section offered within the blog template with all the posts belonging to the same blog

MORE POSTS IN THIS BLOG

- Cloud Adoption Increases as More Educators Use Cloud Services
2013/02/27 16:40:46
- Will Google Glass be the next Segway?
2013/02/27 16:36:11
- Google to Sell Internet Glasses to Contest Winners
2013/02/27 16:30:08
- Microsoft to begin connecting Skype and Lync by June 2013
2013/02/26 16:40:12
- BYO-ID? On the internet, now everybody knows you're not a dog
2013/02/26 16:35:31
- Shuttleworth's one device: The smartphone is the tablet and the PC
2013/02/26 16:30:25

Implementation details

- A new sorting method called **posted date** has been added to the set of BibSort methods in order to sort records by the date given in the MARC tag 269_c. Also a new washer called (sort_transform_format_date has been implemented to transform the format of the date provided by the spider to the standard format %Y/%m/%d %H:%M:%S.

- A new property called **latest_additions** has been added to the **Collection** class, which will contain the list of recids of each collection sorted by the selected sorting method.
- The blueprint **websearch_blueprint.py** has been extended to propagate to the search the sorting parameters introduced to sort the latest additions.
- A new config variable **CFG_WEBSEARCH_INSTANT_BROWSE_AND_SEARCH_SAME_SORTING** has been added in order to enable or disable the option to display the search results in the same order than the one selected to display the latest additions

Implemented By	CERN
-----------------------	------

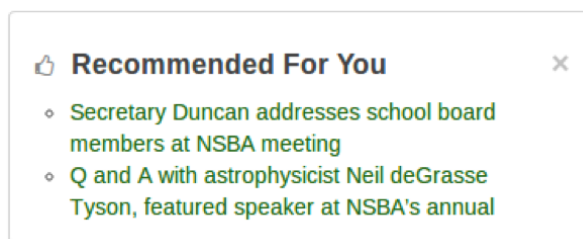
Table 3.5: Implementation Description: RF6

Feature ID	RF1 (Repository Feature 1)
Name	Customizable user dashboard
Effort Spent	2 Weeks
Modules Affected / Created	WebSession
Description of the new feature	
<p>This feature was implemented by BlogForever developers, but ported to next and integrated into Invenio core by Invenio developers. Therefore, now it is a part of Invenio and directly used in BlogForever repository. Description of the functionality of this feature can be found in D4.5.</p>	
Implemented By	SRDC

Table 3.6: Implementation Description: RF1

Feature ID	RF22 (Repository Feature 22)
Name	“Your Preferences” box as part of the user dashboard
Effort Spent	1 Week
Modules Affected / Created	WebSearch, WebSession
Description of the new feature	

A new box containing recommended records based on user search criteria has been added into “Your Account” page.



Implementation details

- Two new database tables, `query_term` and `user_query_term` are introduced. Moreover, `log_query_terms` function is added into `search_engine.py` to keep history of the search terms used by each user.
- `webrecommend.py` module that contains functions to recommend the records to the users are developed. These functions are:
 - `get_unread_records` : Returns the record ID's that are not viewed by the user.
 - `get_query_terms` : Returns the list of the query terms that have been used by the user
 - `get_recommended_content` : Returns the unread records based on word similarity with the query terms used by the user.
- Three configuration parameters have been added into `websession_config.py` :
 - `CFG_RECOMMENDATION_RANK_METHOD` : The name of the word similarity ranking method
 - `CFG_RECOMMENDED_CONTENT_NUMBER` : The number of records recommended in “Your Account” page.
 - `CFG_MOST_FREQUENT_TERM_NUMBER` : The number of most frequent terms considered in recommendation.
- `webrecommend_user_settings.py` is introduced to add “Recommended for you” box to “Your account” page.

Implemented By	SRDC
----------------	------

Table 3.7: Implementation Description: RF22

Feature ID	RF48 (Repository Feature 48)
Name	The archive provides the option to translate its content on demand
Effort Spent	2 Weeks
Modules Affected / Created	BibFormat, WebComment, WebMessage, WebStyle

Description of the new feature

This feature enables users to translate the content of records, messages, reviews and comments. The language used by the user determines the language of the translation.

Mostrar texto original

Descargo de responsabilidad: El contenido de esta entrada de blog es una copia archivada. Haga clic aquí para ir a la original

Publicar en blog: **Texto original:**

Edificio más alto de Nueva York del futuro (1881)

Cuando el dibujante Thomas Nast dibujó este ejemplo de futuro Manhattan para *Weekly de Harper* en 1881, Iglesia de la Trinidad fue el edificio más alto de Nueva York, con su torre y la cruz alcanza 281 metros hacia el cielo. Hasta septiembre de 2001, la Torre Norte del World Trade Center se colocaba como el edificio más alto de la ciudad a 1.368 metros. Hoy en día, el *Empire State Building* es el edificio más alto de Nueva York, a 1.250 metros de altura, pero con un poco de suerte que es probable que cambie pronto (más o menos).

El \$ 3.100.000.000 mucho retraso *One World Trade Center* (antes conocido como Torre de la Libertad) se mantendrá en el antiguo emplazamiento del World Trade Center como el edificio más alto de Nueva York en 1776 pies. Está actualmente programada para ser completada a finales de 2013. Digamos que yo no estoy conteniendo la respiración para esa fecha.

Imagen del Harper 27 de agosto 1881 el semanario y el libro de predicciones: *Predicciones pictóricas del Pasado* por John Durant.

Fecha fijada: 2011/08/18

Fecha archivado: 05/20/2013

Etiquetas:

Licencia:

Creative Commons

The user can translate the corresponding part by clicking “Translate” link over the context. Moreover, the text of the “Translate” link appears in the language of the user. In the sample below, the user using the platform in English translates the message in German to English.

From: admin
Subject: Translate
Sent on: 2013-05-20 17:31:25
Sent to: admin
Sent to groups:

English

Translate

Dies ist die Botschaft, werde übersetzt werden sollen.

The user is able to undo the translation by clicking “Show Original” link.

<div style="border: 1px solid #ccc; padding: 5px; background-color: #f9f9f9;"> <p>From admin Subject Translate Sent on 2013-05-20 17:31:25 Sent to admin Sent to groups</p> </div> <div style="margin-top: 10px;"> <p>English ▾</p> <p>Show original</p> <div style="border: 1px solid #ccc; background-color: #e8f5e9; padding: 5px; min-height: 20px;"> <p>This is the message that will be translated.</p> </div> </div>	
Implementation details	
<ul style="list-style-type: none"> • Google translate gadget has been utilized for translation task. • “Translate” link is added over the records, messages, reviews and comments to translate the content. • To add “Translate” link, webcomment_comments.html , webcomment_reviews.html , webmessage_view.html and some of the bibformat templates are modified, also a new bibformat element, bfe_translate.py , is created. 	
Implemented By	SRDC

Table 3.8: Implementation Description: RF48

Feature ID	RF57-58-61 (Repository Feature 57-58-61)
Name	The archive provides a ranking method based on the user rating of content (RF57) A user can rank archived content based on specific users’ content rating (RF58) The archive ranks blogs based on their views and downloads (RF61)
Effort Spent	1 Week
Modules Affected / Created	Bibrank

Description of the new feature

These three features have been implemented in the same development branch and add two different ranking method templates to the existing ones: “Number of Record Views” (record_view) and “Average Review Score” (average_score).

“Number of Record Views” ranking method calculates the number of the visits to the record. Each visit to the same record is counted as 1 for the visits occurred within a minute.

“Average Review Score” calculates the average scores of the records.

Implementation details

- 2 new ranking templates are added: **template_average_score.cfg** and **template_record_view.cfg**
- **template_record_view.cfg** has a parameter **time_interval** that decides the interval to delete consequent record views, i.e., it does not matter how many times a user views a record in a given time interval, it is counted only once.
- 4 new functions have been implemented in **bibrank_tag_based_indexer.py** :
 - **record_view** : executes **bibrank_engine** method for **record_view** ranking method
 - **record_view_exec** : Ranks total number of record visits without checking the user IP
 - **average_score** : executes **bibrank_engine** method for **average_score** ranking method
 - **average_score_exec** : Ranks average review score for records
- 2 new files have been created: **bibrank_record_view_indexer.py** and **bibrank_average_score_indexer.py**
 - **bibrank_record_view_indexer** contains the functions that are used for indexing visit counts of each record.
 - **bibrank_average_score_indexer** contains the function that are used for indexing average review score of each record.

Implemented By	SRDC
-----------------------	------

Table 3.9: Implementation Description: RF57-58-61

Feature ID	RF64 (Repository Feature 64)
Name	The archive offers the option to login using external (universal) credentials
Effort Spent	3 Weeks
Modules Affected / Created	MiscUtil, WebAccess, WebSession

Description of the new feature	
<p>This feature was implemented by BlogForever developers, but ported to next and integrated into Invenio core by Invenio developers. Therefore, now it is a part of Invenio and directly used in BlogForever repository. Description of the functionality of this feature can be found in D4.5.</p>	
Implemented By	SRDC

Table 3.10: Implementation Description: RF64

Feature ID	RF70 (Repository Feature 70)
Name	6 weeks
Effort Spent	WebAccess, WebSession
Modules Affected / Created	The archive can provide services under some cost using a billing system

Description of the new feature

This feature introduces “premium access to collections” service. With this feature admin can restrict collections for some cost for a finite/infinite time interval.

Premium packages can be managed easily from the admin panel.
(Administration ⇒ Configure Webaccess ⇒ Manage Premium Packages)

Admin may add premium packages for collections:

Add New Premium Package ×

Name

Details

Duration

Price

Collections

Blogs Comments

Current premium packages can be monitored, edited or deleted:

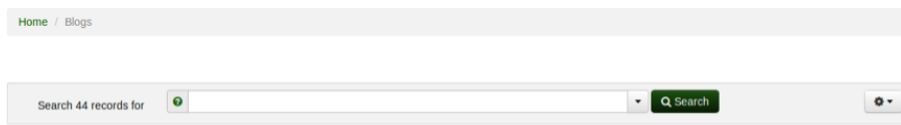
[Edit Premium Packages](#)
[Payment History](#)
[Premium Users](#)

ID	Name	Details	Duration	Price	Access			
1	Blogs and Comments	Display blogs and comments for 3 days!	3 Day	5.0 EUR	Blogs Comments			
2	Posts - 1	Displays posts for a month!	1 Month	19.0 EUR	Posts			
3	Pages	Pages for a year!	1 Year	49.99 EUR	Pages			
4	All in One	All in One unlimited membership!!!	Unlimited	149.99 EUR	Blogs Posts Comments Pages			

[Add new premium package](#)

- Edits the premium package (Displays same form as adding new one)
- Deletes the premium package
- Moves the premium package up and down respectively. The order of the premium packages can be changed through these buttons.

When premium packages are added for certain collections, these collections become restricted.



After edit or delete operations, if a collection loses its premium packages, that collection becomes unrestricted.

By clicking search button, if the user has not bought a premium package, the list of the premium packages related to that collection are shown.

Purchase a premium package

Name	Details	Duration	Price	Access to collections
<input type="radio"/> Blogs and Comments	Display blogs and comments for 3 days!	3 Days	5.0 EUR	Blogs Comments
<input type="radio"/> All in One	All in One unlimited membership!!!	Unlimited	149.99 EUR	Blogs Posts Comments Pages



To purchase a premium package, users can select a suitable package and payment method which are the following:

1) Credit card

Purchase a premium package with credit card

Premium Package Details

Name	Blogs and Comments
Details	Display blogs and comments for 3 days!
Duration	3 Days
Price	5.0 EUR
Access to Collections	Blogs Comments

Credit Card Information

Name on Card	<input type="text"/>
Card Number	<input type="text"/>
Expiration	1 <input type="text"/> 2013 <input type="text"/>
Security Code	<input type="text"/>
First Name	<input type="text"/>
Last Name	<input type="text"/>
Street	<input type="text"/>
City	<input type="text"/>
State / Province	<input type="text"/>
ZIP / Postal Code	<input type="text"/>
Country	AALAND ISLANDS <input type="text"/>

[Upgrade](#)

The user can buy premium packages with his/her credit card through the above form. This is the last screen before completing the transaction. After clicking the upgrade button, if the transaction fails, an error message is shown:

Credit Card Information

This transaction cannot be processed. Please enter a valid credit card number and type.

Otherwise, a confirmation page is displayed.

You have successfully purchased the premium package. Please note that your transaction ID is Paypal-5HG92527WD4206728

Your Premium Memberships

Your current premium membership status:

Collection	Status	
Blogs	Your membership will be expired in 2013-05-23 18:00:24	Extend your membership!
Posts	You don't have any premium membership to display!	Buy a premium package
Comments	Your membership will be expired in 2013-05-23 18:00:24	Extend your membership!
Pages	You don't have any premium membership to display!	Buy a premium package

2) PayPal Express Checkout

BlogForever Repository Demo

Your order summary

Descriptions	Amount
Posts - 1 Item description: Display posts for ... Item price: €19.00 Quantity: 1	€19.00
Item total	€19.00
Total €19.00 EUR	

Review your information

PayPal

[Continue](#)

Payment methods [Change](#)

PayPal Balance \$26.60 USD

PayPal Conversion Rate as of May 22, 2013: 1 U.S. Dollar = 0.714485 Euros

PayPal gift card, certificate, reward, or other discount [Redeem](#)
View [PayPal policies](#) and your payment method rights.

Contact information
alper_1347461937_per@srdc.com.tr

[Continue](#)

You're almost done. You will confirm your payment on BlogForever Repository Demo.

[Cancel and return to BlogForever Repository Demo.](#)

User may choose PayPal express checkout if s/he has a PayPal account. Clicking “Checkout with PayPal” button redirects the user to the PayPal page to login and confirm the transaction. After clicking “Continue Button”, user is redirected back to Invenio site, and confirms his/her order.

Home / Your Account / Your Premium Memberships / Confirm payment

Confirm payment

Name	Details	Duration	Price
Posts - 1	Display posts for a month!	1 Month	19.0 EUR
Total: 19.0 EUR			

User clicks the checkout with PayPal button and confirms the transaction.

You have successfully purchased the premium package. Please note that your transaction ID is Paypal-7JX13215RN579620K x

Your Premium Memberships

Your current premium membership status:

Collection	Status
Blogs	You don't have any premium membership to display! Buy a premium package
Posts	Your membership will be expired in 2013-06-21 15:52:49 Extend your membership!
Comments	You don't have any premium membership to display! Buy a premium package
Pages	You don't have any premium membership to display! Buy a premium package

Users may see their premium group memberships via “Your account page”:

Your Account

Filter widgets

Account ↗ ×

You are logged in as Demo

Premium Memberships ↗ ×

You are able to display 1 of total 4 premium collections.

Baskets WIP ↗ ×

You have 0 personal baskets ▾

Alerts WIP ↗ ×

You own the following 0 alerts ▾

Searches ↗ ×

You have made 0 queries. A detailed list is available with a possibility to (a) view search results and (b) subscribe to an automatic email alerting service for these queries.

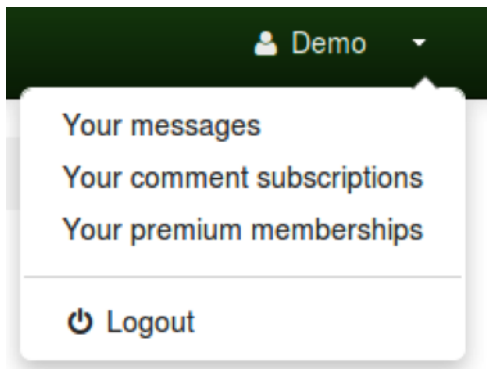
Messages ↗ ×

You have 0 new messages out of 0 messages.

Group ↗ ×

You are not involved in any group.

Users may display its detailed status of premium memberships by clicking the title of the “Premium Memberships” box or “Your premium packages” link on the top of the page.



“Your Premium Memberships” page consists of two main parts: “Premium membership status” and “Available premium packages”

Your Premium Memberships

Your current premium membership status:

Collection	Status
Blogs	Your membership will be expired in 2013-05-25 16:02:07 Extend your membership!
Posts	Your membership will be expired in 2013-06-21 15:52:49 Extend your membership!
Comments	Your membership will be expired in 2013-05-25 16:02:07 Extend your membership!
Pages	You don't have any premium membership to display! Buy a premium package

Available premium packages:

Name	Details	Duration	Price	Access to collections
<input type="radio"/> Blogs and Comments	Display blogs and comments for 3 days!	3 Days	5.0 EUR	Blogs Comments
<input type="radio"/> Posts - 1	Display posts for a month!	1 Month	19.0 EUR	Posts
<input type="radio"/> Pages	Pages for a year!	1 Year	49.99 EUR	Pages
<input type="radio"/> All in One	All in One unlimited membership!!!	Unlimited	149.99 EUR	Blogs Posts Comments Pages



On upper part, the user may display the expiration date of his/her premium memberships, extend it by the “Extend your membership!” or “Buy a premium package” links.

On lower part, user may see all of the available packages and purchase one of them.

The admin may see the transaction history and premium members from the admin panel:

Premium Area selection for WebAccess Admin

Edit Premium Packages Payment History Premium Users

Number of transactions : 2

Total revenue : 24.00 EUR

Transaction Time	User ID	User e-mail	Package ID	Price	Payment Method
2013-05-22 16:02:07	2	jekyll@cds.cern.ch	1	5.0 EUR	Credit Card
2013-05-22 15:52:49	2	jekyll@cds.cern.ch	2	19.0 EUR	Paypal

Premium Area selection for WebAccess Admin

Edit Premium Packages Payment History Premium Users

Total number of premium users: 1

User ID	User E-mail	User Nickname	Collection	Expiration Time
2	jekyll@cds.cern.ch	Demo	Blogs	2013-05-25 16:02:07
2	jekyll@cds.cern.ch	Demo	Comments	2013-05-25 16:02:07
2	jekyll@cds.cern.ch	Demo	Posts	2013-06-21 15:52:49

Give a gift!

The admin can disable paying with credit card, paypal express checkout or whole premium service from **invenio.conf** file. Credit card payment can be done by PayPal or Ogone payment gateways. Admin can select payment methods to use in **invenio.conf** . Also the API credentials of PayPal and Ogone can be set in this file.

If the admin disables the premium service, users cannot buy premium packages but restriction status of collections does not change. If the admin enables it back, users can buy premium packages again.

Implementation details

- **12 new variables added into **invenio.conf** :**
 - **CFG_PREMIUM_SERVICE** : **1** for enabling, **0** for disabling the premium service. If this variable is changed **1** to **0** , users cannot buy premium packages but restriction status of collections does not change. On the contrary, If this variable is changed from **0** to **1** , users can buy premium packages again.

- **CFG_TEST_PREMIUM_SERVICE** : **1** for using test servers of the payment gateways, otherwise **0** .
 - **CFG_CREDIT_CARD_PAYMENT_GATEWAY** : The payment gateway used for purchasing with credit card. It may have 3 options: **“paypal”** , **“ogone”** or **“”** (blank). If it is blank, paying with credit card becomes disabled.
 - **CFG_USE_PAYPAL_EXPRESS_CHECKOUT** : The variable to decide to use “Paypal Express Checkout” or not. **1** for enabling, **0** for disabling.
 - **CFG_PAYPAL_API_USERNAME** , **CFG_PAYPAL_API_PASSWORD** , **CFG_PAYPAL_API_SIGNATURE** , **CFG_PAYPAL_API_VERSION** : The credentials to use PayPal API.
 - **CFG_OGONE_API_PSPID** , **CFG_OGONE_API_USERID** , **CFG_OGONE_API_PSWD** : The credentials to use OGone API.
 - **CFG_PREMIUM_GROUP_SUFFIX** : The suffix of the premium group names.
- **4 new tables added into database:**
 - **premium** : Keeps the information (name, details, duration, price and display order) about premium packages.
 - **hstPAYMENT** : Keeps the payment history.
 - **premium_collection** : keeps the premium package - collection mapping.
 - **collection_accrole** : keeps the collection - role mapping
 - **If a collection needs a premium membership to be accessed, it is checked when displaying corresponding collection**

search function in **websearch_blueprint.py** is modified to accommodate billing system. It checks if the premium service is enabled and if there is a premium package to access that collection. If a user does not have a right to access that collection, the list of the premium packages that allows to access is displayed.
 - **The admin panel to manage premium packages is introduced**

With the admin panel, premium packages can be added, edited and deleted. In addition, their display order may be changed. Payment history can be monitored and the list of the users who is a member of the premium groups can be displayed in the admin page. To achieve these functionalities, necessary functions are added into **webaccess_admin_blueprint.py** and **webaccess_admin_premiumarea.html** is added to templates.
 - **WebPayment module is introduced** It handles the cases about premium service, contains necessary functions and classes. **webpayment.py** contains the following functions:

- [add_new_premium_package](#) : Adds a premium package to the database. Arranges the restrictions for collections of the new premium package.
- [edit_premium_package](#) : Edits the premium package. Arranges the restrictions for collections of the premium package.
- [fix_roles_and_authorizations](#) : Arranges the restrictions for the premium collections.
- [add_role_and_authorization](#) : Restricts the given collection. Adds a role and an authorization to restrict the collection.
- [grant_user_access](#) : Give a user the access to the collections restricted a given premium package.
- [gift_premium_package](#) : Gives a premium package to a user as gift.
- [get_possible_packages](#) : Returns the premium packages to display given collection.
- [get_package_collection_map](#) : Returns a dictionary that contains information about which premium package allows to access which collections.
- [register_payment_history](#) : Registers the transaction to payment history.
- [delete_premium_package](#) : Deletes the premium package with given ID. Arranges the restrictions of the collections that can be displayed by this premium package.
- [get_user_premium_membership](#) : Returns total number of premium packages that the user has purchased.
- [get_membership_expiration_dates](#) : Returns the expiration dates of the premium memberships of the user with given ID.

WebPayment module provides [webpayment_query.py](#) file which contains database related functions of WebPayment module. It also provides [webpayment_base.py](#) which includes the following required classes to implement payment gateways:

- [PaymentGatewayResponse](#) : This class is inherited from dict class. It ensures that the dictionary contains corresponding premium package, success state of the transaction, transaction ID, error messages and additional data if exists.
- [PaymentGateway](#) : This is an “abstract” class from which all of the payment gateways should be inherited. The fields which should be **overridden** are following:
 - * [SERVER](#) :The URL of the payment gateway API.
 - * [TEST_SERVER](#) :The test URL of the payment gateway.
 - * [_additional_inputs](#) :Some payment gateways require more data than the credit card information. This field is used to specify the additional information required to payment gateway.
 - * [name](#) :The name of the payment gateway.
 - * [_accept_types](#) :List of the credit card types that accepted.
 Types can be [PaymentGateway](#) . [VISA](#) ,

`PaymentGateway` . **MASTERCARD** , `PaymentGateway` . **DISCOVER** , `PaymentGateway` . **AMERICANEXPRESS** or `PaymentGateway` . **MAESTRO** constants.

If the payment gateway is used for purchasing with credit card, `process` method should be **overridden**. `process` performs the credit card transaction and return the response (`PaymentGatewayResponse`) including transaction ID if succeeded, error messages if failed.

If the payment gateway redirects the user to a 3rd party site to complete the payment, `construct_checkout_url` , `get_transaction_details` and `complete_transaction` methods should be overridden:

- * `construct_checkout_url` :This method should return the response with 3rd party site URL to checkout in the `data` field of the response. If fails, it should return response with error messages. The return URL when calling the payment gateway api should be in the form of **CFG_SITE_SECURE_URL** /webpayment/re-view?id_package=self.premium_package.id&payment_method=self.name. If you want to show the user what s/he is buying, the endpoint should be **review** and override `get_transaction_details` , or you may complete the payment after returning from 3rd party site by setting endpoint as **complete**.
- * `get_transaction_details` : Checks if the transaction is appropriate for the payment gateway. If it is, it returns the HTML code of the button for completing the transaction in `data` key of the response. Otherwise, it should return a response with error messages. If you want to skip this step, just do not override this function.
- * `complete_transaction` : Should complete the transaction. If the transaction is succeeded, it should return a response with transaction id. Otherwise, it should return a response with error messages.

`webpayment_paypal.py` and `webpayment_ogone.py` modules contain classes derived from `PaymentGateway` . These two both contain necessary methods to buying with credit card. In addition, `webpayment_paypal.py` contains methods for completing payment in 3rd party site (PayPal Express Checkout).

- Payment methods can be managed from `webpayment_config.py` and implemented methods should be added into this file.

CFG_CREDIT_CARD_PAYMENT_METHODS is a dictionary which contains the classes implemented for purchasing

with credit card. Its keys are the name of the payment methods (with uppercase letters) and values are only the corresponding classes (not instances).

CFG_PAYMENT_METHODS is also a dictionary which contains all the payment methods for purchasing with 3rd party site (like PayPal Express Checkout). Its values are the name of the payment methods and values are the corresponding class. In addition it has the key **cc** for credit card payment whose value is determined by **invenio.conf**.

- **New modules are added for the web interface**

- **webpayment_user_settings.py** is introduced to add “Your premium memberships” box to “Your account” page.
- **webpayment_blueprint.py** and templates **webpayment_display.html**, **webpayment_index.html**, **webpayment_packages.html**, **webpayment_review.html**, **webpayment_upgrade.html** are introduced.

Implemented By	SRDC
-----------------------	------

Table 3.11: Implementation Description: RF70

Feature ID	RF71 (Repository Feature 71)
Name	The archive provides a personalized annotating and highlighting tool for users
Effort Spent	4 weeks
Modules Affected / Created	WebSearch, WebSession, Webstyle, Miscutil, Bibformat

Description of the new feature

This feature enables users to highlight certain parts of the records and add notes on them. It can be enabled/disabled for each collection, So an icon is added to activate the feature at the right of the page. This icon is only visible when a user logged in, otherwise it is not displayed.

Clean Dissertations?



that will propel me through the end of the process and one that I
nd, here is the beginning of some thoughts ...

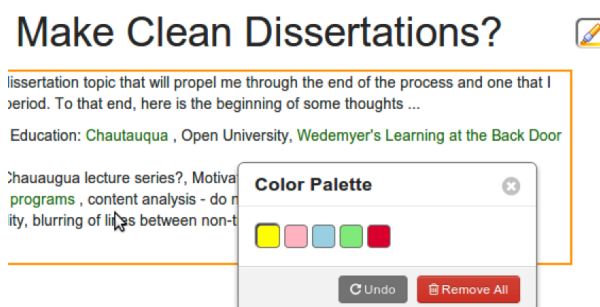
.tauqua , Open University, Wedemeyer's Learning at the Back Door

When the icon is clicked, a color palette containing highlight colors appears and previously saved highlighted items are loaded. 5 colors are chosen as default. These can be changed from **invenio.conf**. Palette has also two more options, **remove all** and **undo**.

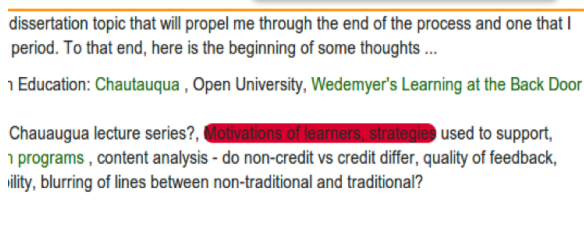
Actions which can be undone by **undo** operation:

- Creating Highlight
- Extending Highlight
- Deleting Highlight
- Adding Note
- Editing Note
- Deleting Note
- Remove All

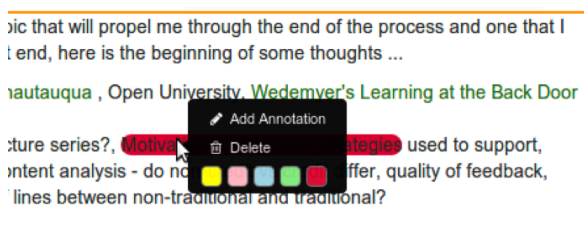
After activating highlight, an orange box appears around the record, which defines the borders of the editable area.



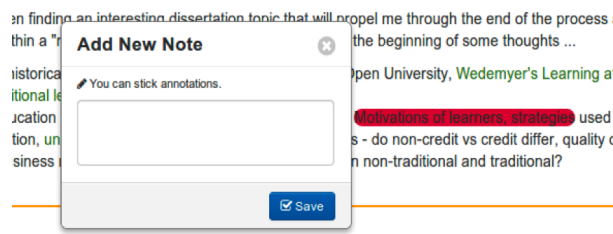
The user is able to highlight the record by holding mouse button and dragging the cursor.



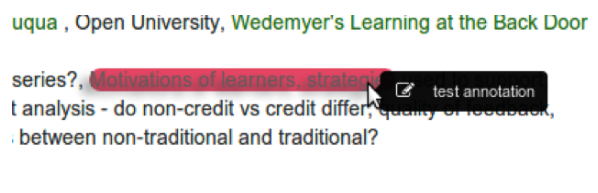
When mouse is rolled over an highlighted area, an **edit menu** is displayed. It provides of 3 options which are **Add Annotation**, **Delete highlight** and **Change Color**.



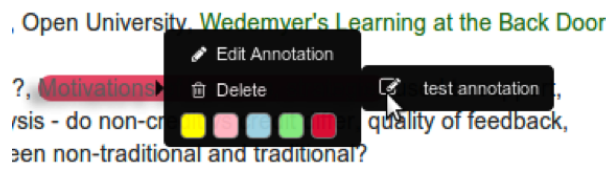
A box is displayed when user clicks on **Add Annotation** so that, the user can enter his/her annotation and save it to the repository.



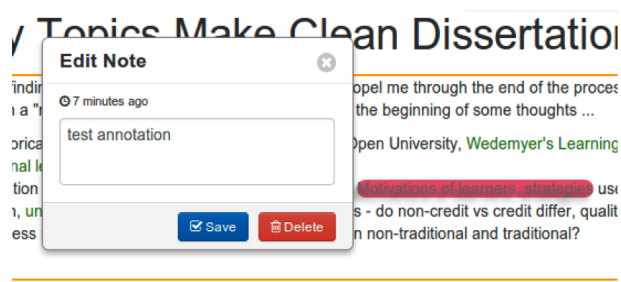
After the note is saved, the text of the corresponding highlight becomes shadowed. When you hover the mouse through annotated section, a tooltip showing the note appears.



When the icon at the leftmost of the annotation tooltip is clicked, a menu similar to the edit menu appears. It provides of 3 options namely **Edit Annotation**, **Delete Highlight** and **Change Color**.



When a highlighted text that has annotation or **Edit Annotation** link illustrated above is clicked, the corresponding note can be seen. It can be edited and saved or removed completely.



When the color palette is closed, a new icon to reopen the color palette appears just below the highlight activation icon.

Clean Dissertations?

hat will propel me through the end of the process and one that I
d, here is the beginning of some thoughts ...

lauqua , Open University, Wedemyer's Learning at the Back Door

a series?, **Motivations of learners, strategies** used to support,
nt analysis. do non credit vs credit differ quality of feedback



Highlighted texts can be extended. If the selected text contains any highlighted part and if the selection color is same with its color, they are merged. Moreover, two neighbor highlighted parts are merged, if they have same color after a color change is applied.

Since there are different nodes in HTML based records, the user selection is resulted in divided highlighted parts. When the user adds an annotation to one of them, it is also added to all highlight nodes with the same identifier. Highlighted parts still seem divided, but they are logically unified.

If selection contains a part of a MathJax expression, it highlights whole MathJax element in order not to ruin structure of MathJax.

Implementation details

- This feature can be enabled for specific collections by adding collection name to **CFG_HIGHLIGHT_AND_ANNOTATION_COLLECTIONS**. **Posts** and **Comments** are in this list by default.
- To enable highlighting feature for any content, it is just enough to surround the content with `<div class='highlightable'></div>` tags.
- **selectionRange** object is mainly used for highlight. Therefore, this feature is applicable on Internet Explorer from version 9.
- To highlight the selected text, an element with tag **highlight** is inserted around the selection.
- A **highlight** element does not contain any other **highlight** elements.
- The main aim is keeping the **highlight** elements with minimum depth in DOM tree. To do so, after each selection, the recently inserted **highlight** nodes are traversed. If all its siblings are **highlight** nodes, then just highlight the parent instead of all children. This reduces the costs to save highlights.
- For each text selection, the resulted **highlight** nodes are given same identifier to logically unify them. This identifier is unique for each selection. The identifiers are also used as annotation ID's.
- Highlights are saved as serialized JSON string. On page load, the DOM tree is reconstructed with this JSON object. An example json string:

```

{
  "leaves":{
    "43":[{"s": 22, "e": 35,"a": 0, "id": "2",
           "n": 0, "c": "rgb(255, 255, 0)"}],
    "73":[{"s": 357, "e": 406, "a": 1, "id": "0",
           "n": 0, "c": "rgb(255, 255, 0)"}]
  },
  "nodes":{
    "57":{"c": "rgb(255, 255, 0)", "id": "1", "a": 0}
  }
}

```

'leaves' key corresponds to highlight elements around text nodes. It has keys as numbers which are ids of each dom element. These ids are assigned when highlight mode is on. For the highlight nodes around text nodes, their parent nodes are used as keys i.e 43, 73. Highlight nodes under them are listed in dictionary format.

- 's' denotes start position
- 'e' denotes end position
- 'a' denotes whether that highlight element has an annotation. if yes **1** , otherwise **0** values are used.
- 'id' denotes id '**high_anno_id**' attr value of the **highlight** nodes. It is used to keep track of seperated highlight nodes which are result of a selection. For example, the user makes a selection starting from a **<p>** element and ending to another **<div>** . There are more than one highlight nodes as a result of this selection. This value is also used for annotation id.
- 'n' denotes the child number that the indices are valid on.
- 'c' denotes the color.

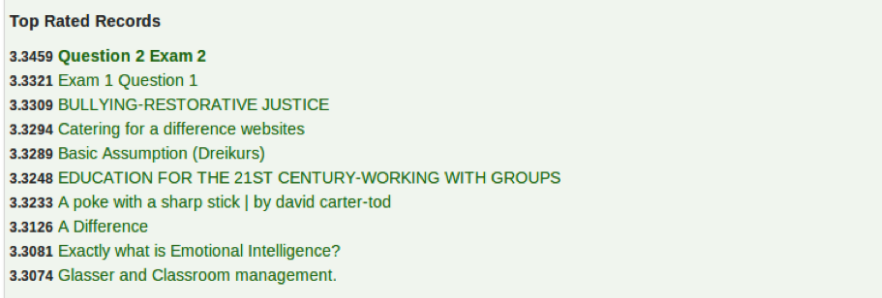
'nodes' key corresponds to a node level highlighting information. For nodes, there is no need to save indices, it is enough to keep color information, since each node will have a unique identifier (57 in example above) and they will be highlighted directly. The 'c' , 'id' , and 'a' keys have same meanings as above.

- Each change is directly saved into the database.
- Before loading highlights, record's last modification date and highlight date are compared to understand whether the record is changed or not. If the comparison indicates a change on the record, the user is warned about possible distortion on highlights.
- Two new tables namely **bibrec_highlights** and **bibrec_annotations** have been inserted into database to keep highlights and annotations.
- In **websession_webinterface.py** , **savehighlights** , **loadhighlights** , **saveannotation** , **getannotation** and **removeannotation** methods have been added for communication between server and client sides.

<ul style="list-style-type: none"> • In <code>webuser.py</code> , <code>check_bibrec_modification_date</code> , <code>set_user_bibrec_annotation</code> , <code>get_user_bibrec_annotation</code> , <code>delete_user_bibrec_annotation</code> , <code>set_user_bibrec_highlights</code> , <code>get_user_bibrec_highlights</code> functions have been implemented for database transactions. • In <code>websession_templates.py</code> , <code>tmpl_highlight_tools</code> , <code>tmpl_annotation_box</code> , <code>tmpl_color_palette</code> methods have been implemented to create HTML codes for highlight tools such as color palette and annotation box. • In <code>dateutils.py</code> , <code>difference_between_times</code> function has been inserted to calculate elapsed time in units such as second, minute, hour etc. • <code>CFG_COLOR_PALETTE</code> parameter has been added to define highlight colors. As default, four colors have been set. • Also in <code>inveniocfg.py</code> and <code>search_engine.py</code> , some minor modifications have been made. 		
<table border="1"> <tr> <td>Implemented By</td> <td>SRDC</td> </tr> </table>	Implemented By	SRDC
Implemented By	SRDC	

Table 3.12: Implementation Description: RF71

Feature ID	RF73 (Repository Feature 73)
Name	The archive recommends blogs to users based on the ratings and preferences
Effort Spent	1 Week
Modules Affected / Created	BibRank, WebSearch
Description of the new feature	
<p>This feature introduces a new ranking method to rank the records by their weighted averages.</p> <p>A portalbox which shows the Top Rated Records has been added into main page.</p>	



A portalbox which shows last added records has been added into main page.



Implementation details

- Portal boxes have become associated with ranking methods.
 - `bibrank_portalbox` table, which keeps which ranking method is related with which portal box in which language, has been inserted into database.
 - `update_bibrank_portalbox` and `drop_bibrank_portalbox` functions have been added into `bibrank_record_sorter.py`.
 - * `update_bibrank_portalbox` function updates the portal boxes when bibrank is run.
 - * `drop_bibrank_portalbox` function removes the entries related to given bibrank method when either the ranking method is deleted or the variable keeping the number of records shown in portalbox is set to `0`.
 - For portalboxes `bibrank_portalbox.html` template is added.
- Ranking with “Weighted Average” has been introduced.
 - To rank the records by their weighted average, `bibrank_weighted_average_indexer.py` module has been created. This module contains the function `weighted_average_to_index` which calculates the weighted average with “Bayesian estimate” which is the following formula:

$$\frac{N}{N+m} * A + \frac{m}{N+m} * G$$

where

- * **N**: the number of reviews of corresponding record
 - * **m**: minimum number of reviews required to calculate the rank of the record
 - * **A**: Average score of corresponding record
 - * **G**: Average score of all of the records
- **bibrank_weighted_average_template.cfg** containing the parameters for the ranking method has been created. These parameters are:
 - * **show_relevance** : **1** to show the score on search page, **0** otherwise.
 - * **minimum_review_number** : minimum number of reviews required to be ranked
 - * **display_on_portalbox_count** : the number of the records will be displayed on the portalbox. If it is **0** , portalbox disappears and entries related to that ranking method is removed from database.
 - Archived content indexer has been added as a ranking method to create “Recently Added Records” portalbox.
 - **bibrank_archived_content_indexer.py** has been introduced to rank the records in a time interval.
 - **template_recently_archived_content.cfg** containing the parameters for the ranking method. These parameters are:
 - * **latest_records_number** : the number of the lastly added records, if **0** , ranks all of them.
 - * **date_type** : **creation** for ranking by creation date, **modification** for ranking by modification date.
 - * **start_date** : the beginning of the time interval.
 - * **end_date** : the end of the time interval.
 - * **interval** : the sql like time interval. (i.e. **3 HOUR** , **1 DAY**)
 - * **display_on_portalbox_count** : the number of the records will be displayed on the portalbox. If it is **0** , portalbox disappears and entries related to that ranking method is removed from database.
 - Some modifications have been occurred in **bibrank.py** and **bibrank_tag_based_indexer.py** to accommodate new ranking method.

Implemented By	SRDC
----------------	------

Table 3.13: Implementation Description: RF73

3.2.4 Features ported to the new interface

This list presents those features of which front-end has been notably improved by introducing the new technologies, Jinja2 templates and Twitter Bootstrap, described in Section 2.3. The improvements that the user interface has gone through after adapting the user interface elements to the new technologies, can be appreciated in Figures 2.20, 2.21, 2.22, 2.23 and 2.24.

- RF4 - Bibformat output templates to display blogs and blog posts differently
- RF9 - The archive stores and displays accordingly all record metadata received from the spider
- RF17 - The archive displays a disclaimer about the originality of the content
- RF23 - The archive stores the comments of blog posts and displays them as part of the blog posts
- RF24 - Links to other sources within the blog posts are displayed separately
- RF25 - The archive displays the tags of blogs and blog posts
- RF28 - The archive displays the author of blog posts and comments
- RF47 - Description of how to cite archived records is presented prominently with each record

3.2.5 Postponed features

The benefits that basing the repository on the new version of Invenio gives to the platform were explained in Section 2.3, but of course it comes with a work overload that obviously has delayed the implementation of the features. The remaining 3 months were planned for the integration of the spider and the repository components, but both, the spider and the repository design, and development, have taken this into account since the beginning of the project. This results in having a complete design of the communication between the 2 components and an implementation already working, as can be verified in the Case Studies conducted by WP5 so far in the 3 test servers: BF1, BF3 and BF5. This leaves us enough of time to finish the development of the following features within the project deadlines.

- RF14 - Descriptive statistics are offered by records
- RF34 - The archive displays and suggests similar records to the user
- RF36 - The archive identifies and stores the topic of blogs and blog posts to let users navigate through the archive by topic
- RF46 - Users can create personal collections of their favourite blogs
- RF56 - The archive provides a journal view of the new blog
- RF78 - The archive displays content after filtering it with user preferences
- RF87 - The archive transforms the SIPS received from the spider to AIPs
- RF89 - The archive carries out the normalization and/or migration of the media attachments

3.3 Features not retained

In order to concentrate on blog-specific must-have repository features, some decisions have been taken regarding nice-to-have feature requirements identified in D4.1 [5] user survey. The list of non-retained features is presented below.

The following features were not retained because the complexity of the topic is falling out of scope of blog archiving platform. The solutions developed outside of platform could be integrated with the blog archive in later stages.

- RF42 - The archive extracts bibliographic metadata from content embedded in blogs
- RF68 - The archive provides information diffusion analysis mechanisms
- RF75 - The archive can do sentiment analysis on the content

The following features were not retained because sufficient alternative approaches exist:

- RF50 - The archive offers the option to disseminate newly archived content in external social platforms (alternative: instead of pushing to social platforms, the apps can pull from the blog archive platform via customised RSS feeds)
- RF49 - The archive distinguishes institutional/corporate blogs from personal blogs (alternative: user tags blogs as corporate upon submission, see RF31 description in Section 3.2 of D4.5 [2])

The following features were not retained because of low demand and high complexity of the topic:

- RF76 - The archive detects content's originality and ranks it accordingly

Chapter 4

Conclusions

The aim of this report was to present the implementation process of the final BlogForever Weblog Digital Repository Component. In Chapter 2 the design of D4.4 [3] has been compared to the final implementation. In Chapter 3 the implementation details were described in more detail. In general, the implementation has closely followed the original design.

The agile development and testing of early repository prototypes through collaboration with WP5 case studies has led to putting higher priority for features related to improving user experience. As a result the decision has been taken to base the BlogForever repository on a newer version of Invenio that better answers these needs. The results were positive not only from improving the user experience, but also from the point of view of the flexibility of the platform, making it easier to extend the source code and to facilitate its long-term maintainability. As a drawback, using newer version of Invenio implied extra work to port previously developed features to the new system. Furthermore, working on top of cutting edge source code consumed extra time due to its initial lack of stability. As a result, the implementation of several features has been delayed.

The final parts of WP4 are devoted to implementing delayed features and to finalising the integration between the Spider and Repository components. Thanks to the agile development and testing done collaboratively in WP4 and WP5, the integration of Spider and Repository components has been happening progressively for the user case studies, leaving the remaining workload perfectly affordable to implement delayed features within project deadlines.

Some feature requirements were identified as “low priority” or falling out of scope for the current project. The feature definition threshold in Task 4.1 to accept a sentence from a user interview as a requirement was found to be too permissive, leading to niche features (e.g. detect content originality). Some of the repository features as depicted in the Description of Work were too ambitious to implement within project deadlines and given resources (e.g. sentiment analysis). Some of the features are common to blog and non-blog repositories (e.g. extraction of metadata from attached content) and may come to the blog archive platform in that way.

References

- [1] V. C. Venkatraman, “Enhancing scalability of invenio - digital library software,” tech. rep., École Polytechnique Fédérale de Lausanne (EPFL), May 2013.
- [2] S. Postacı, A. Çınar, G. B. Laleci, J. García Llopis, R. Jiménez Encinar, V. Banos, and A. P. and, “*D4.5: Initial Weblog Digital Repository Prototype*,” work package, SRDC Yazılım Arastırma ve Gelistirme ve Danışmanlık Ticaret Limited Şirketi, Jan. 2013. Work Package Four Deliverables.
- [3] J. García Llopis, R. Jiménez Encinar, *et al.*, “*D4.4: Digital Repository Component Design*,” work package, European Organization for Nuclear Research (CERN), Nov. 2012. Work Package Four Deliverables.
- [4] “Blogforever,” tech. rep., 2011.
- [5] H. Kalb, N. Kasioumis, J. García Llopis, S. Postacı, and S. Arango-Docio, “*D4.1: User Requirements and Platform Specifications Report*,” work package, B. Consortium (Ed.): Technische Universität Berlin, Dec. 2011. Work Package Four Deliverables.
- [6] K. Stepanyan, M. Joy, A. Cristea, Y. Kim, E. Pinsent, and S. Kopidaki, “*D2.2: Weblog Data Model*,” work package, B. Consortium (Ed.): University of Warwick (UW), Oct. 2011. Work Package Two Deliverables.
- [7] Y. Kim, S. Ross, K. Stepanyan, E. Pinsent, P. Sleeman, S. Arango-Docio, H. Kalb, *et al.*, “*D3.1 Preservation Strategy Report*,” work package, Y. Kim & S. Ross (Eds.): University of Glasgow, Sept. 2012. Work Package Three Deliverables.
- [8] M. Rynning, “*D4.3: Initial Weblog Spider Prototype*,” work package, CyberWatcher, Sept. 2012. Work Package Four Deliverables.
- [9] M. Rynning, “*D4.6: Final Weblog Spider Component*,” work package, CyberWatcher, Apr. 2013. Work Package Four Deliverables.
- [10] S. Arango-Docio, P. Sleeman, E. Pinsent, G. Gkotsis, T. Farrell, S. Kopidaki, and M. Rynning, “*D5.1: Design and Specification of Case Studies*,” work package, University of London, June 2012. Work Package Five Deliverables.
- [11] “Invenio documentation.” <http://invenio-demo.cern.ch/help/admin/>. [Retrieved November 25, 2012], 2012.

- [12] J. Kunčar and T. Šimko, “New Features and Technologies in Current and Future Invenio Versions,” tech. rep., European Organization for Nuclear Research (CERN), Oct. 2012.
- [13] H. Kalb, Y. Kim, and P. Lazaridou, “*D2.3: Weblog Ontologies*,” work package, A. I. Cristea & M. Joy (Eds.): Technische Universität Berlin (TUB), May 2012. Work Package Two Deliverables.
- [14] H. Kalb, P. Lazaridou, and M. Trier, “*D4.2: Weblog spider component design*,” work package, B. Consortium (Ed.): Technische Universität Berlin (TUB), June 2012. Work Package Four Deliverables.
- [15] S. Arango-Docio, V. Banos, K. Stepanyan, and M. Joy, “*D2.1: Survey Implementation Report*,” work package, University of London, Aug. 2011. Work Package Two Deliverables.
- [16] “Invenio Project.” <http://invenio-software.org/>. [Retrieved November 11, 2012], 2012.

