Thesis
Extending the Core Knowledge 2.0 Platform

Supervisor: Dr Hans-Wolfgang Loidl
Second lecturer: Phil Trinder
Student: Victor Godayer HW ID: H00107316

August 15, 2012
Declaration

I, Victor Godayer, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed:
Date:
Acknowledgments

First of all, I would like to thank Dr Hans-Wolfgang Loidl, who supervised me during this thesis and has been always helpful. I would also like to thank Amir Ghaffari who has been eager to give us technical information and explanations on the client side application. I also thank Alexios Karapetsas who is being very cooperative and thus ease our work on this project.
Abstract

E-Publishing and knowledge dissemination has been increased practice through scholarly web repositories since the Web 2.0 have emerged. The Web 2.0 and its features have turned out to be one of the most powerful architecture to spread and harvest knowledge even out of the boundaries of the scholarly knowledge dissemination.

In the context of scientific publication, even though these technologies are marvelous tools to lower edition and publication costs, the quality of the knowledge aimed to be disseminated on the web still relies on the principle of peer reviewing made by pairs gathered under an editorial board.

One of the new aspect to use Web 2.0 for knowledge dissemination is its strength to leverage bi directional communication between the producer and the consumer. By its features it allows to instantiate formative and summative functionalities such as rating and commenting systems respectively.

The CKP 2.0 is an ongoing project aiming to be a scholarly web repository. Its main goals is to implement the conventional peer review process, and the modern approach of public reviewing for peer-reviewed and approved papers thanks to the features the Web 2.0 comes with.

To stay in the Web 2.0 compliance sphere, the CKP 2.0 is implemented as a web service. Therefore, this thesis focuses on performance considerations. Features such as caching, and cache prefetching will be added atop the existing implementation of CKP 2.0 to provide a high quality of service for the client side application. This later, named Front End, is also a project running in parallel with the CKP 2.0.
# Contents

## Contents

## List of Figures

## List of Tables

1. **Introduction**
   1.1 Abstract ........................................ 12
   1.2 What is Core Knowledge Platform 2.0 ........................ 13
   1.3 Context .......................................... 13
   1.4 Aims ........................................... 14
   1.5 Objectives ...................................... 15
   1.6 Structure of document .................................. 16

2. **Literature Review**
   2.1 Web 2.0 ........................................... 17
       2.1.1 Characteristics .................................. 17
       2.1.2 Features ........................................ 18
           2.1.2.1 Commenting ..................................... 18
           2.1.2.2 Rating .......................................... 19
           2.1.2.3 Social Tagging .................................. 20
       2.1.3 Web Services ..................................... 21
           2.1.3.1 Definition and Purpose ......................... 21
           2.1.3.2 Web Service Approach Comparison .............. 22
           2.1.3.3 The choice of REST for CKP 2.0 ............... 23
       2.1.4 External Interfaces ................................ 24
           2.1.4.1 Open Archive Initiative (OAI) .................. 24
           2.1.4.2 Google Scholar .................................. 25
   2.2 Performances ........................................ 26
       2.2.1 Motivations ..................................... 26
       2.2.2 Techniques Overview ............................... 26
       2.2.3 Web Server Gateway Interface ....................... 27
       2.2.4 Cache ........................................... 27
       2.2.5 Django and Memcached ............................... 29
       2.2.6 Other potential techniques to enhance performance .......................... 29
       2.2.7 CKP2.0-Specific performance possibilities ............ 30

5
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.1</td>
<td>User - Tag - Information</td>
<td>20</td>
</tr>
<tr>
<td>2.1.2</td>
<td>User - Tag - User</td>
<td>20</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Information - Tag - Information</td>
<td>20</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Roles and underlying concepts of a Web Service</td>
<td>22</td>
</tr>
<tr>
<td>2.1.5</td>
<td>OAI-PMH General Idea</td>
<td>25</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Peer Review Flow Chart</td>
<td>31</td>
</tr>
<tr>
<td>5.2.1</td>
<td>System architecture</td>
<td>38</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Django application structure</td>
<td>38</td>
</tr>
<tr>
<td>5.5.1</td>
<td>New data model</td>
<td>40</td>
</tr>
<tr>
<td>5.9.1</td>
<td>Prefetching application process</td>
<td>42</td>
</tr>
<tr>
<td>5.9.2</td>
<td>Prefetching work-flow</td>
<td>43</td>
</tr>
<tr>
<td>5.10.1</td>
<td>Search - CKP work-flow</td>
<td>44</td>
</tr>
<tr>
<td>5.10.2</td>
<td>OAI service provider interface design</td>
<td>47</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Prefetcher engine and its policies</td>
<td>52</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Gantt Chart</td>
<td>64</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Resource distribution</td>
<td>66</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Summary (include RelatedPublications policy)</td>
<td>66</td>
</tr>
<tr>
<td>8.2.3</td>
<td>No cache and no prefetching</td>
<td>67</td>
</tr>
<tr>
<td>8.2.4</td>
<td>Prefetching with no policy and objects cached</td>
<td>68</td>
</tr>
<tr>
<td>8.2.5</td>
<td>Prefetching with PageRendering policy</td>
<td>68</td>
</tr>
<tr>
<td>8.2.6</td>
<td>HTTP Responses cached</td>
<td>69</td>
</tr>
<tr>
<td>8.2.7</td>
<td>Search and Next Search prefetched</td>
<td>70</td>
</tr>
<tr>
<td>8.2.8</td>
<td>Search and Resumption Token refining</td>
<td>71</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Platform comparison</td>
<td>32</td>
</tr>
<tr>
<td>8.2</td>
<td>Comparison of publications extraction methods CKP/Pure</td>
<td>73</td>
</tr>
<tr>
<td>8.3</td>
<td>Requirements summary</td>
<td>74</td>
</tr>
<tr>
<td>10.1</td>
<td>Author</td>
<td>77</td>
</tr>
<tr>
<td>10.2</td>
<td>Comment</td>
<td>77</td>
</tr>
<tr>
<td>10.3</td>
<td>Esteem</td>
<td>78</td>
</tr>
<tr>
<td>10.4</td>
<td>Keywords</td>
<td>78</td>
</tr>
<tr>
<td>10.5</td>
<td>Esteem</td>
<td>78</td>
</tr>
<tr>
<td>10.6</td>
<td>PaperGroup</td>
<td>78</td>
</tr>
<tr>
<td>10.7</td>
<td>PeerReview</td>
<td>79</td>
</tr>
<tr>
<td>10.8</td>
<td>Publication</td>
<td>79</td>
</tr>
<tr>
<td>10.9</td>
<td>Rating</td>
<td>79</td>
</tr>
<tr>
<td>10.10</td>
<td>ReferenceMaterials</td>
<td>80</td>
</tr>
<tr>
<td>10.11</td>
<td>ResearchArea</td>
<td>80</td>
</tr>
<tr>
<td>10.12</td>
<td>Tags</td>
<td>80</td>
</tr>
<tr>
<td>10.13</td>
<td>UserProfile</td>
<td>80</td>
</tr>
<tr>
<td>10.14</td>
<td>Vote</td>
<td>81</td>
</tr>
</tbody>
</table>
## List of Algorithms

| 6.1 Virtual namespace retrieval | .................................................. 50 |
| 6.2 Cached page retrieval | .................................................. 50 |
| 6.3 Cached pages invalidation | .................................................. 51 |
| 6.4 Author search | .................................................. 55 |
| 6.5 Date filtering | .................................................. 55 |
| 6.6 Resource combination search | .................................................. 56 |
| 6.7 User search | .................................................. 57 |
| 6.8 Related papers | .................................................. 57 |
| 6.9 XML parsing | .................................................. 60 |
### Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASP</td>
<td>Active Server Pages</td>
</tr>
<tr>
<td>CGI</td>
<td>Common Gateway Interface</td>
</tr>
<tr>
<td>CKP</td>
<td>Core Knowledge 2.0 Platform</td>
</tr>
<tr>
<td>COM/COM+</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create Read Update Delete</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Model</td>
</tr>
<tr>
<td>DNA</td>
<td>Distributed interNet Applications Architecture</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DOI</td>
<td>Digital Object Identifier</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Description Language</td>
</tr>
<tr>
<td>ISBN</td>
<td>International Standard Book Number</td>
</tr>
<tr>
<td>JSON</td>
<td>Javascript Object Notation</td>
</tr>
<tr>
<td>LFU</td>
<td>Last Frequently Used</td>
</tr>
<tr>
<td>LRU</td>
<td>Last Recently Used</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>OAI</td>
<td>Open Archives Initiative</td>
</tr>
<tr>
<td>OAR</td>
<td>Open Access Repositories</td>
</tr>
<tr>
<td>OOP</td>
<td>Object Oriented Programming</td>
</tr>
<tr>
<td>ORM</td>
<td>Object Relational Mapper</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PEP</td>
<td>Python Enhancement Proposals</td>
</tr>
<tr>
<td>PHP</td>
<td>Hypertext Preprocessor</td>
</tr>
<tr>
<td>PMID</td>
<td>PubMed identifier or PubMed unique identifier</td>
</tr>
<tr>
<td>RBAC</td>
<td>Role-Based Access Control</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>ROAR</td>
<td>Registry of Open Access Repositories</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WBS</td>
<td>Work Breakdown Structure</td>
</tr>
<tr>
<td>WSD</td>
<td>Web Service Description</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSGI</td>
<td>Web Server Gateway Interface</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Abstract

Academic publication is a practice known and used since 1665, when “Le journal des savants” was first published (Jinha [2010]). It is considered as a basic principle to keep researchers up to date each other from their work. Therefore, they can use new shared knowledge to conduct farther their own work.

Nevertheless, the quality of academic papers must be checked before they are allowed to be published. One of the principles acting as a quality filter is a process called peer review. In this process, peers are mandated to assess publication of a scientist work. Selecting these peers, submitting to them the paper to get their quality estimation of this latter, publishing on a wide scale the paper in case of acceptation by the community and all the administrative overhead takes a lot of time.

Since the widening use of the Internet and Web 2.0, the traditional publication process has been seen challenged by those new technologies lowering production cost, process overhead and hugely multiplying the knowledge dissemination possibilities. Helped by the exponential growing output of publication (Jinha [2010]), the Web 2.0 appeared to be also a very good tool to relief traditional peer review actors by its ubiquitousness.

The Core Knowledge Platform 2.0 (CKP 2.0) aims to be a web service based framework for dissemination of quality controlled knowledge enhanced by Web 2.0 technologies. Specifically, the main goals will be to include a peer review management system and also a rating and feedback systems on published papers. A focus on these motivations is to improve and speed-up the quality assessment. Besides, a particular attention will be paid to implement a prefetching system for the cache in order to speed up the server response time for common repeated queries made by the client application.

All those systems must be made available to the client application, the Front End, which is a Rich Internet Application (RIA) providing the final user interface trough a web site. A deeper description of the aims and objectives of this project is given in sections [1.4 on page 14] and [1.5 on page 15].
1.2 What is Core Knowledge Platform 2.0

The Core Knowledge Platform 2.0 (CKP 2.0) is a web service based framework for knowledge dissemination using Web 2.0 technology (a thorough explanation of what a web service is given in section 2.1.3). The CKP is the server component and the service it provides is consumed by the complementary client side which is the Front End application, a Rich Internet Application. These both components aim is to interact with each other in order to provide the following features to the end user:

▷ Allow knowledge management by allowing users to upload documents.

▷ Provide a quality assessment system to control the quality of a paper before and after it is published. The concrete instances of these ideas are:

- An online peer review process system. This will include sub features such as role management to insure restrictive access, process steps and statuses on a publication resource.

- A system allowing readers to use an esteem based rating system on papers, and comments. Rating system will be made of ranged grade to give to a publication as well as votes. This latter will apply for publications and comments.

- An esteem system to give more judgment weight to a more experienced user than a new registered user. This system will be a complete backend of the CKP.

▷ Employ Web 2.0 features to facilitate knowledge dissemination:

- Insure bi-directional communication during the peer review process between referees and authors

- Leverage feedbacks on published papers, thanks to comments posted by the readers

▷ Harness metadata related to a publication from other existing platforms.

In the following section we will discuss the achievement of the upper described features. A simple page gathers the functionalities available on the platform. It can be found at this URL: [http://www2.macs.hw.ac.uk/ckp/functionalities/](http://www2.macs.hw.ac.uk/ckp/functionalities/)

1.3 Context

The CKP 2.0 is a project which has started to be developed the previous year by Florian Bergmann. The Front End application (client component) is being developed from the past year toward now by Amir Ghaffari. Here we will describe the features implemented in this previous work from the top level specification set given in the section 1.2 for both server and client components.

▷ Server side

- A robust and extensible data model has been built for storage of publications and their classic metadata.
The database interface is insured with the data representation (XML).

Some basic form of peer reviewing has been implemented but needs to be extended.

Bergmann’s implementation uses Web 2.0 technologies such as comments only for published papers. However the bi-directional communication has to be implemented for the peer review process.

The system provides user management and administration interface systems. Management of roles and permissions will be available after their implementation based on the Role-Based Access Control (RBAC).

A rudimentary interface to external platforms is implemented but does not work: a non-generic interface with CiteSeerX\footnote{http://citeseer.ist.psu.edu/index} is established to screen scrap data only if a valid Digital Object Identifier (DOI) is present in a publication resource. The concrete API of CiteSeer is not used.

**Client side**

- Upload of document is not implemented.
- Tagging and keywords systems are also implemented and available respectively for reader and publication submitter.
- The quality assessment (peer review) and bi-directional communication during this process are not implemented. The rating system relies on the simple implementation of rating and esteem on the server side which will change.

A more thorough description of what has been implemented on the client side is available in the Alexios Karapetsas report who works on behalf of Amir Ghaffari.

### 1.4 Aims

Previous work focused on implementing a first version of CKP 2.0. It has been realised thanks to Django, a Model-View-Controller web framework written in Python.

Our work is aimed to:

- Combine classic peer reviewing with metadata management and Web 2.0 techniques.

- Implement the peer review process. The current platform does not provide the interface to insure the implementation on the client application.

- Assess and improve the performance and usability of the current implementation. Design and implementation will enhance the performance of common, repeated queries.

- Enhance Web 2.0 features. Votes on publications, messaging between authors, referees and editors, esteem-based rating system will be implemented to leverage interactions between users of the platform as well as bi-directional communication between actors of the peer review process.
Provide interfaces to external databases. The knowledge dissemination on the Internet is highly related to its harvesting compliance. Agents which can harvest and being harvested increase their interaction with other Web 2.0 agents. This is why a particular effort must be applied on the active and passive harvesting capacities.

Our approach will pay particular attention to the software design, keeping the actual object oriented paradigm bound with the REST architecture. Keep lightweight user interface for each component is a principle we will respect for insuring easy cooperation with any other kind of component or web service. The established REST API will be let as it is to remain compatible with the front end code developed.

1.5 Objectives

Our objectives can be classified according each aim they refer to. Many sub objectives can constitute one aim.

Design and Implementation Review

▷ A deep analysis in source code, data model, comments, documentation and report provided by Florian Bergmann.
▷ An overview to the front end code and related report (Ghaffari 2011).

This will portray us a global view of the system implementation as well as its strength and weaknesses.

Peer Review Process

▷ Define and modelise the roles data model for the different agents involved in the process.
▷ Define and modelise the set of permissions attached to each of these roles. This will follow the RBAC model.
▷ Implement the process regarding these modelisations.

Performance and Usability Assessment

▷ Deploy the current system on the Heriot Watt web server
▷ Build and execute a generic procedure allowing to assess and record the main performance-related variables on this deployed version. To be relevant this procedure must be executed first in a native configuration and then in performance configuration. Has a cache finds its relevance for large amount of data, the tests will be performed on a database holding a lot of records (6000). Since cache is allowed only on GET request, we will test the efficiency of the cache on its ability to cache the publication pages and the related page holding more metadata.
▷ Draw up a comparison of the results obtained in both native and performance configurations.
CHAPTER 1. INTRODUCTION

Performance Enhancement

▷ Compare the different solutions offered to a Django application to speed up response time from the server side.

▷ Integrate the solution(s) chosen.

Web 2.0 Compliance Enhancement

▷ Add functionality for peer feedback. Comments within the framework of peer review process has to be sent in a bi-directional anonymity or in a simple anonymity from the referee to the author.

▷ Provide a new and more efficient coupled system between Esteem and Rating. The rating of a user will be pondered by his esteem.

▷ Improve mechanisms for low-volume publication feedback

Probe possible interfaces with existing databases.

▷ Find a way to add in a more generic manner Open Access Repositories (OAR). OAR are repositories providing public access to their publications.

▷ Implement interface to allow metadata harvesting with the added OAR.

▷ Inspect the service type and data format used by Pure platforms to know what interactions are possible with them.

1.6 Structure of document

The chapter 2 is dedicated to explores the different backgrounds required to understand the several concepts underlying the CKP 2.0 implementation. Then a the chapter 3 will gives some details about the research methodologies used for this dissertation. The following chapter 4 will give a precise description of the functional and non-functional requirements.

The sections 5 and 6 will describe respectively the approach chosen and the implementation details related to the requirements listed in the previous section.

Finally the chapters 7 and 8 will detail the issues and their domains involved, and an analysis of the tasks and planning of our project.
Chapter 2

Literature Review

This chapter holds the overview of the different concepts taking part into the CKP 2.0 implementation. We will provide definitions and explanations of these different ideas as well as the opinions variances. When it will be relevant, more explanations will be given about the way a specific feature or topic will be implemented in the current system.

2.1 Web 2.0

2.1.1 Characteristics

Web 2.0 was first used in 2001 in a meeting after the dot-com bubble burst. This latter had left a lot of companies in the state of bankrupt and only a few get through (O’Reilly, 2005). The companies that had survived, had something in common that O’Reilly qualified as the Web 2.0 characteristics. This later gave a definition of this term by describing it as a platform which must respect these competencies:

- **Harnessing collective intelligence**
  The fact that Internet is decentralized, based on a network of nodes has to be used to harness the users resources such as data or calculation power. An implemented view of the architecture of participation is to consider a client also as a server. For instance the service Bit-Torrent automatically gets better the more people use it.

- **Harnessing the long tail**
  Most of the Internet content is hosted on small sites. The aim of a Web 2.0 application is to tie to each other these content nests. Therefore the customer self-service and algorithm for data management must be considered.

- **The perpetual beta**
  The web applications turn out to be release cycle free. Compared to software, a web service or application has to be in a perpetual development state. According to Christopher Alexander, this idea is strongly bound with the fact that the users must be integrated as co-developer, testers, or at least draw up analytic tools allowing to evaluate the service usage.
CHAPTER 2. LITERATURE REVIEW

▷ **Software above the level of a single device**

Device types are growing more and more with mobiles, GPS, game platform, personal computer, and thus any service must be cross platform compliant.

▷ **Cooperate, Don’t control**

The Web 2.0 is based on services cooperation. Building loosely coupled applications thanks to lightweight user interfaces, and programming models is a key point to insure this principle. Moreover cooperation must not be melt with coordination which is not wished. A service should not come under another service’s request. For instance RSS and REST-ful applications respect this idea.

More recently Tim O’Reilly notices that the most important piece underlying the Web 2.0 remains harnessing the collective intelligence especially in the data way. He points out that the data generation is not solely driven by the user typing on its keyboard, but more and more by sensors. These sensors are the smart-phones and camera which can record and process our location, our motion and also our voice. They “are being turned into eyes and ears for applications”?

Moreover, Christian Briggs introduces a new notion of value creation for the Web 2.0 business models. The first established for the Web 1.0 was more linked to the product in itself whereas the Web 2.0 definition focuses on the decentralization capability of the product value: “value is a system of relations around the product” [Briggs (2009)]. This new definition allows to see the Web 2.0 as a new usage of the technologies and not solely as new technologies.

2.1.2 Features

As described in the section 2.1.1 Web 2.0 compliance relies partly on the implementation of features allowing cooperation between users, and collective intelligence harnessing. The features we will describe here have a particular importance with the CKP 2.0 since they may strongly help to leverage quality assessment of published papers, give informal feedback to the author, and link together the related contents.

All of the following presented features are already implemented in the current platform. However they are not well correlated to allow dynamically weighted ratings on the authors, readers and papers during the “post-filtering” [Surowiecki and Silverman, 2007] period, that is to say when the publication is made accessible for common readers.

Moreover the logic to provide related publications based on criteria (such as the author, its affiliation, its author group, the tags or keywords) is not fully integrated.

2.1.2.1 Commenting

Commenting has become more and more widespread on the web as a formative tool. Sites such as blogs and forums often integrate this feature\(^1\). As we have mentioned above, the bidirectional communication between the producer (the author) and the consumer (the reader) is very important. This is the most basic yet efficient way to give feedbacks to the author.

\(^1\)See http://stackoverflow.com for instance
Because of the importance of the bi-directional communication in our context we will heavily use this feature.

In CKP 2.0 comments need to be used for two distinct purposes. Firstly during the “pre-filtering” (Surowiecki and Silverman [2007]) period, that is to say during the peer review process where the referee may need to comment anonymously the author publication draft. The second purpose is to make comments a “post-filtering” tool by allowing readers to comment the final publication. Then the author can take benefit from these forms of feedback to keep his work relevant and up-to-date.

The CKP 2.0 integrates already the possibility to comment published papers.

Since the peer review process is not yet implemented, this feature is not made available in this context. Nevertheless, the commenting ability during the peer review process has to fulfill one more constraint. During the aforesaid process, a bi-directional or a one-way blindness may be wanted between the author and the referee. The actual data model includes these constraints partially only by allowing to set an open peer review or a simple blindness (the author does not know the referee). The data model will be modified to allow at least open peer review, simple blindness, and double blindness.

### 2.1.2.2 Rating

On the other hand rating is a summative tool which is a way to insure quality control and give feedbacks on a publication: for Nadasdy [1997] it is a “post-filtering” tool which “substitute peer review with democracy”. In our case we will keep using the peer review process, in addition we will use an open system of rating. As it is quantifiable, it is possible to give a more formal semantic on the rating data and introduce a notion of reputation on each agent of the system. Web sites such as Stackoverflow make use of both comments and ratings to build a system of reputation. Any reader can give a rate to any post to mean if it is a good answer or not. On this base the answer provider and the rated author will gain or loose reputation.

Besides Mizzaro [2003] submits a rating system in the context of scholarly publishing where each agent (author, paper, and reader) has a score (i.e. a reputation). Comments are useful to give feedbacks on a publication, but they don’t really give an ultimate, dynamically weighted, judgment to assess the quality of a paper. Rating system described in the article of Mizzaro provides this feature by modifying the reputation and the rating weight of each agent after any reader has rated a paper. Therefore the rating process does not rely on the user status, but only on the previous grades he gave. We will take the concept of dynamic weighted rating to enhance the existing implementation of esteem which is currently static.

We want to make this rating system available for the different actors in the post-filtering process. First of all this feature has to be accessible for the main roles, such as other publication authors, referees and editors removed from their context. Additionally rating must be enabled for two kind of readers. The first is clearly identified with appropriate information, the second remains anonymous, and would provide only the bare essentials to get access to the platform. This later type of reader rises authority and security issues. In order to prevent from unwanted behavior and unreliable data provided by this kind of user, we will have to draw up a policy accordingly.
2.1.2.3 Social Tagging

Social tagging is a way to classify content by letting the choice of tags to the user. There is already an implementation of tagging in the existing project.

As Florian Bergmann states in his report it exists two sorts of tagging in the current system: “The difference between keywords and tags will be that keywords can only be defined by the person that added the publication to the system, whereas tags can be added by every user of the system.”

[Tonkin et al. (2008)] defines three models of tag use:

- **User - Tag - Information** (Figure 2.1.1)
  
  In this case the tag is used to link an information to the user. It is a manner to show how the user consider an information for a given tag.

- **User - Tag - User** (Figure 2.1.2)
  
  This model links the user by their use of tag (even though they don’t have the same meaning for the of tag they use). This model is the most social-networking oriented since it links the human agents of the model.

- **Information - Tag - Information** (Figure 2.1.3)
  
  This last model relates directly the information with the same tag. The usefulness of this model is in its power to take metadata from the user. Therefore, search engines will use these relationships between information to find the related information for a given one.

![Figure 2.1.1: User - Tag - Information](image1)

![Figure 2.1.2: User - Tag - User](image2)

![Figure 2.1.3: Information - Tag - Information](image3)
CHAPTER 2. LITERATURE REVIEW

The current database design is built on the Information-Tag-Information model: any publication entity can refer to one or more keywords and tags. This relationship allows to deal with related publications easily and construct data representation holding these relationships.

2.1.3 Web Services

Web services are instantiations of the Web 2.0 characteristics listed above. In this section we will cover a definition of what a web service is, make a comparison of different web service architectures, then explain the choice of a web service implementation and its REST architecture for this project.

2.1.3.1 Definition and Purpose

According to Haas and Brown (2004), a web service is “a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”. We will see that there is also other sort of web service approach in the Section 2.1.3.2.

In their technical report, the W3C Web Services Architecture Working Group (McCabe et al., 2004) identifies several roles enrolled in a web service architecture:

▷ **Agents and Services**

The agent is the artifact implementing the service. For instance, the Core Knowledge 2.0 Platform service is to record and provide information about publications in different manner according an established semantic. The agents concerned for implementing this service are the database holding the data, the web server and the software involved to send and receive messages. Therefore, web services agents can be software or hardware pieces.

▷ **Requesters and Providers**

The requester side is composed by the entity (person or organisation) who wants to use the web service’s provider entity (person or organisation). The requester entity wish will be achieved by using a requester agent to communicate with the web service provider’s agent. This process involves messages exchange, thus a service description and a semantic must be agreed from both sides requester and provider.

▷ **Service Description**

As implemented agents are machines and softwares, the service availability is insured by a message exchange mechanism between machines. A specification of the web service interface must be described and machine-processable. This description is called the Web Service Description (WSD). It gives basic required information in order to be able to interact with it. These information could be the message formats, the data types, the transport protocol, and the data serialization.

▷ **Semantic**
Semantic is related to the meaning of the web service. It allows a less formal description of the expectations the provider and requester entities have from it. Purpose and meaning of the exchange done between both entities are parts of these expectations.

The following figure summarize the whole concept of a web service (McCabe et al., 2004).

![Roles and underlying concepts of a Web Service](image)

**Figure 2.1.4: Roles and underlying concepts of a Web Service**

The definition of a WSD aims to let the web service to be used by machines. Then one of the purpose of web services is “to ease automation of data exchanges between entity such as computer systems” (Gray, 2003 p492).

Moreover, in opposition with Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA) standard and the Microsoft proprietary products (COM,COM+, DCOM, DNA), web services are built atop HTTP protocol. It could be built on any other communication protocol, but it’s most commonly used with HTTP. They are several advantages of building them over HTTP. It avoids trouble with firewall configurations, it eases trouble shouting since HTTP is a readable protocol. The large amount of deployed HTTP agents also helps for a cost-effective deployment phase.

Nevertheless, web services have also disadvantages. As the data format is readable, the overhead on a communication link may reach high weight.

### 2.1.3.2 Web Service Approach Comparison

To achieve the following comparison we need to introduce the resources term. On the web a resource can be viewed as a set of data referenced by a key where the later is an URI (Uniform Resource Identifier). Different approaches to deal with these resources exist in which two mains has emerged:

- **SOAP** (Simple Object Access Protocol):
It is a message-oriented protocol with a document format, integrating machine processable descriptions, defining a set of very extensible technologies (Haas, 2005, Slide 3). Snell (2004) states also that SOAP is "activity-oriented", in a way that the application using it cares more about what action the user will perform than the resource on which they will operate.

**REST** (Representational State Transfer):

It follows closely the architecture of the Web (Haas, 2005, Slide 3) and its resources concept. In a REST-ful web service the user can access and perform a set of operations on these resources regardless of the type of them (Snell, 2004). To achieve these goals, it uses the HTTP uniform interface *GET*, *POST*, *PUT*, and *DELETE* and use XML as a representation format.

In the development point of view, REST is an "architectural style for distributed hypermedia systems".

From these different ways to approach the notion of web service, pros and cons (often antagonist) exist on both sides. As they are antagonist we give a list of cons only below (Haas, 2005, Flanders, 2009, Snell, 2004):

**Against Web services**

- They try to reinvent the web with their message protocol, instead of using the current resource oriented architecture of the web.
- They are not easy to build and deploy, there is a lot of tool dependencies and version restrictions.
- Each request and response is wrapped in an (XML) envelope which increase the payload on the bandwidth.
- REST-ful services are less CPU expensive since there is less parsing, marshaling and unwrapping.
- SOAP use POST tunneling for its message transfers. Thus caching is not easy at all since POST response are not cacheable.

**Against REST-ful Web services**

- It is an utopia to want to modelise only by using the HTTP methods POST, GET, PUT, and DELETE.
- How do you represent the common features of a service such as reliability, transactions?
- There are no development tools. It implies a lot of work left to the developer.

### 2.1.3.3 The choice of REST for CKP 2.0

The REST approach tends to be more Web 2.0 compliant than SOAP-based web service and more adapted to our context.


- **User Scalability:** REST based services make use of GET HTTP method to access resources, which is cacheable. On the other hand, SOAP only use POST HTTP method which is not cacheable (Fielding et al., 1999, § 9.3 and 9.5). For our performance goal, cacheability of HTTP response is one of the most important point.

- **Interoperability:** The big advantage of REST against SOAP is that the only thing it needs is a HTTP stack. Nowadays most of platforms are equipped of this later. In his article, Flanders (2009) states that even if SOAP is supposed to be interoperable, “the large number of different standards (and versions of each of those standards) to choose” implies a lack of interoperability. Then the platform-interoperability is limited by the standards chosen on each entity using SOAP.

- **Loose coupling:** The web does not check the integrity of the resources in order to avoid tight coupling with the user. A more simple concept of status code is used to warn about any resource state changes.

- **Stateless:** REST respects the statelessness of HTTP whereas SOAP introduces states. Keeping a pattern design statelessness compliant allow to use load balancing on the server in case of heavy use of the web service.

- **Simplicity of Operations:** As REST bases all of its operation on the HTTP methods, the set of operation is very restrictive and keep interactions with resources as minimal and simple as possible.

This comparison, the REST-ful implementation of the current system and the cacheability goals we have to improve the performance justify the choice of a REST-ful architecture for our work.

### 2.1.4 External Interfaces

This section states the different ways in which we can implement interactions with other platforms having the same goal of scholarly knowledge dissemination. Two approaches has been retained because they respect the Web 2.0 framework.

#### 2.1.4.1 Open Archive Initiative (OAI)

The OAI is an organisation which aims to provide standards and technological frameworks to ease dissemination of content, especially in a research and education context. Particular attention is paid to avoid any kind of coupling between the standard definitions they provide and both the type of content and the economic mechanisms surrounding that content. This objective is particularly important in order to keep the Web 2.0 principles such as interoperability, lightweight interface, and loosely coupled application possibilities. The following paragraph provide a description of the one of the current projects held by this organisation.

**OAI - Protocol for Metadata Harvesting**\(^2\) OAI-PMH project provides a set of specifications and implementation rules to build interoperable archive repositories based on low-barrier mechanism. In this protocol, two types of entity are defined: the Data Providers which are the

\(^2\) Lagoze et al., 2008
repositories able to give structured metadata usable by an OAI-PMH implementation, and the Service Providers which use OAI-PMH service requests to harvest metadata given by the Data Providers. Those interaction between Data Providers and Service Providers are made by using “a set of six verbs or services that are invoked within HTTP”. Although this protocol based his metadata on the Dublin Core standard. This metadata format defines 15 not mandatory elements to describe metadata, such as title, description, subject, publisher. The following figure gives a general idea of OAI-PMH:

![Figure 2.1.5: OAI-PMH General Idea](image)

Integration of OAI - PMH in CKP 2.0  
A python interface exists already for the PMH. This interface is integrated in the current platform as a module, but it only does screen scraping over the html retrieved by the data provider. Therefore an improvement on the way the metadata are harvested from the data provider has to be made. In particular, a more accurate request must be built upon the metadata held in the CKP 2.0 database. Since there is no real documentation on how to use the python implementation of the OAI-PMH interface, these improvements rely on a thorough understanding of the python module implementation which will be made later on.

Moreover we plan to provide an OAI compliant Data Provider implementation for CKP 2.0 in order to give an interoperable access point for the stored publications.

2.1.4.2 Google Scholar

Google Scholar operates as tiers agent to gather metadata and open publications thanks to its search engine. It also ranks the documents it indexes based on the place it was published, the publisher, the author, and how often and recently the document has been cited in other scholarly documents. Google Scholar allows to index scholarly literature from any platform under the condition that each document must provide an abstract (at least) to be visible for any user coming from Google Scholar or Google.

The technical documentation of Google Scholar notices that the indexing of the university repositories to their search engine is done by most of the current major dissemination knowledge platform such as eprints\(^3\), digitalcommons\(^4\) or DSpace\(^5\). As we aim to develop our CKP 2.0, we will follow the technical documentation.

---

\(^3\)http://www.eprints.org/

\(^4\)http://digitalcommons.bepress.com/

\(^5\)http://www.dspace.org/
2.2 Performances

2.2.1 Motivations

We have chosen to implement the CKP 2.0 as a web service. Consequently every data is represented as a resource. The tests performed on the previous year CKP 2.0 and Front End applications has shown that the wall clock time needed to display publication data and their related information was very expensive (an average of 2.1 seconds was evaluated to fetch a publication and its 10 related resources). The related information can be tags, keywords and comments but also related publication and further metadata harvested from external platforms. The reason of this long round trip delay is due to the fact that each related information need to be retrieved by a dedicated request to the wanted resource. Thus the request rate made for one page becomes dramatically high and increase the overall time to retrieve the final web page.

Moreover on the cross-platform harvesting aspect, a performance issue has been also noticed. When the platform is asked to harvest more metadata from other platforms (CiteSeer for the current implementation), the wall clock time appeared to be very long. This issue has also motivated our performance concern.

In this context tools and techniques to lower the wall clock time for requests and their associated response is expected.

2.2.2 Techniques Overview

As state Levy-Abegnoli et al. (1999) in their paper, performance of a Web server may depend on several criteria. First of all the platform on which it is executed play an important part. Other constituent parameters are the operating system scheduler and the interruption processes. They also mention the fact that a lot of time is required for a request to be served, since the later cross several physical and logical layers when it arrives on the server. Other factors such as the size of the request and the response may implies high round trip delay. Here is a list of the different physical, architectural, built-in, and strategical known techniques to improve performances of web servers given by Iyengar et al. (2000):

- **Hardware Redundancy**: It aims to use more hardware materials in order to dispatch processing time in case of high request rate of the web site.

- **Load Balancing**: It is a child concept of the hardware redundancy. It is used to distribute requests over a set of web servers thanks to an intermediate load balancer (a router for instance).

- **Web Server Accelerator**: It is a physically or logically externalised entity placed between the client and the server serving the web pages. Its main purpose is to be very fast in serving pages from its implemented cache. Iyengar et al. also advises this entity should use memory as storage device since disk is too slow for writing and reading. Nowadays it exists several implemented accelerator, with sub features.

- **Efficient Dynamic Data Serving**: It holds two underlying ideas, the cache replacement policy (that is to say the algorithm used) and also the way the CGI is instantiated for creating dynamic data. We will discuss the choice of both respectively in the sections 2.2.4 and 2.2.3.
In addition to pure technical solutions, there is also ideas which has been developed around the order of the request should be processed. Harchol-Balter et al. (2003) say in their article that if an order is defined on the size of requests and the server serves them following the ascendant order and according the Shortest Remaining Process Time (SRPT) algorithm, then the response time of every request is improved.

2.2.3 Web Server Gateway Interface

WSGI is “standard interface between web servers and Python web applications or frameworks, to promote web application portability across a variety of web servers.”

Iyengar et al. (2000) state in their paper that the CGI work by creating process for each new request received. The excessive fork of processes causes “considerable overhead”. Better mechanism where Gateway Interface processes have a longer run time or run as daemon induces significant decay of this overhead.

Our deployment version of the CKP will run on the Heriot-Watt server which is an Apache. A specific module named mod_wsgi will be added to the Apache in order to integrate the Django CKP application. This CGI module can be configured in a manner to run as a daemon. Therefore, the recommendations specified by Iyengar et al. to improve dynamic data serving will be applied.

2.2.4 Cache

As explained in section 2.2.1 our main issue is the wall clock time between the client request and the response provided by the server. The large delay occasioned is due either to the number of requests made by the client side to the server side or to the requests made to external platforms to harvest further metadata for a publication. Wessels (2001, Chapter 6.2) mentions two reasons in which the use of cache is vindicatory for our case:

- The server will provide HTTP response faster
- The load on the servers and on the network will be lowered.

Therefore our choice is oriented to the cache technique.

As described in Wessels (2001) in his book, the server response time increases with the request rate. He gives two approaches for caching data.

The first is to use a proxy, that is to say an external user agent. Therefore the proxy must play two roles in the same time: the client role to deal with the server, and the server role for the client who asks for a resource. When a client user asks for a resource, the request is first processed by the proxy. If the resource asked is not cached (or not cacheable), the proxy forward the request to the server. Once the server’s response sent to the proxy, the later caches the result if it is cacheable and forwards it back to the client.

The second approach keeps this idea of proxy but add the fact that this proxy should work “on behalf of the origin server rather than a user agent”. Those kind of cache architectures is usually called reverse caches.
CHAPTER 2. LITERATURE REVIEW

Wessels gives four usual terms used in cache systems that we need to know in order to have a better understanding of the picture and for the further performance improvement evaluation:

- **Cache Hit**: It corresponds to the fact that the object requested by the end user is present in the cache.

- **Cache Miss**: It stands for the retrieving fail when the cache system tried to find the object in its storing space. Therefore a request to the server to obtain the object is needed.

- **Freshness vs Stale**: Freshness is the principle related to the fact that a cached resource can be considered as up to date. In opposition when this resource turns out to be out of date then it swaps to a Stale state.

- **Validation**: Whenever a cache hit occurs, it can require Validation. This Validation need arises when a cache hit occurs but the object corresponding is in Stale state. In this situation the cache system needs to validate the state of the object by asking to the server if the cached response is still valid.

A cache allows to decrease the load of a server and the response time. Nevertheless, many factors affect the ratio of a cache hit and must be taken into account. Wessels (2001, Chap 2.4) gives three factors affecting hit ratio: cache size, number of clients, and freshness heuristics. Especially, it is known that the relation cache hit - cache size is not linear but pseudo-logarithmic. Wessels provides advices that we will use during the development phase to configure appropriate cache size.

Another important aspect during the deployment of a cache system is the storage method. It currently exists three storage method, and each of them has their strength and weaknesses:

- **Disk**: huge storage space, persistence even in servers crashes because write into files, but slow in I/O, and crash-prone (Fitzpatrick 2004, p1).

- **Memory**: getting larger in space over time and really fast in I/O, but it looses data in case of crash. This can be lowered by distributed caching system such as memcached.

- **Database**: works best if with fast disk and well-indexed database server.

As we can see from this listing, the memory caching is far the best when coupling with a distributed caching politic. Distributed caching consists to spread all the keys/values of the cache on different machines. All the web processes running must be able to access simultaneously these keys/values and see the any change made by another instantly. The idea to spread all the keys/values on different machines came up to Brad Fitzpatrick because he considers that “memory is cheap, networks are fast” and he doesn’t “trust servers to stay alive” (Fitzpatrick 2004, p2).

Another important issue with cache, especially within the context of distributed caches and multi core architectures, is to pay attention to keep the algorithm lock free: clients dealing with the caches must not get into a mutual exclusion state. In our case it means that the access to the cache data must be lock-less. If one client with a bad connection is updating an
object, and many other clients are reading this object, any read-write locking state must be avoid. A judicious choice in data structures allows to avoid such a problem. Buffers, stack, list, queue are likely to be used [Fraser (2004)]. Memcached uses multi-versioned and count reference on all its objects in order to ensure lock-less [Fitzpatrick (2004), p2].

Finally, there is an architecture support to insure. CKP 2.0 may be subject to be deployed on different kind of hardware architectures such as a multi cored server. Therefrom we have to choose a cache system which meshes with this eventuality.

Over these technical considerations, we have to make a choice between two kind of cache systems. A first type of them would be placed between the server and the Gateway Interface. A second solution would be to place it between the Gateway Interface and the web framework Django. The first proposal is not satisfactory because it does not allow to distribute the cache system on several machines. Likewise tuning aspect is not implementable with this approach. The second approach is much more attractive since a module such as memcached can be easily integrated into the current system, support different hardware architectures, and can be featured and configured thanks to its provided python interface. The next section deal with the usage of the module memcached with Django.

2.2.5 Django and Memcached

Memcached is a distributed memory object caching system. In other words, this cache system has the ability to distribute his storage of key/values on several machines. Moreover it is multi-threaded by default, thus when it is deployed with a thread support enabled, dealing with multi-core architecture becomes possible.

One of the most important aspect to pay attention within the CKP 2.0 context, is the prefetching of the data from an identified request type. As we stated in the Section 2.2.1 there are a lot of sided requests made to retrieve related publication information.

As the side requests are always the same for a publication request, the related data attached to a publication may be fetched into the cache. The module memcached allows this prefetching, so we will make use of this feature coupled with a configuration interface, providing setting parameters such as the field on which we want to prefetch the cache.

All those prefetching operations are made available thanks to the python’s interface provided by python-memcache. This later will be consequently installed beside the memcached module and the Django framework.

2.2.6 Other potential techniques to enhance performance

It exists additional ways to increase response time on both sides client and server.

On the first one, the most famous technique is the usage of Ajax. It allows to perform requests lazily, according the user behavior. Therefore, only the additional information the user asks for are retrieved from the server side. Early host name resolution can be also used on the information held in the retrieved document (HTML).

On the server side compression of data before sending them onto the network can be used. The decompression phase is let to the client side.
CHAPTER 2. LITERATURE REVIEW

2.2.7 CKP2.0-Specific performance possibilities

There are specific solutions linked with the core business of CKP 2.0 that can be added to increase performance. This solutions can be separated into two parts: one from the server side that the Front End has from CKP, and another of CKP as a client when it harvests metadata from external platform. Will deal with these performance solutions according this two point of view:

**Server side**  Django provides a very strong ORM (Object-Relational Mapping) layer. Therefore with a caching business logic, it would be possible to store every objects accessed in the cache. This logic could also be used for a prefetching logic: for most likely requests to be performed by the service consumer we prefetch the cache with the object corresponding to these requests. As Django provides as well a signal concept, it would be possible to use them to invalidate any object stored in the cache that is related to another object that has changed.

In addition of caching and prefetching at the object level an even more efficient way to increase performance is to use page caching. That is to say that we could cache the entire HTTP response generated by CKP given a request received. As CKP is built on the REST architecture, every type of data is canonically defined. Therefore it is a strong asset for managing cache integrity and intelligent invalidation. This resource typing coupled with unique identifier for each resource type instance we could push the invalidation policy at a very high granularity level.

Finally as Ghaffari (2011) states in his master thesis, it would be also possible to directly integrate the related resource of a publication resource in the former response, thus no more requests are needed to get the authors, the keywords and so on. Even though it breaks down a bit the REST approach, coupling issues are avoided thanks to Django features such as template inclusion into other templates.

**Client side**  Harvesting and finding related metadata that match a CKP publication from external platform can turn out to be very slow. Indeed, there are a lot of platform offering a web service to harvest metadata (through OAI-PMH or not). The long delay is mostly due to the wall clock time to send an appropriate request to the web service, getting the response, and finally parse it and refine it to provide accurate results to the CKP service consumer.

One of the most approachable way to lower this time consuming process would be to translate these different steps onto a parallel process that would be performed asynchronously. Therefore we always have a local version of harvested metadata with a freshness that relies only on the frequency the asynchronous harvesting is repeated. Then only parsing and refining to get the accurate related metadata to a publication remains to be done, and once it is done it can be cached. This latter option will increase dramatically the performance in term of response time from a CKP client request.

2.3 Peer review process

The peer review process is a key step for knowledge dissemination. It is a powerful mean to assess work before making it public in the research realm.
As shown in Figure 2.3.1 (Bergmann, 2011), this process involves many sub-processes that rely on people involvement and their assessing skills. These several sub-processes and the number of people involved in these latter generate delay in the overall process.

From the figure showing the peer review flow, we can give a formal list of the time-consuming step in the physical peer reviewing way:

- Submission paper.
- Gathering the committee for pre-screening.
- Review period and delay between request and response.
- Waiting for more reviewing judgments after a first publication decision.
- If the publication is accepted, the publication process in the appropriate scientific journals.

CKP is a platform that lower this time-consuming sub-processes by taking advantage of the electronic technologies and more particularly by using web 2.0 technologies.

The literature review on peer review made by Florian Bergmann is very well documented and will provide a more thorough insights about peer review process (Bergmann, 2011, Section 2.3.1).

2.4 Current Systems

In the same way that it had been done in the previous report, we will compare different platforms and websites on the criteria listed in the first column of the table 2.1.
<table>
<thead>
<tr>
<th>Features/Platform</th>
<th>ePrints</th>
<th>Open Journal Systems</th>
<th>Fedora Commons</th>
<th>DPubS</th>
<th>Digital Commons</th>
<th>Dspace</th>
<th>Pure</th>
<th>CKP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper submission</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Peer review process</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Commentary collaboration</td>
<td>Partial</td>
<td>✓</td>
<td>✗</td>
<td>?</td>
<td>✓</td>
<td>Partial</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Rating and Esteem</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Metadata harvesting (OAI compliant)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Main user roles (Admin omitted)</td>
<td>Editor, Deposit, Reader</td>
<td>Editor, Reader, Author, Reviewer</td>
<td>?</td>
<td>Editor. Role creation possible</td>
<td>Editor, Reviewer, Author</td>
<td>Editor, Author. Role creation possible</td>
<td>Basic User, Editor, Referee</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Platform comparison
As we can see many platforms do not provide a commentary collaboration or only partially by emailing between referees.

Moreover most of these publication systems miss completely the esteem and rating features. *Pure* provides an esteem principle but only based on awards such as a prize, a medal, a honorary degree.

The CKP requirements overview dressed in this table are more detailed in the chapter 4.
Chapter 3

Methodology

Research methodologies are methods to underpin the process of research. It can be divided in two distinct areas: quantitative and qualitative methods. A third method inherits from these later by combining both.

Usually the quantitative method is used to confirm existing theories or assumptions thanks to statistical tools such as questionnaires. On the other hand, the qualitative method aims to help in understanding a meaning, a concept and describe it further to integrate it in the current research performed. The combination of both of these methods is often used in order to have a “double” view of the research subject.

For this project, we believe that using combination of those methods is a good choice. The qualitative approach will help us to understand and modelise the high level concepts such as peer review process and rating and esteem systems. This phase is mainly leaded by reading publications and articles about this concern. The quantitative method will assist us to confirm the assumptions made on the end-user customs and behavior on the CKP 2.0. Therefrom, a main sketch of what kind of data may be retrieved at once to lower response latency will be dressed. For the quantitative phase, a particular attention will be paid to the selection of people taking part into the survey in order to avoid irrelevant statistics.
Chapter 4

Requirements Analysis

4.1 Functional requirements

The sub sections below will list the requirements which can be formally described (contra-positive of the non-functional definition given by Roman (1985)).

4.1.0.1 Performance

The cache must:

▷ P01: provide an interface to allow trivial configuration of the cache system instance(s)\(^{(2012)}\)
▷ P02: prefetching the cache intelligently according the request
▷ P03: Allow tuning possibilities on the prefetching rules
▷ P04: Pay attention to insure good performance for any external metadata platform harvesting

4.1.1 Peer Review Process

▷ PR01: Modelise each stakeholder of the peer review process
▷ PR02: Define all the permissions related to peer review process and its stakeholders
▷ PR03: Automatic notification for referee’s deadline reminder

4.1.2 Web 2.0 Technologies

▷ WT01: Enhance the users Esteem implementation
▷ WT02: Allow Rating and attach Esteem to anonymous users
▷ WT03: Tighten the Rating system on the Esteem system (see 2.1.2.2)
▷ WT04: Integrate dedicated search framework
▷ WT05: Add more fields on which related papers are retrieved such as author affiliations
4.1.3 User Management

- UM01: Provide an interface allowing definition of permission groups
- UM02: Provide an interface to assign group of permissions to specific roles
- UM03: Provide an interface to manage the permissions of each role in a RBAC manner

4.1.4 User Interface

- UI01: Insure the registration of the different roles types
- UI02: For each registration, a profile with specific information related to the role must be created

4.1.5 Data Model

- DM01: Modify database design according the Peer Review data model

4.1.6 External Interface

- EI01: Extends in a more generic way the list of data providers implementing OAI - PMH thanks to ROAR XML formatted list
- EI02: Integrate external and internal data management using a python module implementing the OAI - PMH
- EI03: Implement the OAI Data Provider interface for CKP 2.0
- EI04: Enable the platform content to be available on Google Scholar (see 2.1.4.2)

4.2 Non functional requirements

As states Roman (1985), here is listed the requirements which cannot be formally specified. We implicitly consider the previous non functional requirements listed in the Florian’s report\(^1\).

- Any new gathering of functionalities must be packed into a module or equivalent concept
- Any enhancement of the actual system must be well interfaced
- The system must show a performance gain in an heavy use case and for external metadata harvesting
- The restriction of resource access must be insured between the different roles involved in the peer review process
- User roles must be easy to manage through an administration interface
- The connection between the Front End application and the CKP 2.0 will be insured

\(^1\) Bergmann, 2011
Chapter 5

Design & Software Architecture

This chapter will give the general design decisions starting from the most general point of view and going down to a more refined view. As the system has been improved from an existing project, we will skip the system’s organisation and leave the reader referring to Bergmann’s Master Thesis to get more details about this point.

After a quick reminder about the design goals, the global architecture of the system and the improvements added to the data model, we will describe the architectural and functional solutions chosen to overcome final achievements of the requirements.

5.1 Design goals

The project started the previous year was chosen to be developed with Python programming language aside with the MVC Web Framework Django. More details about these choices are described in Florian’s Master Thesis.

Another important point about design during this project was to respect as much as possible the Object Oriented Programming architectural style. This goal is easily achieved since Python support this paradigm and Django is built in an Object Oriented manner.

Moreover we wanted to use as much as possible the modular approach to clearly separate the different feature implementations and to keep the loose coupling idea real.

5.2 System Architecture

The system architecture strongly relies on the Django framework and its web application concept. The Figure 5.2.1 shows the overall system architecture where each package stands for a Django application. The settings file gathers all the main configurations data needed by the whole project and server when the project is deployed. The url module gather the core url routing information. Finally the wsgi module instantiate the Common Gateway Interface using the Apache module WSGI.

Briefly, the application chronograph is used by the application oai in order to be able to manage automatic update of OAI repositories. The core_web_service application is the core application of the project. The application prefetching gathers the business code for the prefetching feature, and the south application is used to perform database migrations.
5.3 Django-applications

In order to keep the web MVC architecture chosen we have used the application concept wrapped in the Django framework. Therefore the performance and external harvesting requirements are implemented in two different Django-applications. Based on this concept, we used tiers party applications for the implementation of the prefetching feature and the external harvesting.

Each Django application contains at least the following package structure:

```
<table>
<thead>
<tr>
<th>views</th>
<th>models</th>
<th>admin</th>
<th>tests</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>.py</td>
<td>models.py</td>
<td>admin.py</td>
<td>tests.py</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The `views` module gathers the controllers handling URLs described in the `urls` module. The `models` module contains all the data model design and its business constraints such as the foreign keys, cross reference abstraction, primary keys, and the authorized value for the different model entities.

Finally the `admin` module allows to override or add some admin interface functionalities. We will have a look in the different Django applications implemented in the section 6.

5.4 Interfaces implementation

Each interface implementation with external database are gathered in a python module into the Django application dedicated for the external harvesting: oai.

The implementation for the OAI-PMH service provider and data provider interfaces are respectively gathered in the sub packages `service_provider` and `data_provider` of the `oai` application.
The Mendeley client API implementation is gathered in one module \textit{mendeley\_client} stored in the \textit{oai} application.

### 5.5 Data model

Before going deeper in the data model design decisions it is good to have an overview of the whole modified data model in order to understand the following sub-parts (Figure [5.5.1]).

**Messages**  As we can see on the Figure [5.5.1], a message data model was added to insure bi-directional communication between an author user and an editor or referees.

**Author role and author data model**  The main modifications is between the author data model and the generic user profile data model. This new relation intend to fix the following tricky situation.

Firstly there is the \textit{author data model}. This is defined to gather information about an author involved in publication redaction. It is important not to confuse this model with the \textit{author role}.

As we will see in the next section, an \textit{author role} is also defined. In the system this role is held in the \textit{auth\_group} data model. This role can be attached to any \textit{User} model instance, as described in figure [5.5.1].

We can wonder why there is an author data model if there is also an author role defined for users.

The answer is that a user who submits a new publication has to provide the list of co-authors involved in this publication (if any). These co-authors might not be registered in the system. Therefore we need to store information of these co-authors in a different data entity: the author data model. In addition of that, the submitter should match the author data model instance created with the user registered in the system if there is any.

In the case that no registered user matches the new author data model instance created, the match will have to be done when a user will register as an author. He will have to select its corresponding author profile.

As we can see in the data model, the matching between the author and the user model is made by a foreign key from the author referring to the user. This foreign key can be null since an author profile could not have any user matched.

**Publication votes and Comment votes**  The first implementation provided only the vote feature for the comments. A publication vote data model is added. It contains a reference to a publication as well as a reference the vote caster (User data model).

### 5.6 Peer Review Process

As planned in the requirements, it was needed to identify exactly the different stakeholders of the system. The literature review helping, it comes up that four essential roles are clearly identifiable with a different set of permissions for each of them. The tables describing the permissions granted given a role and a resource type are provided in annexe [10.1].
Figure 5.5.1: New data model
CHAPTER 5. DESIGN & SOFTWARE ARCHITECTURE

reader: it gathers all the permissions referring to a common user of the platform. They can rate, vote and comment publication as well as vote existing comments.

author: it puts all the permissions about creating a new publication together.

referee: permissions for papergroup in which they are member, and publication and peer review access for publication under their review.

editor: permissions for papergroup access, and publication and peer review access.

As an author could be as well a reader in another context, and similarly with an editor or a referee, these three later are always coming with the reader role.

The Section [6.2.1](#) will give the implementation of this role modelisation.

5.7 Esteem rating and rating system

As stated in Section [5.1](#) we implemented these features in a backend system of the CKP. It is gathered in a package called *esteem_rating*.

The backend approach allows to define different esteem rating implementations. To choose a backend implementation, it relies only on the declaration of this latter in the project *settings* file.

5.8 Caching

Caching is the most generic way to improve performance compare to prefetching. One of its strength is that it is application independent, therefore it is used in this project by exploiting the implemented interface of memcached. Nevertheless, caching as the complementary flaw of its advantage: it could not be used to apply a specific policy to prefetch related resources of a publication, or related search queries. Therefore, prefetching was chosen to apply this requirement.

5.9 Prefetching

As prefetching is more application dependent we dedicated a Django application to this feature. Tiers party application was also used in order to overcome additional sub features that were needed to implement prefetching.

We wanted to prefetch at the database entity instance level. Besides we decided to use the ORM layer, to provide automatic invalidation of the appropriate database entity instances stored in cache when changes were performed into the database. All these requirements turned out to be already implemented in a Django application called *johnny-cache* ([Moiron and Self](#) 2012). As this application shown a good online support and documentation we chose to integrate it to the project and built the prefetching application upon this latter.

Thereafter, we designed the global architecture of the prefetching application and defined the typical work-flow of how prefetching shall work. The Figure [5.9.1](#) shows the global process the application implements whereas the Figure [5.9.2](#) shows the most likely work-flow to be executed under the Front End context for the *PageRendering* policy.

In the process, we identified different elements:
The activity that can be matched with an interaction performed on the Front End side. For instance there is an activity matching the fact that a front end user is accessing a specific publication web page. There is as well another activity for a front end user performing a search. These two activities are the most likely to be concerned by prefetching.

The policy that describe how the policy-specialised prefetching engine should prefetch. The policy are described according a set of rules. By combination of several rules we can describe different situations and simulate simple boolean operators between them (such as OR and AND). A more thorough attention will be paid in the rule syntax and semantic in the implementation section (6.4.2).

A simple URL parameter that trigger the prefetching feature by mentioning the policy to use.

5.10 Search

The search feature is one of the most important because of the most used. Therefore the performance issues were aimed to be handled for this feature as well.

5.10.1 Internal search

The search for an internal publication is one of the most likely work-flow we can expect from a end-user using the Front End. This work-flow directly impacts on the way we can improve performance on the search service that CKP provide to the service consumer. Wherefore, we had to think about a way to keep an already performed search query to reuse it without having to execute the business code corresponding to the query again.

The detailed work-flow identified was as follows:

1. The end-user search for a publication according one or more field, for instance a publication title, an author name, a research area, a tag or keyword.

2. The request received on CKP, we perform the search in the database.
Figure 5.9.2: Prefetching work-flow
3. We generate a *resumption token* that correspond to the query and store in the cache the search result using the resumption token as key.

4. Then the response containing the result is sent to the user.
   a) If the end-user wants to refine the search we take the previous result from the cache and refine it according the new field inputs provided by the user.
   b) Once the narrowing of the search is generated we apply the same Steps 3 and 4.

Figure 5.10.1 shows the flow chart of the aforesaid described work-flow with the cache feature and the resumption token.

Moreover, to improve as much as possible the performance aspect, this feature uses the same page caching policy described further down in this document in Section 6.4.1.1.

5.10.2 External metadata harvesting

As for the prefetching feature, the external metadata harvesting feature is wrapped into a Django application. This application contains an implementation of both *data provider* and *service provider* interfaces (see Section 2.1.4.1). Moreover it embeds a client implementation using the Mendeley API (Mendeley 2012). The performance concerns about the currently described feature will be addressed in the section 6.6.4.

As the *data provider* interface merely implements the protocol description (Lagoze et al. 2008), we will give its implementation in the listings 6.6.3.

On the other hand we had to specify what was aimed to be our *service provider* interface and the client implementation using the Mendeley API.
5.10.2.1 Service provider interface with OAI-PMH

The external harvesting is about two concerns. A first is to get related paper's metadata for a CKP publication, and the second one is to get further metadata for a CKP publication. In both cases the result is sent to the service consumer.

OAI-PMH chose its native metadata format to be the Dublin Core. A drawback of using this format is that there is no exact rules defining how the XML elements referring to related resources should be used. Therefore we can't build any implementation on this element to harvest additional metadata that match exactly a paper being viewed. Thus, we added another implementation to match this requirement, and it is explained in the next Section 5.10.2.2.

To get further external metadata we use an intermediate platform called the Registry of Open Access Repositories (ROAR). This platform implements the OAI-PMH interface to provide a wide list of Open Archive Repositories. As it implements OAI-PMH the results given to the client are in a machine processable representation (XML). The CKP service parses this XML, stores it in the CKP database to leverage the performance aspect when a CKP service consumer will use our OAI service.

Though, OAI-PMH is based on the harvesting approach, our implemented service gather functionalities of both harvesting and search approaches described by Muller and Wang (2003).

We list characteristics of both approaches thereafter:

The cross search approach is summarized by (Muller and Wang, 2003):

1. Search queries can directly be encoded (e.g. author="Schulz").
2. The service provider does not need a central database for the metadata.
3. The service provider has to deal with duplication, ranking and merging problems synchronously.
4. The service performance depends on the slowest data provider.

Whereas the harvesting approach is shaped by (Muller and Wang 2003):

1. No or only rudimentary search vocabulary.
2. The service provider needs an own database to store all metadata of the participating archives. Metadata are duplicated when using the harvesting approach.
3. The service provider has to regularly harvest the participating metadata archives in order to always provide the newest information. The service provider has to cope with update and deletion problems.
4. Providing added value services as well as tackling duplication, ranking and merging problems is not time-critical for service providers because these processes can be conducted asynchronously.

According our performance constraint and the service we wanted to provide to the Front End we kept

- an intermediary of the 2nd search approach’s bullet and the 2nd harvesting approach’s bullet: for all the repository harvested we store their full content in a file.
from the harvesting approach the bullets:

- 1, as we will see in the section we improved the native vocabulary
- 2, in order to have as quick as possible response time from the client request. Therefore an asynchronous update of the database is needed to serve fresh results.
- 3

5.10.2.2 Mendeley

As said above OAI-PMH is not a good protocol for harvesting additional metadata for an exact paper. Therefore we use the Mendeley web API to palliate this lack.

The Mendeley API requires the client to provide a certain type of URI that refer to the publication. The allow URI types are arxiv, DOI, ISBN, PMID, scopus or ssm (Mendeley, 2012). We chose to keep the current Publication data model design that contains the doi and isbn attributes. Whenever arxiv, PMID, scopus or ssm would be provided it is still possible to save them by using the FurtherField data model.

To use this API, and keep the modular architecture we gather all the business code related to this requirement in a specific module mendeley_client.

5.10.2.3 Tiers party applications used

Besides, we used the oaipmh python package, which is not a Django application, to implement the data provider interface. This package provides an abstract layer of an OAI data provider interface.

As said before, performance concerns was very important. Asynchronous update of the repository metadata harvested was a strong element to guarantee good performance level. Therefore to handle this asynchronous update, we needed to use a Django-application and a UNIX feature.

The first is django-chronograph\(^\text{1}\). This application allows to select and plan the frequency of asynchronous jobs through the django admin interface. Therefore, a python script written to update the OAI repository is launched by this application asynchronously at a day frequency.

The second feature used, crontab, allows us to launch django-chronograph backend at a wanted frequency, and this latter will run the scripts selected in the admin interface. By this combination of these two applications, we were able to update OAI repository metadata once a day at midnight and thus insure a good freshness of the results provided to the CKP - OAI service consumer.

In order to summarize up the constraints, the features wanted, and the tiers party applications used that shaped the service design we provide the Figure 5.10.2.

\(^\text{1}\)https://bitbucket.org/wnielson/django-chronograph/
Figure 5.10.2: OAI service provider interface design
Chapter 6

Implementation

6.1 System deployment and basic interface decisions

Server and CGI  The deployed version of CKP run under an Apache 2.2 server with WSGI implementation of the CGI.

HTTP Status Codes  In addition of the existing status code that CKP sends embedded in the HTTP response, the status code 409 (Fielding et al., 1999, section 10) was added to notice database integrity errors. Typically it stands for duplicated data according the uniqueness constraints defined on the data model given a resource type.

6.2 Peer Review Process

6.2.1 Roles and Permissions

In order to implement the modelisation made (see Section 5.6 and 10.1) we built a dictionary matching these descriptions. Then, deployment becomes easily onto the database thanks to python functions. All these functions and dictionary are gathered in a module called permissions in the package business_logic of the Django core application core_web_service (see listing 10.1).

6.2.2 HTTP Basic Authentication

The security has been thoroughly improved based on the HTTP Authentication. Most of the views associated with the request methods POST, PUT and DELETE requires HTTP Authentication. Based on the HTTP Authentication header provided, the system makes use of the data model designed to get the matching role for the given user credentials to grant access. An exhaustive description of which URL needs authentication according the method used is given in the service documentation.

From those two components, a large amount of the views use a first decorator to enforce HTTP Authentication to be provided and then another decorator to restrict the authenticated user to be part of a specific group (mentioned in the decorator parameter). These two decorators has been implemented in the authentication_decorators module (see listing 10.2).
Moreover the fact that we use HTTP Authentication pushed further the loose coupling idea that characterise the REST architecture. And it will be a strong asset for the page caching feature that we will see in Section 6.4.1.1.

6.2.3 Privacy

There are different levels of privacy. The esteem rating system applies for any kind of user, but the users may want to be either a public member of the rating system or an internal actor of peer review process (such as an author, a referee or an editor). Therefore different set of constraints on the user profile information must be satisfied for these different roles.

The reader role asks only for a user name, password and email for registration. After validation of the account, the user is able to rate and vote public publications. Vote casting on comments is also made available.

On the other hand the author, referee and editor roles are asked to provide further mandatory information. These are a first and last name.

6.3 Esteem and rating systems

As explain in section 5.7 the esteem rating system is gathered in the module listed in listing 10.3.

Currently, two backends has been implemented. A basic one which does not take into account the esteem of the user when he does further judgments as vote casts and ratings. Secondly, the Mizzaro model has been implemented but not yet tested.

The whole esteem rating logic is in the module implementation, and the only thing needed to trigger the process is to call the backend when a new resource such as vote or rating is saved in database. The backend will handle from this resource the assignment of the new esteem value and the rate to the user(s) and the publication.

6.4 Performance

6.4.1 Caching

Caching was the first thing to take into account in order to improve response time. This technique can be applied at two levels: at the python object level, which is made relatively easy with the Django ORM abstraction. Another level, even more efficient is at the HTTP response level. This latter caches the whole response content that is sent to the client thus increases dramatically the response time as the business code is run only the first time until the next invalidation.

6.4.1.1 Page caching

Page caching is performed with a more clever way than a fix expiring time for invalidation. We did not choose this latter approach because it has the flaw not to be accurate on the modification made by the service consumer on the resources (pure modification as well as creation and deletion).
In order to palliate this problem we use an advanced key prefixing technique that we will call virtual namespace chaining. This technique coupled with the REST architecture turns out to be very good to have a fine invalidation granularity. It uses virtual namespaces (Memcached Team [2011]) coupled with a recursive refinement of the namespace generation based on the parent virtual namespace retrieved from the cache and a string cache key.

In order to understand the whole implementation, the listing 6.1 gives the function that retrieve a virtual namespace from the cache according its parent namespace and the string key that identify the searched namespace.

**Algorithm 6.1 Virtual namespace retrieval**

```python
def _get_namespace(key, parent_namespace):
    key = key + str(parent_namespace) if parent_namespace else key
    namespace = cache.get(key)
    if not namespace:
        namespace = time.time() + random.random() * 100
        if not cache.add(key, namespace):
            namespace = cache.get(key)
    return namespace
```

**Cached Page retrieval** Based on this logic, it is now possible to cache the page, helped with the REST architecture. In the first workaround proposed we need a first raw cache key string to create a first root namespace. Afterwards we use the class design used to implements the REST architecture: we take the class name of the current resource accessed as cache key to generate a new namespace coupled with the root one. Finally with this latter namespace generated we couple it with the parameters that identify the resource accessed to get the final namespace under which the cached page is stored.

**Security** A security concern must not be forgotten. All the pages cached must be addressed according the user identity. Therefore, we added the HTTP authentication header on the last namespace generated to insure that the delivered response correspond to the user role and permissions.

The listing 6.2 shows the described logic for retrieving a cached page.

**Algorithm 6.2 Cached page retrieval**

```python
method = nonalpha_re.sub('', request.method.upper())
root_namespace = RestView._get_namespace(RestView.root_key_namespace, None)
view_namespace = RestView._get_namespace(self.__class__.__name__, root_namespace)
if method == 'GET' and CACHING_ENABLED:
    req_key=request.path
    if request.META.has_key('QUERY_STRING'):
        req_key += request.META.get('QUERY_STRING')
    request_namespace = RestView._get_namespace(req_key, view_namespace)
    auth_key = hashlib.md5(request.META.get('HTTP_AUTHORIZATION', '')).hexdigest()
    final_key = str(request_namespace) + auth_key
    ret = cache.get(final_key)
    if ret: return ret
```

**Cache coherency protocol** Invalidation of the cached page highly relies on our choice for namespace naming policy. As we chose to use the class name corresponding to the resource type
accessed as well as the URL parameters that identify uniquely the resource, we can invalidate page that are under the same class name namespace.

The approach of invalidation is as follow:

- if a POST or DELETE request is received, we invalidate all the cache because there are many cross references in the different pages cached, under the form of atom links. So to preserve cross reference integrity it is necessary to invalidate everything.

- if a PUT request is received then we know that this resource is modified but still exists. So we can only invalidate the related resource that depends on the state of this modified resource.

In order to achieve the last bullet requirement, we added an attribute in the super class RestView that contains in a tuple all the related class names that refers to resource that must be invalidate.

The listing 6.3 gives us the algorithm of invalidation.

Algorithm 6.3 Cached pages invalidation

```python
def _invalidate_resources(self, method, root_namespace, view_namespace):
    # NB: incr() is ATOMIC with MEMECACHED backend => insure cache integrity
    if method == 'POST' or 'DELETE':
        try:
            cache.incr(RestView.root_key_namespace, 1)
        except ValueError:
            cache.set(RestView.root_key_namespace, time.time())
    else:
        s_root_ns = str(root_namespace)
        key_to_invalidate = self.__class__.__name__ + s_root_ns
        cache.incr(key_to_invalidate, 1)
        except ValueError:
            cache.set(key_to_invalidate, time.time())
        for ns_to_invalidate in self.involved_key_namespaces:
            try:
                key_to_invalidate = ns_to_invalidate + s_root_ns
                cache.incr(key_to_invalidate, 1)
            except ValueError:
                cache.set(key_to_invalidate, time.time())
```

6.4.1.2 Database query caching

As mentioned in the section 5.9, we used a tiers party Django application called [Johnny-cache](Moiron and Self, 2012). This application allows to cache the Django querysets, python objects that represent and store results from database hits.

In order to cache these queryset, it monkey-patches the Django ORM engine. Therefore, when a piece of code makes use of the Django ORM layer, Johnny-cache catches all the calls and add a layer of cache management.

Basically Johnny-cache generates a key for the database query generated from the ORM engine. Then it looks up if the results are in the cache. If the results are not in the cache, then the database hit is performed and the result is stored in the cache. Otherwise the results are directly given back when the queryset is evaluated.
Invalidation  Johnny-cache is powerful in his ability to automatically invalidate cache entries by using brilliantly the ORM abstraction layer. Whenever a modification is made in the database, all entries concerned by foreign keys and cross reference tables referring to the entry modified (or added or deleted) are invalidated. A strong corollary of this feature is that any queryset can be cached with an infinite time in the cache.

**Note:** It is important to remember that cached querysets are referenced by a digest of the database query string for the next section 6.4.2.

### 6.4.2 Prefetching

As explained in the previous section 6.4.1.2, querysets are cached with an infinite time and automatically invalidated. We wanted to keep this strong feature to couple with our prefetching feature, therefore the prefetching application is based on the johnny-cache application.

As we said before, querysets are cached by johnny-cache according the database query string. This particularity implies that the prefetching engine must use exactly the same querying syntax to prefetch querysets.

Moreover, making the prefetching easily tunable by a the client consumer was one of the main requirement to satisfy. There are two main artifacts that implements this latter. An engine module that contains the core business logic to perform prefetching at the model entity level, and a rules module destined to be modified by an end user with almost no skills in python language.

**Engine**  The figure below (6.4.1) give the UML structure of the engine, the policy base class and its policy-specified inheriting classes. Refers to the appropriate design Section 5.9 to have a reminder about the activity and policy keywords.

![Prefetcher engine and its policies](image)

**Rules**  The rules module is designed to be easily editable by a non python-skilled person. It takes the approach of the rule based language. This approach has been taken because it fits well the web service approach. Indeed the URL parameters given to the web service can be easily allied with the rules’ conditions. The syntax of the pseudo rule language built is based on the following scheme
CHAPTER 6. IMPLEMENTATION

▷ Activity A:

– Policy AA:

\[ \text{Condition } C_1 \]
\[ \ast (\text{param}, \text{value})_1, \ldots, (\text{param}, \text{value})_n \rightarrow \text{resource type}_1, \ldots, \text{resource type}_n \in R^n \subseteq F \]

\[ \text{Condition } C_2 \]
\[ \ast (\text{param}_2, \text{value}_3)_1, \ldots, (\text{param}_1, \text{value}_4)_n \rightarrow \text{resource type}_4, \text{resource type}_5 \in R^2 \subseteq F \]

\ast ...

– Policy AB:

\[ \text{Condition} \]
\[ \ast (\text{param}, \text{value})_2, \ldots, (\text{param}, \text{value})_3 \rightarrow \text{resource type}_2, \ldots, \text{resource type}_n \in R^{n-1} \subseteq F \]

\[ \text{Condition} \]
\[ \ast (\text{param}_2, \text{value}_3)_3, \ldots, (\text{param}_1, \text{value}_4)_4 \rightarrow \text{resource type}_3, \text{resource type}_4 \in R^2 \subseteq F \]

\ast ...

▷ Activity B:

– ...

– ...

The CKP rule semantic corresponding for this rule syntax is:

\[ R \] is the set of all the resource types.
\[ S_e \] is the set of all the sub fields that compose the resource \( e \in R \).
\[ A \] is the set of all the extra URL parameters handle-able by the prefetcher engine.
\[ P_a \] the set of all the parameter values possible for \( a \in A \).
\[ \sigma_r \] is the set of all the possible conditions applying for the resource type \( r \in R \).
\[ C \] the condition containing a list of tuple \((f, sf)\) where \((f, sf) \in R \times S_f \) or \((f, sf) \in A \times P_f \).

\[ F \] is the set of the related resource types contained in \( R \), that will be prefetched.

With these definition the semantic of the rule language is:

If \( C \subseteq \sigma_r, \forall e \in F : \text{fetch}(e) \)

A informal and readable example will be probably meaningful:

For the activity that search for publications:

If the URL parameters look for a publication \((\in R)\) title \((\in S)\) and the URL parameter contains the extra argument \textit{limit} with the value \texttt{0,10};

then prefetch the related resources \textit{keyword, tags, and author}, of the results matching the query request.

As mentioned above, the rules are described in the module \textit{rules}. Therefore the python language has been chosen to describe the rules. This choice is justified mainly by the fact that
it avoids the prefetching engine to parse the rules from another language that could generate noticeable overhead processing time. Furthermore, the dictionary data structure provided by python is very similar to the JSON or YAML, so it is easy for a person that does not know much about python to edit the rules.

**Possibilities** As pointed out during the design section (6.4.2), it is possible to combine several rules for a policy. By this way it is possible to modelise different configurations that can occurs from the request received from the service consumer.

For instance in the search activity, if we want to trigger prefetching only if the limit argument is provided with the value 0,10 AND the search asks for a publication title, then we just have to specify these two conditions in one rule. On the other hand, for the same activity, if we want to trigger the prefetching if the limit argument is provided with the value 0,10 OR the search ask for a publication title, we put these two conditions in two different rules.

The right part of the rule allow so far only to mention resource type to prefetch. It is not possible to mention for instance another policy to apply prefetching with.

Implementation of the full prefetching application is provided in listing 10.5.

**Policies implemented**

▷ Activity: Display publication:

- *PageRendering*: The logic is to prefetch the related resources from the database on a first request received from the service consumer. Therefore additional requests received from the client to get the related resources won’t generate database hit: the data will be taken from the cache.

- *RelatedPublication*: This policy aims to prefetch the most likely further action the user would do once a publication page is displayed. As the Front End proposes to find related publication by keywords, tags, authors, and research areas, CKP can prefetch the result of these typical searches.

▷ Activity: Search for publications:

- *PageRendering*: This work the same way as the *RelatedPublication* policy for the previous activity.

- *NextSearch*: This aims to prefetch the most likely search the user would do after a first search result.

### 6.5 Internal Search

**Multiple values per argument** First of all, an improvement about the number of values for an argument provided was added. Before we could only mention a value for an argument such as:

```
/publication/author/name=Crocker&affiliation=HW.
```

Now it is possible to mention different value for one argument as well as the operator that combine the values:

```
/publication/author/name=Crocker&name=Andre[&searchtype=and].
```

By default the searchtype is set to *or*.
Author’s affiliation filtering  The author’s search integrates now a search possibility on the research area field. As the research area data model is referring to the author data model, we had to add some changes at the generic function that searched for authors (algorithm 6.4):

Algorithm 6.4  Author search

```python
def search_authors(search_items):
    result = Author.objects.none()
    if 'research_area' in search_items:
        search_items = search_items.copy()
        research_area = search_items['research_area']
        del search_items['research_area']
        research_areas = ResearchArea.objects.filter(title__icontains=research_area)
        for r in research_areas:
            result |= r.core_web_service_userprofile_related.filter(Q(user__groups__name = 'author')).distinct()
    if search_items:
        search_type, search_items = _get_search_type(search_items)
        query = build_query(search_items)
        if not result:
            result = _perform_search(Author, search_type, query)
        else:
            result = result | _perform_search(Author, search_type, query)
    result = set(result)
    return result
```

Date filtering  Moreover a filter by date was added (algorithm 6.5). It allows to filter from a specific date as well as until a specific date. The arguments can be specified separately. A usage example could be: /publication/from=1985-05-13/ until=2012-06-03.

Algorithm 6.5  Date filtering

```python
def filter_by_dates(publications, from_terms=None, until_terms=None):
    time_format = '%Y-%m-%d'
    if from_terms:
        import time, datetime
        date = datetime.datetime.fromtimestamp(time.mktime(time.strptime(from_terms, time_format)))
        publications = publications.filter(date__gte=date)
    if until_terms:
        import time, datetime
        date = datetime.datetime.fromtimestamp(time.mktime(time.strptime(until_terms, time_format)))
        publications = publications.filter(date__lte=date)
    return publications
```

Publication fields combination  In the previous implementation of the search module, it was only possible to combine with boolean operators AND and OR, fields of the resource type we were filtering publication with. It was made by this way for instance:

/publication/author/name=Crocker&affiliation=HW&searchtype=and/until=1987-09-03

Now it is possible to specify a boolean operator that links the the resource types between themselves:

/publication/author/name=Crocker&affiliation=HW&searchtype=and/until=1987-09-03/cro=(or|and).
The following algorithm 6.6 shows the modified code from the previous implementation to provide this feature.

**Algorithm 6.6 Resource combination search**

```python
def search_publications(publication_terms=None, author_terms=None, keyword_terms=None, tag_terms=None, from_terms=None, until_terms=None, cross_resource_operator=None, order_by=None, limit=None, publication_list=[], user=None):
    cross_resource_operator = cross_resource_operator.lower() if cross_resource_operator else cross_resource_operator
    if cross_resource_operator == "or" or publication_list:
        pub_ids = [pub.id for pub in publication_list]
        q_pub_ids = Q(id__in=pub_ids)
        if publication_list:
            publication_list = Publication.objects.filter(q_pub_ids)
    if publication_terms:
        publications = _search_publications(publication_terms, publication_list)
    else:
        if publication_list:
            publications = Publication.objects.filter(q_pub_ids)
        else:
            publications = Publication.objects.all()
    if author_terms:
        authors = search_authors(author_terms)
        author_ids = [author.id for author in authors]
        if cross_resource_operator == "or":
            publications = Publication.objects.filter(Q(authors__id__in=author_ids) | q_pub_ids)
        else:
            publications = publications.filter(authors__id__in=author_ids)
    if keyword_terms:
        ...
    if tag_terms:
        ...
    publications = filter_by_dates(publications, from_terms, until_terms)
    if not user or user.is_anonymous() or not user.groups.filter(name__in=['editor', 'referee', 'author']):
        publications = publications.filter(review_status=Publication.PUBLIC_STATUS)
    if order_by:
        publications = publications.order_by(order_by)
    if limit:
        offset, limit = limit.split(',
    publications = publications[int(offset):int(offset) + int(limit)]
    return publications.distinct(True)
```

**User search** This feature was improved in order to allow search on the native User data model as well as the profile. It also retrieves the results ordered by esteem value. See algorithm 6.7 for the implementation.
Algorithm 6.7 User search

```python
def search_user(search_items, limit=None):
    try:
        search_items['password']
        raise AttributeError('Can not search for password')
    except KeyError:
        pass
    mutable_search_items = search_items.copy()
    result = UserProfile.objects.all()
    for key, val in mutable_search_items.items():
        if not hasattr(UserProfile, key):
            del mutable_search_items[key]
            mutable_search_items['user__' + key] = val
    query = build_query(mutable_search_items)
    if query:
        result = result.filter(reduce(operator.or_, query))
    result = result.order_by('-esteem__value')
    if limit:
        offset, limit = limit.split(',')
        result = result[int(offset):int(offset) + int(limit)]
    return result
```

Related papers An addition relationship search was implemented: now it is possible to search from a publication, related publications by research areas and affiliations (algorithm 6.8):

Algorithm 6.8 Related papers

```python
def get_publications_by_research_areas(publication_obj):
    publications = Publication.objects.none()
    for author in publication_obj.authors.all():
        if author.user:
            for r in author.user.profile.research_areas.all():
                publications |= search_publications(author_terms=QueryDict('research_area=%s' % r.title))
    return publications
def get_publications_by_affiliation(publication_obj):
    publications = Publication.objects.none()
    for a in publication_obj.authors.all():
        if a.affiliation:
            publications |= search_publications(author_terms=QueryDict('affiliation=%s' % a.affiliation))
    return publications
```

6.6 External Harvesting

The external harvesting implementation used the same REST architecture built by Florian Bergmann. Therefore there are two views that distinguish service interface and data provider interface.

The service provider is even split up in two sub services (see Section 5.10.2): the OAI service and the Mendeley service. Both of the services are called and triggered according the URL parameters provided by the service consumer. They both return an XML content that is separated from the publication XML. We chose this approach in order to allow the usage of Web
2.0 techniques on the client side such as Ajax to load the external metadata asynchronously from the publication. The listing 10.11 shows the service implementation call.

6.6.1 OAI - Service provider Interface

Primarily, this service retrieves a list of publication that are matched according CKP publication fields criteria selected by the service consumer. The service takes different parameters:

- a set of IDs that refers to set of OAI repositories listed from the ROAR.
- a publication field that is used as search criteria to find related papers metadata.

For instance we can ask the service to look for related papers metadata from the ROAR set 1,3,5 according the publication field *keywords*. The URL for such a request would be:

```url
{service_url}/oai/metadata/publication/{publication_id}/related_by=keywords&roar_set_ids=1,3,5/
```

6.6.2 Mendeley

In opposition of the OAI service, the Mendeley service uses the Mendeley client API to harvest metadata that match exactly a publication stored in the CKP database. Therefore it is likely that there is no results sent back if the unique identifier used (DOI or ISBN) does not match any record in the Mendeley database. Nevertheless, if there is a result matching, the result retrieved from Mendeley platform is converted from JSON to XML.

The XML content contains generally an URL that points to the document location, the Mendeley URL of the metadata harvested, different statistics about the paper or journal, the abstract of the document, a set of identifiers, the authors, and the Mendeley categories.

The client implementation that uses the Mendeley API is provided in the listing 10.12.

6.6.3 OAI - Data provider Interface

The interface is strongly based on the python module *oaipmh*. We implemented and gathered all the business code related to this interface into a sub package of the oai Django application, called *data_provider*. Its content is made of one *settings* file that contains few helper functions, a *server* module and a *metadata* module. The server module contains a class that implements the *ResumptionOAIPMH* interface in order to provide the required resumption token feature *(OAI-PMH, 2003)*. Moreover it contains a factory function that return an instantiated OAI-PMH implemented server. In this factory function, we provide to the server constructor the metadata writer, that is aim to translate a CKP publication data model instance to the Dublin Core metadata format. This metadata writer is implemented in the *metadata* module.

The implementation of the data provider interface can be found in listing 10.13.

6.6.4 Performance

A simple page caching with the namespace logic is used for the data provider responses.

On the other hand, for the service provider responses, we cache according another approach. As the results sent back to the service consumer comes from XML files locally saved, we cache the result for the period of time that correspond to the frequency on which XML file are
retrieved. This frequency as been set to one day so we cache the HTTP Response for 86400 seconds.

Moreover, this module integrates a more refined caching policy. When the service consumer ask to look for external related papers metadata, he might ask according the keywords, tags, author, reference materials, organisation fields and so on. The implementation of the service provider interface will look and perform Xpath queries in the local XML files that match the constraints. As this constraints can be specified according different way, we cached the result of a Xpath query corresponding to each field constraint given. For instance if a second request is received from the service consumer to find related papers metadata with a constraint on the organisation field, if the Xpath query was already performed, then we just have to extract the value from the cache and send it back.

### 6.7 Synchronization

Getting Front-End and CKP working together was a important part of the work. As the previous implementation of both CKP and Front-End had not been tested together, several bugs and incoherences had to be corrected.

The main correction done on CKP was in the insertion concern. First the dates format was improved: the previous version included date in the publication data model regarding the bibtex format. Therefore there were only a month and year data field which were separately available. This format is good to insert publication metadata from bibtex file, though it is not good if we want to provide query expressiveness to the service consumer. Thus we added a date typed field in the data model, and now it is possible to query publications according date order criteria with the URL parameters from and until. The support for this new field is also insured for the bibtex inserter: it takes the month and the date if provided and build the date object before assigning it to the data model.

We also added functions that insert publication vote and a messaging system between two users.

Thorough modifications was added to the insertion of a new user profile. As there was not a registration system built we had to integrate such a system coupled with a registration validation process. Moreover a process to reset a forgotten password was implemented.

Additionally, we had to correct most of the insertion functions on the update (PUT) implementation. For instance, if a PUT request was received for a papergroup resource, with a new set of publications to assign, the insertion function took the the papergroup and only add the publication provided. This implementation had the flaw not to consider the XML received as a new state of the resource. Therefore the old publication that was assigned in the papergroup being modified remained in this papergroup whereas the XML received did not mention these publications.

Last but not least, we had to modify the core function that parse the XML received from the service consumer. The previous function did not parse deeply enough though it was supposed to parse until 2 levels. So the new function implemented (recursive) is able to parse XML as in the listing 6.9.
CHAPTER 6. IMPLEMENTATION

Algorithm 6.9 XML parsing

```xml
<owners>
  <owner>
    <atom:link rel="owner" type="application/xml" href="http://www2.macs.hw.ac.uk/ckp/user/1/">ckpuser</atom:link>
  </owner>
  <owner>
    <atom:link rel="owner" type="application/xml" href="http://www2.macs.hw.ac.uk/ckp/user/72/">author1</atom:link>
  </owner>
  <owner>
    <atom:link rel="owner" type="application/xml" href="http://www2.macs.hw.ac.uk/ckp/user/73/">author2</atom:link>
  </owner>
  <owner>
    <atom:link rel="owner" type="application/xml" href="http://www2.macs.hw.ac.uk/ckp/user/74/">author3</atom:link>
  </owner>
</owners>
```

6.8 Security, Safety and Privacy

Coupled with the RBAC model implementation we insure a good security level by using HTTP Basic Authentication. Moreover each user password is salted and SHA1-encrypted before being stored in database.

The privacy is also insured by the restriction of the sensible resources using authentication and the role of the user.

Specific measure has been taken with the page caching in order to avoid cached page to be accessed by non-authorized user: each page that is concerned by restrictive access is cached with the HTTP Authentication header provided by the service consumer as part of the cache key.
Chapter 7

Professional, Ethical and Legal Issues

This chapter outlines the different issues we have to pay attention during the development of the project.

7.1 Professional and Ethical Issues

There are several points on which Professional and Ethical issues apply:

- **Modular Software Design:**
  The most important aspect in this professional issue is that the CKP 2.0 is subject to be taken over by next students. Therefore a modular design with associated interfaces must be respected in order to guarantee a high interoperability between the different entities developed over the years.

- **Social Interactions:**
  As CKP 2.0 development is bound with the development of the Front End application, a frequent and fructuous communication link between the students working on both of these applications must be kept. This is the basis to insure a good interoperability between these applications.

- **User confidence:**
  After referring to the section 7.2 the legal issues related to cache systems can be summarized to matters of trust. As a copy of a work or private data is allowed to be stored temporary in the cache system for performance purpose, the user may trust the platform and its administrator they will not break down the privacy and the copyrights of those assets.

- **User profile data trustworthiness:**
  As we mentioned in the section User Interface a new user must provide personal information related to its role into the platform. The application will rely on the fact that any information provided by the user for its profile is reliable. If any user inserts unreliable profile’s data, it may be subject to see his account deleted or banned by the administrator of the system.
7.2 Legal Issues

In their article, Dodig-Crnkovic and Feldt (2009) state among other the different kind of legal issues involved in Software Engineering:

- Intellectual property:
  
  In the context of the CKP 2.0, the intellectual property concern is directly tight to the authors copyrights. Copyrights of authors are legislated in the UK under two main areas: the copyright law (1988), and the data protection law (1998). The current application already respects the specification given by Bergmann (2011) about these two laws.

  Nevertheless, there is a new issue to consider regarding the cache system. As the cache system may keep stored (even for a short period of time) a copy of an author’s work, his copyrights are involved. However The Copyright and Related Rights Regulations Act (2003, Part 2, paragraph 8, sub-paragraph 2, (a)) gives answer to this problem: temporary copy of a publication is allowed if its sole purpose is to transmit “the work in a network between third parties by an intermediary”.

- Privacy an civil liberties:

  In our case privacy concerns are related to the data provided by the user and the cache logs.

  For users data privacy concern, the report of Bergmann (2011, Chapter 3.3) states particularly well the different components in which we have to pay attention.

  On the other hand, the privacy concern on the cache logs must be explained in more details.

  As states Wessels (2001, Chapter 3.1), “privacy is a very important issue on the Internet as a whole and the Web in particular” because wherever we go on the Internet, we let information of our activity. In particular, cache systems might keep track of personal user data such as profile information. Therefore to guarantee privacy of such kind of private data stored on the cache system, the application will respect the Data Protection Act (Crown 1998) and not make abusive use of these data.
Chapter 8

Project Plan

This chapter is dedicated to outline the steps to take towards completion with milestones and risk assessment. The stepped tasks will be detailed in the section 8.1. The section 8.2 will detail the performance and system evaluations.

8.1 Work Breakdown Structure and Gantt Chart

The Work Breakdown Structure gives the step tasks in order to provide project’s end objectives and primary elements. This WBS includes the tasks from the CKP and from the Front End since development of both of this sides are tighten. The Gantt Chart represents the graphical view of the WBS. See figure 8.1.1.

Another milestone which is not visible in the WBS is shown on the Gantt chart on the 24th of April. This milestone aims to establish the structure and definition of the REST API, CKP 2.0 must provide to the Front End. Of course this may change after the design and implementation phases but it still allow to have a sketch of the overall API.

8.2 Evaluation

The evaluation will focus on different views.

First of all we will assess the performance aspects by testing different configurations of the system. The section 8.2.1 will give us the exact plan to estimate this aspect on CKP 2.0 by testing 4 different configurations from the combination of caching and prefetching. Both of these features aims to improve performance, we will measure the impact of each feature in isolation and in combination.

A second Section 8.2.2 will evaluate the concrete features implemented regarding the functional requirements.

Finally we will summarize up in Section 8.2.3 the achieved requirements and give headlines for suggestions about future work and further improvements possible for the CKP 2.0.

8.2.1 Caching and prefetching

In order to realistically assess performance, we consider two states the system can be in:
<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Jan '12</th>
<th>Feb '12</th>
<th>Mar '12</th>
<th>Apr '12</th>
<th>May '12</th>
<th>Jun '12</th>
<th>Jul '12</th>
<th>Aug '12</th>
<th>Sep '12</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Background reading &amp; Analysis requirements phase</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>3</td>
<td>Research</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>4</td>
<td>Understanding CRP</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>5</td>
<td>Deliverable literal review</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>7</td>
<td>Design Phase</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>8</td>
<td>CRP - Design roles of the platform</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>9</td>
<td>CRP - Design peer review process</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>10</td>
<td>CRP - Design external rating process</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>11</td>
<td>CRP - Deliverable first design</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>12</td>
<td>Implementation Phase</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>13</td>
<td>Front End - User registration system and Profile creation system</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>14</td>
<td>Front End - Main functionalities of the system</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>15</td>
<td>Deliverable first edition with basic functionalities</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>16</td>
<td>Front End, CRP - Improvements in the current system according to feedbacks</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>17</td>
<td>Front End, CRP - Implementing web 2.0 features</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>18</td>
<td>CRP - Cache Prefetching</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>19</td>
<td>CRP - Enhance external platform harvesting</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>20</td>
<td>Deliverable, second edition of the platform (full edition)</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>21</td>
<td>Front End, CRP - Improvements to the current system according to the feedbacks</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>22</td>
<td>Evaluation Phase</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>23</td>
<td>Writing Phase</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>24</td>
<td>Thesis writing</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
<tr>
<td>25</td>
<td>Deliverable thesis</td>
<td>01</td>
<td>07</td>
<td>18</td>
<td>29</td>
<td>11</td>
<td>03</td>
<td>04</td>
<td>01</td>
<td>03</td>
</tr>
</tbody>
</table>

Figure 8.1.1: Gantt Chart
1. Resources have never been accessed, therefore nothing is cached at the object level neither at the HTTP response level.

2. Resources have been accessed.

The impact of caching on these states is as follows:

- In case 1, the resources will be fetched from the cache rather than from the database.
- In case 2, the resources will be fetched from the database and cached.

The impact of prefetching on these states is as follows:

- In case 1, the related resources that may be requested by the client will be prefetched in the cache.
- In case 2, prefetching business code will be executed again (if no caching), but there will be no database hits.

As described in Section 6.4.2, the prefetching operates according an activity and a parameterizable policy. Though different activities were identified, the most important to test in performance terms is the view of a publication page and the view of a search result. For the sake of simplicity we use acronyms (N)C for (No) Cache and (N)P for (No) Prefetching thereafter.

8.2.1.1 Publication page: a test representative for the reader role

To perform performance test on a publication page view we wanted to take the approach of the client consumer. From that point of view we built a script that imitate the behavior of this client to fetch all the resources needed to display a publication web page.

As we said the policy is parameterizable. Therefore, we will organize the tests according different policy parameterisations, on the base of the PageRendering policy (see [6.4.2] for more information on this policy). We will see as well according our evaluation that the RelatedPublication policy tends to be less efficient.

Moreover we set up a specific resource distribution for each of the typical publication resources we test. Each publication has 1 more related resource than the previous tested. Referring to the figure 8.2.1 will give a better explanation than a long paragraph.
Finally each of the following paragraph will give the wall clock time to fetch all the resource related to a publication needed to build up the web page. The X abscissa is the same as the ordinate given in Figure 8.2.1 that is to say the number of related resource’s links given a publication.

As we wanted to perform the evaluation in a production environment, the script used to evaluate performances was executed on remote machine that sent requests to the server where the production version of CKP is hosted. Though, As Front End is hosted on the same machine as CKP, we made an estimation of the round trip delay existing between the remote machine and the server to have results more close to the real usage.

An important details has also to be recalled: the cache replacement algorithm chosen by Brad Fitzpatrick for memcached is LRU (Last Recently Used).

Below is the summary of all the configuration tested:

Figure 8.2.2: Summary (include RelatedPublications policy)

Separated essential plots presented below, and an outcome discussion is provided further
down.

Figure 8.2.3: No cache and no prefetching
Figure 8.2.4: Prefetching with no policy and objects cached

Figure 8.2.5: Prefetching with PageRendering policy
CHAPTER 8. PROJECT PLAN

Outcomes  As we could expect the wall clock time is at the lowest when the entire HTTP response as been cached. On the other hand, the highest wall clock times is when no cache neither prefetching is turned on. Between those both cases, comes the situation where no prefetching is performed but the objects are already cached.

Finally the PageRendering policy turns out to be faster than RelatedPublications. This can be explained if we keep in mind that we are in the context of the Front End that want to generate the web page associated to the publication resource. Therefore all the requests towards the related resource to build the web page are performed, and, additional hits to database are performed to prefetch the search result of related publications.

A first conclusion about these results is that PageRendering policy turns out to be the most appropriate in term of policy choice. Besides, the fact that the prefetching feature is built upon the johnny-cache application bring a strong asset: several resources have a many-to-many relationships (such as tags, keywords, authors, research_areas), therefore once these related resources have been prefetched for a given publication, if they are involved in another publication, prefetching them won’t hit the database anymore.

The second conclusion is that the best way to improve performance on CKP, is to combine the different assets of prefetching, and page caching. This combination would be to uses both page caching and prefetching, where prefetching would take place when the page is not yet cached. When the result is sent back to the client, we cache the whole XML page generated. Therefore the next request that will ask for this same resource, the response time will be dramatically increased.
8.2.1.2 Search: a test representative for a reader role

The search is the object of two performance features where both of them aim to support the work-flow in two different ways.

**Prefetching:** This first approach aims to prefetch the result of the most likely next search to be performed by the end user once a first search as been performed.

For instance if a user searches for a publication title, it is likely he would perform another search later on with one of the tag contained in the publications of the first search result.

The performance result of such a work-flow is given in Figure 8.2.7.

![Figure 8.2.7: Search and Next Search prefetched](image)

**Resumption Token:** This second approach is based on the same concept that OAI-PMH uses. See the Figure 5.10.1 to have the understanding of the work-flow design. The performance results are shown on Figure 8.2.8.
As we can see the resumption token approach does not seem to be such efficient for a small amount of publications retrieved. Moreover the wall clock time seems to be quite similar between the first search and the first refining until the limit threshold reaches 35. Forward this limit it is clear that using the resumption token lower the response time.

**Conclusion** In a nutshell, both of these features can be combined together, where the resumption token can be an additive option that the end user can trigger through the Front End search interface. As we’ve seen, the feature that uses the resumption token technique becomes efficient only for a limit threshold about 35. A good configuration could be then to set the limit threshold for all the search to 40, and provide the resumption token as an option to the user.

Finally if we consider the page caching with this prefetching and resumption token techniques the wall clock time will be dramatically lowered on repetitive search requests.

### 8.2.2 Web 2.0 features and Software Design

#### 8.2.2.1 External Harvesting

The external harvesting has been implemented through two services: the OAI service provider interface and the Mendeley client API. Furthermore the performance issues involved in these two service implementations were addressed by using the low level cache API provided by Django. Each request that differs from any previous one has its results cached. Therefore any repeated request to the services are replied much more quickly.

Moreover the data provider interface has been implemented according the protocol description given by the OAI([Lagoze et al.](Lagoze et al., 2008)). Though the set hierarchy, advised to be implemented in the protocol notes but not mandatory has not been implemented.
Finally the freshness of the results for the services implemented is insured by an automated process that update every day the local data harvested from the remote repositories.

\subsection*{8.2.2.2 Expressiveness of search query}

The expressiveness of a query to search for publication has been improved. To summarise and show the capacities of CKP in terms of search queries we give the following table that compare expressiveness of CKP queries and Pure queries (Figure 8.2). The feature comparison was done from the web service documentation document [Her].

As described in the Section \[5\] we took care to develop more business code and keep the good object oriented practice and web MVC model, by using classes, methods as well as the web-application architecture provided by the Django framework.

\subsection*{8.2.3 Fullfilled requirements}

We give below the list of which requirements were fullfiled.
<table>
<thead>
<tr>
<th>Feature</th>
<th>CKP 2011</th>
<th>CKP 2012</th>
<th>Pure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limiting publications result</td>
<td>limit and offset number as well as date range or threshold</td>
<td>limit and offset as well as created/modified/published resource date threshold</td>
<td></td>
</tr>
<tr>
<td>Ordering</td>
<td>According publication fields.</td>
<td>According fields provided for ordering publication</td>
<td></td>
</tr>
<tr>
<td>Reverse Ordering</td>
<td>Available</td>
<td>Available</td>
<td></td>
</tr>
<tr>
<td>Rendering format</td>
<td>Native CKP-XML format</td>
<td>+ Dublin Core metadata format for the OAI data provider interface</td>
<td>configurable: the two guaranteed are “xml_short” (publication overview) and “xml_long” (detailed publication page).</td>
</tr>
<tr>
<td>Search</td>
<td>based on publication-related resources as such author’s name and affiliation, as well as keyword, tags, publication title, editor, publisher.</td>
<td>+ date, submission_date, last_modification_date are available fields</td>
<td>Search for name, title, description, and other indexed publication fields. Uses:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>▶ operator to combine query string words: OR/AND/NOT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>▶ Wildcard: ?/*/~</td>
</tr>
<tr>
<td>Search filtering</td>
<td>same as previous cell</td>
<td>same as previous cell</td>
<td>Search can be restricted by:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>▶ publication presented in the profile’s CV,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>▶ type or keyword classification (description from another URL),</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>▶ user-defined keywords (CKP tags),</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>▶ associated persons (CKP authors)/owners(CKP owners)/organisation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>▶ external source / source_id</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>▶ work-flow/publication state</td>
</tr>
<tr>
<td>Unique publication retrieval</td>
<td>retrieved thanks to the unique id provided</td>
<td>retrieved thanks to the unique id provided</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.2: Comparison of publications extraction methods CKP/Pure
CHAPTER 8. PROJECT PLAN

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Status</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>✔</td>
<td>A simple flag triggers the cache.</td>
</tr>
<tr>
<td>P02</td>
<td>✔</td>
<td>Through the prefetching_policy URL parameter</td>
</tr>
<tr>
<td>P03</td>
<td>✔</td>
<td>Thanks to the rules module</td>
</tr>
<tr>
<td>P04</td>
<td>✔</td>
<td>Uses page caching</td>
</tr>
<tr>
<td>PR01</td>
<td>✔</td>
<td>Roles</td>
</tr>
<tr>
<td>PR02</td>
<td>✔</td>
<td>Gathered in permissions module</td>
</tr>
<tr>
<td>PR03</td>
<td>✔</td>
<td>Done on Front End side</td>
</tr>
<tr>
<td>WT01</td>
<td>✔</td>
<td>Backend Esteem implementation and integration of Mizzaro model</td>
</tr>
<tr>
<td>WT02</td>
<td>✔</td>
<td>Made possible with a reader profile</td>
</tr>
<tr>
<td>WT03</td>
<td>✔</td>
<td>Done in backend implementation</td>
</tr>
<tr>
<td>WT04</td>
<td>✗</td>
<td>Original search module kept and improved</td>
</tr>
<tr>
<td>WT05</td>
<td>✔</td>
<td>As described in section 6.5</td>
</tr>
<tr>
<td>UM01</td>
<td>✔</td>
<td>Through the Django admin interface</td>
</tr>
<tr>
<td>UM02</td>
<td>✔</td>
<td>Through the Django admin interface</td>
</tr>
<tr>
<td>UM03</td>
<td>✔</td>
<td>Through the Django admin interface</td>
</tr>
<tr>
<td>UI01</td>
<td>✔</td>
<td>Integrate in the insertion modules</td>
</tr>
<tr>
<td>UI02</td>
<td>✔</td>
<td>Integrate in the insertion modules</td>
</tr>
<tr>
<td>DM01</td>
<td>✔</td>
<td>As described in 5.5</td>
</tr>
<tr>
<td>EI01</td>
<td>✔</td>
<td>By using ROAR</td>
</tr>
<tr>
<td>EI02</td>
<td>✔</td>
<td>Through Data provider and Service provider implementations</td>
</tr>
<tr>
<td>EI03</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>EI04</td>
<td>~</td>
<td>The system must be in a production mode to be asked to be integrated in google scholar</td>
</tr>
</tbody>
</table>

Table 8.3: Requirements summary

8.2.4 Further Work

**Interface with external platform**  As mentioned in the section 8.2.2.1 the set hierarchy in the OAI data provider interface is not implemented. In order to introduce even more structure in the publication’s metadata storage, implementing this feature could improve CKP data provider interface compliance for selective harvesting performed by external platforms.
Page Caching  As described in section 6.4.1.1, the page caching follow a chained virtual
namespace logic to cache the pages return to the service consumer. This namespace chaining
logic could be pushed further to increase the granularity of the invalidation policy of the cached
pages by CKP when POST, PUT, or DELETE method are received.

Template Architecture  As it was pointed out, a flaw of the strict respect of the REST
architecture is to represent resources canonically. Therefore it involves a lot of side requests to
gather all the resources in order to build a web page in the case of Front End. By using Django
template tags to integrate XML of related resources directly in the core XML response, the
wall clock time needed to build up a web page could dramatically decrease. The architecture
would be slightly less REST oriented but still thoroughly structured.

Prefetching engine and rules  In the rules module it would be possible to add more pos-
sibilities for the right part of the rules. In addition of allowing to mention resource types to
prefetch, we could make possible to mention to prefetch according another policy described.

Propose the most famous search on the Front End  This idea is a good way to use as
much as possible the cached search pages and thus increase the use of the cache.

Password recovery  The process that allow to get a new password when the user has lost his
password could be improved in terms of security. The actual process generates a new password
for the given user without any authentication needed and send this new password through the
network to the service consumer that send an email with this new password.
It could be great to avoid the new password to be viewed by any spoofer or man in the
middle.

SSL  Setting up SSL could dramatically increase the security aspect. All the data exchanged
between CKP and the service consumer would be encrypted. On the server there is not much
to do except enabling the Apache SSL module on which CKP is running on.
Chapter 9

Conclusion

This project aimed to evaluate the previous implemented system and add specific features that uses web 2.0 techniques to leverage quality assessment of papers. Performance improvements and interaction with external database interfaces were the two others objectives of this work.

Therefore, we described in the literature review an overview of the different concepts used in the previous implementation such as the REST architecture, the main web 2.0 features. Moreover we provided a review of OAI-PMH, the protocol used to implement external harvesting feature. Finally we gave an overview of the different performance features existing and the ones chosen to be implemented in our system to improve the wall clock time.

From on these overviews, we added several new features. On the peer review aspect, we completed the query expressiveness, implemented the roles permissions system and integrated and tighten esteem-based rating system with all the user types defined.

We also had a strong focus on the performance issues. We set up a two-level caching system and implemented a service-consumer triggerable prefetching system that can take into account a user defined policy.

Finally, while we developed the OAI-PMH data provider and service provider interfaces as well as the client Mendeley API, we paid a strong attention to insure good performance results for these features when they are used. More specifically we saw that a combination of page caching and prefetching with the PageRendering policy was the best combination of settings for high-performance.

All these added features were implemented upon the system built by Florian Bergmann, using the Python language and the web framework Django. All the work done was dramatically helped by the very extensible code written by Florian Bergmann as well as all the actor of the Web 2.0 that give their little help to everyone through forums such as Stackoverflow or their own blogs. Additionally the Django community and the numerous open source Django applications shared eased development of certain features.
Chapter 10

Annexes

10.1 Permissions tables

When OK word is used, it stands for an entity-level permission granted. When ‘X’ is used it means that the access is not granted to the role. When OL (Object-Level) is used it specifies that the access for this resource with the given role is restricted to an object level. Most of the time the object level restriction correspond to a restriction access to the owner of the object. Therefore when it is this case it won’t be noticed, but if it is for another reason, an explanation will be provided. Sometimes there are additional conditions on top of the OL restriction and they will be mentioned.

▷ **Author resource**

<table>
<thead>
<tr>
<th></th>
<th>Anonymous/Reader</th>
<th>Author</th>
<th>Referee</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Add</td>
<td>X</td>
<td>OK</td>
<td>X</td>
<td>OK</td>
</tr>
<tr>
<td>Change</td>
<td>X</td>
<td>OL</td>
<td>X</td>
<td>OK</td>
</tr>
<tr>
<td>Delete</td>
<td>X</td>
<td>OL</td>
<td>X</td>
<td>OK</td>
</tr>
</tbody>
</table>

Table 10.1: Author

▷ **Comment resource**

<table>
<thead>
<tr>
<th></th>
<th>Anonymous</th>
<th>Reader</th>
<th>Author</th>
<th>Referee</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Add</td>
<td>X</td>
<td>OK</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Change</td>
<td>X</td>
<td>OL</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Delete</td>
<td>X</td>
<td>OL</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 10.2: Comment
### Esteem resource

<table>
<thead>
<tr>
<th></th>
<th>Anonymous</th>
<th>Reader</th>
<th>Author</th>
<th>Referee</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Add</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Change</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Delete</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 10.3: Esteem

### Keyword resource:

<table>
<thead>
<tr>
<th></th>
<th>Anonymous</th>
<th>Reader</th>
<th>Author</th>
<th>Referee</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Add</td>
<td>X</td>
<td>X</td>
<td>OK</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Change</td>
<td>X</td>
<td>X</td>
<td>OL</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Delete</td>
<td>X</td>
<td>X</td>
<td>OL</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 10.4: Keywords

### Message resource:

<table>
<thead>
<tr>
<th></th>
<th>Anonymous</th>
<th>Reader</th>
<th>Author</th>
<th>Referee</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>X</td>
<td>X</td>
<td>OL</td>
<td>OL</td>
<td>OL</td>
</tr>
<tr>
<td>Add</td>
<td>X</td>
<td>X</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Change</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Delete</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 10.5: Esteem

### PaperGroup resource:

<table>
<thead>
<tr>
<th></th>
<th>Anonymous</th>
<th>Reader</th>
<th>Author</th>
<th>Referee</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>OL</td>
<td>OL</td>
</tr>
<tr>
<td>Add</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>OK</td>
</tr>
<tr>
<td>Change</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>OL</td>
</tr>
<tr>
<td>Delete</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>OL</td>
</tr>
</tbody>
</table>

Table 10.6: PaperGroup
PeerReview resource:

<table>
<thead>
<tr>
<th>Anonymous/Reader</th>
<th>Author</th>
<th>Referee</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>OK [condition]</td>
<td>OK [condition]</td>
<td>OL</td>
</tr>
<tr>
<td>Add</td>
<td>X</td>
<td>X</td>
<td>OL</td>
</tr>
<tr>
<td>Change</td>
<td>X</td>
<td>X</td>
<td>OL</td>
</tr>
<tr>
<td>Delete</td>
<td>X</td>
<td>X</td>
<td>OL</td>
</tr>
</tbody>
</table>

Table 10.7: PeerReview

Publication resource:

<table>
<thead>
<tr>
<th>Anonymous/Reader</th>
<th>Author</th>
<th>Referee</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>OK [condition]</td>
<td>OL</td>
<td>OL</td>
</tr>
<tr>
<td>Add</td>
<td>X</td>
<td>OK</td>
<td>X</td>
</tr>
<tr>
<td>Change</td>
<td>X</td>
<td>OL[condition2]</td>
<td>X</td>
</tr>
<tr>
<td>Delete</td>
<td>X</td>
<td>OL[condition3]</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 10.8: Publication

Rating resource:

<table>
<thead>
<tr>
<th>Anonymous</th>
<th>Reader</th>
<th>Author</th>
<th>Referee</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Add</td>
<td>X</td>
<td>OK</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Change</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Delete</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 10.9: Rating

ReferenceMaterial resource:
Table 10.10: ReferenceMaterials

ResearchArea resource:

Table 10.11: ResearchArea

Tag resource:

Table 10.12: Tags

UserProfile resource:

Table 10.13: UserProfile
Vote resource:

<table>
<thead>
<tr>
<th></th>
<th>Anonymous</th>
<th>Reader</th>
<th>Author</th>
<th>Referee</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Add</td>
<td>X</td>
<td>OK</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Change</td>
<td>X</td>
<td>OL</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Delete</td>
<td>X</td>
<td>OL</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 10.14: Vote

10.2 Listings

Listing 10.1: Permissions

```python
# Created on 1 May 2012

Create the different roles of the system and associate permission
to each of them accordingly
To find out more about the roles-permissions assignation see the
reference document.

Reference Document: CKP_design.pdf

@author: victorinox

from core_web_service.models import Comment, Rating, Tag, CommentVote,
    ResearchArea,
    Esteem, Author, Keyword, PaperGroup, Publication, ReferenceMaterial,
    PublicationVote
from django.contrib.auth.models import Permission

GROUPS_PERMISSIONS = {
    'reader':{
        'add':[Comment, Rating, Tag, CommentVote, ResearchArea, 
            PublicationVote ],
        'change':[Esteem, Tag, CommentVote, PublicationVote],
        'delete':[Tag, CommentVote, PublicationVote],
    },
    'author':{
        'add':[Author, Keyword, PaperGroup, Publication, CommentVote, 
            ReferenceMaterial, ResearchArea ],
        'change':[],
        'delete':[],
    },
    'referee':{
        'add':[ResearchArea],
        'change':[],
        'delete':[],
    },
}
'editor': {
    'add': [PaperGroup, Author, ResearchArea],
    'change': [PaperGroup, Author, ],
    'delete': [PaperGroup, Author, ],
}

def reset_permissions(group_instance):
    group_instance.permissions.clear()

def get_permission_codenames(group_instance):
    group_permissions = GROUPS_PERMISSIONS[group_instance.name.lower()]
    perm_codenames = []
    for type, model_classes in group_permissions.items():
        for model_class in model_classes:
            perm = '%s_%s' % (type, model_class.__name__.lower())
            perm_codenames.append(perm)
    return perm_codenames

def get_perm_queryset(perm_codenames):
    from django.db.models import Q
    perms_queryset = Permission.objects.filter(
        content_type__app_label="core_web_service")
    q = None
    for p in perm_codenames:
        cq = Q(codename=p)
        if q:
            q.add(cq, Q.OR)
        else:
            q = cq
    if q:
        return perms_queryset.filter(q)
    else:
        return None

def set_permissions(group_instance):
    keys = GROUPS_PERMISSIONS.keys()
    if group_instance.name.lower() not in keys:
        raise KeyError("The role %s is not in one of these: %s" % (
            group_instance.name, keys))

    perm_codenames = get_permission_codenames(group_instance)
    perm_queryset = get_perm_queryset(perm_codenames)
    if perm_queryset:
        group_instance.permissions.add(*perm_queryset)
        group_instance.save()
    return group_instance

Listing 10.2: Authentication decorators
from django.contrib.auth.decorators import user_passes_test
from functools import wraps
import logging, os
from django.http import HttpResponse
from settings import LOGFILE
from django.contrib.auth import authenticate, login

logger = logging.getLogger('myproject.custom')
if os.path.exists(LOGFILE):
    os.remove(LOGFILE)

hdlr = logging.FileHandler(LOGFILE)
formatter = logging.Formatter('%(asctime)s/uni2423%(levelname)s/uni2423%(message)s')
hdlr.setFormatter(formatter)
logger.addHandler(hdlr)
logger.setLevel(logging.INFO)

def http_authenticate(request):
    user = request.user
    if request.META.has_key('HTTP_AUTHORIZATION'):
        authmeth, auth = request.META['HTTP_AUTHORIZATION'].split('uni2423 ', 1)
        log = logger.info("HTTP/uni2423auth/uni2423method/uni2423:/uni2423%s/uni2423" % authmeth)
        logger.info("HTTP/uni242364/uni2423encoded/uni2423string/uni2423:/uni2423%s/uni2423" % auth)
        if authmeth.lower() == 'basic':
            auth = auth.strip().decode('base64')
            username, password = auth.split(':', 1)
            logger.info("HTTP/uni2423username/uni2423:/uni2423%s/uni2423" % username)
            user = authenticate(username=username, password=password)
        if user:
            login(request, user)
        logger.info("user/uni2423authenticated/uni2423:/uni2423%s/uni2423" % user)
        return func(request, *args, **kwargs)
logger.info("user\u001aauthentication\u001a\u001afailed")
response = HttpResponse()
response.status_code = 403
logger.info(response)
return response

return _decorator

def group_required(*group_names):
    """
    Requires user membership in at least one of the groups passed in.
    @author: msanders
    @source: http://djangosnippets.org/snippets/1703/
    """
    def in_groups(u):
        if u.is_authenticated():
            if bool(u.groups.filter(name__in=group_names)) | u.is_superuser:
                return True
        return False
    return user_passes_test(in_groups)

Listing 10.3: Esteem Rating init module

###
Created on 8 May 2012

Package helper function to get the esteem−rating backend according the one given in settings project file.

:author: Victor G

```
from django.core.exceptions import ImproperlyConfigured
from settings import ESTEEM_RATING_BACKEND

# Python 2.7 has an importlib with import_module; for older Pythons,
# Django's bundled copy provides it.
try: from importlib import import_module # pragma: no cover
except ImportError: from django.utils import importlib import import_module # pragma: no cover

def get_backend():
    """
    Return an instance of a registration backend, given the dotted Python import path specified in settings.py (as a string) to the backend class.

    If the backend cannot be located (e.g., because no such module exists, or because the module does not contain a class of the appropriate name), **django.core.exceptions.ImproperlyConfigured** is raised.
    """
    path = ESTEEM_RATING_BACKEND
```
i = path.rfind('.')
module, attr = path[:i], path[i + 1:]
try:
    mod = import_module(module)
except ImportError, e:
    raise ImproperlyConfigured('Error/uni2423loading/uni2423 registration/uni2423backend/uni2423%s 
""" % (module, e))
try:
    backend_class = getattr(mod, attr)
except AttributeError:
    raise ImproperlyConfigured('Module "%s" does not define a esteem backend named "%s"
""" % (module, attr))
return backend_class()

Listing 10.4: Esteem rating backends

Created on 8 May 2012

Contains the different esteem-rating backends
Each new implementation must inherit "BaseBackend".

from abc import abstractmethod, ABCMeta
from core_web_service.models import Esteem, CommentVote, Rating, Publication,
UPVOTE, DELETED, DELETED
import logging

class BaseBackend(object):
    ""
    Define the method used during the esteem rating process.
    These methods are called from the models before or after save call according their purpose.
    ""
    __metaclass__ = ABCMeta
    def __init__(self):
        super(BaseBackend, self).__init__()

    @abstractmethod
def calculate_average_rating(self, publication_instance, **kwargs):
        ""
        Calculate average rating of a publication
        ""
        pass

def create_esteem_for_profile(self, userprofile_instance, **kwargs):
    if userprofile_instance.pk is None:
        return Esteem.objects.create()
    else:
        return userprofile_instance.esteem

    @abstractmethod
def calculate_esteem_from_vote(self, vote_instance, **kwargs):
        ""
        Update esteem of profile related to this vote.
        They might be concerned by a vote if they are the author of the comment
        ""
        pass
being voted, or if they are caster of the vote
""
pass

@abstractmethod
def calculate_esteem_from_rating(self, rating_instance, **kwargs):

""" Update esteem of profile related to this rating.
They might be concerned by a rating if they are author of the
publication being rated, or if they are caster of the rating ""
pass

class SimpleBackend(BaseBackend):

"""
Implements esteem rating according the following specs:
− A Publication resource/model can be rated by user (except owners)
  Owners of the publication being rated will see their esteem updated
  according the rate given.
− Publications store their average rating into a field
  average_rating
− A Comment resource/model can be upvoted or downvoted
  Esteem value of the comment author will be updated according
  the vote type
"""
def __init__(self):
    super(SimpleBackend, self).__init__()

def calculate_average_rating(self, publication_instance, **kwargs):
    sum_of_values = 0
    number = publication_instance.rating_set.count()
    average_rating = 0
    for rating in publication_instance.rating_set.all():
        sum_of_values += rating.rating
        if number != 0:
            average_rating = str(float(sum_of_values) / float(number))
    else:
        average_rating = str(0.0)
    publication_instance.average_rating = average_rating
    return publication_instance


def calculate_esteem_from_vote(self, vote_instance, **kwargs):

"""
Update esteem of the comment author
If the vote caster is one of the owners of the publication the
comment is related to, esteem attributed is raised.
Operates change on commentator esteem only.
"""
new_esteem = 0
vote = vote_instance
m = 1 if vote.votetype == UPVOTE else -1
commentvote = kwargs['commentvote']
if commentvote:
    comment = vote.comment
    publication = comment.publication
    try:
        user = comment.user # The user who has submitted the comment
        old_esteem = user.profile.esteem.value
        esteem = user.profile.esteem
        if user:
            publication = comment.publication
            publication_owners = publication.owners.all()
            caster = vote.caster # The user who voted on a comment
            if caster in publication_owners:
                # The caster is one of the owner of the publication
                # being commented.
                new_esteem += 20 * m
            else:
                new_esteem += 5 * m
        temp_new_esteem = old_esteem + new_esteem
        new_esteem = temp_new_esteem if temp_new_esteem >= 0 else 0.01
        esteem.value = str(new_esteem)
        esteem.save()
    except AttributeError, e:
        logger = logging.getLogger('myproject.custom')
        logger.error(e)
else:
    owners = vote.publication.owners.all()
    for owner in owners:
        esteem = owner.profile.esteem
        tmp_new_esteem = esteem.value + 5 * m
        new_esteem = tmp_new_esteem if tmp_new_esteem >= 0 else 0.01
        esteem.value = str(new_esteem)
        esteem.save()

def calculate_esteem_from_rating(self, rating_instance, **kwargs):
    """
    A rating is a number between 0 and SCALE constant.
    To update esteem, a 'shift' constant is calculated: SCALE / 2
    the given rate value is calculated.
    Then a 'factor' is deduced by subtracting the rate by 'shift'
    Therefore -'shift' <= 'factor' <= 'shift'
    The esteem is then updated by adding an ADDITIVE_CONSTANT
    multiplied by a 'factor'.
    Operates change on publication owners' esteem only
    """
    ADDITIVE_CONSTANT = 5
    shift = Rating.SCALE / 2
    factor = int(rating_instance.rating) - shift
    publication = rating_instance.publication
    owners = publication.owners.all()
    for owner in owners:
        esteem = owner.profile.esteem
        esteem_value = esteem.value
        new_esteem_value = esteem_value + factor * ADDITIVE_CONSTANT
class MizzaroBackend(BaseBackend):
    
    THIS BACKEND IS NOT TESTED
    This backend is based on Mizzaro’s work about ‘Quality Control
    In Scholarly Publishing’. This implementation makes use of the steadiness field added in
    the models Publication, and Esteem. It also exclude the time dimension used in original Mizzaro work,
    in order to avoid very long database lookups.
    Mizzaro uses the ‘score’ vocabulary whereas we defined in our data
    model the equivalent under ‘average_rating’ for publications and
    ‘esteem.value’ for user profiles.
    @author: Stefano Mizzaro
    @email: mizzaro@dimi.uniud.it
    @document: http://www.dimi.uniud.it/mizzaro/research/papers/EJ−JASIST.pdf

    def __init__(self):
        super(MizzaroBackend, self).__init__()

    def calculate_average_rating(self, publication_instance, **kwargs):
        
        all_ratings = publication_instance.rating_set.all()
        steadiness = 0
        t = 0
        average_rating = 0

        for rating in all_ratings:
            caster = rating.caster
            caster_esteem = caster.profile.esteem.value
            t = t + rating.rating * caster_esteem
            casters_esteem_sum = steadiness + caster_esteem

        average_rating = t / steadiness if steadiness != 0 else 0
        publication_instance.average_rating = str(average_rating)
        publication_instance.steadiness = str(steadiness)

        # Authors (owners) esteem update
        for owner in publication_instance.owners.all():
            pubs = Publication.objects.filter(owners__in=[owner])

            if publication_instance.pk:
                # Exclude the current publication being updated
                pubs.exclude(pk=publication_instance.pk)

            a, owner_steadiness = 0, 0
            owner_esteem = owner.profile.esteem
            for pub in pubs:
                ...
a = a + pub.steadiness * pub.average_rating
owner_steadiness = owner_steadiness + pub.steadiness

# We add the freshest steadiness and average_rating values of the current publication being updated.
# a = a + publication_instance.steadiness * publication_instance.average_rating
owner_steadiness = owner_steadiness + publication_instance.steadiness

new_esteem = a / owner_steadiness if owner_steadiness != 0.0 else 0.01
owner_esteem.value = str(new_esteem)
owner_esteem.steadiness = str(owner_steadiness)
owner_esteem.save()

return publication_instance

def calculate_esteem_from_vote(self, vote_instance,**kwargs):
    """ Update esteem of profile related to this vote. They might be concerned by a vote if they are author of the comment being voted, or if they are caster of the vote """
    if vote_instance.votetype == DELETED:
        return
    comment_author_esteem = vote_instance.comment.user.profile.esteem
caster = vote_instance.caster
estime = caster.profile.esteem
judgement = 1 if vote_instance.votetype == UPVOTE else -1
judgement = judgement * esteem.value

    # If the vote is of type UPVOTE the comment's author # has his steadiness increased by the caster esteem value. Otherwise decreased
comment_author_esteem.steadiness = comment_author_esteem.steadiness + judgement
comment_author_esteem.save()

def calculate_esteem_from_rating(self, rating_instance,**kwargs):
    """ Update esteem of profile related to this rating. They might be concerned by a rating if they are author of the publication being rated, or if they are caster of the rating """
    import math
publication = rating_instance.publication
caster = rating_instance.caster
judgement = int(rating_instance.rating) / Rating.SCALE # [0,1]
caster_ratings = Rating.objects.filter(caster=caster).select_related('publication')
caster_ratings = caster_ratings.only('publication__average_rating', 'publication__steadiness')
if rating_instance.pk:
    caster_ratings.exclude(pk=rating_instance.pk)
a, b = 0, 0
for rating in caster_ratings:
    pub_steadiness = rating.publication.steadiness
CHAPTER 10. ANNEXES

pub_average_rating = rating.publication.average_rating

goodness = 1 - math.sqrt(math.fabs(judgement - pub_average_rating))

a = a + goodness * pub_steadiness
b = b + pub_steadiness

goodness = 1 - math.sqrt(math.fabs(judgement - publication.average_rating))
a = a + goodness * publication.steadiness
b = b + publication.steadiness

estem = caster.profile.esteem
new_estem = a / b if b != 0 else 0

estem.value = str(new_estem)
estem.steadiness = str(b)
estem.save()
publication.save()
raise BaseException('No "%s" activity' % activity)
self.activity = getattr(rules, activity)

if not self.activity.has_key(policy):
    raise BaseException('No "%s" policy' % policy)

self.policy = policy
if not self._find_applying_rules(input_fields):
    self.enabled = False
    return

if not queryset_or_model_instance:
    self.model_list = []
    return

if isinstance(queryset_or_model_instance, QuerySet) or isinstance(queryset_or_model_instance, list):
    self.model_list = queryset_or_model_instance
elif isinstance(queryset_or_model_instance, Model):
    self.model_list = [queryset_or_model_instance]
else:
    raise BaseException("Object used for prefetching must be a queryset or a model instance")

def _find_applying_rules(self, input_fields):
    
    Find the set of rules from the rules module corresponding of the input_fields.
    If not input_fields are described in the rules module the the rule apply directly.
    
    :param input_fields: criteria to find the applying rule
    
    
    self.rules = []
    all_rules = self.activity[self.policy]['rules']
    for rule in all_rules:
        if not rule['input_fields']:
            self.rules.append(rule)
            continue
        rule_apply = True
        for key, val in input_fields.items():
            if val:
                if isinstance(val, QueryDict) and (key, val.keys()) not in rule['input_fields']:
                    rule_apply = False
                    break
                elif isinstance(val, basestring) and (key, [val]) not in rule['input_fields']:
                    rule_apply = False
                    break
        if rule_apply:
            self.rules.append(rule)
    return self.rules
def prefetch(self, *args, **kwargs):
    
    Prefetching factory.
    Get and instantiate the prefetcher matching the policy chosen.
    Then call the business method prefetch() of the instantiated prefetcher
    
    if not self.enabled: return
    import sys
    policy_class = "%sPolicy" % self.policy
    thismodule = sys.modules['__name__']
    policy_prefetcher = getattr(thismodule, policy_class)(self.rules,
                            self.model_list, *self.args, **self.kwargs)
    for rule in self.rules:
        if rule.has_key('extra_args'):
            args = rule['extra_args']
            if args:
                policy_prefetcher._apply_rules_extra_args(args)
            policy_prefetcher.prefetch(*args, **kwargs)
    return

class BasePolicy(object):
    
    Base class to subclass for implementing policy-based prefetcher
    :ivar args: list of supplementary arguments for future improvements
    :ivar kwargs: list of supplementary keyword arguments for future improvements
    :ivar rules: rules applying for this prefetcher
    :ivar model_list: list of model instances prefetcher will use to prefetch related resource from this model instance.

    def __init__(self, rules, model_list, *args, **kwargs):
        self.args = args
        self.kwargs = kwargs
        self.rules = rules
        self.model_list = model_list

    @abstractmethod
def _apply_rules_extra_args(self, args):
        
        Handle extra args given in the rule description (module rules)
    
    pass

    @abstractmethod
def prefetch(self, *args, **kwargs):
        
        Prefetch according rules selected and policy class
    
    pass
class PageRenderingPolicy(BasePolicy):
    
    ***
    Prefetch related resources which are related to a publication object.
    Will prefetch only the output fields specified in the 'rules' module
    for the activity 'publication_details' and the policy 'PageRendering'
    ***

    def _apply_rules_extra_args(self, args):
        pass

    def prefetch(self, *args, **kwargs):

        for publication in self.model_list:
            fields_to_prefetch = self.rules[0]['output_fields']
            if 'keyword' in fields_to_prefetch:
                [repr(Keyword.objects.get(id=keyword.id)) for keyword in publication.keywords.all()]
            if 'tag' in fields_to_prefetch:
                [repr(Tag.objects.get(id=tag.id)) for tag in publication.tags.all()]
            if 'comment' in fields_to_prefetch:
                for comment in publication.comment_set.all():
                    c = Comment.objects.get(id=comment.id)
                    repr(c)
                for cv in c.commentvote_set.all():
                    commentvote = CommentVote.objects.get(id=cv.id)
                    repr(commentvote.caster)
                    repr(User.objects.get(id=c.user.id))
            if 'author' in fields_to_prefetch:
                for author in publication.authors.all():
                    a = Author.objects.get(id=author.id)
                    if a.user:
                        repr(a.user)
            if 'owner' in fields_to_prefetch:
                for owner in publication.owners.all():
                    u = User.objects.get(id=owner.id)
                    repr(Author.objects.get(user=u))
                    prof = u.profile
                    try:
                        repr(ProfileField.objects.get(user_profile=prof))
                    except ProfileField.DoesNotExist:
                        pass
                    repr(prof.esteem)
                    for ra in prof.research_areas.all():
                        repr(ResearchArea.objects.get(id=ra.id))
            if 'rating' in fields_to_prefetch:
                for r in publication.rating_set.all():
                    repr(Rating.objects.get(id=r.id))
            if 'publicationvote' in fields_to_prefetch:
                for pv in publication.publicationvote_set.all():
                    pubvote = PublicationVote.objects.get(id=pv.id)
                    repr(pubvote.caster)
class RelatedPublicationsPolicy(BasePolicy):
    '''
    Prefetch related publications for a given publication
    Policy implementation available only for the activity "publication_details".
    '''
    def _apply_rules_extra_args(self, args):
        self.start, self.range = 0, 5
        if args.has_key('limit'):
            self.start, self.range = 0, int(args['limit'])
    def prefetch(self, *args, **kwargs):
        publication = self.model_list[0]
        fields_to_prefetch = self.rules[0]['output_fields']
        for field in fields_to_prefetch:
            l = '%s,%s' % (self.start, self.range)
            pubs = search.get_related_publications(publication, related_by='%s' % field, limit=l)
            [repr(Publication.objects.get(id=pub.id)) for pub in pubs]

class NextSearchPolicy(BasePolicy):
    '''
    Prefetch the most likely searches to be performed on the next action from the end user.
    In order to know what are the most likely search to be done we spot the input parameters given for the search ('rules' module), and we apply the prefetching on the output fields described in the rules that matched these input fields.
    '''
    def _apply_rules_extra_args(self, args):
        self.start, self.range = 0, 3
        if args.has_key('limit'):
            self.start, self.range = 0, int(args['limit'])
    def prefetch(self, *args, **kwargs):
        pub_list = self.model_list[self.start:self.range]
        limit = kwargs.pop('limit', '0,10')
        output_fields = []
        done = []
        for rule in self.rules:
            output_fields = output_fields + rule['output_fields']
        for pub in pub_list:
            if 'keyword' in output_fields:
for keyword in pub.keywords.all():
    if keyword.keyword not in done:
        repr(search.search_publications(keyword_terms=QueryDict('keyword=%s' % keyword.keyword), limit=limit))
        done.append(keyword.keyword)
if 'tag' in output_fields:
    for tag in pub.tags.all():
        if tag.name not in done:
            repr(search.search_publications(tag_terms=QueryDict('name=%s' % tag.name), limit=limit))
            done.append(tag.name)
if 'research_area' in output_fields:
    for author in pub.authors.all():
        if not author.user: continue
        for r in author.user.profile.research_areas.all():
            if r.title not in done:
                repr(search.search_publications(author_terms=QueryDict('research_area=%s' % r.title), limit=limit))
                done.append(r.title)
if 'affiliation' in output_fields:
    for author in pub.authors.all():
        if not author.affiliation: continue
        if author.affiliation not in done:
            repr(search.search_publications(author_terms=QueryDict('affiliation=%s' % author.affiliation), limit=limit))
            done.append(author.affiliation)

Listing 10.6: Rules

""
This module contains the rule description for the policy-oriented prefetcher engine.
So far 3 activities are defined with their different implemented policies:
- 'publication_details': ("PageRendering", "RelatedPublications")
- 'search': ("PageRendering", "NextSearch")

Description of the vocabulary for the rules:
Each policy must contain a list of dictionary.
Each dictionary must have the 3 keys 'input_fields', 'output_fields', 'extra_args'.
If no value is passed to any of this keys, an empty list or dictionary is the default datastructure to give.

Available output and input fields for a 'PageRendering' policy are:
'keyword', 'tag', 'comment', 'publicationvote', 'commentvote', 'author', 'owner', 'rating'

Available input fields for the 'NextSearch' policy are:
'QueryDict(order_by=[-](publication_field))', 'QueryDict(limit=offset, limit >)""
The extra args value authorized are:
- limit, which restrict on how many publication object the prefetching will be performed.

```
from django.http import QueryDict

# This dictionary describe the vocabulary that can be used for the 'rules' dictionary, though it is not used to check the validity of the rules you'll dress up for performance reasons.

publication_details = {
    'PageRendering': {
        'rules': [
            {'input_fields': [],
             'output_fields': ['keyword', 'tag', 'comment', 'publicationvote', 'commentvote', 'author', 'owner', 'rating', 'further_field', 'reference_material', 'peer_review'],
             'extra_args': {}},
            {'input_fields': [QueryDict('order_by=-average_rating'), QueryDict('limit=0,10')],
             'output_fields': ['keyword', 'tag', 'author'],
             'extra_args': {'limit': 5}},
            {'input_fields': [QueryDict('order_by=-average_rating')],
             'output_fields': ['keyword', 'tag', 'comment'],
             'extra_args': {'limit': 5}}
        ]},
    'RelatedPublications': {
        'rules': [
            {'input_fields': [],
             'output_fields': ['keyword', 'tag', 'author'],
             'extra_args': {'limit': 5}},
            {'input_fields': [QueryDict('order_by=-average_rating')],
             'output_fields': ['keyword', 'tag', 'comment'],
             'extra_args': {'limit': 5}}
        ]},
}

search = {
    'PageRendering': {
        'rules': [
            {'input_fields': [QueryDict('order_by=-average_rating'), QueryDict('limit=0,10')],
             'output_fields': ['keyword', 'tag', 'comment'],
             'extra_args': {'limit': 5}},
            {'input_fields': [QueryDict('order_by=-average_rating')],
             'output_fields': ['keyword', 'tag', 'comment'],
             'extra_args': {'limit': 5}}
        ]},
    'RelatedPublications': {
        'rules': [
            {'input_fields': [],
             'output_fields': ['keyword', 'tag', 'author'],
             'extra_args': {'limit': 5}},
            {'input_fields': [QueryDict('order_by=-average_rating')],
             'output_fields': ['keyword', 'tag', 'comment'],
             'extra_args': {'limit': 5}}
        ]},
}
```
CHAPTER 10. ANNEXES

'NextSearch':{
    'rules':[
        {
            'input_fields':[( 'order_by', [ '−date' ]), ( 'limit', [ '0,10' ])],
            'output_fields': ['keyword', 'tag'],
            'extra_args': {'limit': 5},
        },
        {
            'input_fields': [( 'publication', [ 'title' ]), ( 'limit', [ '0,10' ])],
            'output_fields': ['keyword', 'tag'],
            'extra_args': {'limit': 5},
        },
        {
            'input_fields': [( 'publication', [ 'booktitle' ]), ( 'limit', [ '0,10' ])],
            'output_fields': ['keyword', 'tag'],
            'extra_args': {'limit': 5},
        },
        {
            'input_fields': [( 'author', [ 'name' ]), ( 'limit', [ '0,10' ])],
            'output_fields': ['affiliation', 'research_area'],
            'extra_args': {'limit': 5},
        },
        {
            'input_fields': [( 'keyword', [ 'keyword' ]), ( 'limit', [ '0,10' ])],
            'output_fields': ['affiliation', 'research_area'],
            'extra_args': {'limit': 5},
        },
        {
            'input_fields': [( 'tag', [ 'name' ]), ( 'limit', [ '0,10' ])],
            'output_fields': ['affiliation', 'research_area'],
            'extra_args': {'limit': 5},
        },
        {
            'input_fields': [( 'tag', [ 'name' ]), ( 'keyword', [ 'keyword' ]), ( 'limit', [ '0,10' ])],
            'output_fields': ['affiliation', 'research_area'],
            'extra_args': {'limit': 5},
        }
    ]
}
CHAPTER 10. ANNEXES

Listing 10.7: URLs

from django.conf.urls.defaults import patterns, url

urlpatterns = patterns('prefetching.views',
    url(r'^flush_cache/$','execute_cache_command',
        {'command' : 'flush_all'},
        name='flush_cache'),
    url(r'^reset_stats/$','execute_cache_command',
        {'command' : 'stats', 'args': ['reset']},
        name='reset_stats'),
    url(r'^evaluate/(?P<resource_type>\[\w\]+)/repetition=(?P<repetition>\[\d\]+)/$','evaluate'),
    url(r'^evaluate/(?P<resource_type>\[\w\]+)/(?P<id>\[\d\]+)/repetition=(?P<repetition>\[\d\]+)/$','evaluate'),
    url(r'^evaluate/(?P<resource_type>\[\w\]+)/(?P<id>\[\d\]+)/$','evaluate'),
    url(r'^descriptors/policies/$','policies_descriptor'))

Listing 10.8: Views

# Create your views here.
from django.http import HttpResponse, HttpResponseRedirect
from django.core.urlresolvers import reverse
from django.contrib import messages
from core_web_service.views import RestView

def execute_cache_command(request, command, args=[]):
    '''
    Execute 'command' from the dictionary passed in the 'urls' module
    and send back an appropriate HTTP Response
    '''
    from prefetching import commands
    com = commands.MemcachedCommand()
    com.execute(command, *args)
    url = reverse('admin:prefetching_prefetchersetting_changelist')
    messages.success(request, "Cache/\%s/\%s executed with success" %
        (command))
    return HttpResponseRedirect(url)

    return HttpResponse('Cache flushed')

def evaluate(request, resource_type, id=None, repetition=1):
    from prefetching.evaluation import evaluators
    from prefetching.evaluation import tictoc, base_url, log
    suffix = resource_type
log_str = ""
if hasattr(evaluators, '%s' % resource_type):
    evaluator = getattr(evaluators, '%s' % resource_type)
else:
    evaluator = evaluators.generic_evaluator
evaluator = tictoc(evaluator)
evaluator = log(suffix)(evaluator)
log_str = log_str + evaluator(base_url=base_url, suffix=resource_type, id=id, repetition=repetition)
return HttpResponse("<pre>" + log_str + "</pre>"

def policies_descriptor(request):
    ""
    Return a list of the different prefetching activities and their policies
    ""
    from prefetching.rules import search, publication_details
    policies = [{
        'activity': 'search',
        'policies': search.keys()
    },
    {
        'activity': 'publication_details',
        'policies': publication_details.keys()
    }]
    return RestView.render_response(request, 'descriptors/policies', {'list': policies})

Listing 10.9: Commands

""
Implements an the admin web–interface to execute basic commands on memcached server.
""
import telnetlib
from django.dispatch import Signal
memcached_command = Signal(providing_args=["command"])

class MemcachedCommand(object):
    ""
    Execute memcached commands thanks to telnetlib
    ""
    def __init__(self, *args, **kwargs):
        ""
        Establishes connection to the telnet terminal
        ""
        from settings import CACHE_LOCATION
        HOST, PORT = CACHE_LOCATION.split(':')
        self.tn = telnetlib.Telnet(HOST, PORT)
    def write(self, command, *args):
        ""
        Write command into the terminal object
str_arg = ""
for arg in args:
    str_arg = str_arg + "," + arg
self.tn.write(command + str_arg + "\n")

def execute_telnet_buffer(self):
    """
    Append 'quit' keyword and execute commands buffered.
    """
    self.tn.write('quit
')
    self.tn.read_all()

def execute(self, command, *args):
    """
    Wrapper to write and execute command in telnet object.
    """
    self.write(command, *args)
    self.execute_telnet_buffer()
    memcached_command.send(sender=self, command=command)

Listing 10.10: Admin Interface

Created on 6 Jun 2012
@author: victorinox

from django.contrib import admin
from prefetching.models import PrefetcherSetting
from django.contrib import messages
class PrefetcherSettingAdmin(admin.ModelAdmin):
    actions = ['disable_prefetchers', 'enable_prefetchers']

def disable_prefetchers(self, request, queryset):
    queryset.update(enabled=False)
    messages.success(request, "Prefetchers disabled with success")
    disable_prefetchers.short_description = "Disable selected prefetchers"

def enable_prefetchers(self, request, queryset):
    queryset.update(enabled=True)
    messages.success(request, "Prefetchers enabled with success")
    enable_prefetchers.short_description = "Enable selected prefetchers"

admin.site.register(PrefetcherSetting, PrefetcherSettingAdmin)

Listing 10.11: Service call switch

def retrieve_metadata(self, publication):
    if eval(self.ckp_params["related_papers"]):
        return self._harvest_related_papers(publication)
    else:
        return self._harvest_further_metadata(publication)
def _harvest_related_papers(self, publication):
    from oai.refiner import XMLRefiner
    from django.core.cache import cache
    import hashlib
    params_val_flat = self.flatten(self.ckp_params.values())
    result_cache_key = ''
    for p in params_val_flat:
        result_cache_key += str(p)
    result_cache_key = hashlib.md5('%s%s' % (publication.id, result_cache_key)).hexdigest()
    results = cache.get(result_cache_key, {})
    if not results:
        if self.ckp_params['roar_set_ids'] == 'all':
            roar_set_ids = [rrs.id for rrs in ROARRepositorySet.objects.all()]
        else:
            roar_set_ids = self.ckp_params['roar_set_ids']
        for set_id in roar_set_ids:
            roar_set = ROARRepositorySet.objects.get(id=set_id)
            repos = OAIRepository.objects.filter(sets__id__in=[roar_set.id])
            for repo in repos:
                if repo.valid:
                    file_obj = self._retrieve_file(repo)
                    if file_obj:
                        refiner = XMLRefiner(file_obj, repo, related_papers=self.ckp_params['related_papers'])
                        related_by_fields = self.ckp_params['related_by'].split(',')
                        res = refiner.get_results(publication, *related_by_fields)
                        if res:
                            results = self._update_results(res, results)
                            file_obj.close()
        if results:
            cache.set(result_cache_key, results, 3600 * 24)
    return results

def _harvest_further_metadata(self, publication):
    from settings import MENDELEY_CONSUMER_KEY, MENDELEY_CONSUMER_SECRET
    options = None
    mc = mendeley_client.MendeleyClient(MENDELEY_CONSUMER_KEY, MENDELEY_CONSUMER_SECRET, options)
    try:
        mc.load_keys()
    except IOError:
        mc.get_required_keys()
        mc.save_keys()
    ret = None
    if publication.doi:
        ret = mc.details(publication.doi, type='doi')
    if not ret or isinstance(ret, dict) and ret.has_key('error') or hasattr(ret, 'status'):
        if publication.isbn:
            isbn = re.sub('^\\d', '', publication.isbn)
            ret = mc.details(isbn, type='isbn')
        if not ret or isinstance(ret, dict) and ret.has_key('error') or hasattr(ret, 'status'):
            isbn = re.sub('\d', '', isbn)
            ret = mc.details(isbn, type='isbn')
if isinstance(ret, dict):
    ret = dict2xml(ret)
else:
    ret = dict2xml({'error': 'Document/uni2423not/uni2423found'})

return ret

Listing 10.12: Mendeley Client API Implementation

Created on 26 Jul 2012

@author: victorinox

from oauth2 import Token

Mendeley Open API Example Client

Copyright (c) 2010, Mendeley Ltd. <copyright@mendeley.com>

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

For details of the Mendeley Open API see http://dev.mendeley.com/

Example usage:

>>> from pprint import pprint
>>> from mendeley_client import MendeleyClient
>>> mendeley = MendeleyClient('<<consumer_key>>', '<<secret_key>>')
>>> try:
...     mendeley.load_keys()
... except IOError:
...     mendeley.get_required_keys()
...     mendeley.save_keys()
... results = mendeley.search('science')
... pprint(results['documents'][0])
{u'authors': None,
 u'doi': None,
 u'id': u'8c18bd50-6f07-11df-b8f0-001e688e2dcb',
 u'mendeley_url': u'http://localhost/research//',
 u'publication_outlet': None,
 u'title': None,
 u'year': None}
>>> documents = mendeley.library()
>>> pprint(documents)
{u'current_page': 0,
import oauth2 as oauth
import pickle
import httplib
import json
import urllib
import logging, os
from settings import LOGFILE, STATIC_ROOT

logger = logging.getLogger('myproject.custom')
if os.path.exists(LOGFILE):
    os.remove(LOGFILE)

hdlr = logging.FileHandler(LOGFILE)
formatter = logging.Formatter('%(asctime)s/uni2423%(levelname)s/uni2423%(message)s')
hdlr.setFormatter(formatter)
logger.addHandler(hdlr)
logger.setLevel(logging.INFO)

class OAuthClient(object):
    """ General purpose OAuth client """
    def __init__(self, consumer_key, consumer_secret, options=None):
        if options is None: options = {}
        self.host = options.get('host', 'api.mendeley.com')
        self.port = options.get('port', 80)
        self.access_token_url = options.get('access_token_url', '/oauth/access_token/ ')
        self.request_token_url = options.get('request_token_url', '/oauth/request_token/ ')
        self.authorize_url = options.get('authorize_url', '/oauth/authorize/ ')

        if self.port == 80: self.authority = self.host
        else: self.authority = '%s:%d' % (self.host, self.port)

        self.consumer = oauth.Consumer(consumer_key, consumer_secret)

    def get(self, path, token=None):
        url = "http://%s%s" % (self.host, path)
        request = oauth.Request.from_consumer_and_token(
            self.consumer, 
            token, 
            http_method='GET', 
            http_url=url,
            parameters={})
        return request.sign(self.consumer, token, 
                            http_method='GET', 
                            http_url=url, 
                            parameters={})
```python
def post(self, path, post_params, token=None):
    url = "http://%s%s" % (self.host, path)
    request = oauth.Request.from_consumer_and_token(
        self.consumer,
        token,
        http_method='POST',
        http_url=url,
    parameters=post_params)
    return self._send_request(request, token)

def delete(self, path, token=None):
    url = "http://%s%s" % (self.host, path)
    request = oauth.Request.from_consumer_and_token(
        self.consumer,
        token,
        http_method='DELETE',
        http_url=url,
    )
    return self._send_request(request, token)

def put(self, path, token=None, body=None, body_hash=None, headers=None):
    url = "http://%s%s" % (self.host, path)
    request = oauth.Request.from_consumer_and_token(
        self.consumer,
        token,
        http_method='PUT',
        http_url=url,
    parameters={'oauth_body_hash': body_hash}
    )
    return self._send_request(request, token, body, headers)

def request_token(self):
    response = self.get(self.request_token_url).read()
    token = oauth.Token.from_string(response)
    return token

def authorize(self, token, callback_url="oob"):
    url = 'http://%s%s' % (self.authority, self.authorize_url)
    request = oauth.Request.from_token_and_callback(token=token, callback_url=callback_url, http_url=url)
    return request.to_url()

def access_token(self, request_token):
    response = self.get(self.access_token_url, request_token).read()
    return oauth.Token.from_string(response)

def _send_request(self, request, token=None, body=None, extra_headers=None):
    request.sign_request(oauth.SignatureMethod_HMAC_SHA1(), self.consumer, token)
    conn = self._get_conn()
    if request.method == 'POST':
conn.request('POST', request.url, body=request.to_postdata(),
headers={"Content-type": "application/x-www-form-urlencoded"})

elif request.method == 'PUT':
    final_headers = request.to_header()
    if extra_headers is not None:
        final_headers.update(extra_headers)
    conn.request('PUT', request.url, body, headers=final_headers)

elif request.method == 'DELETE':
    conn.request('DELETE', request.url, headers=request.to_header())
else:
    conn.request('GET', request.url, headers=request.to_header())
return conn.getresponse()
filename = filename[1].strip('"
')
except:
    pass

if mime == 'application/json':
    # HTTP Status 204 means 'No Content' which json.loads cannot deal
    if status == 204:
        data = ''
    else:
        data = json.loads(body)
    return data
elif attached == 'attachment':
    return {'filename': filename, 'data': body}
else:
    return response

class MendeleyClient(object):
    # API method definitions. Used to create MendeleyRemoteMethod instances
    methods = {
        'details': {
            'required': ['id'],
            'optional': ['type'],
            'url': '/oapi/documents/details/%(id)s/','
        },
        'categories': {
            'url': '/oapi/documents/categories/','
        },
        'subcategories': {
            'url': '/oapi/documents/subcategories/%(id)s/','
            'required': ['id'],
        },
        'search': {
            'url': '/oapi/documents/search/%(query)s/','
            'required': ['query'],
            'optional': ['page', 'items'],
        },
        'tagged': {
            'url': '/oapi/documents/tagged/%(tag)s/','
            'required': ['tag'],
            'optional': ['cat', 'subcat', 'page', 'items'],
        },
        'related': {
            'url': '/oapi/documents/related/%(id)s/','
            'required': ['id'],
            'optional': ['page', 'items'],
        },
        'authored': {
            'url': '/oapi/documents/authored/%(author)s/','
            'required': ['author'],
            'optional': ['page', 'items'],
        },
        'public_groups': {
            'url': '/oapi/documents/groups/','
            'optional': ['page', 'items', 'cat']
        }
    }
CHAPTER 10. ANNEXES

`public_group_details`: {
    'url': '/oapi/documents/groups/%(id)s/',
    'required': ['id'],
},

`public_group_docs`: {
    'url': '/oapi/documents/groups/%(id)s/docs/',
    'required': ['id'],
    'optional': ['details', 'page', 'items'],
},

`public_group_people`: {
    'url': '/oapi/documents/groups/%(id)s/people/',
    'required': ['id'],
},

`author_stats`: {
    'url': '/oapi/stats/authors/',
    'optional': ['discipline', 'upandcoming'],
},

`paper_stats`: {
    'url': '/oapi/stats/papers/',
    'optional': ['discipline', 'upandcoming'],
},

`publication_stats`: {
    'url': '/oapi/stats/publications/',
    'optional': ['discipline', 'upandcoming'],
},

`tag_stats`: {
    'url': '/oapi/stats/tags/%(discipline)s/',
    'required': ['discipline'],
    'optional': ['upandcoming'],
},

```
# User Specific Resources

library_author_stats`: {
    'url': '/oapi/library/authors/',
    'access_token_required': True,
},

`library_tag_stats`: {
    'url': '/oapi/library/tags/',
    'access_token_required': True,
},

`library_publication_stats`: {
    'url': '/oapi/library/publications/',
    'access_token_required': True,
},

`library`: {
    'url': '/oapi/library/',
    'optional': ['page', 'items'],
    'access_token_required': True,
},

`create_document`: {
    'url': '/oapi/library/documents/',
    # HACK: 'document' is required, but by making it optional here it
    # will get POSTed
    # Unfortunately that means it needs to be a named param when
    # calling this method
    'optional': ['document'],
}
'access_token_required': True,
'method': 'post',
},

'upload_pdf': {
    'url': '/oapi/library/documents/%(id)s/',
    'required': ['id'],
    'optional': ['data', 'file_name', 'oauth_body_hash', 'sha1_hash'],
    'access_token_required': True,
    'method': 'post'
},

'download_file': {
    'url': '/oapi/library/documents/%(id)s/file/%(hash)s/',
    'required': ['id', 'hash'],
    'access_token_required': True,
    'method': 'get'
},

'download_file_group': {
    'url': '/oapi/library/documents/%(id)s/file/%(hash)s/%(group)s/',
    'required': ['id', 'hash', 'group'],
    'access_token_required': True,
    'method': 'get'
},

'document_details': {
    'url': '/oapi/library/documents/%(id)s/',
    'required': ['id'],
    'access_token_required': True,
},

'documents_authored': {
    'url': '/oapi/library/documents/authored/',
    'access_token_required': True,
},

'delete_library_document': {
    'url': '/oapi/library/documents/%(id)s/',
    'required': ['id'],
    'access_token_required': True,
    'method': 'delete'
},

'contacts': {
    'url': '/oapi/profiles/contacts/',
    'access_token_required': True,
    'method': 'get'
},

'contacts_of_contact': {
    'url': '/oapi/profiles/contacts/%(id)s/',
    'required': ['id'],
    'access_token_required': True,
    'method': 'get'
},

'add_contact': {
    'url': '/oapi/profiles/contacts/%(id)s/',
    'required': ['id'],
    'access_token_required': True,
    'method': 'post'
},

# Folders methods #
CHAPTER 10. ANNEXES

'folders': {
    'url': '/oapi/library/folders/',
    'access_token_required': True,
},

'folder_documents': {
    'url': '/oapi/library/folders/%(id)s/',
    'required': ['id'],
    'optional': ['page', 'items'],
    'access_token_required': True,
},

'create_folder': {
    'url': '/oapi/library/folders/',
    # HACK: 'folder' is required, but by making it optional here it’ll get POSTed
    # Unfortunately that means it needs to be a named param when calling this method
    'optional': ['folder'],
    'access_token_required': True,
    'method': 'post',
},

'delete_folder': {
    'url': '/oapi/library/folders/%(id)s/',
    'required': ['id'],
    'access_token_required': True,
    'method': 'delete',
},

'add_document_to_folder': {
    'url': '/oapi/library/folders/%(folder_id)s/%(document_id)s/',
    'required': ['folder_id', 'document_id'],
    'access_token_required': True,
    'method': 'post',
},

'delete_document_from_folder': {
    'url': '/oapi/library/folders/%(folder_id)s/%(document_id)s/',
    'required': ['folder_id', 'document_id'],
    'access_token_required': True,
    'method': 'delete',
},

# Groups methods #

'groups': {
    'url': '/oapi/library/groups/',
    'access_token_required': True,
},

'group_documents': {
    'url': '/oapi/library/groups/%(id)s/',
    'required': ['id'],
    'optional': ['page', 'items'],
    'access_token_required': True,
},

'group_doc_details': {
    'url': '/oapi/library/groups/%(group_id)s/%(doc_id)s/',
    'required': ['group_id', 'doc_id'],
    'access_token_required': True,
},

'group_people': {
    'url': '/oapi/library/groups/%(id)s/people/',
}
'required': ['id'],
'access_token_required': True,
}
'
create_group': {
'url': '/oapi/library/groups/',
'options': ['group'],
'access_token_required': True,
'method': 'post',
}
'
delete_group': {
'url': '/oapi/library/groups/%(id)s/',
'required': ['id'],
'access_token_required': True,
'method': 'delete',
}
'
leave_group': {
'url': '/oapi/library/groups/%(id)s/leave/',
'required': ['id'],
'access_token_required': True,
'method': 'delete',
}
'
unfollow_group': {
'url': '/oapi/library/groups/%(id)s/unfollow/',
'required': ['id'],
'access_token_required': True,
'method': 'delete',
}
'
delete_group_document': {
'url': '/oapi/library/groups/%(group_id)s/%(document_id)s/',
'required': ['group_id', 'document_id'],
'access_token_required': True,
'method': 'delete',
}

# Group Folders methods #
'group_folders': {
'url': '/oapi/library/groups/%(group_id)s/folders/',
'required': ['group_id'],
'access_token_required': True,
}
'
group_folder_documents': {
'url': '/oapi/library/groups/%(group_id)s/folders/%(id)s/',
'required': ['group_id', 'id'],
'options': ['page', 'items'],
'access_token_required': True,
'
create_group_folder': {
'url': '/oapi/library/groups/%(group_id)s/folders/',
'required': ['group_id'],
# HACK: 'folder' is required, but by making it optional here it'll get POSTed
# Unfortunately that means it needs to be a named param when calling this method
'options': ['folder'],
'access_token_required': True,
'method': 'post',


'delete_group_folder': {
    'url': '/oapi/library/groups/%(group_id)s/folders/%(id)s/',
    'required': ['group_id', 'id'],
    'access_token_required': True,
    'method': 'delete',
},
'add_document_to_group_folder': {
    'url': '/oapi/library/groups/%(group_id)s/folders/%(folder_id)s/
    /%(document_id)s/',
    'required': ['group_id', 'folder_id', 'document_id'],
    'access_token_required': True,
    'method': 'post',
},
'delete_document_from_group_folder': {
    'url': '/oapi/library/groups/%(group_id)s/folders/%(folder_id)s/
    /%(document_id)s/',
    'required': ['group_id', 'folder_id', 'document_id'],
    'access_token_required': True,
    'method': 'delete',
},

# DEPRECATED METHODS
# Deprecated
'collections': {
    'url': '/oapi/library/collections/',
    'access_token_required': True,
},
# Deprecated
'sharedcollections': {
    'url': '/oapi/library/sharedcollections/',
    'access_token_required': True,
},
# Deprecated
'collection_documents': {
    'url': '/oapi/library/collections/%(id)s/',
    'required': ['id'],
    'optional': ['page', 'items'],
    'access_token_required': True,
},
# Deprecated
'sharedcollection_documents': {
    'url': '/oapi/library/sharedcollections/%(id)s/',
    'required': ['id'],
    'optional': ['page', 'items'],
    'access_token_required': True,
},
# Deprecated
'sharedcollection_members': {
    'url': '/oapi/library/sharedcollections/%(id)s/members/',
    'required': ['id'],
    'access_token_required': True,
},
# Deprecated
'delete_collection': {
    'url': '/oapi/library/collections/%(id)s/',
    'required': ['id'],
    'access_token_required': True,
}
'method': 'delete',
},
# Deprecated
'delete_sharedcollection': {
    'url': '/oapi/library/sharedcollections/%(id)s/',
    'required': ['id'],
    'access_token_required': True,
    'method': 'delete',
},
# Deprecated
'create_collection': {
    'url': '/oapi/library/collections/',
    # HACK: 'collection' is required, but by making it optional here
    # it'll get POSTed
    # Unfortunately that means it needs to be a named param when
    # calling this method
    'optional': ['collection'],
    'access_token_required': True,
    'method': 'post',
},
# Deprecated
'create_sharedcollection': {
    'url': '/oapi/library/sharedcollections/',
    'optional': ['sharedcollection'],
    'access_token_required': True,
    'method': 'post',
},
# Deprecated
'add_document_to_collection': {
    'url': '/oapi/library/collections/add/%(collection_id)s/%(document_id)s/',
    'required': ['collection_id', 'document_id'],
    'access_token_required': True,
    'method': 'post',
},
# Deprecated
'remove_document_from_collection': {
    'url': '/oapi/library/collections/%(collection_id)s/%(document_id)s/',
    'required': ['collection_id', 'document_id'],
    'access_token_required': True,
    'method': 'delete',
},
# Deprecated
'delete_sharedcollection_document': {
    'url': '/oapi/library/sharedcollections/%(collection_id)s/%(document_id)s/',
    'required': ['collection_id', 'document_id'],
    'access_token_required': True,
    'method': 'delete',
}

def __init__(self, consumer_key, consumer_secret, options=None):
    self.mendeley = OAuthClient(consumer_key, consumer_secret, options)
    # Create methods for all of the API calls
for method, details in self.methods.items():
    setattr(self, method, MendeleyRemoteMethod(details, self.api_request))

def api_request(self, url, access_token_required=False, method='get',
            params=None):
    if params == None: params = {}
    if access_token_required: access_token = self.access_token
    else: access_token = None
    if method == 'get':
        if len(params) > 0:
            url += '?%s' % urllib.urlencode(params)
        response = self.mendeley.get(url, access_token)
    elif method == 'delete':
        response = self.mendeley.delete(url, access_token)
    elif method == 'put':
        headers = {'Content-disposition': 'attachment;filename=%s' % params.get('file_name')}
        response = self.mendeley.put(url, access_token, params.get('data'),
            params.get('oauth_body_hash'), headers)
    else:
        response = self.mendeley.post(url, params, access_token)
    return response

def get_required_keys(self):
    self.request_token = self.mendeley.request_token()
    # self.request_token = Token.from_string('oauth_token_secret=
c23816e7ed29b884dbb9944452b6637&oauth_token=900217977001
b5330a16dc72ffe23cd050126955')
    # auth_url = 'http://api.mendeley.com/oauth/authorize/?oauth_token=908
fb87d1a05a5391bde4d198a8305d050126849&oauth_callback=oob'
    auth_url = self.mendeley.authorize(self.request_token)
    # raise Exception(auth_url + ' ' + self.request_token.to_string())
    print 'Go to the following url to authenticate:
        %s % (auth_url,)
    verifier = raw_input('Enter verification code:
        %s' % (auth_url,))
    # verifier = '06c97c1ce5'
    self.request_token.set_verifier(verifier)
    self.access_token = self.mendeley.access_token(self.request_token)

def load_keys(self):
    data = pickle.load(open('%s/mendeley_api_keys.pkl' % STATIC_ROOT, 'r'))
    self.request_token = data['request_token']
    self.access_token = data['access_token']

def save_keys(self):
    data = {'request_token': self.request_token, 'access_token': self.access_token}
    pickle.dump(data, open('%s/mendeley_api_keys.pkl' % STATIC_ROOT, 'w'))

Listing 10.13: OAI Data Provider - settings
CHAPTER 10. ANNEXES

REPO_NAME = 'CKP'
ADMIN_EMAIL = ['godayerv@gmail.com']

METADATA_FORMATS = {
    'oai_dc': 'OAIDC',
}

DELAY = None
SETS_ALLOWED = []
SETS_DISALLOWED = []
FILTER_SETS = []
UNIQUE_IDENTIFIER_FORMAT = "oai:macs-ckp:%s"
SETS_DELETED = []

BATCH_SIZE = 20

def get_internal_identifier(identifier):
    
        :return: CKP Publication id from the OAI-formatted identifier
        
        return identifier.rsplit(':')[1]

def get_oai_identifier(publication_id):
    
        :return: OAI 2.0 identifier from a Publication model instance
        
        return UNIQUE_IDENTIFIER_FORMAT % publication_id

def get_internal_set_id(set):
    
        No set support.
        :raise: NotImplementedError
        
        raise NotImplementedError()

def get_writer(prefix):
    
        :return: metadata writer given a prefix
        :raise: CannotDisseminateFormatError if no writer exists for the given
        prefix
        
        from oai.data_provider import metadata
        from oai.data_provider.settings import get_writer, get_internal_identifier
        from oai.data_provider import settings
        from oai_pmh import common, error
        from django.core.urlresolvers import reverse
        from oai_pmh.error import NoSetHierarchyError
        
        mf = METADATA_FORMATS.get(prefix, None)
        if not mf or not hasattr(metadata, mf):
            raise CannotDisseminateFormatError
        return getattr(metadata, METADATA_FORMATS.get(prefix))(prefix)

Listing 10.14: OAI Data Provider - server

from datetime import datetime
from oai.data_provider import get_writer, get_internal_identifier
from oai.data_provider.settings import get_writer, get_internal_identifier, SETTINGS
from oai_pmh import common, error
from django.core.urlresolvers import reverse
from oai_pmh.error import NoSetHierarchyError
from core_web_service.models import Publication
from django.db.models import Q

class OAIServer(common.ResumptionOAIPMH):
    """
    OAI Django specific server implementation
    Subclasses the ResumptionOAIPMH.
    Implement the resumption token feature.
    """

def identify(self):
    """
    :return: information about the CKP repository.
    """
    result = common.Identify(
        repositoryName=settings.REPO_NAME,
        baseURL=reverse('oai_data_provider'),
        protocolVersion='2.0',
        adminEmails=settings.ADMIN_EMAIL,
        earliestDatestamp=datetime(2012, 7, 1, 10, 00),
        deletedRecord='transient',
        granularity='%Y-%m-%dT%H:%m:%sZ',
        compression=['identity'],
        toolkit_description=False)

    return result

def listMetadataFormats(self, identifier=None):
    """
    :type identifier: string
    :param identifier: criteria to filter metadata format listing to a
                      specific record
    :return: metadata formats available from CKP repository
    """
    result = []
    for prefix in settings.METADATA_FORMATS.keys():
        writer = get_writer(prefix)
        ns = writer.get_namespace()
        schema = writer.get_schema_location()
        result.append((prefix, schema, ns))

    return result

def listSets(self, cursor=0, batch_size=20):
    """
    Not implemented.
    :type cursor: integer
    :param cursor: offset from where to filter returned sets
    :type batch_size: integer
    :param batch_size: size of the batch of sets returned.
    :return: OAI sets
    """
    raise NoSetHierarchyError()

def listRecords(self, metadataPrefix, set=None, from_=None, until=None,
               cursor=0, batch_size=10):
List CKP records according selective harvesting parameters provided.

:param metadataPrefix: criteria to filter records returned
:type metadataPrefix: string

:param set: OAI set specification
:type set: string

:param from_: criteria to filter records having a modification date up to 'from_' (inclusive)
:type from_: datetime (UTC)

:param until: criteria to filter records having a modification date before 'until' (inclusive)
:type until: datetime (UTC)

:param cursor: offset from where to filter returned sets
:type cursor: integer

:param batch_size: size of the batch of sets returned.
:type batch_size: integer

:return: records from CKP repository

```python
self._checkMetadataPrefix(metadataPrefix)
for record in self._listQuery(set, from_, until, cursor, batch_size):
    header, metadata = self._createHeaderAndMetadata(record)
    yield header, metadata, None
```

def listIdentifiers(self, metadataPrefix, set=None, from_=None, until=None, cursor=0, batch_size=10):

```python
self._checkMetadataPrefix(metadataPrefix)
for record in self._listQuery(set, from_, until, cursor, batch_size):
    yield self._createHeader(record)
```

def getRecord(self, metadataPrefix, identifier):

```python
Returns an individual metadata record from a repository
:type metadataPrefix: string
```
def _checkMetadataPrefix(self, metadataPrefix):
    if metadataPrefix not in settings.METADATA_FORMATS.keys():
        raise error.CannotDisseminateFormatError

def _createHeader(self, record):
    """Create metadata header from a record""
    oai_id = record['record']['id']
    datestamp = record['record']['when_modified']
    sets = [self.config.get_setspec_id(s) for s in record['sets']]  
    deleted = record['record']['deleted']
    for deleted_set in settings.SETS_DELETED:
        if deleted_set in record['sets']:
            deleted = True
            break
    return common.Header(oai_id, datestamp, sets, deleted)

def _createHeaderAndMetadata(self, record):
    header = self._createHeader(record)
    metadata = common.Metadata(record['metadata'])
    metadata.record = record
    return header, metadata

def _listQuery(self, set=None, from_=None, until=None, cursor=0, batch_size=10, identifier=None):
    """List records according selective input arguments.
    :param set: OAI set specification
    :param from_: criteria to filter records having a modification date up to 'from_' (inclusive)
    :param until: criteria to filter records having a modification date before 'until' (inclusive)
    :param cursor: offset from where to filter returned sets
    :param batch_size: size of the batch of sets returned.
    :param identifier: criteria to filter metadata format listing to a specific record
    :return: single record (metadata + header)
    """
    self._checkMetadataPrefix(metadataPrefix)
    header = None
    metadata = None
    for record in self._listQuery(identifier=identifier):
        header, metadata = self._createHeaderAndMetadata(record)
        if header is None:
            raise error.IdDoesNotExistError(identifier)
    return header, metadata, None
param identifier: criteria to filter metadata format listing to a specific record

return: a list of records.

```python
if identifier:
    identifier = settings.get_internal_identifier(identifier)
if set:
    set = settings.get_internal_set_id(set)

now = datetime.now()
if until != None and until > now:
    # until should never be in the future
    until = now

if set is None:
    filtersets = []
else:
    filtersets = [set]

sets = settings.SETS_ALLOWED
filtersets += settings.FILTER_SETS
notsets = settings.SETS_DISALLOWED

return self._db_oai_query(cursor=cursor, batch_size=batch_size,
    sets=sets, not_sets=notsets,
    filter_sets=filtersets,
    from_=from_, until=until,
    identifier=identifier)
```

def _db_oai_query(self, cursor=0, batch_size=20, sets=[], not_sets=[],
    filter_sets=[], from_=None, until=None,
    identifier=None):

    Perform the request to the database to retrieve Publication instance that match the selective input arguments.
    :type sets: list
    :param sets: OAI set specifications
    :type not_sets: list
    :param not_sets: set of OAI set specifications to exclude from the harvesting
    :type filter_sets: list
    :param filter_sets: set of OAI set specifications for filtering
    :type from_: datetime (UTC)
    :param from_: criteria to filter records having a modification date up to "from_" (inclusive)
    :type until: datetime (UTC)
    :param until: criteria to filter records having a modification date before "until" (inclusive)
    :type cursor: integer
    :param cursor: cursor from where to filter returned sets
    :type batch_size: integer
    :param batch_size: size of the batch of sets returned.
    :type identifier: string
:param identifier: criteria to filter metadata format listing to a specific record
:return: a list of records.

```python
if batch_size < 0:
    batch_size = 0

# make sure until date is set, and not in future
if until == None or until > datetime.now():
    until = datetime.now()
else:
    until = datetime.strptime(until, "%Y-%m-%dT%H:%M:%SZ")

q = Q(last_modification_date__lte=until)
records = Publication.objects.filter(q)

if from_:
    from_ = datetime.strptime(from_, "%Y-%m-%dT%H:%M:%SZ")
    records = records.filter(last_modification_date__gte=from_)

if identifier is not None:
    records = records.filter(id=get_internal_identifier(identifier))

# No Set support so no refining with set
records = records.distinct()[cursor:cursor + batch_size]
for publication in records:
    record = {
        'id': settings.get_oai_identifier(publication.id),
        'when_modified': publication.last_modification_date or datetime.utcnow(),
        'deleted': False,
        'sets': tuple(),
    }
    yield {
        'record': record,
        'sets': record['sets'],
        'metadata': publication,
        'assets':{}
    }
```

def OAIserverFactory():
    
    Create a new OAI batching OAI Server
    
    from oaipmh.metadata import MetadataRegistry
    from oaipmh.server import BatchingServer

    metadata_registry = MetadataRegistry()
    for prefix in settings.METADATA_FORMATS.keys():
        metadata_registry.registerWriter(prefix, settings.get_writer(prefix))

    return BatchingServer(OAIserver(),
                           metadata_registry=metadata_registry,
                           metadata_registry=metadata_registry,
resumption_batch_size=settings.BATCH_SIZE

Listing 10.15: OAI Data Provider - metadata

from lxml.builder import ElementMaker
from oai.data_provider import settings
from core_web_service.models import FurtherField, ReferenceMaterial
from django.db.models.query import ValuesListQuerySet
from django.core.urlresolvers import reverse
from core_web_service.views import service_url

XSI_NS = 'http://www.w3.org/2001/XMLSchema-instance'

class MetaDataFormat(object):
    
    """
    Base class for metadata format description
    """
    :ivar prefix: prefix of the metadata format

    def get_namespace(self):
        """
        Return the namespace corresponding to the prefix.
        """
        return self.ns[self.prefix]

    def get_schema_location(self):
        """
        Return schema location url according the prefix
        """
        return selfschemas[self.prefix]

class MetadataFormatMapper(object):
    
    """
    Base class to contain the logic for mapping an Django Publication model to
    a specific metadata format.
    """
    :ivar _publication: Publication object to be mapped to an specific
    metadata format.
    
    set_publication_object(self, publication_obj, *args, **kwargs):
        """
        Setter for '_publication' attribute.
        
        :type publication_obj: core_web_service.models.Publication
        :param publication_obj: publication model instance
        """
        self._publication = publication_obj

    def get_url(self):
        """
        :return: the URL for the given Publication model instance.
        """
return reverse('publication_detail', args=[self._publication.id])

def get(self, field, default=[]):
    """Call dynamically the appropriate method 'get_<field>' to return the value mapped from the Publication model.
    :type field: string
    :param field: Metadata-specific field name.
    :type default: list
    :param default: default value for the metadata field in case of no equivalent for the Publication model.
    :return: List of values or empty list.
    """
    if field == 'url':
        return '%s/%s' % (service_url, self.get_url()) or default
    if hasattr(self, 'get_%s' % field):
        ret = getattr(self, 'get_%s' % field)() or default
        if not ret:
            return default
        if isinstance(ret, list):
            return ret
        else:
            if isinstance(ret, ValuesListQuerySet):
                return list(ret)
            else:
                return [ret]
    return default

class OAIDCMapper(MetadataFormatMapper):
    """Concrete class implementing the mapping from the OAI_DC (OAI Dublin Core) format to the Publication model."""
    def get_title(self):
        return self._publication.title
    def get_creator(self):
        return self._publication.authors.values_list('name', flat=True)
    def get_subject(self):
        return
    def get_description(self):
        return self._publication.abstract
    def get_publisher(self):
        return self._publication.publisher
    def get_contributor(self):
        return self._publication.editor
    def get_type(self):
        return 'Text'
    def get_format(self):
        return 'application/xml'
    def get_identifier(self):
        return settings.get_oai_identifier(self._publication.id)
def get_source(self):
    r = []
    if self._publication.doi:
        r.append('doi:%s' % self._publication.doi)
    if self._publication.full_text:
        r.append('url:%s' % self._publication.full_text)
    if self._publication.isbn:
        r.append('isbn:%s' % self._publication.isbn)
    else:
        sources = ['url', 'bdsk-url-1', 'bdsk-url-2', 'biburl', 'issn', 'pmid', 'arxiv', 'scopus', 'ssm']
        f = None
        for source in sources:
            try:
                f = FurtherField.objects.get(publication=self._publication, key=source)
                r.append('%s:%s' % (f.key, f.value) if f else None)
                break
            except FurtherField.DoesNotExist:
                pass
    return r

def get_language(self):
    return

def get_date(self):
    ret = self._publication.last_modification_date \\
        or self._publication.submission_date \\
        or self._publication.date or None
    return ret.strftime('%Y-%m-%dT%H:%m:%sZ') if ret else None

def get_relation(self):
    references = ReferenceMaterial.objects.filter(publication=self._publication)
    f = '%s/uni2423-%s/uni2423-%s'
    return [f % (ref.name, ref.url, ref.notes) for ref in references]

def get_coverage(self):
    r = []
    if self._publication.date:
        r.append(self._publication.date.strftime('%Y-%m-%dT%H:%m:%sZ'))
    if self._publication.address:
        r.append(self._publication.address)
    return r

def get_rights(self):
    return

class OAIDC(MetaDataFormat):
    """
The standard OAI Dublin Core metadata format.  
Every OAI feed should at least provide this format.  
\: attention: This class implementation is highly inspired from the 
MOAI implementation made by the Infrae  
Name: MOAI  
Version: 1.1.2"""
def __init__(self, prefix):
    self.prefix = prefix
    self.metadata_mapper = OAIDCMapper()
    self.ns = {'oai_dc': 'http://www.openarchives.org/OAI/2.0/oai_dc/',
               'dc': 'http://purl.org/dc/elements/1.1/'}
    self.schemas = {'oai_dc': 'http://www.openarchives.org/OAI/2.0/oai_dc.xsd'}

def __call__(self, element, metadata):
    data = metadata.record
    publication_obj = data['metadata']
    self.metadata_mapper.set_publication_object(publication_obj)

    OAI_DC = ElementMaker(namespace=self.ns['oai_dc'], nsmap=self.ns)
    DC = ElementMaker(namespace=self.ns['dc'])

    oai_dc = OAI_DC.dc()
    oai_dc.attrib['{%s}schemaLocation' % XSI_NS] = '%s%s' % (self.ns['oai_dc'],
                                                            self.schemas['oai_dc'])

    for field in ['title', 'creator', 'subject', 'description',
                  'publisher', 'contributor', 'type', 'format',
                  'identifier', 'source', 'language', 'date',
                  'relation', 'coverage', 'rights']:
        el = getattr(DC, field)
        for value in self.metadata_mapper.get(field, []):
            if field == 'identifier':
                value = self.metadata_mapper.get('url', None)
            if value:
                oai_dc.append(el(value))

    element.append(oai_dc)
Bibliography


