

*Master Thesis*

---

TARGETED COMMENTING AND ANNOTATION OF  
HETEROGENEOUS SCIENTIFIC LITERATURE AND DATA RESOURCES

---

ADRIAN-TUDOR PĂNESCU  
ADRIAN.PANESCU@EPFL.CH, CERN.CH

SUPERVISORS:

TIBOR ŠIMKO  
CERN/ IT/ CIS/ DLT  
TIBOR.SIMKO@CERN.CH

CHRISTINE VANOIRBEEK  
EPFL/ IC/ MEDIA  
CHRISTINE.VANOIRBEEK@EPFL.CH



*Switzerland, 2014*



## **Abstract**

This project aims at enhancing the scientific user community features related to shared commenting and annotating of scientific literature and data resources. The current commenting practices in digital repositories often cover global, per-document commenting options only. Within the Large Hadron Collider experiment collaborations at CERN, more targeted needs include per-section, per-line, or per data item commenting granularity.

In the context of Invenio, a digital library platform originating in the high-energy physics community, a framework which allows annotating various types of data resources has been developed. By leveraging it, a general Web page annotator, and a targeted document commenting system, which allows referring to specific elements such as pages, sections, figures, or mathematical equations have been delivered.

State of the art technologies have been employed in order to deliver solutions which are in line with current Web application practices. The framework includes an annotation dissemination mechanism, which adheres to standards such as JSON-LD and REST services, and emerging specifications such as the Open Annotation RDF data model.



# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Invenio Digital Library Platform . . . . .	1
1.1.1	CERN Document Server . . . . .	2
1.2	Motivation . . . . .	2
1.2.1	Use Cases . . . . .	3
<b>2</b>	<b>State of the Art</b>	<b>7</b>
2.1	Commenting and Annotation Systems . . . . .	7
2.2	Linked and Structured Data . . . . .	12
2.2.1	Open Annotation . . . . .	15
<b>3</b>	<b>System Design</b>	<b>17</b>
3.1	Annotation Framework . . . . .	18
3.2	Invenio Framework . . . . .	20
<b>4</b>	<b>Implementation</b>	<b>23</b>
4.1	General Resource Annotator . . . . .	23
4.2	Targeted Document Annotations . . . . .	26
4.3	Dissemination . . . . .	32
4.3.1	JavaScript Object Notation for Linked Data . . . . .	32
4.3.2	Annotation Publishing . . . . .	36
4.4	Future Directions . . . . .	42
<b>5</b>	<b>Conclusions</b>	<b>43</b>
	<b>Appendices</b>	<b>50</b>
A	Open Repositories and AAHEP7 Submissions . . . . .	51
B	Source Code Contributions . . . . .	55

# List of Figures

1.1	Example of comments during the paper review phase on CDS . . . . .	4
1.2	Reference correction form as implemented by INSPIRE. . . . .	5
2.1	Disqus commenting widget embedded on a web blog . . . . .	8
2.2	Interface of the Annotator project . . . . .	9
2.3	Interface of the Pundit project . . . . .	10
2.4	Document annotated using Xournal . . . . .	11
2.5	Example RDF graph . . . . .	13
2.6	RDF/XML Example . . . . .	13
2.7	XTiger template fragment for annotations . . . . .	14
2.8	Basic Open Annotation RDF Graph . . . . .	16
2.9	Twitter message used to annotate an image region using Open Annotation. .	16
3.1	Basic annotation data model . . . . .	19
3.2	Annotation system data flow . . . . .	20
4.1	Base JSON model for annotations . . . . .	23
4.2	Form allowing to add annotations on any Invenio page . . . . .	24
4.3	Web page annotation workflow . . . . .	25
4.4	Invenio commenting form allowing annotation markup . . . . .	28
4.5	Regular expression marker definition for sections . . . . .	28
4.6	Document annotation workflow . . . . .	29
4.7	Document annotation previewer . . . . .	31
4.8	Tree used for annotation aggregation. . . . .	31
4.9	JSON-LD document describing a person . . . . .	33
4.10	Expanded and flattened JSON-LD document . . . . .	34
4.11	Information extracted from JSON-LD document in RDF N-Quads format . .	35
4.12	Annotation serialised as an expanded JSON-LD document . . . . .	38
4.13	Annotation dissemination workflow . . . . .	41

# 1. Introduction

## 1.1 Invenio Digital Library Platform

Invenio [Inv02] is a software suite which allows running an online, large-scale document repository or digital library. Its features cover all aspects of such repositories, including document ingestion, classification, ranking, indexing, curation and dissemination [Kap10, GILMv13]; the architecture of the software is highly modular, each component being in charge of one the mentioned aspects or other additional features. Currently, it is being developed by an international developer community and is released as open source software under the second version of GNU General Public License.

Invenio complies with domain-specific standards such as the Open Archives Initiative (OAI) metadata harvesting protocol. This is a low-barrier mechanism for repository interoperability, in which HTTP requests can be made to providers in order to request bibliographic record metadata [LVdSNW08].

The software platform is built using the Python programming language and currently supports the MySQL relational database. Records are represented internally using an Extensible Markup Language (XML) derivative of the MARC 21 bibliographic format, which is widely used for the representation and exchange of bibliographic, authority, holdings, classification, and community information data in machine-readable form [Lib99].

The Invenio project originated in the world of high-energy physics, being used in production by both the CERN<sup>1</sup> Document Server (CDS), and the INSPIRE repository which is curated by a collaboration of scientist from the Deutsches Elektronen-Synchrotron (DESY), Fermi National Accelerator Laboratory (FNAL), and SLAC National Accelerator Laboratory. Nevertheless, the continuous evolution of the project made it suitable for other use-cases, such as the SAO/NASA Astrophysics Data System (ADS), the scientific information portal of École Polytechnique Fédérale de Lausanne, the Labordoc document repository of the International Labour Organisation (ILO), or the European Open Access implementations OpenAIRE and ZENODO; more information about repositories using Invenio can be found at <http://invenio-software.org/wiki/General/Demo>.

---

<sup>1</sup>European Organisation for Nuclear Research

### 1.1.1 CERN Document Server

The CERN Document Server (CDS) is the main repository of the European Organisation for Nuclear Research. Since 2002, it stores more than 1.300.000 records [Eur02] in over 500 collections [Mar09], both related to the scientific activities of the organisation (research papers, theses), and the administrative and public outreach ones. While the majority of the records include textual content, images, videos and other data formats are also distributed.

Apart from being the first user, CDS was also the initial motivation behind the Invenio software platform. Due to the specific needs of the high-energy physics community in terms of document publishing (e.g., collaborations comprising of hundreds or even thousands of scientists<sup>2</sup>) and dissemination ([GBMH<sup>+</sup>08] showed that users prefer domain-specific search engines), a new solution that apart from providing storage would also facilitate the entire workflow was deemed necessary.

CDS also inspired the modular architecture of Invenio, as this facilitates its deployment, administration and improvement: “The key feature of CDS Invenio’s architecture lies in its modular logic. Each module embodies a specific, defined, functionality of the digital library system. Modules interact with other modules, the database and the interface layers. A module’s logic, operation and interoperability are extensible and customisable” [PBG<sup>+</sup>05].

## 1.2 Motivation

The Invenio platform provides certain features, such as a commenting facility or document collections, that allow users to discuss and augment bibliographic records. While these cover a number of requirements, they do not satisfy all users’ collaborative needs, especially those of large scientific communities such as the ones using CDS. The project identified three possible axis of improvement inside the Invenio ecosystem:

- Targeted commenting: lower granularity levels should be allowed when annotating data items. For example, document comments should support references to specific sections, paragraphs, or text lines.
- Rich feature set: various methods of augmenting data items, such as, but not limited to, document reviews, tags, or record collections should be delivered in a consolidated manner, having a common base upon which new similar features can build by extending existing components.
- Dissemination: communication with external entities should be possible, not only in terms of record bibliographic data, but also with regard to user-generated metadata. This includes both making internal data available to third-parties, and also the ingestion of items from other online or offline systems.

The project’s motivation is further explained by the use-cases described in the next subsection. Both a conceptual design and concrete implementation of a solution to the three points above has been proposed, taking into account state of the art technologies and patterns.

---

<sup>2</sup>As of February 2012, the ATLAS experiment included over 3000 scientist from 174 institutions [Eur].



### 1.2.1 Use Cases

The first considered use-case is one that originated on CDS. Here, the record commenting facilities of Invenio are often employed by the high-energy physics collaborations in order to review new papers in their pre-publication stage. These collaborations use special workflows with multiple commenting rounds for restricted drafts, necessary for amending potential issues before making research results public [Mar12].

In order to reference specific parts of the discussed papers, reviewers often use a format similar to the following:

- “*P1 - equation system is not presumed*”: a correction targeting the first page of the draft.
- “*P1/L143 - equation system is not presumed*”: a variation of the above which also states the targeted text line.
- “*Fig. 2 - equation system is not presumed*”: a comment on a figure.
- “*P1, E2 - equation system is not presumed*”: a comment on Equation 2 on the first page.

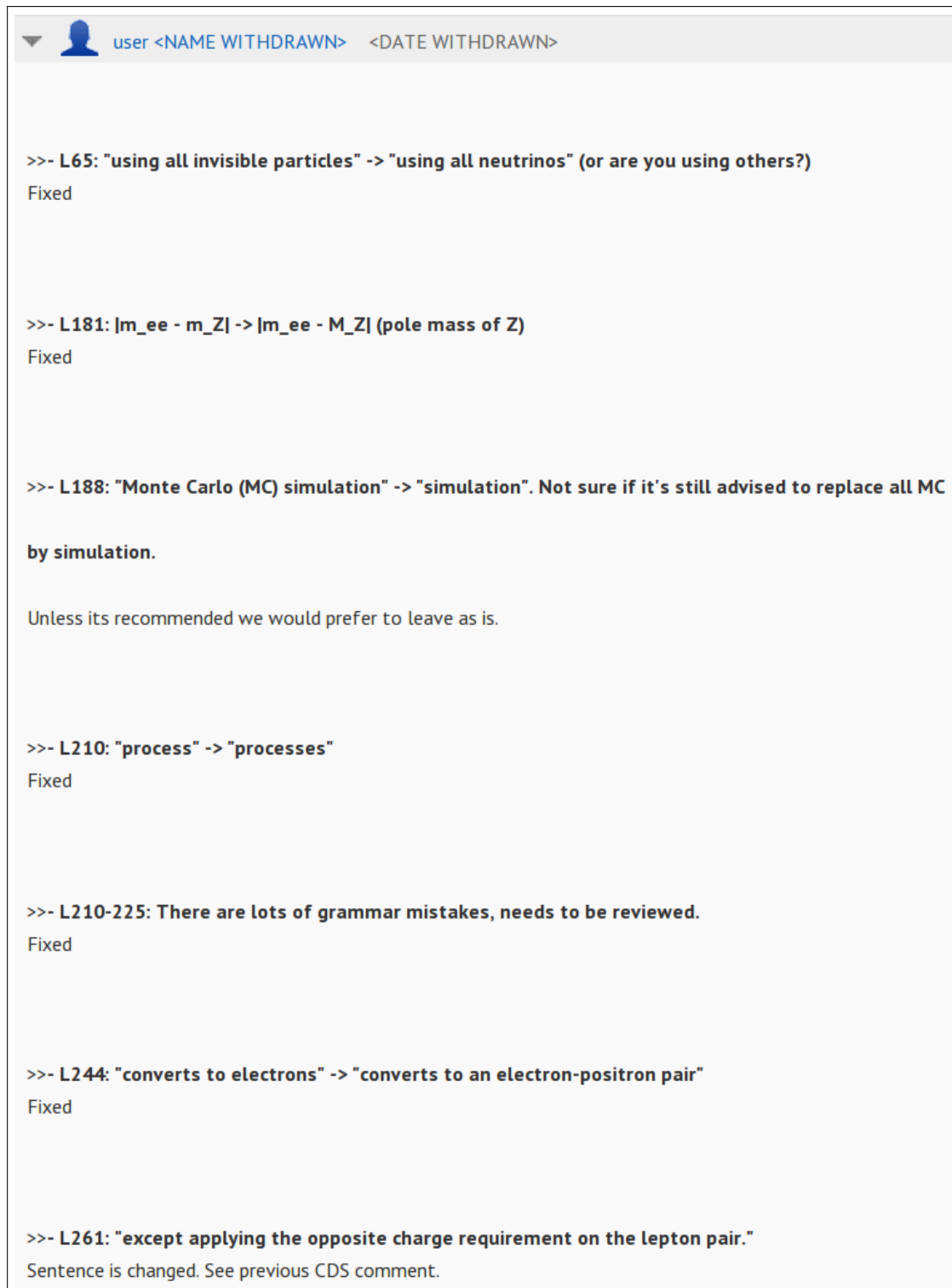
This format is also employed when the original paper authors respond to the corrections, as shown in Fig. 1.1.

While being easy to use and clear regarding the corrections, this method also presents a number of issues:

1. Non-standardised: even within the same commenting round of one draft, reviewers may use different variations of the markup; for example, to reference the first page, both “P1” and “Pag1” could be employed.
2. Difficult to follow and aggregate: this issue emerged from the limitations of Invenio’s commenting system which was not purposed for such structured input. Namely, both targeted remarks and free textual content could be combined in single comments, impeding reviewers from quickly identifying the required corrections to be made. Moreover, as commenting rounds can consist of hundreds of comments coming from multiple reviewers, aggregation is further hindered. For example, in a restricted CDS review round, we have identified over one thousand targeted remarks in 180 comments coming from fifty persons.

Thus, a solution for solving these issues, while preserving the reviewing workflow to which the end-users got accustomed to was required.

Aside from pre-publication reviews, such targeted notes could also be seen useful for annotating resources (textual, multimedia or other formats). While standard comments are sufficient for discussing general aspects regarding the content, allowing annotations on various elements such as pages, figures or paragraphs could prove useful for enriching the content and facilitating its dissemination. Moreover, although comments often include a social aspect, annotations could also be used in a private manner, for each user’s self study.



**Figure 1.1:** Example of comments during the paper review phase on CDS. The author of a paper responds to corrections by referencing text line numbers.

Another requirement emerges from the need to consolidate a number of features present in Invenio and its deployed variants (CDS, INSPIRE) which share a number of similarities:

1. Comments and reviews: allow general remarks on records; reviews include a “*star score*”.
2. Tags: a new feature to be released in the next major version of Invenio, allows users to add brief annotations to records mainly in order to organise and categorise them. Can be public, personal to each user, or shared between groups.
3. Baskets: allow users to create collections of records and annotate them. Similar to tags, can be private or shared.

Apart from these general Invenio features, certain deployments integrate their own homologous functionalities. One example is INSPIRE, which implemented a custom form for allowing users to suggest corrections for the citation list of papers as the one in Fig. 1.2. This can be seen as a particular use-case of the record document review practices previously described, in which only the reference list is considered.

Your Email  Required

Your Name  (Optional)

Comments  (Optional)

[HELP with this form](#)

Click on + to insert new references below existing reference

Reference ID	Reference Text
<input type="button" value="+"/>	<input type="text"/> <a href="#">Search for Citation</a>
<input type="button" value="+"/>	Astrophys.J.,242,772 Abramowicz, M. A., Calvani, M., & Nobili, L. 1980 Astrophys.J.,242,772
<input type="button" value="+"/>	Ann.Rev.Astron.Astrophys.,40,439 <a href="#">Shapes and shaping of planetary nebulae - Balick, Bruce et al.</a> Ann.Rev.Astron.Astrophys. 40 (2002)
<input type="button" value="+"/>	IAU Symp.,259,35 Blackman E. G. 2009 IAU Symp.,259,35
<input type="button" value="+"/>	Astrophys.J.,546,288 <a href="#">MHD stellar and disk winds: Application to planetary nebulae - Blackman, Eric G. et al.</a> Astrophys.J.,546,288
<input type="button" value="+"/>	Mon.Not.Roy.Astron.Soc.,199,883 <a href="#">Hydromagnetic flows from accretion discs and the production of radio jets - Blandford, R.D. et al.</a> Mon.Not.Roy.Astron.Soc.,199,883
<input type="button" value="+"/>	Astron.Astrophys.,377,868 Bujarrabal V. et al. 2001 Astron.Astrophys.,377,868

**Figure 1.2:** Reference correction form as implemented by INSPIRE. Users are invited to suggest amends, a structured form being used for guiding data input.

Finally, the high demand for a metadata dissemination facility has motivated this project. As mentioned, Invenio already includes features enabling record harvesting in compliance with the OAI protocol, but currently does not implement any specific mechanism for exporting user-generated metadata, which can provide more context to consumers.



## 2. State of the Art

### 2.1 Commenting and Annotation Systems

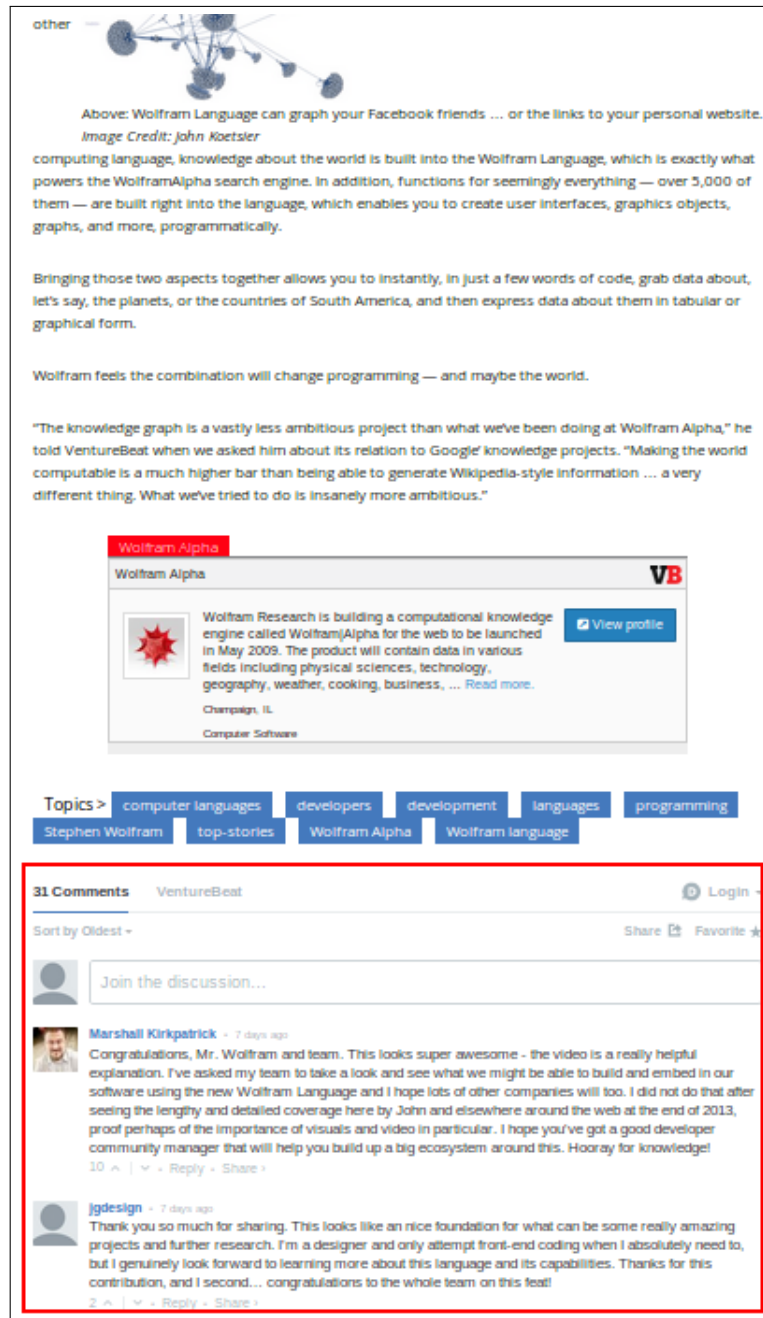
Most Web application nowadays include a social aspect, mostly in the shape of message forums or commenting capabilities. Apart from platforms fully targeted at communication among their users, such as message boards, other applications include facilities that allow commenting the principal features, displayed content, or simply facilitate user engagement. Notable examples include most web blogs, which usually allow comments on posts, YouTube, which encourages users to discuss videos, or Facebook which augments the social aspect by embedding comments on all updates posted by its users.

An emerging trend among web applications is represented by the usage of the social capabilities of one platform for augmenting another application or website. For example, Facebook offers the possibility of embedding its widgets, such that users can comment on external sources using their existing social network account. A more full-fledged solution in this direction is Disqus [Dis07], which provides commenting as a service to other applications, as exemplified in Fig. 2.1. Such solutions offer advantages from the points of view of both the application provider and the end user. The provider saves development time by not having to build a custom commenting system, while users are given the opportunity to aggregate comments and annotations generated across applications on various information resources.

Most commenting solutions propose a plain text format for inserting notes, but alternatives exist. Rich text editors, such as CKEditor [Kna03] are often used to allow standard formatting actions (e.g., bold text) on Web forms. More customisation options can be implemented by using special markup, such as XML or  $\text{\LaTeX}$ , which is converted to the target format (e.g., Extensible HyperText Markup Language (XHTML) for Web applications). Content structure can be enforced by implementing special forms (see Inspire example in Fig. 1.2) or by pre-defining constraints for the users to follow when inserting plain text (see Section 2.2 for an example).

While comments satisfy the needs of most applications in terms of social interaction with and between the end users, they fall short in terms of adding private notes or enriching existing content by augmenting it with user-generated material. A trend among the Web community is related to annotations, which are small pieces of information attached to various resources, regardless of their type (images, (hyper)text, videos).

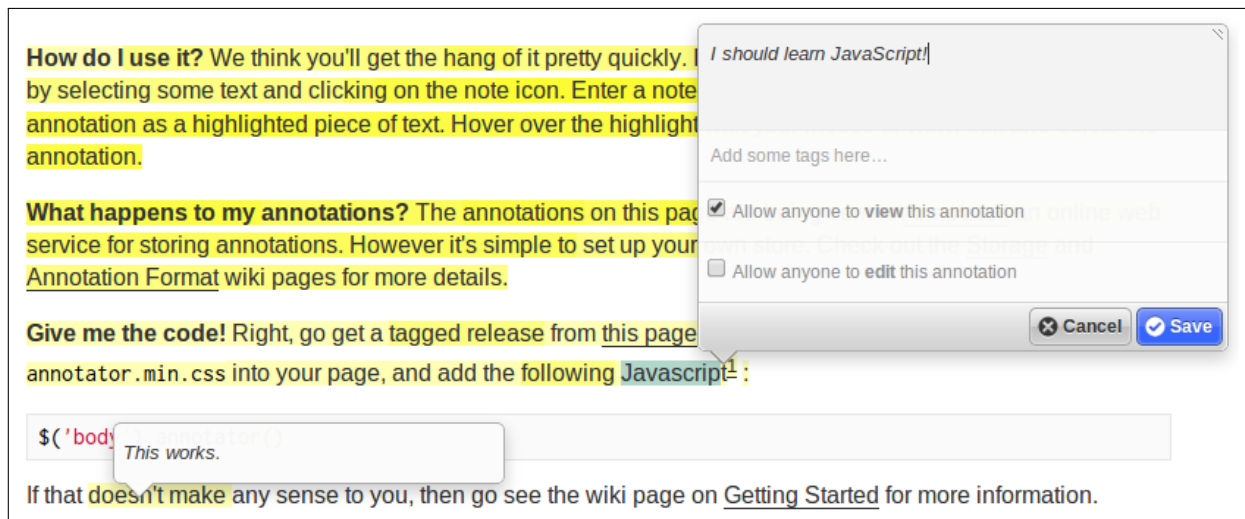
Kahan et al. [KKPS02] proposed the Annotea system, which allows adding structured notes on Web Documents using the Resource Description Framework (RDF) format. Annotations are viewed as statements made by the author about the document. Annotea uses the Amaya [Wor96] editor on the frontend, and a generic RDF database accessible through a HTTP server for storage.



**Figure 2.1:** Disqus commenting widget (highlighted in red rectangle) embedded on a web blog.

The Annotator project [Ope09] allows developers to embed a special widget on their web pages, in order to enable users to annotate any available textual content, as exemplified in Fig. 2.2. It features a JavaScript front-end, annotations being saved in the JavaScript Object Notation (JSON) format in order to be stored on the server side (the reference implementation features a Python wrapper around Elasticsearch [Ela10]).

While the Annotator project provides a limited number of features, it has an open plugin interface, which allows third-parties to provide various additions, such as an image tagging facility, an offline editing mode, or various visualisation utilities. It is interesting to note



**Figure 2.2:** Interface of the Annotator project. It allows adding annotations on any Web page element; in this example, a form for annotating a piece of textual information is displayed.

that, in contrast to the commenting solutions discussed previously, this project regards private annotations as first-class citizens, support for shared content being provided only by a plugin. While the main purpose of the system is to be deployed over existing applications, a stand-alone version which allows users to maintain a collection of annotations over any Web resource is also provided.

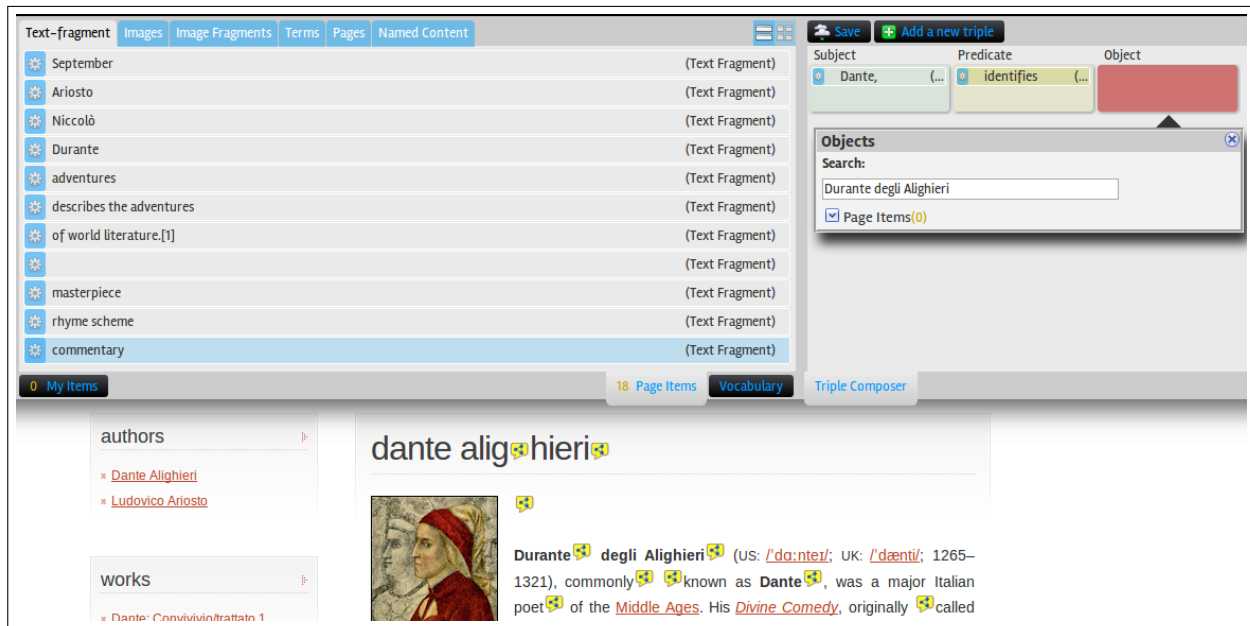
A similar project, funded by the European Union’s Seventh Framework Programme, is Pundit [GMN<sup>+</sup>12], which was initially developed in order to enable humanities researchers to work with manuscripts by leveraging linked data. Similar to Annotator, it features a JavaScript interface that allows users to annotate any Web page, but it provides more features in terms of formatting and structuring the annotations. For example, it allows generating RDF triples, as exemplified in Fig. 2.3.

The main focus of Pundit is allowing users to collaboratively build a knowledge graph, annotations being targeted not only at augmenting Web pages with confined information, but also at linking various resources residing on different domains.

Pundit is designed as a client–server application. The JavaScript client communicates with the backend through the exposed Representational State Transfer (REST)<sup>1</sup> interface, which can also be used by third party plugins. Annotations are stored in an RDF triple store which exposes a standard SPARQL [PS08] endpoint. The open nature of the software facilitates the extension of the platform; for example, video resource tagging was implemented into Pundit [GMN].

It is important to note that between Annotea and newer projects such as Annotator and Pundit, the community seemed to move from defining a strict structure for annotations to a more loose format or to even dropping any content restrictions. While this might impede the consumption of annotations by automated systems, it also makes the process more accessible to users allowing for more metadata to be collected, better reflecting the web resources’ meaning from the users’ point of view [WZY06].

<sup>1</sup>See Subsection 4.3.2 for more details regarding the REST architecture.



**Figure 2.3:** Interface of the Pundit project. Like Annotator, it allows annotating Web pages, featuring a rich editor including RDF triples creation support (right-hand-side), and the possibility to link various pieces of information (left-hand-side).

While the main focus of this project are Web applications, considering the annotation capabilities of traditional *offline* software is also of interest. While these applications mostly lack any possibility of linking data across documents, they also surpass current online applications in terms of possible methods of highlighting and annotating content. Figure 2.4 shows a brief usage example of the Xournal open source tool, which was inspired by Microsoft Windows Journal. Applications such as Adobe Portable Document Format (PDF) Reader [Ado14] or the Mendeley [Men08] reference manager provide similar capabilities, along with the possibility of storing and sharing annotations online. Moreover, it is worth noting that the interfaces used by such applications are usually more familiar to users, due to their longer life span. The resource linking capabilities of online tools along with the annotation editing capabilities of offline software could prove to be a powerful combination.



Actual  
CDS  
Examples

## 1.2 Motivation

The initial motivation for this project arose from a use-case originating on CDS. Here, the record commenting facilities of Invenio are often used by the high-energy physics collaborations in order to review new papers in their pre-publishing stage. These collaborations use special workflows with multiple commenting rounds for restricted drafts, necessary for amending potential issues before making research results public [Mar12].

In order to reference specific parts of the discussed papers, reviewers often used a format similar to the following:

- “P1 - equation system is not presumed”: a correction targeting the first page of the draft.
- “P1/L143 - equation system is not presumed”: a variation of the above which also states the targeted text line.
- “Fig. 2 - equation system is not presumed”: a comment on a figure.
- “P1, E2 - equation system is not presumed”: a comment on Equation 2 on the first page.

This format is also employed when the original paper authors respond to the corrections, as shown in Fig. 1.

While being easy to use and clear regarding the corrections, this method also presents a number of issues:

1. Non-standardised: even within the same commenting round of one draft, reviewers used different variations of the markup; for example, to reference the first page, both “P1” and “Pag1” could be used.
2. Difficult to follow and aggregate: this issue emerged from the limitations of Invenio’s commenting system which was not purposed for such structured input. Namely, both targeted remarks and free textual content could be combined in single comments, impeding reviewers from quickly identifying the required corrections to be made. Moreover, as commenting rounds can consist of hundreds of comments coming from multiple reviewers, aggregation is further hindered. For example, in a restricted CDS review round, we have identified over one thousand targeted remarks in 180 comments coming from fifty persons.

Thus, a solution for solving these issues, while preserving the reviewing workflow to which the end-users got accustomed to was required.

Aside from pre-publication reviews, such targeted notes could also be seen useful for annotating resources (textual, multimedia or other formats). While standard comments are sufficient for discussing general aspects regarding the content, allowing annotations on various elements such as pages, figures or paragraphs could prove useful for enriching the content and facilitating its dissemination. Moreover, while comments often include a social aspect, annotations could also be used in a private manner, for each user’s self study.

Another motivation emerges from the need to consolidate a number of features present in Invenio and its deployed variants (CDS, INSPIRE) which share a number of similarities:

1. Comments and reviews: allow general remarks on records; reviews include a “star score”.
2. Tags: a new feature to be released in the next major version of Invenio, allows users to add brief annotations to records mainly in order to organise and categorise them. Can be public, personal to each user, or shared between groups.
3. Baskets: allow users to create collections of records. Similar to tags, can be private or shared between users.

2      Make sure this enumeration does not spread over multiple pages.

Figure 2.4: Document annotated using Xournal

## 2.2 Linked and Structured Data

While annotations facilitate individual note taking for users and improve engagement inside platforms deploying such mechanisms, it is also desirable that metadata can be shared across ecosystems in order to allow for new connections between information resources to be established. Thus, the usage of standardised annotation dissemination formats is necessary.

The systems presented in the previous section make use of different forms of RDF structures in order to represent annotations both internally and externally. RDF [Wor04] is a model defined by the World Wide Web Consortium (W3C) to be used for online data interchange. The main idea behind it is related to making statements about resources in the form of subject–predicate–object expressions, such as:

```
<Dave Beckett> <edits the> <RDF syntax grammar specification>
```

Due to this structure, these statements are named *triples* in the RDF terminology. A collection of RDF statements will constitute a directed multi-graph, also named the *context* of the concerned triples; contexts use Internationalised Resource Identifiers (IRI)<sup>2</sup> to link resources. A simple example is shown in Fig. 2.5

It can be observed that this model is well-suited for the annotations use case. Metadata can be seen as a statement about a resource in the form (but not restricted to):

```
<Resource identified by IRI> <has metadata> <annotation body text>
```

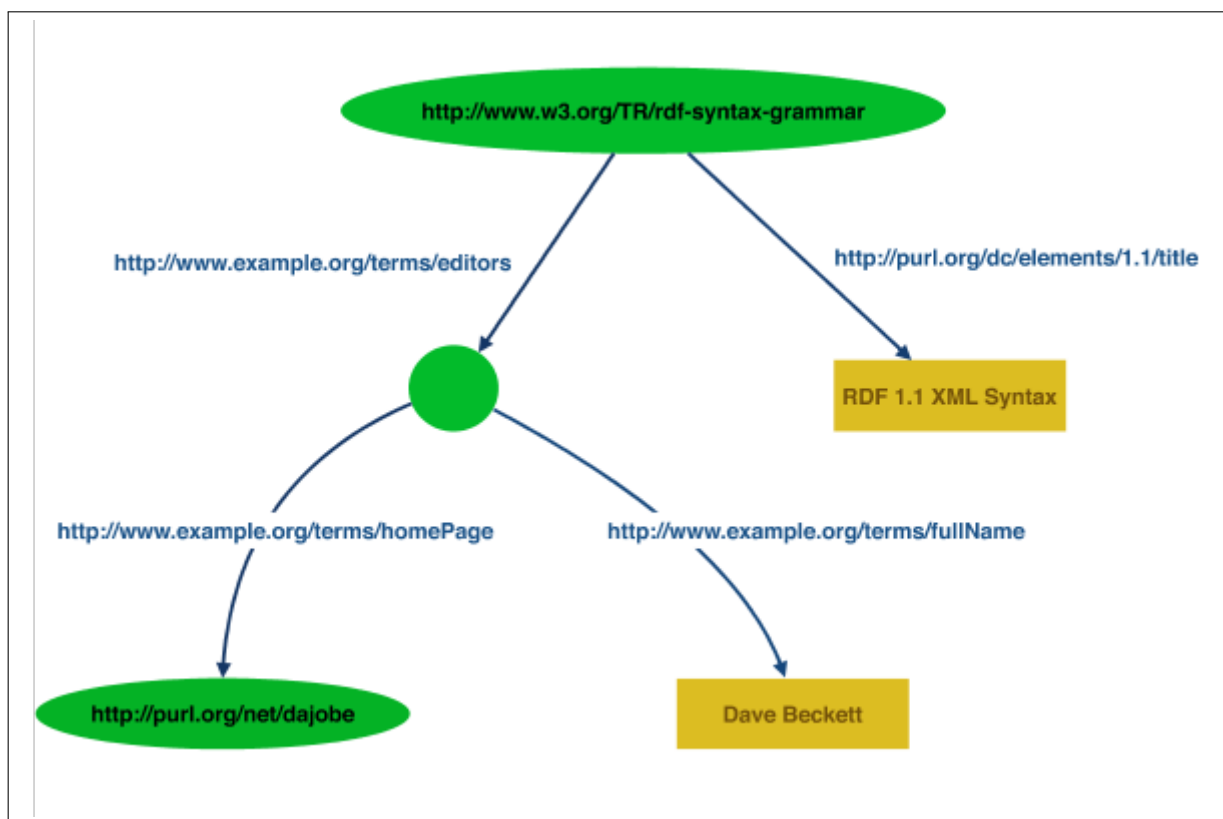
The RDF data model can be serialised in different formats, such as Turtle, N-Triples, N-Quads (a superset of N-Triples), Notation 3 (N3), RDF/XML or JSON-LD. Note that the RDF/XML format is commonly referred as simply RDF, due to its widespread usage, leading to some confusion between the *data model* and the *serialisation format*. A typical RDF/XML document will look as the one in Figure 2.6, which encodes the graph in Fig. 2.5. In order to be encoded in XML, the RDF graph nodes and predicates need to be represented in the specific structural terms: element and attribute names, element contents and attribute values. Note that the example XML uses three collections to define the elements, namely the RDF standard set, the Dublin Core Metadata Element Set [Dub12] and a placeholder set; in real use-cases, each Web application can define its own set of elements, depending on the specific domain knowledge.

Similar to RDF/XML, an augmented version of the JavaScript Object Notation (JSON) format, JSON-LD, can also be used to serialise the data model; an in-detail description, in the usage context of the project, is included in Subsection 4.3.1.

Relational databases, while not suitable for the graph structure of the RDF model, are commonly used solution for storing such data. If a database stores only the triples it is called a *triplestore*, and if the context is also included, a *quad* store is employed. These stores can be queried using the SPARQL [PS08] language.

---

<sup>2</sup>IRI is a generalisation of Uniform Resource Identifiers (URI), that may contain any Unicode character; a Uniform Resource Locator (URL) is a type of URI that identifies a resource through a representation of its primary access mechanism. Throughout this thesis, only the IRI term will be utilised.



**Figure 2.5:** Example RDF graph, source [GS14]. This graph can be summarised briefly as: The RDF syntax specification, identified by an IRI, has a person characterised by its full name and personal Web page as editor.

---

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:ex="http://example.org/stuff/1.0/">

  <rdf:Description rdf:about=
    "http://www.w3.org/TR/rdf-syntax-grammar"
    dc:title="RDF1.1 XML Syntax">
    <ex:editor>
      <rdf:Description ex:fullName="Dave Beckett">
        <ex:homePage rdf:resource=
          "http://purl.org/net/dajobe/" />
      </rdf:Description>
    </ex:editor>
  </rdf:Description>
</rdf:RDF>
```

---

**Figure 2.6:** RDF/XML Example, source [GS14]. Specific XML elements are used to encode the RDF context in Fig. 2.5.

While RDF provides a standard method for disseminating data, it is mainly purposed for machine manipulation. Taking the RDF/XML format as an example, it can be seen that editing such a document by hand is not optimal for end-users. Nevertheless, capturing input and filling an XML template, while accounting for invalid data or out-of-scope information, can become expensive in terms of computing resources.

A simple solution is to deploy standard forms for guiding the users in the data collection process. The previously discussed Pundit project uses such a visual form for allowing users to input RDF statements. While this solution is fast to develop due to the prevalence of the practice in Web applications, it also lacks in terms of extensibility and scalability; it can be seen that in order to be able to express complex RDF graphs, a considerable effort must be invested.

The XTiger [MED10] XML language allows defining templates, in order to guide an editing tool for building documents that follow a predefined model. It does this by pre-defining a set of typed structures and constructors for these (*components*); for example, a minimal template for inputting annotations could be defined as in Figure 2.7. The XTiger template can be used to generate documents in target languages, for example XHTML or RDF/XML.

---

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xt="http://ns.inria.org/xtiger"
      xml:lang="en" lang="en">

  <xt:head version="1.0">
    <xt:component name="annotation-form">
      <xt:component name="target">
        <xt:use types="text">Annotation Target (IRI)</xt:use>
      </xt:component>

      <xt:component name="annotation">
        <xt:use types="text">Annotation Body</xt:use>
      </xt:component>
    </xt:component>
  </xt:head>

  <body>
    <xt:use types="annotation-form" label="annotation"/>
  </body>

</html>
```

---

**Figure 2.7:** XTiger template fragment for annotations. A custom component, consisting of two input fields, is defined in the `xt:head` element and used to build the form inside the XHTML body.

Special constructs for repeating components across the resulting document can ease the creation of complex structures. RDF graphs can be constructed by using the RDF/XML serialisation format as the target for an XTiger template and by leveraging the special constructs that allow repeating components across the produced document (e.g., `xt:repeat`). It is worth noting that XTiger borrows certain elements from XML Schema, being possible to define the semantics of XTiger as an extension of XML Document Type Definitions (DTD); thus the constraints defined by an XTiger template can be expressed in schema language and an XSLT transformation can be employed to perform the translation to XML Schema [QRSV10].

To allow even further flexibility, the Adaptable XML Editing Library (AXEL) [MED11] can be used. This JavaScript library allows users to freely combine XTiger components, while still producing conforming documents as the end-result.

Another possibility is to use RDFa [ABMH13], a specification for attributes to express structured data in any markup language. For example, RDF attributes can be inserted into the XHTML documents used to present and capture metadata, facilitating the annotation workflow from both the perspective of the end-user and the backend system.

### 2.2.1 Open Annotation

Exporting metadata in a syntactically standard format is a necessary step towards interoperability between systems collecting user annotations, but not a sufficient one. The semantics of metadata also need to have an agreed form; in the context of RDF, applications exporting annotations could also supply an ontology which describes what the meaning of data is.

The Open Annotation [Wor13] data model (denoted by *OA* in the rest of this document) is a specification proposed by W3C<sup>3</sup> that aims at solving this exact issue. It proposes a general vocabulary of terms that can be used to describe the most common use-cases of annotations.

The minimal basis of OA is defined as an RDF graph in which annotations have a target identified by an IRI, and a body; an example graph is included in Figure 2.8. Further restrictions regarding the types of these elements are not enforced; for example, the target of an annotation can be another annotation. Nevertheless, it is expected that the type is specified when serialising the representation, in order to facilitate third-party consumption.

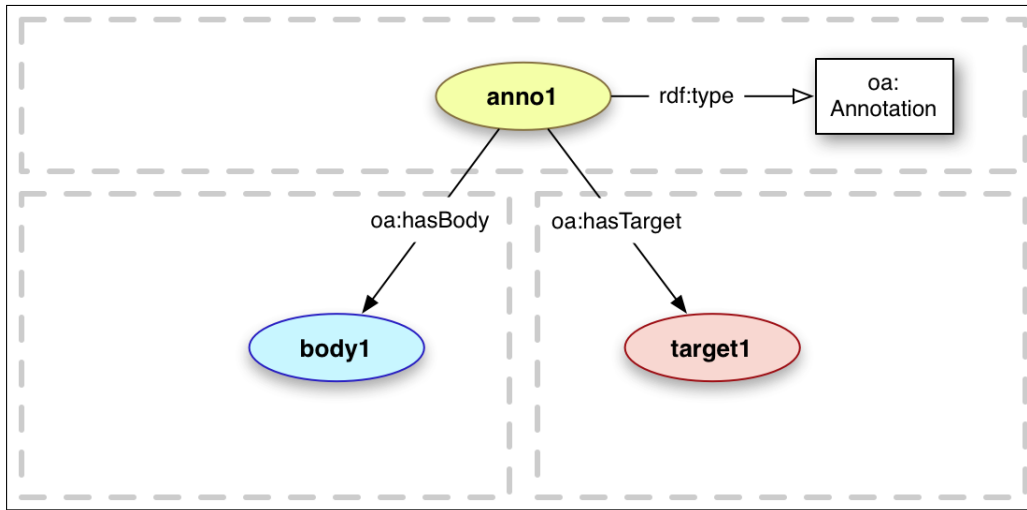
The body and target attributes can be extended by adding miscellaneous attributes. For example, Figure 2.9 presents an annotation where the target is constrained to a specific region of an image. Similarly, the body can also be constrained; for instance, it could be specified that only certain pages of a textbook are describing a target image.

Other elements supported by OA include annotation tagging, versioning the target or body or constraints on the validity of the annotation (e.g., HTTP headers than need to be used to retrieve the correct target). Further details can be found in the specification [Wor13].

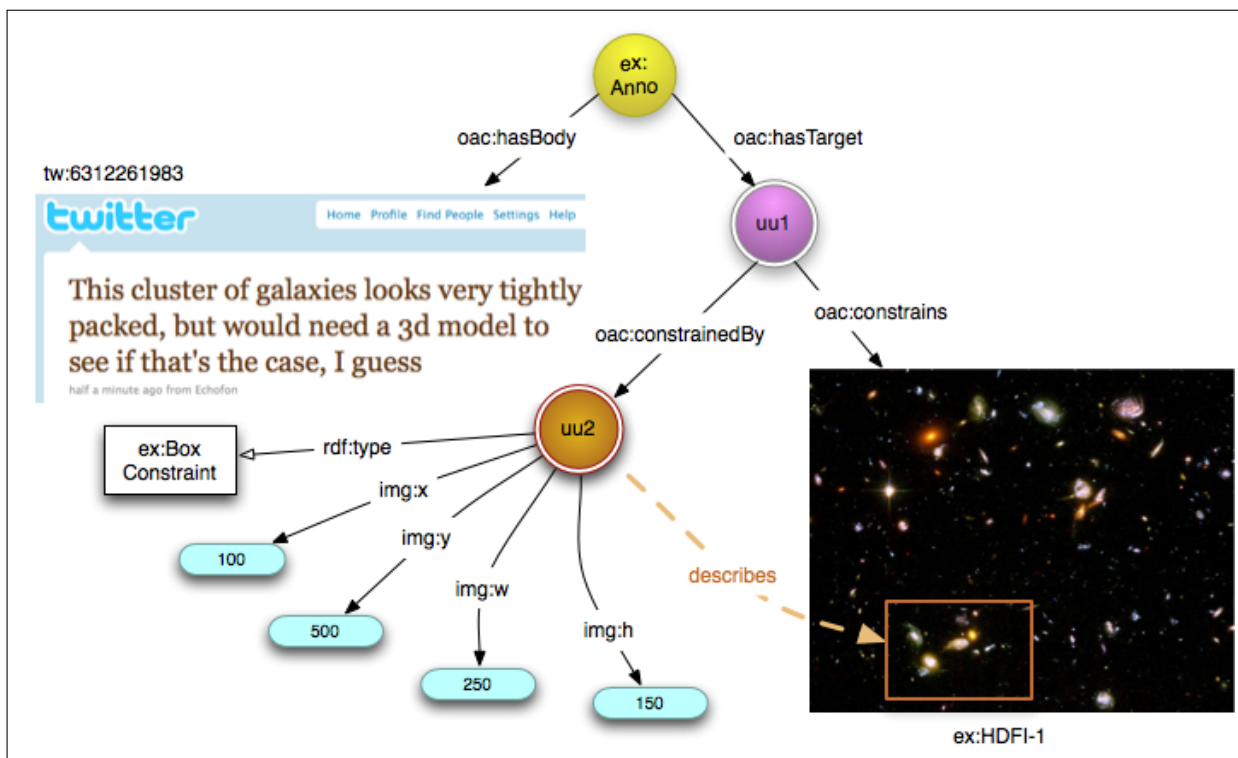
As Open Annotation model instances are RDF graphs they can be exported in any of the supported serialisation formats; the guideline is to use JSON-LD [Wor13], the option being further explored by this project, with details in Section 4.3.

---

<sup>3</sup>A W3C *Community Draft* as of February 2013.



**Figure 2.8:** Basic Open Annotation RDF Graph, source [Wor13]. Annotations need to specify a body and target identified by an IRI (e.g., a Web page).



**Figure 2.9:** More complex example usage of Open Annotation, source [SVdS10]. A Twitter message is used as the body for an annotation targeted at a specific region of an image. The *target* attribute points at the whole image IRI, but is constrained by rectangle coordinates  $(x, y, w, h)$ .

### 3. System Design

Considering the current features of Invenio and the state of art, we defined the following requirements for adding annotation capabilities to the platform:

- The system should be extensible and allow consolidating the existing commenting, reviewing, tagging and grouping of records either by superseding the existing feature set, or by providing a base framework upon which other features can build. Future similar capabilities should be considered by means of development time reduction.
- The integration of the new system and features should be backward compatible and transparent to end-users. Basically, the Invenio application should continue to function in the same manner, with an identical feature set, as before this project.
- The targeted commenting use-case, presented in Section 1.2 should be considered of high-priority, as the described workflow, while frequently used by the community, is inefficient.
- The new system should seamlessly integrate with the access control features of Invenio. For example, during the review period, records on CDS are restricted to only the participating authors and reviewers. Thus, any associated metadata should also obey the access control rules.
- Disseminating annotations should be possible in a standardised manner, alike the harvesting mechanisms employed for records.

The first considered option was using one of the systems previously presented, namely Annotator or Pundit. The first one seems a promising solution, as it uses a loose data model that can be easily extended to various use cases and annotation target types. Unfortunately, the software is involved more on the frontend aspect, with only minimal infrastructure for dissemination and storage. Also, the project focuses more on annotating fragments of HTML documents, which does not fit the targeted commenting use case; records in Invenio provide the full text usually as PDF documents and such, a proper solution must be implemented in order to allow making references inside them.

The Pundit project is more complete in terms of features, allowing to annotate various content types out of the box (text, multimedia), and providing a rich editor which allows embedding RDF graphs and attaching external files. This option was designed more as a standalone application and not as a lightweight wrapper like Annotator, providing features such as authentication or a complete backend including both storage and dissemination by means of a REST interface. This made it unsuitable for a seamless integration with Invenio,

which would have meant stripping any unnecessary components while preserving the required functionality.

One downside shared by both Annotator and Pundit is related to the targeted document annotation exemplified in Fig. 1.1. While it is possible to simulate such a feature in both systems, it is worth noting that paper reviewers frequently take notes in offline mode (e.g., during commute), this being one of the reasons behind the employed markup. Using any of the two systems in offline mode for this purpose is mostly impossible, as they rely on rich graphical interfaces for specifying the exact annotation targets. For this project's purposes, both an interface and an offline editing mode should be provided.

The careful evaluation of the existing alternatives lead us to consider implementing a custom solution which, would satisfy all the user requirements, and also allow for a gradual integration in the existing Invenio framework. Nevertheless, existing components and ideas would be reused and standard practices would be strictly followed, especially in terms of metadata dissemination.

The next section will give an abstract overview of the proposed system, while Chapter 4 will detail the delivered feature set and implementation details. Due to the overlap of this project with a major refactoring of the Invenio project, certain miscellaneous tasks were carried out in order to provide a working base for the new features, this being briefly discussed in Section 3.2.

## 3.1 Annotation Framework

For implementing the annotation system, an extensible design was considered, in which different classes of metadata are constructed using a minimal set of base elements. By accounting for the targeted use cases and standard formats, such as OA, the minimal information set we reckoned necessary for any type of annotation consists of:

1. The target of the annotation (e.g., a document, a Web page, etc.)
2. The content; we target mostly plain text, with possible markup (e.g.,  $\text{\LaTeX}$ ), other types, such as multimedia, being possible by extending the base models (see below).
3. The author of the annotation
4. Time stamp
5. Permissions of the annotation; arguably, this could also be provided as an extension, but as access control is an important aspect of Invenio, we considered that all metadata should have restriction rules integrated. Public access can also be encoded using this parameter.

Starting from this base, further models can be defined. For example, to allow attaching external files, only a new field referencing such data needs to be added. Extensions should also be possible by slightly modifying one of the base fields; for example, while in the general case the target field is an IRI, for annotations targeting specific locations in Invenio record documents, supplementary information (e.g., page number) needs to also be supplied, in the same manner OA uses constraints. The model is briefly summarised in Fig. 3.1.



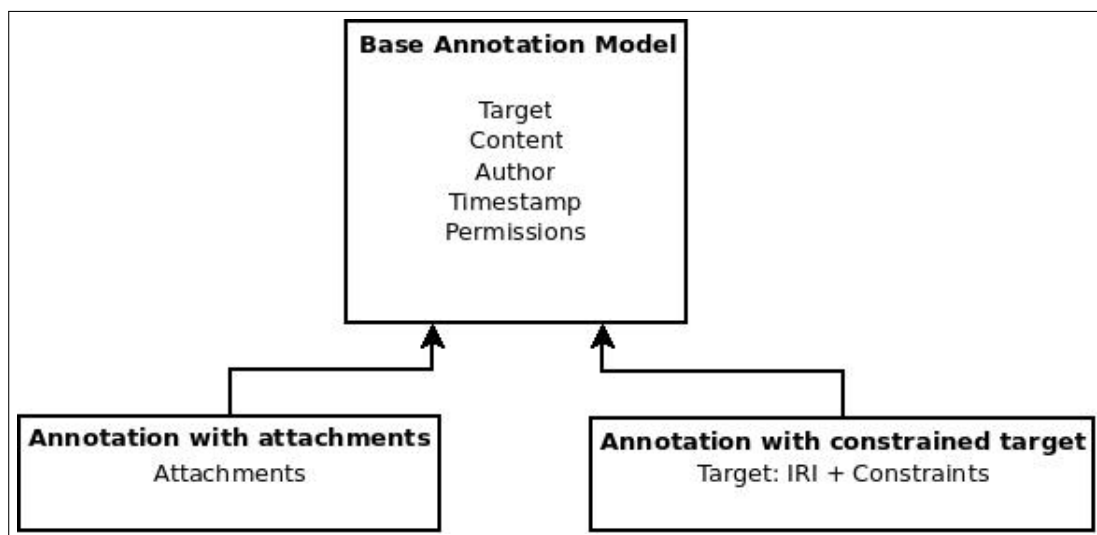
In terms of the system that should provide the annotation facilities, the following components need to be considered:

- Front-end annotation editor; this is presented to end-users, who should fill in the target and content fields, the other information being assumed. Different annotation models can implement different editors, in order to guide users into inputting complete and valid data.
- Annotation persistent storage.
- Annotation exporter; this should provide access to metadata to third parties, in a standardised format, and with regard to the access control policies.

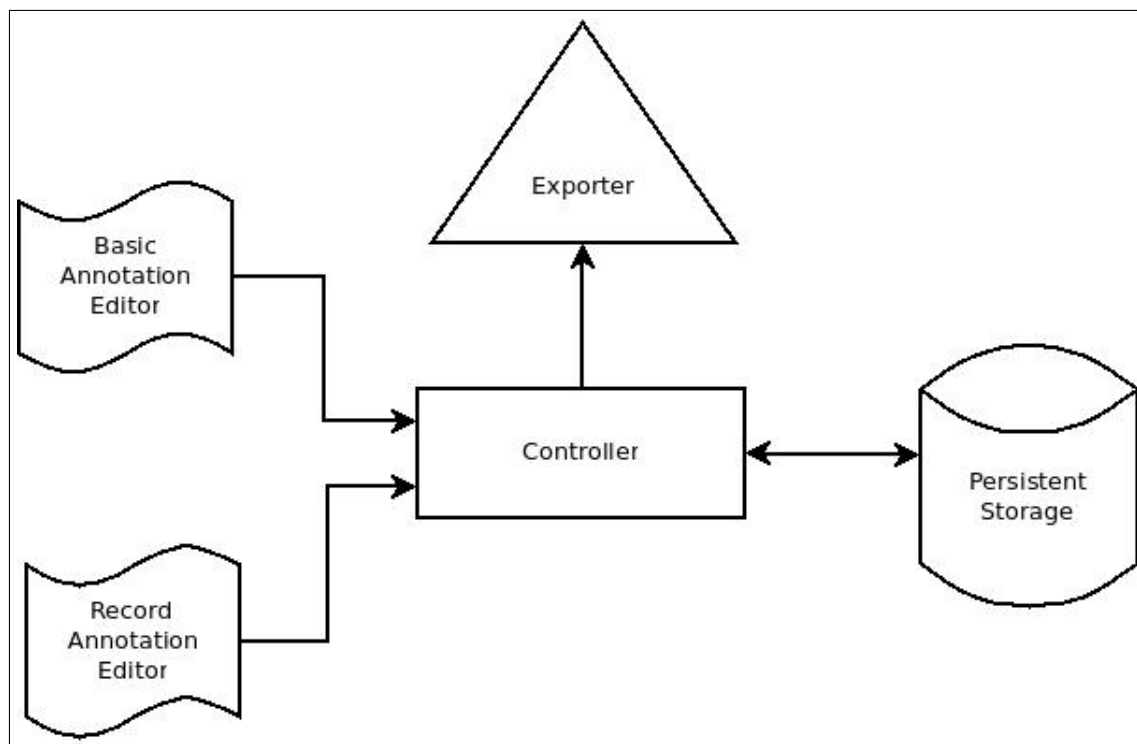
Clearly, seamless communication between these components must be provided, through an equivalent of the controller in the Model–View–Controller pattern. Thus, the data flow of the annotation system can be summarised as in Fig. 3.2.

One issue that needs to be considered regarding the extensible annotation model regards the method used for data persistence. Invenio currently uses an SQL database, and this model does not fit well the proposed definitions. For example, if the three models in Fig. 3.1 are considered, it can be seen that certain difficulties in expressing them in relational terms will be encountered. Using separate tables for each model is clearly not optimal in terms of future scalability, while using one table by, for example, encoding data in a free text field with a loose syntax, is difficult to maintain, error-prone, and possibly sub-optimal in terms of system performance.

Fortunately, the new version of Invenio brings support for storing JSON documents, and the project fully leveraged this possibility, as explained in the next sections. Apart from solving the issue described above, this method also facilitates distribution of annotation data to third parties, as JSON is widely used on the Web for these purposes.



**Figure 3.1:** Basic annotation data model. A series of base fields are defined, upon which various annotation types can build more complex structures, both by adding new fields or modifying existing ones.



**Figure 3.2:** Annotation system data flow. End-user create new annotations using the specialised editors, content being persisted in a database. Metadata can be publicised to interested third parties.

## 3.2 Invenio Framework

As the Invenio software is now being used at large-scale by a considerable number of organisations across various activity domains, the ability of easily adapting to various use cases became a stringent requirement. As a result, a major refactoring of the project, in order to make it more customisable, has been undertaken. The main focus was on changing the delivery package from a fairly tightly integrated digital library platform to a coherent collection of repository components. As this project overlapped with the reorganisation and contributed a number of required additions and fixes to the main source tree, a brief description of the process is included in the following.

The first step in rewriting the platform targeted the reorganisation of the source code. Invenio already featured a modular design [PBG<sup>+</sup>05], but further efforts were made to better separate utilities (record commenting, user messaging etc.) from core bibliographic modules. This also allowed the identification of components that could be reused outside Invenio, being general enough for any other Web application. For example, a component offering support for external user authentication sources is now distributed separately of core Invenio.

The second important change regards the use of external tools for simplifying the code base and workflows. A modern framework for building Web applications, Flask [Ron10] was employed for replacing the custom Web Server Gateway Interface (WSGI) components used since Invenio’s first iterations. Similarly, a Structured Query Language (SQL) toolkit and object-relational mapper (ORM), SQLAlchemy [SQL06], was employed in order to simplify the code used for interacting with the database backend and eliminate the need for hand

crafted queries. This is implemented as a thin wrapper around standard Python classes, that abstracts database operations from developers. An added benefit of using such a solution is the possibility of switching the database system; the Invenio collaboration currently explores the move from MySQL to PostgreSQL for its relational needs.

Other important backend changes involved the data workflow, both inside and outside Invenio. Support for other formats than MARC was added by a new module, JSONAlchemy, which is also of interest as it provides abstractions for handling JSON documents in the same manner SQLAlchemy facilitates the handling of relational models. Another module that facilitates the creation of REST interfaces was deployed, a primer usage scenario being an Application Programmable Interface (API) for document deposition. Other additions include a customisable record ingestion workflow engine, developed within the INSPIRE project, and support for cloud storage by means of a filesystem abstraction module.

Another targeted aspect relates to the user interface, a modern and customisable solution being of high interest. Customisability has been achieved by deploying a templating engine, Jinja version 2, which is already integrated with the Flask framework, for creating highly modular HTML pages. It features an extension model in which templates inherit blocks of markup from each other, similar to object oriented inheritance; thus, if an organisation using Invenio desires to use its own custom design elements, it just needs to extend or overload the base templates, modifying only the required parts (e.g., logo or colour scheme).

Modernisation wise, a series of HTML5 technologies have been employed; the Bootstrap [OT11] frontend framework is being used for defining a number of interface elements (dialogs, menus, etc.), and updated versions of various JavaScript libraries employed for delivering a fluid user experience in line with modern practices and patterns.

A new deployment process has been developed, replacing the GNU build system with a Python-specific process for supplying the backend dependencies, and a specialised workflow for *static* files, such as JavaScript libraries and style sheets.

Performance wise improvements were also targeted, indirectly by adopting new practices while re-witting legacy code, and directly by employing new technologies such as Redis [San09] for distributed caching, or Celery [Sol07] for task scheduling and queuing.

The contributions made by this project to the refactoring are as follows:

- Fixed various issues regarding user account handling, such as authentication or validation. Also, certain features such as the password change facility were ported from the legacy code to the new structure.
- Contributed with fixes and additions to the developer workflow; for example, a brief summary of the steps required for setting up an Invenio development environment was compiled and maintained. Certain maintenance tasks were also carried out regarding the external dependencies used by Invenio, such as version upgrading and source discovery and correction.
- Improved various user interface aspects, by fixing existing issues, further modularising Jinja templates, reorganising source code and developing new utilities. For example, a routine for simplifying the internationalisation of text strings used inside JavaScript sources was developed.
- Reorganised the file tree structure for a number of modules, especially in regards to better separating JavaScript and Cascading Style Sheets (CSS).

- Tested the system in order to discover or confirm any issues resulting from the code reorganisation, and, where possible, proposed fixes.

Apart from these prompt contributions, a number of systematic and more complex tasks were carried out. As, in the initial phase, the project built around the commenting facilities already provided by Invenio, this module was constantly maintained and improved, any general additions being pushed back to mainline.

Other additions have been the result of identifying new requirements while developing the annotation capabilities, namely:

1. Implementing an easy-to-use annotation interface requires displaying a preview of the targeted documents, in order to provide the offline editor experience described in Section 2.1. While this is still an open problem in Invenio, a prototype of such a facility for PDF documents has been implemented in the existing previewer module, which was also improved to fit the new use-case. A brief research of existing solutions was carried out, PDF.js [Gal11] and Multivio [MM09] being considered for future work.
2. Extending the JSONAlchemy module in order to allow greater flexibility in terms of usage scenarios and allow exporting data in the JSON-LD RDF serialisation format. This will be discussed further in Section 4.3.

Finally, in order to provide a demonstrator for the new annotation features, a special instance of Invenio, Invenio Labs, was deployed at <http://labs.invenio-software.org>. This required maintaining a running version in an environment close to production ones, while also providing customisation additions. A significant encountered issue relates to the Python environment version (2.6) used on CERN servers running the Scientific Linux operating system. For example, the external library used for generating JSON-LD documents, PyLD [Dig11] was back-ported in order to work properly.

Building this demonstrator was facilitated by the introduction of a new distinct package, `invenio-demosite`, which allows developers to quickly set up a custom Invenio instance. A number of modified HTML templates were created, a JavaScript website tutorial feature was developed, and record data along with the necessary deployment routine were put in place over this standardised customisation package.

The contributions made by this project in terms of source code are enumerated in Appendix B, along with references to the involved repositories.

## 4. Implementation

Starting from the requirements and abstract design presented in the previous chapter, an implementation of the Invenio annotation facility is proposed, the following sections detailing its various aspects.

It is worth noting that due to various constraints, the project started by implementing the targeted document annotation facility, which built upon the existing SQL based backend already used for record commenting. The first iteration was deployed on Invenio Labs, in order to gather information about possible issues and shortcomings of the design from end-users. Using the experience gained during this process, the general annotation framework was developed and the targeted commenting facilities adapted to using it, this including moving from the SQL backend to a JSON document store based one.

Nonetheless, the description of the developed components will follow the more logical top-down approach, from the highest level of abstraction to the more specific use-cases.

### 4.1 General Resource Annotator

The concrete implementation of the basic annotation model presented in Section 3.1 consists of a JSON representation similar to the one in Fig. 4.1.

---

```
{ _id:    UUID ,  
  who:    UserID ,  
  where:  IRI ,  
  what:   String ,  
  when:   Timestamp ,  
  perm:   Object }
```

---

**Figure 4.1:** Base JSON model for annotations; each field is denoted by its name and held data type.

Each annotation should be distinguished by a Universally Unique Identifier (UUID) and have its target associated either explicitly or implicitly (through application specific translations) with an IRI. Other fields include the author of the annotation (an Invenio user), its textual body, time stamp, and access restrictions.

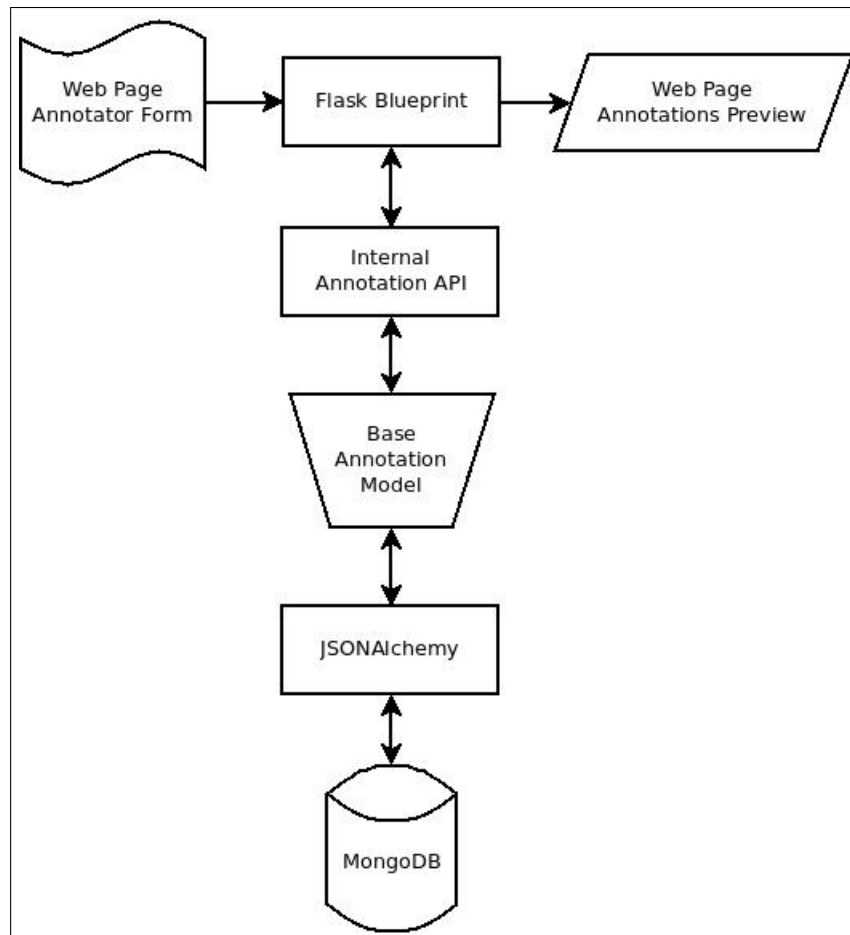
This model facilitated the implementation of an annotation system which allows adding metadata on any Web page of Invenio; users can add such annotations by using a dialog similar to the one in Fig. 4.2.

The image shows a screenshot of the Invenio web interface. A modal window is open for adding an annotation. The modal has two tabs: 'View' and 'Add'. The 'Add' tab is active, showing a text input field with the placeholder text 'Adding an annotation on collection/Books&Reports...'. Below the text field is a 'Public' checkbox, which is checked. At the bottom of the modal is a blue 'Send' button. The background of the page shows the Invenio search results page, with a search bar at the top and a sidebar on the left listing various collections like 'Articles & Preprints', 'Books & Reports', etc.

**Figure 4.2:** Form allowing to add annotations on any Invenio page. Users can add a textual body and specify if the annotation should be public, visible to any Web page visitor, or private, visible only to them.

The communication between the frontend interface and backend system is mediated by view functions defined inside a so-called Flask *blueprint*. Apart from allowing users to input data, displaying annotations is also possible.

Annotations are stored in a MongoDB database [Mon09] using the facilities provided by JSONAlchemy. This module allows defining a schema for JSON documents, in which the collection of fields and various restrictions on them can be defined; for example, the UUID is required for all annotations, and the time stamp should have a standard format. Moreover, JSONAlchemy allows querying the stored objects (e.g., get all annotations with the target being the main page of CDS), and modifying them. A thin wrapper for adapting these operations to the annotation use case is included in an internal API, which can be used by the Flask blueprint, REST interface (see Subsection 4.3.2), or even other Invenio modules to access annotation data. An overview of the workflow is included in Fig. 4.3.



**Figure 4.3:** Web page annotation workflow; users can add annotations on any IRI-identified resource. The specialised API stores data in a MongoDB database using the JSONAlchemy model. The Flask blueprint provides both annotation creation and display facilities to users through its views.

## 4.2 Targeted Document Annotations

Using the base annotation model as a starting point, the targeted document commenting feature can be implemented. In this scenario, the “*where*” field of the base model needs to encode both the document identifier and the exact target of the annotation (e.g., equation four on second page).

In the current version of Invenio, records can have one or more documents associated, each of them with multiple versions. For example, during the CDS paper review rounds, a single record will be used to upload multiple drafts of the same document. Thus, annotations need to be attached to the correct version of documents, in order to avoid displaying obsolete information. As the document storage and handling workflow is currently changing for the next version of Invenio, we have chosen to implicitly attach annotations to the latest version of record documents; nevertheless, this can be changed with minimal effort due to the customisable nature of the JSON models.

Regarding the document locations to which annotations can refer, after studying over one thousand comments from CDS review rounds, we have chosen a total of nine so-called “*markers*” which can identify valid targets:

1. Page (P)
2. Figure (F)
3. Line (L): this refers to the text line number which is sometimes used in drafts (e.g., L<sup>A</sup>T<sub>E</sub>X documents allow this when using the “*draft*” option for the document class). Optionally, this could also be used in other configurations as, for example, specifying a row in a table, or a relative line in a paragraph.
4. Equation (E)
5. Table (T)
6. Section (S): this could also refer to subsections as long as they possess a proper identifier (e.g., Subsection 4.2.1 can be marked using either *S.4 : S.2* or *S.4.2*).
7. Paragraph (PP): this will most likely refer to relative locations, such as “*the third paragraph on the second page*”.
8. Reference (R): a bibliographic citation, could prove useful in a scenario similar to the INSPIRE citation correction form presented in Section 1.2.
9. General (G): this allows specifying annotations which regard general aspects, such as formatting issues (“*the text is not justified*”), or notes targeting currently nonexistent elements (“*a reference to the 2014 paper needs to be added*”).

The letters between parentheses above denote the special markup that can be employed by users for referring to the various document locations; for example, “*P.2: F.3a: equation system is not presumed*” is a valid annotation on subfigure 3a on the second page. As shown, markers can be combined to form hierarchic structures with no depth restrictions; this should facilitate the specification of precise, unambiguous locations.



The *specifiers* that accompany markers can have one of the following forms:

- Single: “*P.1*” (first page) or “*S.A*” (section A).
- Multiple: “*F.1,3,4a*” (figures one and three, and subfigure 4a).
- Sub-locations: “*S.1.1*” (subsection 1.1). Note that in this case the full stops have different purposes, depending on their positions: the first full stop will always separate markers from specifiers, while further ones will signal sub-locations.
- Bibliographic: “*R.[Doe08]*”; the characters between square brackets identify a bibliographic reference, and multiple such locations can be specified using commas ( “*R.[Doe08],[2]*”).

Markers and specifiers can be combined in any manner; for example, “*P.1: S.1.1: F.2,3a,4: equation system is not presumed*” is a valid annotation. The colon (:) is used as a delimiter between a marker-specifier pair and a block of plain text (the body of the annotations), or a new pair, as shown in the hierarchical annotations examples.

This type of annotations can be inserted using the standard Invenio commenting facilities, in two different manners:

- Online, by using the form in Fig. 4.4, which also provides a brief usage tutorial. Moreover, the aggregated view window in Fig. 4.7 can be used to point-and-click on the document previewer pane to add an annotation on the currently previewed page (the input form is pre-filled with the necessary markup).
- Offline, by using the described markup to edit the annotations while online access to the Invenio instance is not possible. Users can write their review, decorate it with the markup and add it later through the online form described above in order to be included with the rest of the comments. This is the reason for the markup being visible to the users, and not implicit, directly encoded in the system by means of, for example, an interface similar to the one in Fig. 4.7.

All the added record comments will be scanned and valid annotations will be extracted by means of a simple regular expression mechanism. From a high level, annotations are considered valid if complying with the following format:

```
^<MARKER 1>.<SPECIFIER 1>: [<MARKER 2>.<SPECIFIER 2>: ...
  [<MARKER N>.<SPECIFIER N>:]] <FREE TEXT>$
```

Square brackets denote optional elements, and the ^ and \$ symbols the beginning and end, respectively, of the text line; the line break is inserted purely for formatting reasons.

While the final regular expression used for extracting the annotations is quite complex, it is composed of simple building blocks. For example, the section marker is defined in a single line of code, as shown in Fig. 4.5. Nevertheless, if the regular expression mechanism becomes too complex, or the user requirements change by including new semantic validation rules (e.g., all paragraphs need to be specified in relation to a section), the use of a grammar could be considered<sup>1</sup>.

---

<sup>1</sup>Regular expressions have the same expressive power as regular grammars (most restrictive in the Chomsky hierarchy) in formal language theory.

**Figure 4.4:** Invenio commenting form allowing annotation markup. To add remarks on any document element users, can employ the markup described on the right-hand-side.

---

```

MARKERS = {
    # ...
    'S': {'longname': 'Section', 'regex': r'[S]\.' + LOCATION}
}

```

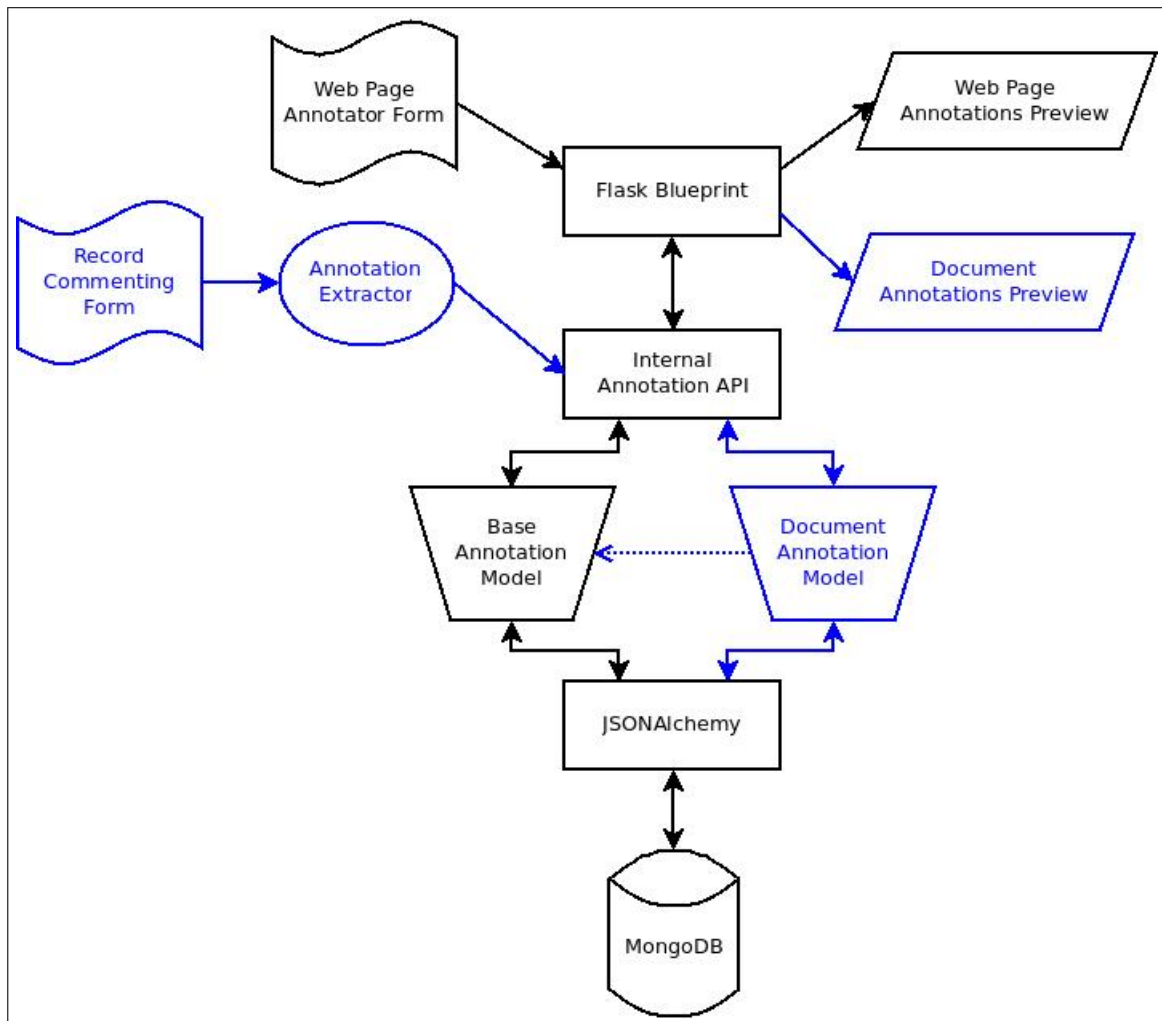
---

**Figure 4.5:** Regular expression marker definition for sections. The regular expression is defined as the marker (*S*), followed by a full stop, followed by list of alpha-numeric characters of length equal or greater than 1 (the global definition of *LOCATION* specifiers is currently used for all markers except citations).

Extracted annotations will be saved in a JSON document complying with the JSON-Alchemy template, in the MongoDB database. It is worth noting that these annotations are stored along with the ones described Section 4.1 in the same *collection* (equivalent of a SQL table); this is made possible by the fact that the database accepts dynamic schemas and not rigid, pre-defined ones.

To summarise, the model for targeted document annotations is similar to the base one presented in Section 4.1, except for the “*where*” field which is composed of the record identifier and document marker(s), and a new field containing the identifier of the comment in which the annotation originated (used for providing context to end-users). A remark here is that in order to use more standard IRI locators for the target, as in the base model, a scheme that allows navigating to a preview of the specified document location could be used. For example, `/record/1/annotations/page1-paragraph2/` could be employed to open a previewer highlighting the specified text region. This is currently implemented only for page numbers, due to technical limitations of the document previewer solution.

For completeness, the workflow of document annotations is included in Fig. 4.6.



**Figure 4.6:** Document annotation workflow; the base annotation model is extended to allow adding notes on specific document locations (e.g., page, figure, equation). Annotations are extracted from basic Invenio comments, by identifying the special markup that can be employed by users in order to refer to locations.

Annotations can be previewed in a manner similar to the one presented in Fig. 4.7. The current version proposes a two-column window, with one column dedicated to previewing the concerned document (currently the supported format is PDF), and the other to displaying annotations.

Coming back to the CDS review process use-case, the annotation display has been designed in a manner addressing two issues:

1. Separate targeted annotations from free-text comments: even if a comment includes both markup decorated and non-structured content, the aggregated view will display only the particular annotations. Considering the following comment:

Dear authors,

This is a very good paper, but I have the following remark:

P.2: F.3a: equation system is not presumed

Best,

only the remark on subfigure 3a on second page will be displayed. Complete comments can still be accessed in the standard Invenio manner.

2. Aggregate remarks by location, such that the authors in charge of amending the paper can systematically follow them. As can be observed in Fig. 4.7, annotations are first grouped by page and then hierarchically, in a top-down manner, from the most general marker to the most particular one.

In order to obtain the aggregated view, two operations need to be applied. First, a query for retrieving the annotations on the requested document and page is issued to MongoDB; while the per-page view is the only one currently supported by the PDF viewer, more granular queries are also supported (e.g., get all annotations on line three of table four).

Secondly, the annotations need to be grouped by their common markers. This is achieved by building a tree structure, in which the leaf vertices contain the annotation objects, and others the various marker-location pairs; the root contains the page marker. For example, consider the following annotations:

- P.1: S.2: F.3: this figure is too small
- P.1: S.2: F.4: this figure is too large
- P.1: S.2: E.3: equation system is not presumed
- P.1: L.23: typo?

The tree in Fig. 4.8 is built for aggregating them. By applying a depth-first traversal, a view similar to the one in Fig. 4.7 can be presented to users.

**Annotations**

« <

Page 5 of 14

Click to add an annotation on the current page

«

»

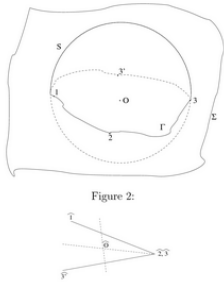


Figure 2:

Figure 3:

$\Sigma$  the curve  $\Gamma$ . (Fig. 2).

From now on, we shall assume that, with respect to some reference "horizontal" plane the slope of the surface does not exceed  $\Theta_M$ , i.e. given  $P$  and  $Q$  on  $\Sigma$ :

$$\left( \frac{\partial \Sigma}{\partial x_i}, \vec{n} \right) \geq \cos \Theta_M, \quad (1)$$

where  $\vec{n}$  is the perpendicular to the horizontal plane. We assume further that at any given point there is a tangent plane to the ground.

**Annotations on Page 5**

Praesent velit velit, consectetur nec mattis ut, dapibus id metus.

— [juliet](#) - 2014-02-27 18:34:24 - [reply to original comment](#)

Phasellus mattis sodales nisl, quis euismod diam consectetur sit amet.

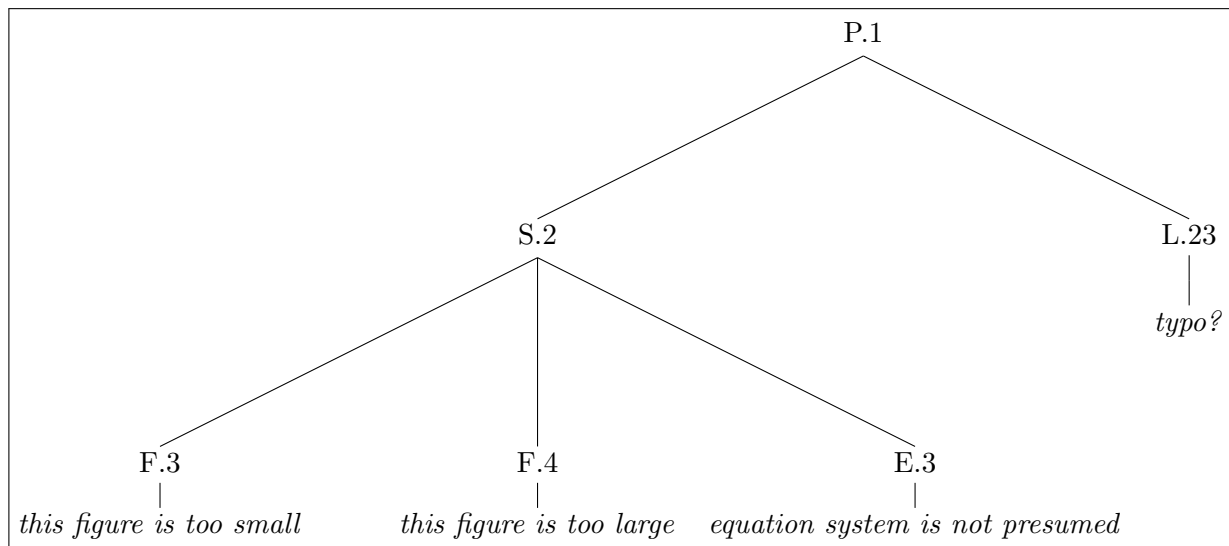
— [benvolio](#) - 2014-02-27 18:34:24 - [reply to original comment](#)

**Annotations on Equation 1**

Quisque vehicula tempor tempor. Vestibulum tempor posuere felis in interdum. Donec quis lorem id felis mattis fermentum.

— [romeo](#) - 2014-02-27 18:34:24 - [reply to original comment](#)

**Figure 4.7:** Document annotation previewer. A PDF document preview is included on the left-hand-side, while the annotations on the currently displayed page are presented on the right-hand-side; annotations are furthermore aggregated, depending on their sub-markers (e.g., Equation 1). Clicking on the previewer pane allows adding a new annotation targeting the displayed page.



**Figure 4.8:** Tree used for annotation aggregation. By traversing the structure in a depth-first manner, annotations can be aggregated by their common markers and location specifiers.

## 4.3 Dissemination

As mentioned in Subsection 2.2.1, an important capability of most annotation systems is metadata dissemination. An example use case, related to the targeted annotation feature, is the routine employed by certain high energy physics collaborations, which use both CDS and an external application (TWiki) for their paper review process. Thus, the possibility of transferring data between applications in a standardised manner could simplify both the users' workflow and the development process for application providers.

An API that allows exporting annotation data from the Invenio platform has been implemented and will be described in Subsection 4.3.2. The system uses the Open Annotation data model for creating standardised RDF graphs describing annotations, and exports them using the JSON-LD serialisation format; a brief description of JSON-LD follows.

### 4.3.1 JavaScript Object Notation for Linked Data

As JSON is a widely used format for transferring Web data, it seems only natural that a method for specifying linked constructs using it to be proposed. This is what JSON-LD achieves by allowing to encode RDF graphs in JSON documents [SLK<sup>+</sup>14].

For this to be possible, the specification defines a number of keywords, all preceded by the “@” token; the most commonly used ones are:

- **@context**: used to define the keys and possibly values employed inside the JSON document; for example, if JSON is used to describe a person identified by a name, the context could specify that the “*name*” field has the meaning as stipulated in a standard ontology (e.g., Friend of a friend (FOAF) [BM14]). The context is itself a JSON object, where the keys are the various elements used inside the main document and the values are identifiers (e.g., IRI of an ontology).
- **@id**: used to uniquely identify JSON elements, usually by means of an IRI; coming back to the above example, the person could be uniquely identified by a homepage such as `http://example.com/JohnDoe/`.
- **@type**: used to specify the data type of values (or more generally of RDF nodes); for example, person names have string type.
- **@value**: used to specify the actual data associated with a property in the RDF graph. In the example above, the “*name*” property could have the value “*John Doe*”.

A JSON-LD document example is included in Fig. 4.9. To transform such a representation to a RDF graph, three simple operations need to be applied<sup>2</sup>:

1. Expansion: this simply removes the context by replacing the keys inside it with the values (usually IRI) across the main object.
2. Flattening: converts the JSON object to a list of RDF graph nodes. Basically, it transforms the graph JSON structure into a flat *deterministic* one, in which all properties of a node (identified by `@id`) are collected in a single JSON object. The resulting graph is represented as a list, the flat structure being easier to parse in an automated manner. Figure 4.10 provides the flattened version of the document in Fig. 4.9.
3. Conversion: from the flattened form to RDF triples. The result of this final step on the document in Fig. 4.9 is included in Fig. 4.11.

---

```
{
  "@context": {
    "name": "http://xmlns.com/foaf/spec/#term_name",
    "Person": "http://xmlns.com/foaf/spec/#term_Person",
    "dc": "http://purl.org/dc/dcmitype/",
    "knows": "http://xmlns.com/foaf/spec/#term_knows"
  },
  "@id": "http://example.com/JohnDoe",
  "@type": "Person",
  "name": {
    "@type": "dc:Text",
    "@value": "John Doe"
  },
  "knows": [
    {
      "@id": "http://example.com/JaneDoe",
      "@type": "Person",
      "name": {
        "@type": "dc:Text",
        "@value": "Jane Doe"
      }
    }
  ]
}
```

---

**Figure 4.9:** JSON-LD document describing a person. The context defines the non-keyword elements used within the document, in order to provide information to the JSON-LD consumers regarding the semantic meaning of data.

---

<sup>2</sup>The complete algorithm is described formally in the documentation.

---

```

{
  "@graph": [
    {
      "@id": "http://example.com/JohnDoe",
      "@type": "http://xmlns.com/foaf/spec/#term_Person",
      "http://xmlns.com/foaf/spec/#term_knows": {
        "@id": "http://example.com/JaneDoe"
      },
      "http://xmlns.com/foaf/spec/#term_name": {
        "@type": "http://purl.org/dc/dcmitype/Text",
        "@value": "John Doe"
      }
    },
    {
      "@id": "http://example.com/JaneDoe",
      "@type": "http://xmlns.com/foaf/spec/#term_Person",
      "http://xmlns.com/foaf/spec/#term_name": {
        "@type": "http://purl.org/dc/dcmitype/Text",
        "@value": "Jane Doe"
      }
    }
  ]
}

```

---

**Figure 4.10:** Expanded and subsequently flattened version of the JSON-LD document in Fig. 4.9. The terms which are not keywords in the JSON-LD vocabulary are replaced by their values from the original context, and the properties of each RDF graph node are collected in a single object. Therefore, the graph (**@graph**) will contain two nodes, one for each person.

Apart from being able to generate RDF graphs, this representation can be used with another purpose, namely to obtain translated versions of the same data. Consider a simple example in which a system supplies information about its users in a format similar to the one in Fig. 4.9. Another application wishes to use this information, but due to its internal workflow, would need to obtain a JSON document in which the “*name*” property identifiers are replaced by “*full-name*”. Clearly, this could be achieved by retrieving the JSON document from the first system and modifying the fields, however, JSON-LD API implementations, such as PyLD [Dig11], provide another method, which basically can move the translation step from consumer- to producer- side. This method requires supplying another context when retrieving the expanded (or flattened) JSON document, which specifies the required translations; coming back to the above example, such a context can be:

```
"@context": {"full-name": "http://xmlns.com/foaf/spec/#term_name"}
```

Note that in both this context and the original one (Fig. 4.9), the *name* and *full-name* keys, respectively, refer to the same ontology IRI. Retrieving an expanded JSON-LD through this method, with a context supplied, is a *compaction* operation; the result will be a new



---

```

<http://example.com/JohnDoe>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/spec/#term_Person> .

<http://example.com/JohnDoe>
<http://xmlns.com/foaf/spec/#term_name>
"John Doe"^^<http://purl.org/dc/dcmitype/Text> .

<http://example.com/JohnDoe>
<http://xmlns.com/foaf/spec/#term_knows>
<http://example.com/JaneDoe> .

<http://example.com/JaneDoe>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/spec/#term_Person> .

<http://example.com/JaneDoe>
<http://xmlns.com/foaf/spec/#term_name>
"Jane Doe"^^<http://purl.org/dc/dcmitype/Text> .

```

---

**Figure 4.11:** Information extracted from the JSON-LD document in Fig. 4.9 in RDF N-Quads format. The five statements, ended by full stop, specify that the resource identified by the IRI `http://example.com/JohnDoe/` is of type *“person”*, with the meaning defined in the FOAF ontology, and has the name *“John Doe”*, of type text (string) as defined in the Dublin Core Metadata Element Set (see *“type”* term in [Dub12]). Moreover, this person has an acquaintance, *“Jane Doe”*, described similarly.

JSON document, in which the translations specified in the new context will be applied (*“name”* is replaced by *“full-name”*). If no new context is supplied, the expanded JSON-LD document is retrieved. This should be the preferred communication method between applications interchanging JSON-LD data. It is interesting to note that the specification allows sending an IRI to a new context through an HTTP Link Header [Not10], thus eliminating the need for sending large JSON payloads.

While the above examples are trivial, more complex modelling and dissemination operations can be achieved using JSON-LD. The next subsection gives an overview on how this technology was employed for publicising annotations.

### 4.3.2 Annotation Publishing

In order to disseminate annotation data in a standardised manner two issues need to be considered:

1. Format annotations using an established specification.
2. Provide an easy to use mechanism for data distribution.

Regarding the data format, we have chosen the Open Annotation data model, which recommends using a JSON-LD RDF serialisation for making annotations public [Wor13]. Basically, the specification provides a context, in which the keys are the various OA model components, and the values IRI to various ontologies, such as FOAF, Dublin Core Metadata Element Set, RDF, or the specific OA vocabulary.

This context can be used by applications to convert their own internal annotation representations to a standard JSON-LD document; if the internal model is already in JSON form, this can be achieved directly by compaction or even by modifying the context keys to match to internal ones.

Despite employing a JSON model in our implementation, we have chosen to programmatically apply the processing steps. Namely, a thin interface over JSONAlchemy was implemented, allowing objects stored in JSON form to specify their custom translation method. This method uses a default context, but custom ones can also be supplied, given that the object class has the required flexibility.

For the annotations, the method will use the OA context and apply a number of simple conversions. Let us consider the targeted annotations use-case, described in Section 4.2, for which the conversions are as follows:

- The main identifier of the resource is an IRI pointing to an endpoint of the Invenio installation, which can supply annotations in the internal JSON format; this should not be frequently used, but is included for completeness. The type of the resource is `Annotation` in the OA vocabulary.
- The OA `annotatedBy` contains information regarding the user that created the annotation. Recall that, internally, we store only an identifier, which can be used to retrieve all other associated information, such as the name or email address. If, instead of applying the explicit conversion process, the JSON-LD compaction routine is used, the mentioned information needs to be directly included in the annotation JSON model, resulting in duplication of data. Another solution would be to provide only minimal information regarding users, namely an IRI pointing to a complete profile<sup>3</sup>.
- For the `annotatedAt` field, the time stamp is provided as-is. A note here is that, as data producers have no knowledge regarding their consumers, time zone information, usually as a value denoting the offset from the Coordinated Universal Time (UTC), needs to be included.
- For the `hasBody` field, the annotation content is provided, along with its format (`format/plain`) and encoding (Universal Transformation Formats-8-bit (UTF-8)).

---

<sup>3</sup>This is currently not available in Invenio.

- The `hasTarget` field is encoded as the IRI to the record of the document being annotated, along with an `OA FragmentSelector` data type instance, which specifies to what part of the document the annotation refers (a marker such as “*second paragraph on third page*”, encoded in the manner described in Section 4.2). Note that if a method of accessing document fragments directly through a valid IRI is provided, specifying the selector is no longer necessary.

For annotations on Invenio Web pages the process is similar, except for the target field, which can be identified uniquely by an IRI.

The end result of the translation operation is a JSON document which, apart from the fields described above, also contains the entire context definitions; this is denoted internally as the “*inline*”<sup>4</sup> or “*full*” format. Other formats, such as the expanded or RDF/ N-Quads ones, can be obtained by simple calls to the PyLD API [Dig11], accessible through the JSONAlchemy wrapper. An example of an expanded JSON-LD representation of a document annotation is included in Fig. 4.12.

Annotation data is served to any interested third-party through a REST Web interface. At an abstract level, REST is an architectural style in which the internal system details are ignored, the focus being on the component roles, the constraints on their interactions, and on the interpretation on significant data elements.

In terms of Web applications, the REST architecture defines a number of formal constraints on the client–server communication protocols:

- Servers and clients are separated by a uniform interface, each having its own distinct set of functions; for example, only servers are concerned with data storage, and user interfaces are handled only by clients.
- Communication is stateless, no client information being stored in the server between requests; this requires clients to supply all necessary data with each request.
- Server responses may be cached and load-balancing may be employed without disrupting clients or corrupting data.

---

<sup>4</sup>As per Open Annotation specification.

---

```

{
  "http://www.w3.org/ns/oa#hasBody": {
    "http://purl.org/dc/elements/1.1/format": "text/plain",
    "http://www.w3.org/2011/content#characterEncoding": "utf-8",
    "@id": "http://www.w3.org/ns/oa#hasBody",
    "@type": [
      "http://www.w3.org/2011/content#ContentAsText",
      "http://purl.org/dc/dcmitype/Text"
    ],
    "http://www.w3.org/2011/content#chars": "Aenean vel gravida [...]"
  },
  "http://www.w3.org/ns/oa#motivatedBy": {
    "@id": "http://www.w3.org/ns/oa#commenting"
  },
  "http://www.w3.org/ns/oa#hasTarget": {
    "http://www.w3.org/ns/oa#hasSelector": {
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "P.2-PP.3",
      "@id": "http://www.w3.org/ns/oa#hasSelector",
      "@type": "http://www.w3.org/ns/oa#FragmentSelector",
      "http://purl.org/dc/terms/conformsTo":
        "http://labs.invenio-software.org/api/annotations/notes_specification"
    },
    "@id": "http://www.w3.org/ns/oa#hasTarget",
    "@type": "http://www.w3.org/ns/oa#SpecificResource",
    "http://www.w3.org/ns/oa#hasSource": {
      "@id": "http://labs.invenio-software.org/record/1"
    }
  },
  "http://www.w3.org/ns/oa#annotatedAt": "2014-03-10T08:50:01+00:00",
  "http://www.w3.org/ns/oa#annotatedBy": {
    "http://xmlns.com/foaf/0.1/name": "dorian",
    "http://xmlns.com/foaf/0.1/mbox": {
      "@id": "mailto:dorian.gray@cds.cern.ch"
    },
    "@id": "http://labs.invenio-software.org/api/accounts/account/?id=4",
    "@type": "http://xmlns.com/foaf/0.1/Person"
  },
  "@id": "http://labs.invenio-software.org/api/annotations/export/?_id=9935ae97-657f-4fc1-970f-f1cbcab65c68",
  "@type": "http://www.w3.org/ns/oa#Annotation"
}

```

---

**Figure 4.12:** Annotation serialised as an expanded JSON-LD document. The internal representation is converted to an OA compliant one, using the standard vocabulary.

The uniform server–client interface is the basis of the REST architecture and can be described in terms of guidelines targeting four aspects:

1. Resource distinguishability: resources are uniquely identified (e.g., in Web applications by IRI) and separated from their internal structure (e.g., an object stored in an SQL database will be served to Web clients in an XML or JSON format).
2. Resource manipulation: given an object representation, a client holds enough information necessary to perform any editing or deletion action over the object.
3. Message description: request responses need to be self-descriptive, specifying the actions clients need to undertake in order to process them. For example, in the case of Web applications, the format of the message (e.g, XML) needs to be included alongside the server response.
4. Hypermedia as the Engine of Application State (HATEOAS): apart from a limited set of simple endpoints, clients do not make any assumptions regarding the possible server actions, but only follow the hyperlinks provided in request responses. For most Web REST interfaces, a brief specification defining the endpoints is provided, and clients cannot step outside these boundaries, except when receiving explicit instructions from the server.

As previously mentioned, the new version of Invenio provides the basic building blocks for REST interfaces. Each module that desires to implement such a facility simply needs to define the endpoints, the methods clients can use to retrieve data (e.g., `GET` or `POST`), and the request template.

For the annotation use-case, one endpoint (`/api/annotations/export/`), accepting two request methods (`GET` and `POST`) is provided. Through the `GET` method, clients can get annotation data in a format similar to the internal one, this being mostly used to build the IRI supplied as the main object’s `@id` in the JSON-LD representation (see Fig. 4.12). The parameters used for this type of request can be used to perform simple annotation queries; for example, to retrieve all annotations targeting the first page of any document a request to `api/annotations/export/?where.marker=P.1` can be performed.

More complex queries, along with requests to JSON-LD serialisations can be performed using the `POST` method. This allows clients to specify the request parameters in a JSON document in which three parameters are of interest:

1. **query**: used to filter the retrieved annotations; the format is as defined by MongoDB for queries [Mon09].
2. **ldexport**: the JSON-LD format to use for serialisation. Can be full, expanded, compacted, flattened, framed (having a custom tree structure, see [LSK<sup>+</sup>13]), or normalised (RDF triples).
3. **new\_context**: the new context to use for compaction, the tree structure for framing, or the normalisation options (e.g., `{“format” : “application/nquads”}`).

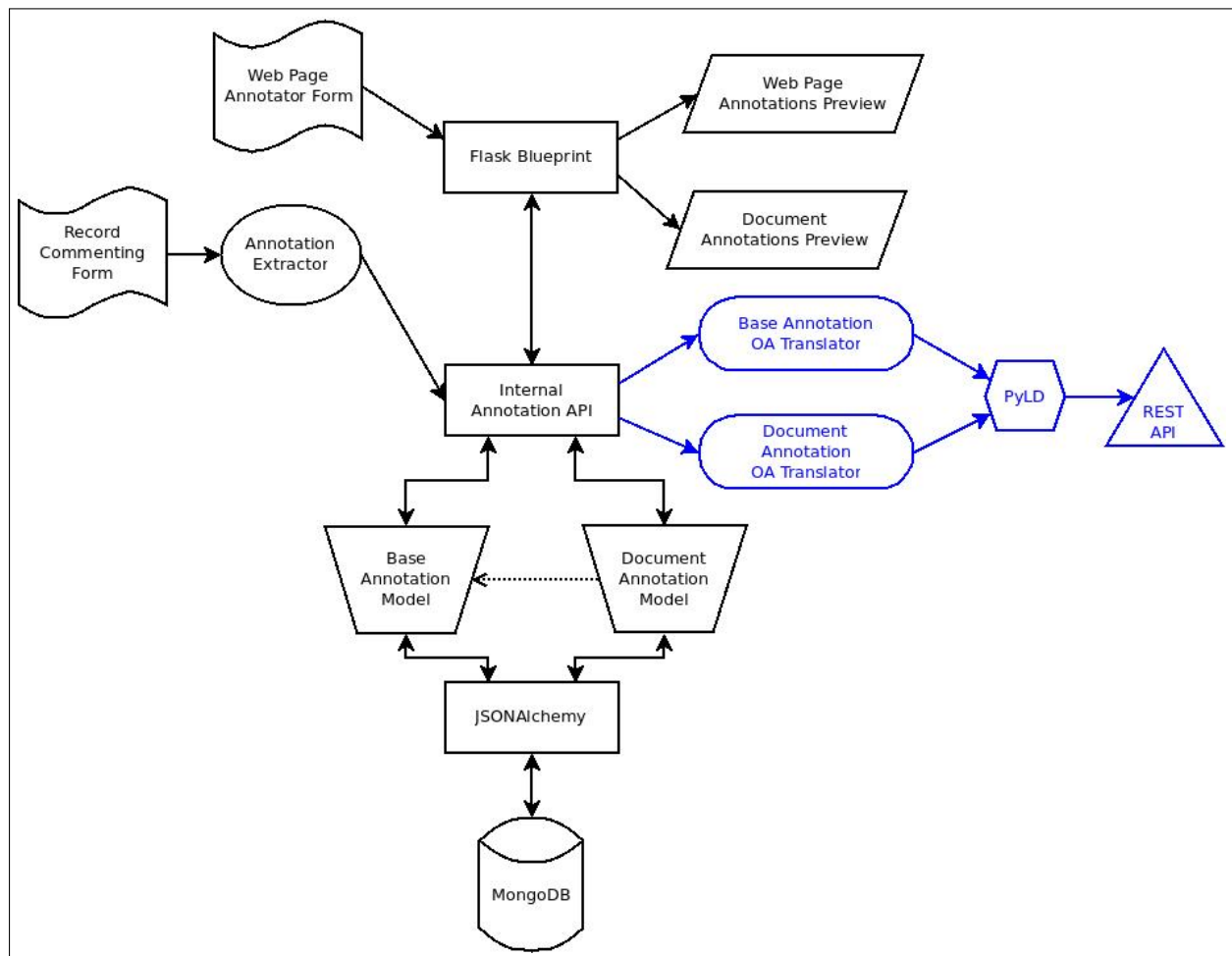
To retrieve the document in Fig. 4.12, the following query can be used:

```
curl labs.invenio-software.org/api/annotations/export/ \
-H "Content-Type: application/json" \
--data '{"query": {"where.marker": "P.2-PP.3"}, \
      "ldexport": "compacted", \
      "new_context": {}}'
```

Recall that JSON-LD expansion and compaction with an empty context are similar operations.

The proposed mechanism closely follows the REST guidelines: only a minimal set of simple, unique, endpoints is provided, clients being allowed to make only the state transitions specified in the responses (e.g., the main annotation IRI identifier), responses contain all the information required to perform any operations over existing annotations, and all the messages from the server contain processing information (the JSON format is advertised inside headers). Moreover, the communication is stateless and responses can be cached as they do not depend on client parameters.

To sum up, Fig. 4.13 provides an overview of the Invenio annotation system, starting from the end-user interface, through the internal API, and ending with the REST dissemination service.



**Figure 4.13:** Annotation dissemination workflow; after being translated to a proper standardised format (Open Annotation JSON-LD) and processed by PyLD, user-added annotations can be served by the Invenio REST API to concerned third-parties.

## 4.4 Future Directions

The implemented elements presented previously provide a strong basis upon which future related projects can build upon, both in order to improve and extend the existing annotation facilities.

In terms of the general framework, more of the currently existing Invenio features for capturing user-side metadata (comments, reviews, tags, baskets) can be adapted to using the new facilities in order to validate the design and backend implementation details. Moreover, new facilities, such as file attachments, decorated formatting, integration with Invenio's search engine, full support for the access control system, or multimedia annotation capabilities may be considered.

Regarding document targeted annotations, a more robust method of displaying the concerned files might be of interest. For PDF documents, Mozilla's PDF.js appears to be a viable solution, as long as a method of rendering complex files on server-side is implemented, and certain cross-browser compatibility issues are resolved; another option can be Multivio. Such a solution could allow for a more attractive interface, employing which users could, for example, visually highlight text fragments, in the same manner desktop applications allow. Another direction regards the validation of user input; either a solution as XTiger can be applied to restrict input on the user-side, or a more complex grammar can replace the current regular expression backend. Moreover, a richer collection of markers could be considered, covering, for example, the differences between reviews targeting stylistic, grammatical, or factual issues. The end-user feedback gathered from the Invenio Labs demonstrator also showed that aggregation options can be considered for more workflow steps, such as the actions required from authors in order to respond to reviews.

Finally, the REST interface could be employed not only for data output, but also for input; this may allow, for example, to import annotations from other systems. The Invenio framework already supports data input through its REST framework, as implemented, for example, by the ZENODO deposition API. Apart from importing linked Web data, annotations from desktop applications such as Mendeley can be of interest. On a higher level, exporting more Invenio resource types, such as user accounts or record metadata, to JSON-LD compliant documents, through the JSONAlchemy API, may provide a consolidated method of producing linked data.



## 5. Conclusions

The aim of this project was to develop a structured annotation facility, the original use-case being identified in the scientific paper pre-publication workflow employed by high energy physics collaborations. Apart from being able to comment at the global document level, collaboration members also require more granular options, such as section, paragraph, figure, or equation targeted annotations. Moreover, aggregating such reviews should be possible in a facile manner, which would simplify the workflow for the authors in charge of amending papers.

After studying a number of use cases and state of the art solutions and related technologies, a general annotation framework for the Invenio digital library platform was designed and implemented. The aim was to provide a standardised method of allowing input of end-user generated metadata regardless of its form: document comments, record collections and tags, or Web page annotations.

The targeted annotator facility was implemented along with a document previewer which facilitates both data input and aggregation for end-users. Furthermore, a method of disseminating annotations, in line with the emerging Open Annotation RDF data model, was implemented. It makes use of modern Web technologies such as the JSON-LD serialisation format and REST services, in order to bestow a streamlined circulation workflow for any third-party interested in accessing user-generated metadata.

The annotation facilities were developed in the context of the new Invenio version, to which the project brought a number of additional contributions. These include fixes and additions to several components of the digital library platform.

The document targeted annotator was released for end-user testing in a controlled environment, the feedback being highly positive. The general consensus is that the new aggregation features significantly simplify the pre-publication review process; future improvements, such as the ones presented in Section 4.4, might improve the workflow even further.



# Bibliography

- [ABMH13] Ben Adida, Mark Birbeck, Shane McCarron, and Ivan Herman. RDFa Core 1.1 - Second Edition. Community draft, World Wide Web Consortium (W3C), August 2013. Electronic record available at <http://w3.org/TR/rdfa-syntax>.
- [Ado14] Adobe Systems, Inc. Adobe Reader XI. <http://adobe.com/products/reader.html>, 2014.
- [BM14] Dan Brickley and Libby Miller. FOAF Vocabulary Specification 0.99. <http://xmlns.com/foaf/spec>, January 2014. Paddington Edition.
- [Dig11] Digital Bazaar. PyLD. <http://github.com/digitalbazaar/pyld>, July 2011.
- [Dis07] Disqus, Inc. Disqus. <http://disqus.com>, October 2007.
- [Dub12] Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, Version 1.1. <http://dublincore.org/documents/dces>, July 2012.
- [Ela10] Elasticsearch BV. Elasticsearch. <http://elasticsearch.org>, February 2010.
- [Eur] European Organisation for Nuclear Research (CERN). The ATLAS Collaboration. <http://home.web.cern.ch/about/experiments/atlas>.
- [Eur02] European Organisation for Nuclear Research (CERN). The CERN Document Server (CDS). <http://cds.cern.ch>, 2002.
- [Gal11] Andreas Gal. PDF.js. <http://mozilla.github.io/pdf.js>, April 2011.
- [GBMH<sup>+</sup>08] Anne Gentil-Beccot, Salvatore Mele, Annette Holtkamp, Heath B. O’Connell, and Travis C. Brooks. Information Resources in High-Energy Physics: Surveying the Present Landscape and Charting the Future Course. *Journal of the American Society for Information Science and Technology*, 60(arXiv:0804.2701. CERN-OPEN-2008-010. DESY-08-040. DESY-2008-040. FERMILAB-PUB-08-077-BSS. SLAC-PUB-13199. 1):150–160, April 2008. Electronic record available at <http://cds.cern.ch/record/1099955>.
- [GILMv13] Patrick Oliver Glauner, Jan Iwaszkiewicz, Jean-Yves Le Meur, and Tibor Šimko. Use of Solr and Xapian in the Invenio document repository software. In *Open Repositories 2013*, Charlottetown, Prince Edward Island, Canada, July 2013.

- [GMN] Marco Grassi, Christian Morbidoni, and Michele Nucci. A Collaborative Video Annotation System Based on Semantic Web Technologies. *Cognitive Computation*, 4:497–514.
- [GMN<sup>+</sup>12] Marco Grassi, Christian Morbidoni, Michele Nucci, Simone Fonda, and Giovanni Ledda. Pundit: Semantically Structured Annotations for Web Contents and Digital Libraries. In Annett Mitschick, Fernando Loizides, Livia Predoiu, Andreas Nürnberger, and Seamus Ross, editors, *Proceedings of the Second International Workshop on Semantic Digital Archives (SDA 2012)*, volume 912, pages 49–60, Paphos, Cyprus, September 2012. Electronic record available at <http://ceur-ws.org/Vol-912/paper4.pdf>.
- [GS14] Fabien Gandon and Guus Schreiber. RDF 1.1 XML Syntax. W3c recommendation, World Wide Web Consortium (W3C), February 2014. Electronic record available at <http://w3.org/TR/2014/REC-rdf-syntax-grammar-20140225>.
- [Inv02] Invenio Collaboration. Invenio. <http://invenio-software.org>, 2002.
- [Kap10] Samuele Kaplun. Invenio: A Modern Digital Library System. In *Open Repositories 2010*, Madrid, Spain, July 2010.
- [KKPS02] Jos Kahan, Marja-Riita Koivunen, Eric Prud’Hommeaux, and Ralph Swick. Annotea: An open RDF infrastructure for shared Web annotations. *Computer Networks*, 39(5):589–608, August 2002.
- [Kna03] Frederico Knabben. CKEditor. <http://ckeditor.com>, March 2003.
- [Lib99] Library of Congress, Network Development and MARC Standards Office. MARC 21 Format for Bibliographic Data. <http://www.loc.gov/marc/bibliographic/>, 1999.
- [LSK<sup>+</sup>13] Dave Longley, Manu Sporny, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. JSON-LD Framing 1.0. W3C Community Group Draft Report, World Wide Web Consortium (W3C), March 2013. Electronic record available at <http://json-ld.org/spec/latest/json-ld-framing/>.
- [LVdSNW08] Carl Lagoze, Herbert Van de Sompel, Michael Nelson, and Simeon Warner. *The Open Archives Initiative Protocol for Metadata Harvesting*. Open Access Initiative Executive, 2008-12-07T20:42:00Z edition, July 2008. Electronic record available at <http://www.openarchives.org/OAI/openarchivesprotocol.html>.
- [Mar09] Ludmila Marian. Ranking Scientific Publications Based on Their Citation Graph, April 2009. Electronic record available at <http://cds.cern.ch/record/1172366>.
- [Mar12] Ludmila Marian. The Workflow of LHC Papers. Computing in High Energy and Nuclear Physics (CHEP), May 2012. Electronic record available at <https://cds.cern.ch/record/1460578>.

- [MED10] MEDIA Research Group, École Polytechnique Fédérale de Lausanne. XTiger XML Language Specification. <http://media.epfl.ch/Templates/XTiger-XML-spec.html>, January 2010.
- [MED11] MEDIA Research Group, École Polytechnique Fédérale de Lausanne. Adaptable XML Editing Library (AXEL). <http://media.epfl.ch/Templates>, January 2011.
- [Men08] Mendeley, Ltd. Mendeley. <http://mendeley.com>, 2008.
- [MM09] Miguel Moreira and Johnny Mariéthoz. Multivio: Generic browser and visualizer for digital objects. <http://multivio.org>, November 2009.
- [Mon09] MongoDB, Inc. Mongoddb. <http://mongodb.org>, 2009.
- [Not10] Mark Nottingham. Web Linking. Request for Comments, Requests Track ISSN-2070-1721, Internet Engineering Task Force (IETF), October 2010. Electronic record available at <http://ietf.org/rfc/rfc5988.txt>.
- [Ope09] Open Knowledge Foundation. Annotator. <http://okfn.org/projects/annotator>, July 2009.
- [OT11] Mark Otto and Jacob Thornton. Bootstrap. <http://getbootstrap.com>, August 2011.
- [PBG<sup>+</sup>05] Alberto Pepe, Thomas Baron, Maja Gracco, Jean-Yves Le Meur, Nicholas Robinson, Tibor Šimko, and Martin Vesely. CERN Document Server Software: the integrated digital library. (CERN-OPEN-2005-018):297–302, April 2005. Electronic record available at <http://cds.cern.ch/record/853565>.
- [PS08] Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. W3c recommendation, World Wide Web Consortium (W3C), January 2008. Electronic record available at <http://w3.org/TR/rdf-sparql-query>.
- [QRSV10] Vincent Quint, Cécile Roisin, Stéphane Sire, and Christine Vanoirbeek. From Templates to Schemas: Bridging the Gap Between Free Editing and Safe Data Processing. In *Proceedings of the 10th ACM Symposium on Document Engineering*, number ISBN:978-1-4503-0231-9 in DocEng'10, pages 61–64, New York, NY, USA, 2010. Association for Computing Machinery (ACM).
- [Ron10] Armin Ronacher. Flask. <http://flask.pocoo.org>, April 2010.
- [San09] Salvatore Sanfilippo. Redis. <http://redis.io>, April 2009.
- [SLK<sup>+</sup>14] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. JSON-LD 1.0: A JSON-based Serialization for Linked Data. W3c recommendation, World Wide Web Consortium (W3C), January 2014. Electronic record available at <http://w3.org/TR/json-ld>.
- [Sol07] Ask Solem. Celery: Distributed Task Queue. <http://celeryproject.org>, 2007.

- [SQL06] SQLAlchemy Authors and Contributors. SQLAlchemy: The Python SQL Toolkit and Object Relational Mapper. <http://sqlalchemy.org>, February 2006.
- [SVdS10] Robert Sanderson and Herbert Van de Sompel. Open Annotation Alpha3 Example: Hubble Deep Field Image. Technical report, Open Annotation Collaboration, October 2010. Electronic record available at <http://openannotation.org/spec/alpha3/examples/hubble.html>.
- [Wor96] World Wide Web Consortium (W3C), French Institute for Research in Computer Science and Automation (INRIA). Amaya. <http://w3.org/Amaya>, July 1996.
- [Wor04] World Wide Web Consortium (W3C). Resource Description Framework (RDF). <http://w3.org/RDF>, February 2004.
- [Wor13] World Wide Web Consortium (W3C). Open Annotation Data Model. <http://openannotation.org/spec/core>, February 2013. Community Draft.
- [WZY06] Xian Wu, Lei Zhang, and Yong Yu. Exploring Social Annotations for the Semantic Web. In *Proceedings of the 15th International Conference on World Wide Web*, number ISBN-1-59593-323-9 in WWW '06, pages 417–426, New York, NY, USA, 2006. Association for Computing Machinery (ACM).

# Appendices





## A Open Repositories and AAHEP7 Submissions

**OR'2014** This appendix contains the paper submitted to the 9<sup>th</sup> International Conference on Open Repositories (OR'2014), taking place in Helsinki, Finland. It is the leading conference in its field, with hundreds of participants from all around the world. The website of the conference is <http://or2014.helsinki.fi/>.

Note that this paper was authored approximately two months before the end of the project, and thus, certain details might be missing or inaccurate in regards to the final state of the deliverable.

Paper follows on next page, the length being of three pages.

**AAHEP7** A presentation regarding the annotation developments in Invenio will be given by Dr. Tibor Šimko at the 7<sup>th</sup> Summit of Information Providers in Astronomy, Astrophysics and High Energy Physics. This summit is attended by representatives of organisations providing access to information in these fields, such as INSPIRE, ADS or arXiv, and major publishers such as the American Institute of Physics (AIP), American Physical Society (APS), Elsevier, Institute of Physics (IOP) or Springer. It will take place at the Stony Brook University, Stony Brook, New York, United States of America, between 1–3 April, 2014. The event's website is <http://indico.cern.ch/event/262430/>.

# Targeted Annotation of Scientific Literature and Data Resources in Invenio Digital Libraries

Adrian-Tudor Pănescu<sup>1</sup>, Tibor Šimko<sup>\*1</sup>, and Christine Vanoirbeek<sup>2</sup>

<sup>1</sup>Department of Information Technology, CERN, Geneva, Switzerland

<sup>2</sup>MEDIA Research Group, EPFL, Lausanne, Switzerland

February 10, 2014

## Abstract

We describe a new targeted annotation framework developed within the Invenio digital library platform in order to answer collaborative annotation needs of high-energy physics user community. After a brief illustration of the most common use case, we describe the new facility and explore its relation to W3C RDF and JSON-LD recommendations as well as to the W3C Open Annotations data model draft standard.

## 1 Use case

The experimental particle physics community, using CERN LHC accelerator to study the basic constituents of matter, consists of multi-national, multi-institute, multi-person collaborations. ALICE, ATLAS, CMS, and LHCb collaborations may comprise up to 3,000 physicists. The publishing workflows of these collaborations include multiple commenting rounds for paper drafts that are restricted within the collaboration before papers become public[1]. The commenting partially happens on an Invenio[2]-powered CERN document server[3].

The structured annotation habits of collaborations include referring to page, line, equation or figure in a free text from. The principal authors of the draft then need to aggregate and process comments left by collaboration members. This task is often non-trivial due to the sheer number of authors and the heterogeneous nature of commenting practices. As a concrete example, one of conference papers received more than thousand annotations in 180 comments coming from 50 persons during its pre-publication stage. (E.g. the most extensive comment contained more than 100 annotations targeted to various parts of the paper draft.)

## 2 Targeted annotation system

In order to answer these targeted annotation needs, a new module was developed for the Invenio digital library platform. The module aims to provide an easy way to capture user annotations in a structured format so that future aggregations can be easily constructed.

When displaying a paper draft, a structured viewer permits to page through the document by showing a page preview on the left-hand-side, while annotations of users for the given page are shown on the right-hand-side, see Figure 1.

Users can enter annotations via so-called “online” and “offline” modes. When online, user can click on a page which brings up a simple structured editor that permits to enter notes for the concrete logical page (P.7), section (S.7), paragraph (PP.7), line (L.7), figure (F.7), table (T.7), equation (E.7), or

---

<sup>\*</sup>Corresponding author.

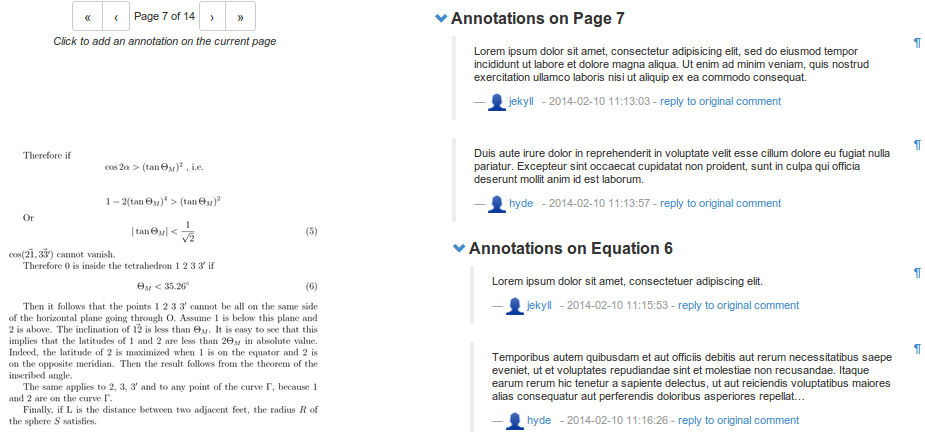


Figure 1: Example of document viewer page (lhs) with aggregated user annotations (rhs).

reference (R.[7]). The leading markers can be nested: for example, marker “P.2: T.3: L.4” refers to line four of the table three on the second page. When offline, users can prepare block of text using the same markers. The text will be then parsed into targeted annotations upon submission.

The framework also permits nested annotations, i.e. to reply to a comment, to annotate an annotation, etc, without any depth restriction.

Finally, the new annotation framework permits to annotate any URI. This makes it possible to use the same commenting and annotating facility to discuss upon any resource handled by the digital library or the application itself, e.g. simple discussion forum on help pages.

It should be noted that similar annotating frameworks exist, such as Annotator [4] or Pundit [5]. While these standalone solutions could cover some of our needs — especially URI-based annotations — our use case usually requires strict restriction of annotations within the given digital library platform. (E.g. collaboration notes on paper drafts remain private even after publication.) The present work therefore attempts at providing an organic evolution of existing commenting and tagging facilities of Invenio towards targeted, structured and linked annotations, all the while providing pluggable interfaces to existing external third-party tools and solutions.

### 3 JSON model and interoperability

The persistence of annotations is achieved by storing an appropriate JSON representation in a document database, namely MongoDB [6]. (Under way is the integration with the PostgreSQL relational database [7] that offers extensive JSON indexing and searching capabilities as of version 9.3.) The schematic JSON document model used for general URI annotations is depicted in Figure 2. The reasons for opting for a JSON representation are two-fold.

The first reason is related to the extendibility of the model; for example, in order to allow for annotations with attachments, a single field specifying the storage location of the files needs to be added to the model described in Figure 2. Coming back to the targeted record commenting use case presented in the previous section, such a specific annotation could be represented by using an extended version of the model, such as the example in Figure 3. (Only specific fields depicted.) All these different types of annotations can be stored homogeneously in a document database, while an SQL representation would likely require distinct tables for each specific annotation type.

The second reason for choosing a JSON representation is related to the possible dissemination of annotations outside the Invenio platform. The Open Annotation Core Data Model [8] is a inter-operable framework proposed by the World Wide Web Consortium (W3C) for creating, connecting, and sharing

```
{
  id: UUID, // annotation identifier
  who: Number, // user identifier
  where: *, // annotation target (e.g. URI, record identifier)
  what: *, // the content and its properties
  when: Timestamp, // annotation creation date and time
  perm: Object // permissions e.g. public, private (group-restricted)
}
```

Figure 2: General JSON model for annotations.

```
{
  where: { URI: "http://cds.cern.ch/record/897121",
    target: "P.2: T.3: L.4" },
  what: { title: "Small correction",
    body: "There is a typo near the word xyzzy." },
  type: "grammatical"
}
```

Figure 3: Example of JSON fields specific for targeted annotations.

associations and annotations across the Web. It uses RDF [9] for modelling and the JSON-LD [10] format is recommended for serialisation. As JSON-LD is a JSON-based format specialized for linked data, this should allow us to easily export annotation data originating on an Invenio platform and use any of the available third-party tools for the semantic study of annotations.

## 4 Conclusions

A new targeted annotation module for Invenio digital library was developed in order to permit structured annotation of scientific literature, as motivated by the needs of high-energy physics experimental collaborations. The module permits “online” and “offline” ingestion of targeted annotations and displays an aggregation of notes around logical units of text and/or data. The captured information is designed to be exportable as JSON-LD or RDF XML.

## References

- [1] L. Marian, *The Workflow of LHC Papers*, Computing in High Energy and Nuclear Physics (CHEP), May 21–25 2012, New York, USA.
- [2] Invenio digital library platform. <http://invenio-software.org/>
- [3] CERN document server. <http://cds.cern.ch/>
- [4] Open Knowledge Foundation, *Annotator*. <http://annotatorjs.org/>
- [5] Net7 srl, *The Pundit*, 2013. <http://www.thepundit.it/>
- [6] MongoDB document database. <http://mongodb.org/>
- [7] PostgreSQL relational database. <http://postgresql.org/>
- [8] The World Wide Web Consortium (W3C), *Open Annotation Data Model*, Community Draft, February 2013. <http://www.openannotation.org/spec/core/>
- [9] The World Wide Web Consortium (W3C) RDF Working Group, *Resource Description Framework (RDF)*, February 2004. <http://www.w3.org/RDF/>
- [10] The World Wide Web Consortium (W3C), *JSON-LD 1.0: A JSON-based Serialization for Linked Data*, Recommendation, January 2014. <http://www.w3.org/TR/json-ld-syntax/>

## B Source Code Contributions

The source code deliverables of the project are as follows:

- The main contributions have already been merged for inclusion in the next version of Invenio, and can be found online at <http://invenio-software.org/repo/personal/invenio-apanescu>. Note that the `pu` (“*Proposed Updates*”) branch refers to the next Invenio version, while the `labs` branch is used for the Labs demonstrator. A total of 79 commits with approximately 6000 lines of code added or modified were recorded by the Git version control system, out of which 3600 lines in 6 commits represent the new annotation facilities. A mirror of this repository can also be found at <http://github.com/adrianp/invenio>.
- The contributions brought to the separate package used for deploying customised Invenio installations can be found at <http://github.com/adrianp/invenio-demosite>. Of interest is the `labs` branch, used for Invenio Labs.
- The Python 2.6 compatible version of the JSON-LD manipulation library, PyLD, used for running the Invenio Labs demonstrator, is available at <http://github.com/adrianp/pyld> (see `python2.6` branch).
- The customised version of the JavaScript library used for the Invenio Labs interactive tutorial, `Intro.js`, can be found at <http://github.com/adrianp/intro.js> (see `inveniolabs` branch). A bug fix has been accepted by the library authors and merged upstream, see <http://github.com/usablica/intro.js/pull/236>.