An Effective Retrieval Approach for Documents related to Past Civil Engineering Projects

Christian Esposito
Department of Electrical Engineering and Information Technology (DIETI)
University of Naples "Federico II"
Napoli, Italy
christian.esposito@unina.it

Oscar Tamburis
Department of Veterinary Medicine and Animal Productions (DMVPA)
University of Naples "Federico II"
Napoli, Italy
oscar.tamburis@unina.it

***Abstract*—During the practice of a public administration or a company, a large volume of documents are typically produced. The recent dematerialization efforts have been pushing the way for a digital form of these documents, so as to make their management more efficient. But, traditional document management systems entail many non-value adding activities such as printing, transport and archiving these documents, making the retrieval of information extremely cumbersome, especially by users without a computer science degree. The research community is putting a lot of attention to realise effective and easy-to-use search capability. This paper investigates such a problem within the context of a document management system for past civil engineering projects, and proposes a solution to offer advanced document retrieval capabilities without demanding a strong background in computer science.**

***Index Terms*—Information Systems, Information Retrieval, Document Management Systems, Inverted Index**

I. Introduction

The dematerialization process undergone in the nineteens has paved the way for the proliferation of informative systems within every aspect of our life and society. Nowadays, this is further boosted by the availability and pervasiveness of Internet, strengthening the exchange of digital documents and further promoting the adoption of ICT solutions within several various application domains. The latest context where such a digital revolution is taking place is the civil engineering, where at the beginning we have witnessed the proliferation of Computer-Aided Drafting (CAD) tools [1] able to substitute manual drawing with electronic drafting when designing building, maintenance system or interior design. The next step has been represented by the arrival of the Building Information Modeling, or BIM [2], which targets the creation and management of digital representations of artefacts and support the plan, design, construction, and maintenance of diverse physical infrastructures. By its definition, such a standard provides architects and civil engineers with a shared knowledge resource for information about any kind of facilities, thus forming a reliable basis for decisions during its life-cycle. At the moment, another revolution in Europe, but also similarly in the rest of the world, is undergoing with the EU Directive 2014/24/EU on public procurement [3], and the relative regulation issued by the State Members, as of the adoption of digital representations in open formats, like BIM, for the design of civil infrastructure and the submission of responses to call-for-tenders by means of certified emails. This is considerably boosting the dematerialization of all civil engineering documents for public procurement and augmenting the volume of digital data held by such a kind of companies.

Offering only storage capabilities of these digital documents is meaningless if it is not coupled with a proper retrieval functionality. In fact, such a large volume of documents can represent an added-value for a company if knowledge can be inferred out of it to support and improve the competitiveness of the company. Specifically, when a call-for-tender is issued by a public authority, the time left for writing a response is extremely limited and can be properly spent if a draft can be quickly obtained by using the past proposals and the lesson learnt from participating to past call-for-tender. Specifically, it should be possible to re-use past similar documents, according to a similar approach put in place for the re-use within the context of software engineering [4]. To this aim, it is extremely important to find all those past documents within the company knowledge base that are somehow related to the call-for-tender of interest and can be exploited to write a first draft of a proposal. However, currently this is not possible as many storage approaches have not been designed for retrieval but only for conservation and historical reasons. Moreover, such documents are not kept at a main server of the company or within the cloud, but at the terminal of each employee so that their existence is not well known by the rest of the company and their accessibility is not guaranteed. Furthermore, current information systems require a solid background on computer science in order to express effective query strings to look for documents in a proficient manner. However, many potential user of such solutions in the context of civil engineering, are not well educated in computer science and not able to use advanced and expressive document retrieval solutions.

Within the context of a research project called PROBIM, a proper data management solution for the documents related to a participation to a call-for-tender of interest has been designed and implemented as a series of RESTful Web Services [5],

so that the explicit use of the HTTP methods can be used to implement the ”create, read, update, and delete” (CRUD) operations for the digital documents for past civil engineering projects or tender responses. We have found, as described in Section II that the best strategy for the storage of such a kind of data set is a Content Management System (CMS) as Alfresco. However, Alfresco support a SQL-like query language, which is not expressive for the document retrieval as it leverage on a term-matching strategy, and is tedious and uneasy to be used by a non computer science degree holder. The documents of our application context of interest are written in natural language, without a previous codification of the terminology, so it is probable to have within the meta-data describing them various synonymous. A term-matching as SQL may lose some document if the term used in the query is singular, but the one in the document description is in its plural form and so on. A more advanced approach in order to have a semantic search by overcoming the obstacles put in the way by the use of singular/plural forms of a term or even their synonymous is demanting, without further exacerbating the learning and use of such an approach by non computer science skilled users. Section II also describes our consideration of such an issue and the proposed solution based on the index of Lucene and the support offered by Solr. On top of Alfresco and Solr, we have built our system, whose design and implementation is detailed in Section III, while Section IV concludes our paper with the sum up on our research conducted so far.

II. Related Works

A series of documents are generated while writing a response to a call-for-tender, and the response itself is composed of multiple documents, where the technical design of a building or a maintenance system is the core document, but there are other additional documents with the economic offer, the work scheduling, the company description, and so on. All of these documents have multiple formats and have been created by different computer applications, spanning from files produced in Microsoft Office (Word is used to write DOCX documents for the administrative part, and Excel used for datasheets containing the economic offer), to PDF or PNG files (containing the administrative/financial documents being printed, signed and scanned). The documents for the technical part of the tender response are made of digital representations of a given civil engineering design produced by a CAD tool, which can have a binary encoding (each tool adopts a given format), a structured format derived from XML and so on, or given Image formats as PNG, JPG or even various vector graphics formats.

Traditionally, such an heterogeneous set of files and digital documents are stored in folders within the file systems of the terminals and/or servers assigned to each employee, with a flat organization where each folder contains all the documentation for a given project/tender response, or a deep organization where the folders have a hierarchical structure (for example a first level divided in years, a second one in classes of project/tender responses, and a third one distinguishing among won and lost applications). This is a naive solution whose main advantage is to be very simple to define, even by operators within a computer science major. The main drawback, on the other side, is related to the difficulties when a specific document needs to be retrieved, as it demands to traverse the overall structure, open each file and check if its content matches the terms of interest. Moreover, all the data of interest for a given project or tender response initiative are not only contained in the produced documents, but also in a set of contextual information and meta-data within the employees' notes and discussions. These meta-data cannot be stored by using such a solution, but on the contrary they can be easily kept within a relational database. Such a solution is associated with a well-structured and expressing query language that helps to retrieve the data of interest by using a precise query expression, which requires operators having a solid background in computer science. Within the tables and tuples, the meta-data can find place, but relational database are not suitable when dealing with files, unless each tuple contains a pointer (such as the file absolute path within the file system) to the file related to the meta-data contained in such a tuple. This requires the user to be aware to keep such a pointer consistent in case of file movements. A Content Management System, on the contrary, represents a hybrid solution among the previous ones: as the first approach, files are contained within hierarchy of folders managed by the file system; while as the second one a relational database with meta-data per each file is defined. The CMS products keeps the two sides of the systems consistent so that by querying the meta-data it becomes possible to retrieve the files of interest. In our envisioned system, we have used a CMS product named Alfresco [6], and on top of it we have designed and implemented our solution. The interaction between Alfresco and our software can occur by means of Alfresco's API, but we have preferred to use the standard interface offered by the Content Management Interoperability Services (CMIS) [7], which is a standard for the interoperability with CMS, so that if we decide in the future to substitute Alfresco with another similar product we do not have to make any changes to our system. CMIS defines in fact an abstraction layer for controlling diverse document management systems and repositories using web protocols, such as REST, AtomPub and so on, which any product like Alfresco implements.

By considering the retrieval of documents, Alfresco and CMIS support a query language whose syntax and semantics is compliant with SQL-92, so that documents are find by means of a term-matching query. A concrete example of this kind of retrieval approach is the following one, where the user is interested to look for all the documents stored within the repository managed by Alfresco with a title equal to “hello world:

```
SELECT *
FROM   cmis:document
```

```
WHERE  cmis:name = 'hello world'
```

Unfortunately the document with title “hello my world or the one with “hi world, and/or all the possible other variants are not retirned as result of this query. To have a more expressive query, a more complex syntax has to be used, but this is particular cumbersome to non-skilled users or those who are not aware of the meta-data modelling determined for the Alfresco repository. It would be more desirable to allow the user to get more results with queries, even with extremely simple keywords. This means returning documents that have slight linguistic variations or synonyms of search terms. In an SQL query for a relational database, a tuple satisfies the query, or not, and the result of a query contains only those tuples that satisfy the search criteria, in an order given by the values in a given column. It would be more desirable to have the results of a search containing the documents that somehow satisfy the search terms, sorted according to the degree of satisfaction.

A classic approach to semantic searches leverages on domain ontologies in order to model the semantics of terms and entities within a given domains by indicating their attributes and relationships, so as to expand the meaning of a particular concept provided as input for a given query. The use of domain ontologies [8] is commonly used in enterprise tools in order to obtain semantic capabilities, and within the context of the BIM standard we can find a proper domain ontology representing knowledge interactions (push/pull) between BIM players, their deliverables and requirements, but ontologies exhibit also drawbacks. Mainly, when a domain ontology is missing, it is extremely difficult to turn specialised domain knowledge (contained in textual document or domain experts) into ontologies, as it demands abstract and effective concept representations. In fact, domain knowledge is typically uneasy to be understood, and contain a lot of contradictions or misinterpretations. Moreover, ontologies fails to perfectly represent synonymy, and the existing tools for building ontologies exhibit a quite heavy learning process making the process of building an ontology quite a time-consuming and discouraging activity for users not holding a computer science degree. Last, expressing semantic queries by using an interrogation language for ontologies, such as SPARQL [9], can be extremely complex.

```
PREFIX bim:<"http_ref">
SELECT ?Location
WHERE  {
   ?document bim:Location ?location;
             bim:Description ?y .
   FILTER regex(?y, "YOUR_REGEX", "i") }
```

where *http_ref* is the location of the OWL file with the BIM ontology, which can be equal to http://ifcowl.openbimstandards.org/IFC4_ADD2\#, *YOUR_REGEX* must be an expression of the XQuery regular expression language, and *i* is an optional flag, meaning that the match is case insensitive. For these reasons we have not exploited ontologies is our work, but they can be integrated in a future possible extension of this study.

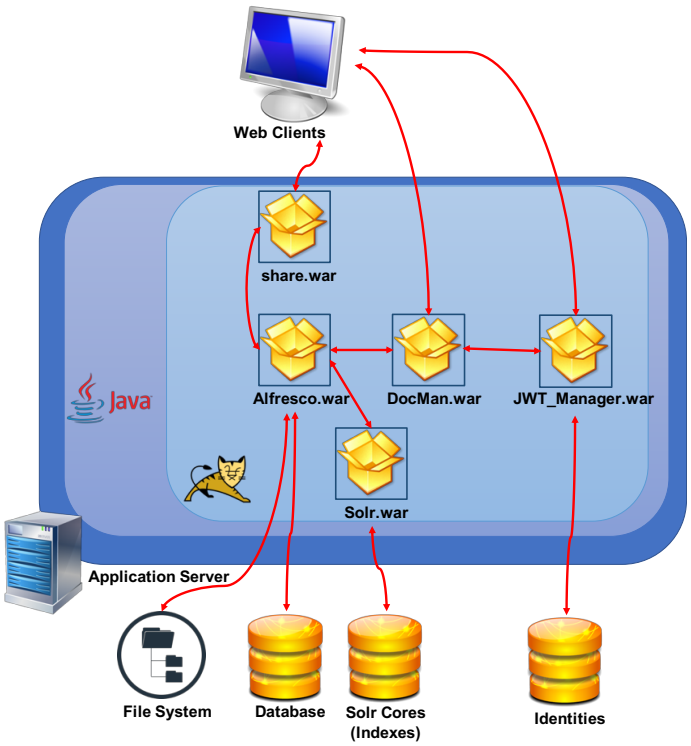


Fig. 1: High-level architecture of the proposed system.

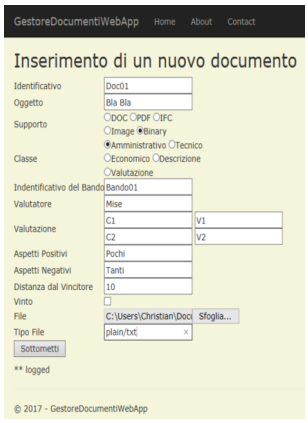
III. Proposed Approach

Figure 1 depicts the main modules and components of our prototype for the proposed solution, where we have the file system storing the documents of the company, the database holding the relative meta-data for the SQL-like search of the documents based on a set of defined properties, and the set of indexes determined by Solr/Lucene from the document content and properties.

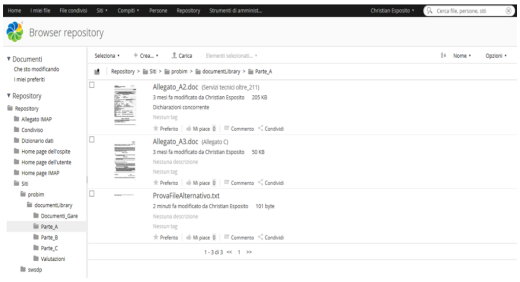
A. Document Management

The Alfresco solution is provided as a war file executed by the web container represented by Apache Tomcat. Such a product provides a management and user console represented by the share.war, so that all the functionalities of the product can be directly invoked by authorised administrators and users, while we have the DocMan.war implementing all the operations of our system (such as the CRUD operations for the kind of documents managed by the system), exposing a RESTful web services implemented within the .NET framework, using JSON as data format, and providing a set of web pages as GUI for the web clients. Figure 2a shows the realised web pages as GUI of our implemented web services, and Figure 2b shows the form for the insertion of a new document filled with some testing text. Figure 2c shows that the new document is eventually available within the repository managed by Alfresco.

(a) Implemented GUI for the implemented RESTful web services.



(b) Example of the use of the prototype to insert a new document.



(c) Check the presence of the new document in Alfresco.

Fig. 2: Prototype testing (local GUI in Italian).

B. *Efficient Search*

A better solution than ontologies or the SQL-like query language of CMIS (and Alfresco) is to allow users to get a query back as a set of documents, together with a set of tools that allow them to refine their search. This search mode is called faceted search [10], which is a technique that involves improving traditional research techniques by applying multiple filters based on the multi-faceted classification of the elements. A particular power and widely known solution within such a kind of search approach is the solution provided by Lucene, a Java library to build and manage an inverted index [11] by writing a proper application or using the RESTful web services offered by Solr [10], as done in our work. Such an

Field	Id		
Sequence Id	Term	Document Frequency	Posting List
0	a	1	1 : 1 : [1] : [0-1]
1	b	1	2 : 1 : [1] : [0-1]
2	c	1	0 : 1 : [1] : [0-1]

Field	title		
Sequence Id	Term	Document Frequency	Posting List
0	building	2	0 : 1 : [1] : [0-7] 2 : 1 : [1] : [0-7]
1	technical	1	0 : 1 : [2] : [9-17]
2	design	2	0 : 1 : [3] : [19-24] 2 : 1 : [3] : [19-24]
3	maintenance	1	1 : 1 : [1] : [0-10]
4	system	1	1 : 1 : [2] : [12-17]
5	restoration	1	2 : 1 : [2] : [9-19]
6	plan	1	2 : 1 : [3] : [21-24]

TABLE I: Inverted Index Example

index resembles the index we can find at the end of a book and consists into an index data structure (stored in the disk for durability) holding a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents. To have a better understanding of this concept let us consider three simple documents, which can be the instances of the meta-data associated to the digital documents managed by Alfresco:

```
Doc0
    {  id:c,
       title:building technical design
},
Doc1
    {  id:a,
       title:maintenance system design
},
Doc2
    {  id:b,
       title:building restoration plan
},
```

The inverted indexes for these three documents are represented in Table I, where the first column is a progressive sequencing of the analysed terms, contained in the second column. The third column contains the term frequency, while the second one indicates where in each document the term is present (*e.g.*, 0 : 1 : [1] : [0-7] means the term is present in the first document, with a frequency equal to 1, and is the second term in the selected field, starting as the first character and finishing with the seventh one). The indexes are built by pre-processing the terms in the documents by removing special characters, stop word, punctuation and applying a given stemmer (in linguistic, stemming consists in reducing inflected (or sometimes derived) words to their word stem or base). Lucene has its own query language (with possibility of term searching in a field of the document or all of them, a set of boolean operators, or special features such as jolly characters, fuzzy search, proximity search and so on), and uses a combination of the Vector Space

Model (VSM) of Information Retrieval and the Boolean model to determine how relevant a given Document is to a User's query. In general, the idea behind the VSM is based on a variant of the *Term Frequency-Inverse Document Frequency* (TF-IDF) named as Classic Lucene Relevancy Algorithm to check the similarity of a document to a given query, which is basically equal to the more times a query term appears in a document relative to the number of times the term appears in all the documents in the collection, the more relevant that document is to the query. It uses the Boolean model to first narrow down the documents that need to be scored based on the use of boolean logic in the Query specification. Such an approach is extremely simple and do not require a specific background on computer science. As a concrete example, if we want to search for a document with 'building' in its title we just have to write *title:building*, if we want the document with a title containing 'building' but not having 'plan' we have to write *title:building -title:plan*, or to find document whose title contains 'building' and 'plan' within 4 words among them (example of proximity search) we have to write *title:"building plan"~4*. It is possible to notice that it is very intuitive to write these queries.

DocMan.war in Figure 1 offers also query capabilities, where the interrogation string can be formalised according to the SQL-like query language provided by Alfresco and supported by the CMIS standard, and the custom query syntax of Lucene, with is not included in CMIS, for querying its indexes managed by Solr. The consistency between the documents in Alfresco and the indexes in Solr is kept by a continuous interaction among them, with occurs without any user intervention. When DocMan.war received an SQL-like query, this is directly passed to Alfresco.war thanks to the CMIS endpoint, while a Solr query can be passed to a specific search service of Alfresco.war, which is not compliant to the CMIS standard and interacts the Solr implementation and returns the obtained outcome, or directly to Solr (as we have preferred to do). The interaction with Solr returns a series of DBID, which represent the internal identifiers that each document has in Alfresco. Subsequently, Alfresco is interrogated in order to obtain the cmis:objectID values, corresponding to the various DBIDs returned by Solr. Given these cmis:objecID, a CMIS query is formulated by DocMan.war to have back the identifications inserted by the users for the calls or documents within Alfresco. Such an interaction for the execution of the Solr query is depicted in Figure 3.

Our prototype has a set of coded queries for which the user has to specify the values to assign to the various parameters of the queries, but also gives the freedom to directly insert the query expression according to the CMIS or Solr syntax, as shown in Figure 4. By running similar queries (looking for documents by their title, types or content and so on) over a similar computing equipment, we noticed that Solr/Lucene handles these queries at least 5 times faster than the SQL-like query capabilities of Alfresco.

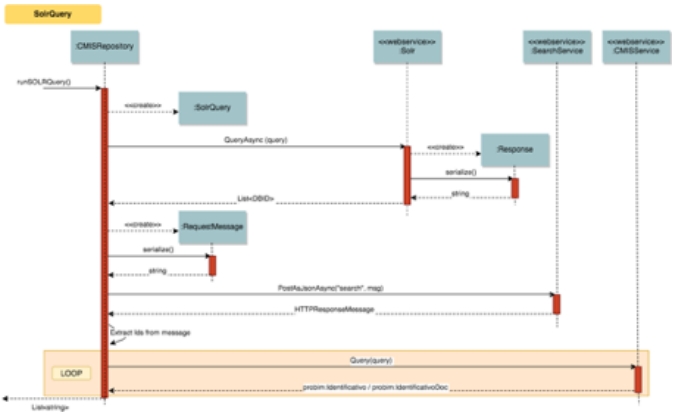


Fig. 3: Solr query implementation.

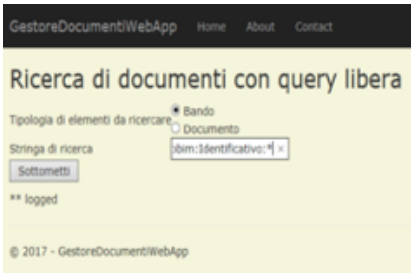


Fig. 4: Query inserting with the GUI.

C. Security Aspects

The interactions of the user with the RESTful web services of our solution needs to be protected from possible misuses and abuses. Implement an authentication and authorization mechanism is the first possible means to guarantee an adequate protection level, so that only legitimate users can invoke the provided operations. Within the context of the web services, the WS-Security standard [12] indicates the mechanisms to be used in such a case by having the Security Assertion Markup Language (SAML) for expressing the identity and security claims to be exchanged within the SOAP header of messages between parties ($e.g.$, requester and web service), the eXtensible Access Control Markup Language (XACML) for modelling the authorisation policies and an architecture of the authorization solution, or WS-Trust for disciplining the trust relationships between participants in a secure message exchange (just to name some of widely known standards). In our solution, we have preferred to use JSON Web Token (JWT) [13] given its simplicity and stateless nature. The users provide its credential as username and password to the server (but more complex claims are possible), which checks if they are present in its identity storage and returns a token in case of success. The adopted model is, therefore, the Access Control List (ACL) [14], where we can find which users are granted access to objects/functionalities of the system (but more advamces access control models are still possible by integrating solutions such as XACML or similar ones). Later on, when requesting a HTTP method, the token is inserted

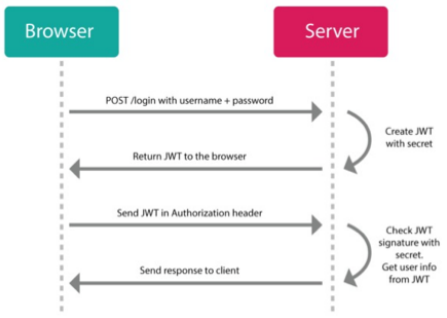


Fig. 5: Client-Server interactions with JWT.

in the SOAP header of the request, and the server grants the access to the incoming requests with valid tokens. Figure 5 shows such interaction. To this aim, the architecture in Figure 1 contains an additional war, named JWT_Manager.war, offering the functionalities of managing the identities, requesting token by giving a valid couple of username and password, checking the validity of a given token, revoke the grant for a given user. It is evident that all our services strongly interact with such an additional set of RESTful web services so as to decide to grant or deny a received request.

IV. Lesson Learnt

This paper described the work to manage and search documents produced during the practice in a civil engineering company when participating to a project and/or responding to a call-for-tender issues by a public administration, but can be easily used in a similar application context, from the healthcare to the manufacturing industry, where unstructured documents must be stored and queried. By exploiting the most effective technologies of Alfresco as the document management system and Lucene/Solr as effective and efficient query system, we have designed and implemented a series of RESTful service in .NET supporting the CRUD and query functions to deal with the life cycle of these documents. As a future work, we aim at extending our solution with with semantic services in addition to the capabilities offered by Lucene/Solr. Our driving idea is to exploit Apache Stanbol [15] to extract features from passed content, integrating an ontology-based reasoner and modelling the knowledge contained in the managed documents.

Acknowledgment

The described work has been partially supported by the PROBIM research project, funded by the Italian Ministry of Economic Development within the context of Horizon 2020 - PON I&C 2014-20.

References

[1] W. J. Luzadder, "Introduction to engineering drawing: The foundations of engineering design and computer aided drafting," in *Prentice Hall PTR*, 1992.

[2] C. Eastman, P. Teicholz, R. Sacks, and K. Liston, "Bim handbook - a guide to building information modeling for owners, managers, designers, engineers and contractors," in *John Wiley & Sons Inc, 3rd edition*, 2018.

[3] EUR-Lex, "Directive 2014/24/eu of the european parliament and of the council of 26 february 2014 on public procurement and repealing directive 2004/18/ec text with eea relevance," in *D. Available on line at https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex\%3A32014L0024*, 2014.

[4] R. Land, D. Sundmark, F. Lüders, I. Krasteva, and A. Causevic, "Reuse with software components-a survey of industrial state of practice," in *International Conference on Software Reuse, 150-159*, 2009.

[5] L. Richardson and S. Ruby, "Restful web services," in *O'Reilly Media*, 2007.

[6] M. Shariff, A. Bhandari, and P. M. V. Choudhary, "Alfresco 3 enterprise content management implementation," in *Packt Publishing*, 2009.

[7] OASIS, "Content management interoperability services (cmis) - version 1.1," in *Available at http://docs.oasis-open.org/cmis/CMIS/v1.1/CMIS-v1.1.html*, 2015.

[8] I. Durán-Muñoz and M. R. Bautista-Zambrana, "Applying ontologies to terminology: Advantages and disadvantages," in *Hermes-Journal of Language and Communication in Business*, vol. 51, 2013, pp. 65–77.

[9] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of sparql," *ACM Transactions on Database Systems (TODS)*, vol. 34, no. 3, p. 16, 2009.

[10] T. Grainger and T. Potter, *Solr in action*. Manning Publications Co., 2014.

[11] M. McCandless, E. Hatcher, and O. Gospodnetic, *Lucene in action: covers Apache Lucene 3.0*. Manning Publications Co., 2010.

[12] Atkinson, B. et al., "Web services security (ws-security). specification," in *Available at https://www.it.iitb.ac.in/~madhumita/research_topics/authentication/WS\%20Security.pdf*, 2002.

[13] M. Jones, J. Bradley, and N. Sakimura, "Json web token (jwt). no. rfc 7519," in *Available at https://jwt.io/*, 2015.

[14] J. Barkley, "Comparing simple role based access control models and access control lists," in *Proceedings of the second ACM workshop on Role-based access control*, 1997.

[15] R. Bachmann-Gmur, *Instant Apache Stanbol*. Packt Publishing Ltd, 2013.