journal homepage: www.intl.elsevierhealt h.com/journals/cmpb

BIOMedical Search Engine Framework: Lightweight and customized implementation of domain-specific biomedical search engines

Alberto G. Jácome [a], *Florentino Fdez-Riverola* [a], *Anália Lourenço* [a,b,*]

[a] *ESEI—Escuela Superior de Ingeniería Informática, Edificio Politécnico, Campus Universitario As Lagoas s/n, Universidad de Vigo, 32004 Ourense, Spain*
[b] *Centre of Biological Engineering, University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal*

ARTICLE INFO

Article history:
Received 26 July 2015
Received in revised form
10 February 2016
Accepted 29 March 2016

Keywords:
Search engine framework
Biomedical literature
Vertical engine
Text mining
Web application

ABSTRACT

Background and objectives: Text mining and semantic analysis approaches can be applied to the construction of biomedical domain-specific search engines and provide an attractive alternative to create personalized and enhanced search experiences. Therefore, this work introduces the new open-source BIOMedical Search Engine Framework for the fast and lightweight development of domain-specific search engines. The rationale behind this framework is to incorporate core features typically available in search engine frameworks with flexible and extensible technologies to retrieve biomedical documents, annotate meaningful domain concepts, and develop highly customized Web search interfaces.
Methods: The BIOMedical Search Engine Framework integrates taggers for major biomedical concepts, such as diseases, drugs, genes, proteins, compounds and organisms, and enables the use of domain-specific controlled vocabulary. Technologies from the Typesafe Reactive Platform, the AngularJS JavaScript framework and the Bootstrap HTML/CSS framework support the customization of the domain-oriented search application. Moreover, the RESTful API of the BIOMedical Search Engine Framework allows the integration of the search engine into existing systems or a complete web interface personalization.
Results: The construction of the Smart Drug Search is described as proof-of-concept of the BIOMedical Search Engine Framework. This public search engine catalogs scientific literature about antimicrobial resistance, microbial virulence and topics alike. The keyword-based queries of the users are transformed into concepts and search results are presented and ranked accordingly. The semantic graph view portraits all the concepts found in the results, and the researcher may look into the relevance of different concepts, the strength of direct relations, and non-trivial, indirect relations. The number of occurrences of the concept shows its importance to the query, and the frequency of concept co-occurrence is indicative of biological relations meaningful to that particular scope of research. Conversely, indirect concept associations, i.e. concepts related by other intermediary concepts, can be useful to integrate information from different studies and look into non-trivial relations.
Conclusions: The BIOMedical Search Engine Framework supports the development of domain-specific search engines. The key strengths of the framework are modularity and extensibility

* *Corresponding author.* ESEI—Escuela Superior de Ingeniería Informática, Edificio Politécnico, Campus Universitario As Lagoas s/n, Universidad de Vigo, 32004 Ourense, Spain.
E-mail addresses: agjacome@esei.uvigo.es, riverola@uvigo.es, analia@uvigo.es (A. Lourenço).

http://dx.doi.org/10.1016/j.cmpb.2016.03.030

in terms of software design, the use of open-source consolidated Web technologies, and the ability to integrate any number of biomedical text mining tools and information resources. Currently, the Smart Drug Search keeps over 1,186,000 documents, containing more than 11,854,000 annotations for 77,200 different concepts. The Smart Drug Search is publicly accessible at http://sing.ei.uvigo.es/sds/. The BIOMedical Search Engine Framework is freely available for non-commercial use at https://github.com/agjacome/biomsef.

1. Introduction

In Life Sciences, the wealth of knowledge in journal publications is of significant importance for researchers in making scientific discoveries. However, the acquisition of such information is becoming increasingly difficult due to the large volume and heterogeneity of articles, and the intense rate of publication in these fields.

The PubMed search engine is quite powerful in that its keyword-based Boolean query interface is practical and easy to use, and it lays on the millions of abstracts available in the Medline database [1]. However, it can be difficult for researchers to explore and search such huge volume of data in an efficient manner. Queries about general topics are likely to return a large number of potentially relevant documents (hundreds or thousands of documents) that require further manual revision. Conversely, queries about very specific topics demand that the researcher knows in advance the most relevant keywords of the topic (which often vary over time).

It is much more intuitive for users to query about concepts relevant to the domain and with which they are familiar with. Within this context, domain-specific literature search is currently of high demand by biomedical researchers [2]. Efforts are being invested in the development of alternative ways to query documents from the Medline database and offer topic-driven search facilities. Alternatives are quite varied in purpose and nature, but most of them take advantage of text mining technologies and domain-specific semantics to offer a more focused and contextualized search experience [3,4].

Today, major biological databases, such as UniProt protein resources, BioCyc pathway knowledge bases and BRENDA enzyme information system enable concept-driven searches of the curated documents (abstracts or full-texts) [5–7]. Furthermore, the annual report of the Nucleic Acids Research journal about biomedical databases and Web services presents a wide variety of specialized literature search engines, created to meet the information needs of particular research communities [8].

Table 1 presents a summary of recent projects that developed domain-specific literature search systems. The purpose is not to provide a complete list of available domain-specific systems but rather to describe the technical standpoint of these applications, most notably the support provided by domain-specific semantics, the use of text mining technologies, and the visual artifacts used to enable concept-driven knowledge discovery. Text mining methods and tools are used to incorporate semantic functionality such as the automatic suggestion of synonyms for user-submitted query terms, the assessment of document relevance, the description of document contents, or the inspection of documents in which concepts are related in specific ways. Controlled vocabularies and ontologies, such as DrugBank [18], UMLS [19], MESH [1], and SNOMEDCT [20], help define the knowledge domain to be captured by the search engine and offer the means to convert keyword-based queries in domain concepts, and navigate within domain semantics [21].

For example, the Protein Interaction information Extraction (PIE) system implements a machine learning approach to provide a PubMed-like search interface that prioritizes documents mentioning protein–protein interactions [9] while Alkemio Web tool uses a naïve Bayesian classifier to predict the relatedness of chemicals to query topics [11]. In turn, the PolySearch2 supports the discovery of associations between human diseases, genes, drugs, metabolites and toxins in MEDLINE abstracts, PubMed Central full-text articles, and text-rich biological databases [13].

Typically, all search engines present results as a ranked list. Semantic annotations are visually highlighted and some semantic facets (based on concept types) may exist to narrow down the list. Graph- or tree-based visualizations are provided as advanced means of navigation, namely, to find indirect associations between concepts of interest.

From a technical standpoint, one obvious observation is that most projects undertake the construction of the domain-specific search engine from scratch, and that all these new implementations include modules for document retrieval, text mining, document indexing and scoring, and query execution. Typically, document retrieval relies on MEDLINE web services and text mining relies on open solutions for linguistic processing and biomedical entity recognition. Moreover, document/concept scoring is often based on well-known algorithms, such as the term frequency inverse document frequency (TF-IDF) algorithm [22].

Understandably, domain requirements and goals of analysis determine the practical choice of methods and tools used, but many technologies and tools have the potential of being applied across domains. In particular, the core infrastructural components linking main operations and resources in most biomedical semantic search engines could be generalized with the help of a web development framework.

There are general purpose and open-source search engine development frameworks, but they are hardly used in bioinformatics applications. These frameworks are typically equipped to deal with distribution and scalability concerns, i.e. enabling the construction of large-scale engines, rather than delivering generalized semantic functionality, i.e. enabling the construction of domain-specific engines (Table 2). So, the learning curve associated with using these frameworks together with the need to program additional biomedical semantic processing and analysis modules discourages bioinformatics application.

Table 1 – Recently published domain-specific literature search systems.

Name	Main goal	Source documents	NLP tools	Controlled vocabulary	Ranking	Concept-driven analysis
PIE (Protein Interaction information Extraction) the search [9] http://www.ncbi.nlm.nih.gov/IRET/PIE/	Web-based ranking system for protein interaction information	MEDLINE abstracts	Dependency parsing, gene mention tagging, and term-based features	Gene corpus	PPI confidence scores	No
Colil [10] http://colil.dbcls.jp/fct/	Search service for citation contexts	PMC-OAS papers	Entity recognition (URI lookup)	BIBO, DoCO, and DCTERMS	Text match score and entity-attribute-value link coefficients	No
Alkemio [11] http://cbdm.mdc-berlin.de/~medlineranker/cms/alkemio	Understand or predict roles of chemicals in biological pathways and diseases	MEDLINE abstracts	Naïve Bayesian classifier and ranker	MESH terms	Precision and false discovery rate	No
FACTA+ [12] http://www.nactem.ac.uk/software/facta/	Finding and visualizing indirect associations between biomedical concepts	MEDLINE abstracts	Dictionary-based entity recognition	DrugBank UMLS, UniProt, CAS, HMDB, and several dbs such as KEGG and DrugBank	Co-occurrence statistics	Indirect concept association based on pivot-target strategy
PolySearch2 [13] http://polysearch.ca	Discovering associations between human diseases, genes, drugs, metabolites and toxins	MEDLINE abstracts, PubMed Central full-text articles, Wikipedia full-text articles and US Patent application abstracts	Dictionary-based entity recognition	Gene ontology, MESH terms, ICD-10 medical codes, biological and chemical taxonomies	Z score, R score	Concept-based filter in document view
G-Bean [14] http://bioinformatics.clemson.edu/G-Bean/index.php	Query documents in MEDLINE database	MEDLINE abstracts	Dictionary-based entity recognition	UMLS, MeSH, SNOMEDCT, CSP and AOD vocabularies	TF-IDF weighting	Graph-based navigation, similar document finding
SCAIView [15] http://purl.bioontology.org/ontology/MSO	Search of genes associated with sclerosis and pathways targeted by approved disease-modifying drug	MEDLINE abstracts	Dictionary-based entity recognition	Multiple sclerosis ontology, DrugBank, MESH	No	Hierarchy tree, entity view
OncoSearch [16] http://oncosearch.biopathway.org	Search of gene expression changes in cancer	MEDLINE abstracts	BANNER and Moara systems, dictionary-based recognizer, Turku event extraction system	NCI thesaurus	Weighted harmonic mean of the confidence scores	Filter by concept type
DigSee [17] http://gcancer.org/digsee	Disease gene search engine	MEDLINE abstracts	ABNER, Turku event extraction system	No	Text evidence relevance (machine learning model)	Graph view

Table 2 – Some general purpose open-source text search engine frameworks.

Tool	Platform	Semantic capacities	Purpose
Elastic https://www.elastic.co/	Java, Lucene.	Simple text and document fields search. Results ranking.	Complete framework on top of Apache Lucene. Real-time search.
Zend_Search_Lucene http://framework.zend.com/manual/1.12/en/zend.search.lucene.overview.html	PHP, Lucene.	Simple text and document fields search. Results ranking.	General purpose text search engine on top of Apache Lucene.
Carrot² http://project.carrot2.org/	Java	Organization of documents into categories, using text clustering algorithms.	Integration of clustering capabilities into existing search engines.

These observations led us to consider building the new and open-source BIOMedical Search Engine Framework. This framework aims to provide a solution to bioinformatics applications in demand of a small to medium size semantic search engine. To meet this goal, the BIOMedical Search Engine Framework integrates core features typically available in search engine frameworks with a set of features useful to perform biomedical semantic analysis. Its architecture is modular and component agnostic, and enables the interchange of different components to build biomedical domain-specific search engines at a reduced programming cost. Basic implementation covers well-established algorithms and state-of-the-art open-source tools, namely, it incorporates modules for document retrieval, text mining, document indexing and scoring, query execution, and knowledge reasoning. Controlled vocabulary is at the core of knowledge representation and reasoning, and the graph data structure helps to scale query and visualization abilities for large literature collections.

The rest of this paper aims to disclose the architecture and application of the framework and is organized as follows. Section 2 outlines the architecture of the BIOMedical Search Engine Framework, and describes how each component contributes to building a customized domain-specific literature search engine. Section 3 discusses the development of the Smart Drug Search (SDS) engine as proof-of-concept of the BIOMedical Search Engine Framework. At the end, we provide some conclusions and identify areas of future work.

2. Methods

2.1. High level description

The BIOMedical Search Engine Framework has been designed to support the construction of domain-specific and highly customized search engines. To meet this purpose, the framework is built on top of state-of-the-art open technologies and standards, and the architecture is completely modularized, to facilitate the extension of features and methods to meet new needs or integrate other technology.

The architecture of the server application has three layers (see Fig. 1). The top layer provides a RESTful API to access data, the middle layer provides all the application logic, and the bottom layer is responsible for database management. These modules are programmed in Scala [23] and integrate technologies from the Typesafe Reactive Platform (http://www.typesafe.com/products/typesafe-reactive-platform). Most notably, the Play Framework (https://www.playframework.com/) is used as a container framework [24], the Slick database query and access library facilitates the functional-relational mapping

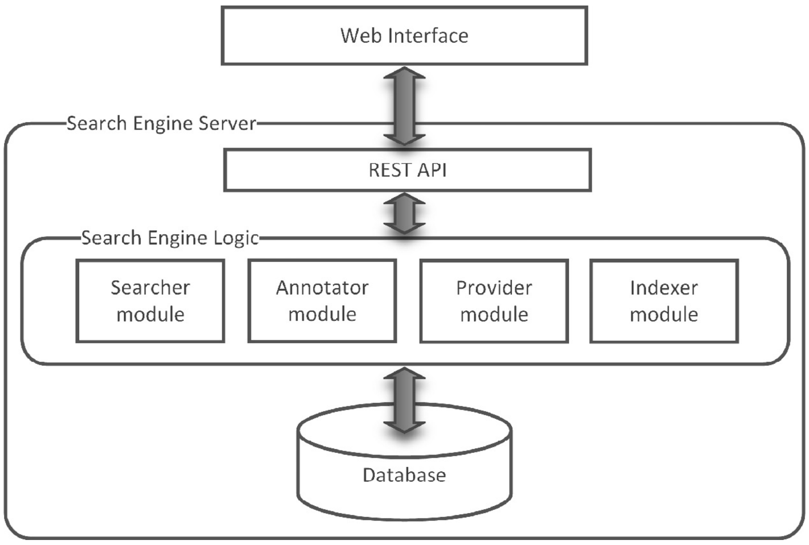


Fig. 1 – Schematic of the architecture of the BIOMedical Search Engine Framework.

```
final class ExampleDrugAnnotator extends AnnotatorAdapter {

  import context._

  lazy val drugAnnotator = new ExternalDrugsAnnotator()

  override def annotate(article: Article): Future[Unit] =
    drugAnnotator.getAnnotationsIn(article.content) flatMap {
      as => persistAnnotations(as, article.id.get)
    }

  private def persistAnnotations(as: Set[DrugAnnotation], articleId: Aticle.ID): Future[Unit] =
    Future.sequence { as.map(persistAnnotation(_, articleId)) }

  private def persistAnnotation(da: DrugAnnotation, aid: Article.ID): Future[Unit] =
    for {
      n <- drugAnnotator.normalizeAnnotation(da)
      k <- super.getOrStoreKeyword(normalized, Drug)
      _ <- super.annotationsDAO.insert(Annotation(None, aid, k.id, da.term, da.start, da.end))
    } yield ()

}
```

Fig. 2 – Code snippet of the ExternalDrugsAnnotator, an annotator trait for recognizing and normalizing mentions to drug-related entities in a text.

(http://slick.typesafe.com/), and the Akka toolkit (http://akka.io/) supports the distribution and parallelization of different parts of the system [25,26]. On the other hand, the client-side web interface is developed using the AngularJS JavaScript framework (https://angularjs.org/) and the Bootstrap HTML/CSS framework (http://getbootstrap.com/).

The modules Provider, Indexer and Searcher encapsulate the most common components of search engines, i.e. the components responsible for document retrieval, document indexing and user query execution. In turn, domain-specific semantic analysis is enabled by the Annotator module. This module includes various text mining methods and tools to be used in the semantic annotation of the documents, the conversion of keyword-based queries in domain concepts, and domain-specific knowledge reasoning over query results. The semantic concept is regarded as the basic knowledge representation unit of the BIOMedical Search Engine Framework, and the user interface is designed to take advantage of concepts as best as possible. Functionality such as query resolution, document relevance assessment, document filtering and traverse, and the discovery of indirect associations are based on the information provided by domain concepts.

2.2. *Provider module*

The document provider module is responsible for incorporating new documents into the database. Currently, the framework implements the PubMed document provider, which is based on the eUtils tools [1], and automatically transforms the XML responses into domain entities that can be easily manipulated within the framework. The way to incorporate more document providers into the system is to implement a new document retrieval class and its respective controller. In future versions, such document provider customization will be facilitated with a Scala trait, which may be refined to any given provider. Similarly to the annotator and searcher modules, the administrator will be able to toggle providers through the system configuration file.

This module operates asynchronously with respect to the rest of the system, i.e. the search engine can still operate fluently while documents are being inserted in the database.

2.3. *Annotation module*

This module integrates text mining technologies, specifically machine learning tools and dictionary-based recognizers based on controlled vocabulary. On implementing a given domain search engine, the system administrator can enable and disable the implemented annotators in the server configuration file, and this module will be responsible for articulating the annotation of the documents accordingly.

Technically speaking, the Annotator module incorporates text mining methods and tools through an adapter interface (a Scala trait). Each annotator is an Akka actor which is able to recognize named entities in the documents and store this semantic information in the database.

The operation of the various annotators is completely asynchronous. All the annotators start to work at the same time, and will respond to the supervisor once they finish. Once all the annotators have finished, and stored all annotations and concepts in the database, the document is marked as already annotated. If any annotator returns an error state to its supervisor, all the document annotations and concepts are deleted, and its status remains as “not annotated” until the system administrator issues a new annotation request.

Fig. 2 illustrates the implementation of an annotator, the ExampleDrugAnnotator, which recognizes and normalizes mentions to drug-related entities in a text. For any given annotator, the developer should implement the method "annotate", i.e. define which concepts are to be annotated and the extent of information to be stored in the database. This annotator may call any available tagger. Here, it calls the ExternalDrugsAnnotator, which has its own method to recognize and normalize mentions to drug-related concepts, and obtains the annotations using the ExternalDrugsAnnotator.getAnnotationsIn method.

2.4. *Indexer module*

The indexer module is the core component of the search engine, as it enables the construction of the search index. Documents have associated domain-specific semantic annotations, i.e. domain concepts that have been detected in the text. These annotations are used to index the documents in a meaningful way to the domain. The relevance of individual documents with respect to each concept is calculated using the TF-IDF algorithm [22].

The indexer module is executed in a scheduled fashion, auto spawning its process after a given time interval has passed since last execution. The reason for generating the index in a scheduled fashion, instead of after each annotation is finished, is efficiency-related. Creating the search index is very resource-consuming and launching the indexer process after producing each individual annotation would deteriorate significantly the performance of the search engine. This scheduling may be configured in the engine configuration file.

2.5. *Searcher module*

The searcher module uses NER tools to normalize queries to a common concept-oriented representation (disambiguate terms or find the stored synonyms in the database). A SearcherAdapter trait is provided to perform any text processing operations necessary to map the terms specified by the user to domain concepts and look for synonyms. Although these searchers are not essential to the correct operation of the search engine, their use is recommended as an additional means of enhancing user search experience. Like with annotators, searchers can be enabled or disabled in the server configuration file. An example of a searcher implementation with a NER tool is shown in Fig. 3.

Search queries are received by the system and further processed to find which keywords and concepts relate to them. Documents are scored based on those keywords and concepts, and the top scores are returned to the user. If no matching concepts are found, the system may be configured to run a simple text search with partial matching of keywords over annotations.

Finally, the retrieved documents are ranked by additively aggregating the precomputed TF-IDF scores for the annotated concepts.

2.6. *User interface*

As illustrated in Fig. 4, the AngularJS JavaScript Model-View-View-Model (MVVM) framework, together with Bootstrap and its complete suite of CSS components, supports the rapid development of highly customizable web applications. Most notably, this framework enables the communication with the server-side RESTful API and the creation of a visually appealing interface.

The BIOMedical Search Engine Framework supports two search interfaces—basic and advanced—that provide access to the underlying TF-IDF index. The basic interface is similar to most search engines where users enter a simple keyword-based query and the relevant documents are returned. The advanced search provides search fields for each searchable field in the index. Along with the query interface, the framework provides an administrative interface from which the administrator can launch new jobs and track the progress of previously started jobs.

Overall, the interface is implemented with full responsiveness. Users may access the search engine using any Internet-capable device, and the interface will resize and accommodate its visual representation to the available screen size. Moreover, since the interface is not coupled with the server application, it can be replaced by a completely new one as long as it uses the existing RESTful API. In fact, the interface may not be even web-based, as HTTP requests can be sent from within a desktop application to any existing server. This allows system administrators to create, refine or re-structure user interface without affecting the search indices or the document collection.

```
final class ExampleDiseasesSearcher extends SearcherAdapter {

  lazy val diseasesTagger = new ExternalDiseasesTagger()

  override def search(query: String): Future[Set[Keyword.ID]] =
    for {
      entities   <- diseasesTagger.findEntities(query)
      normalized <- entities.map(_.getCanonicalName)
      keywordIds <- super.searchNormalized(normalized)
    } yield keywordIds

}
```

Fig. 3 – Example of a SearcherAdapter trait implementation. An ExternalDiseasesTagger service, capable of recognizing and normalizing disease-related entities in a text.

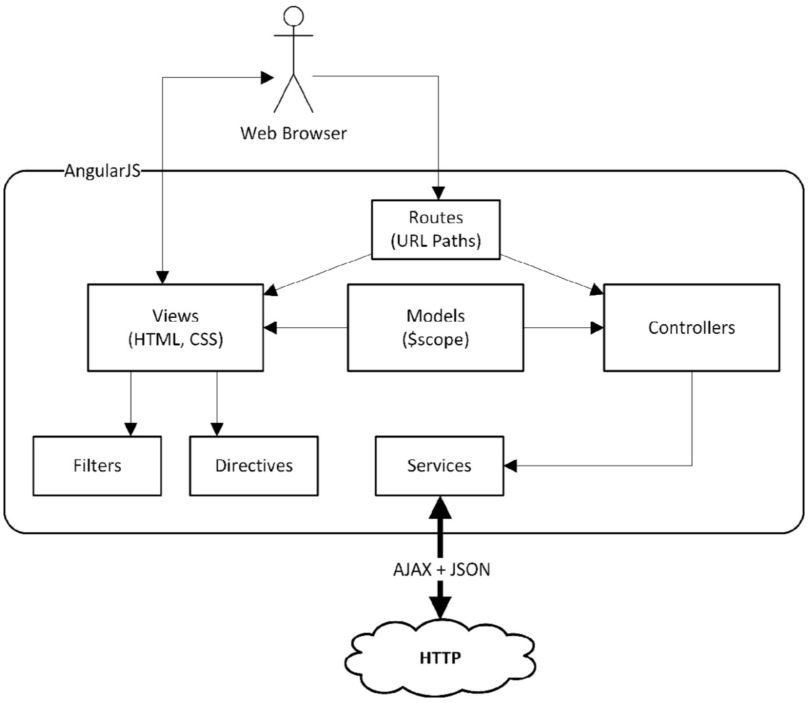


Fig. 4 – Architecture of the web application.

2.7. *Web services*

The RESTful service of the BIOMedical Search Engine Framework is designed for programmatic access over the network. For example, search queries can be performed from another program and results displayed within it. Using common programming languages, any programmer can connect to the service using the different resource-oriented HTTP methods and URLs defined in the route configuration, which follow the common REST architecture practices. Queries and responses will be just plain JSON content, and can be parsed by the program to display whatever information is most relevant in each case.

2.8. *Software verification*

The BIOMedical Search Engine Framework was verified with automated tests covering a great percentage of the main codebase. Unit tests, mostly programmed as property-based tests, verify that the units of code work as intended for any given input and integration tests guarantee that the different components and modules of the framework can function together correctly, and the main workflow is correct. The tools of choice for programming these tests have been the ScalaTest suite, as the main unit- and integration-testing framework, and the ScalaCheck as the property-based testing provider and test case generator.

All these tests are accessible through the main code repository of the software, and any skilled developer can execute them locally with the Simple Build Tool (SBT) runner and its "test" command. Refer to the Mode of availability section to know how to access the code repository of the BIOMedical Search Engine Framework.

3. Results

The BIOMedical Search Engine Framework supported the implementation of the Smart Drug Search (SDS), a publicly accessible drug-related literature search application supporting research on microbial virulence, antimicrobial resistance and novel antimicrobial products (Fig. 5).

Currently, SDS indexes more than 1,186,000 article abstracts and 772,000 concepts, containing an average of 9 annotations per document (see Table 3).

3.1. *Index construction*

In the private area of the SDS, the administrator manages, and monitors, the processes of document retrieval and annotation (Fig. 6). PubMed is the primary source of literature of SDS and MEDLINE eUtils services are used to retrieve documents [27]. Documents may be retrieved using keyword-based Boolean expressions or by indicating a list of PubMed identifiers.

The administrator can also execute document annotation, triggering as many annotator tools as specified in the configuration file. The documents pending (or failing) annotation are easily identified and can be re-issued.

Once the documents are stored in the database, the annotation module is started. Currently, SDS addresses the annotation of drugs and related substances, organisms, diseases, genes and proteins. To perform that annotation, SDS uses the following NER tool: ABNER, that provides information on gene and protein mentions [28]; OSCAR4, that identifies chemical entities and is able to normalize textual mentions to the IUPAC International Chemical Identifier (InChI) notation [29]; and Linnaeus, which is responsible for finding species

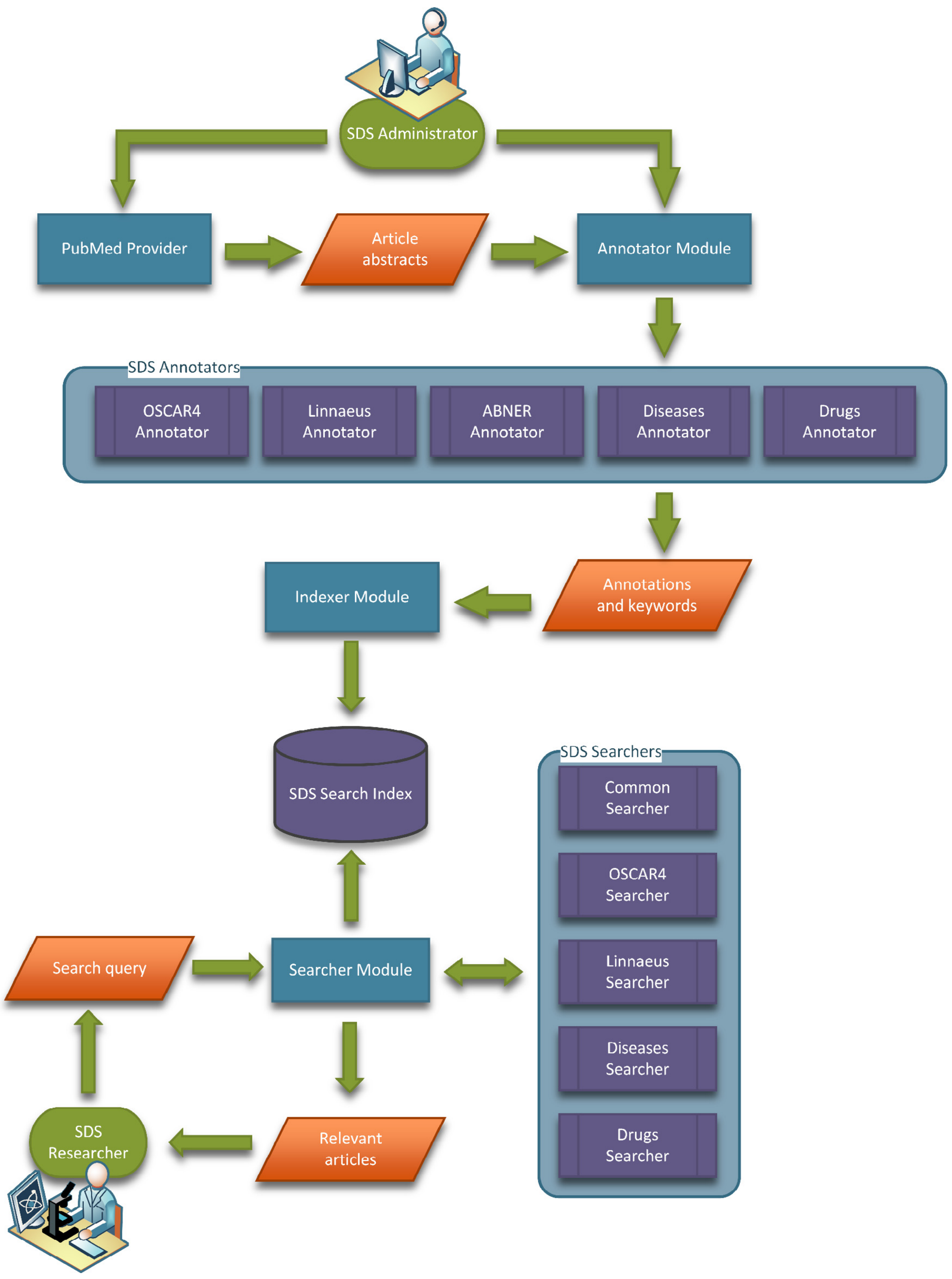


Fig. 5 – The Smart Drug Search workflow developed with the BIOMedical Search Engine Framework. PubMed is the document provider. OSCAR4, Linnaeus and ABNER tools and the dictionary-based disease and drug taggers carry out document annotation and search query mapping.

mentions and normalize them to the scientific or formal name, as specified in NCBI Taxonomy [30]. It also includes a dictionary-based disease annotator based on the Disease Ontology [31] and a dictionary-based drug annotator based on DrugBank controlled vocabulary [18].

The population of the search index is a scheduled operation that will run in configurable intervals. At that point, the indexer will retrieve all the annotated concepts and create a TF-IDF ranked index. Index update does not condition the operation of the search engine. That is, any queries issued while

Table 3 – General statistics about the documents indexed by the SDS system.

Number of documents	1,186,270
Number of concepts	772,356
	Compounds: 31,755
	Drugs: 2,165
	Genes, DNA and RNA: 141,873
	Proteins: 448,661
	Species: 18,086
	Diseases: 3,333
Number of annotations	11,854,377
Minimum number of annotations per document	1
Maximum number of annotations per document	90
Average number of annotations per document	9.993

the index is being updated are resolved on the old index. Once the update finishes, queries will be automatically directed to the updated index.

3.2. *Query execution*

SDS query interface presents a simple and an advanced mode. Keyword-based Boolean expressions are the most common way of describing the query (Fig. 7-1). Users may also use concept-based facets to narrow down the query. For example, the user may be interested in documents that contain compound, gene and drug annotations (Fig. 7-3). Likewise, documents may be filtered by conventional searchable fields, such as publication date (Fig. 7-4).

The retrieved documents are presented as a list (Fig. 7-2). Each document is described in terms of the annotated concepts and the resulting semantic score. Results list may be further filtered by concept classes. For example, we may wish to see only documents that mention DNA entities (Fig. 7-5). Finally, the user may click on the document and read the semantically annotated abstract or access the PubMed record for additional information (Fig. 7-6).

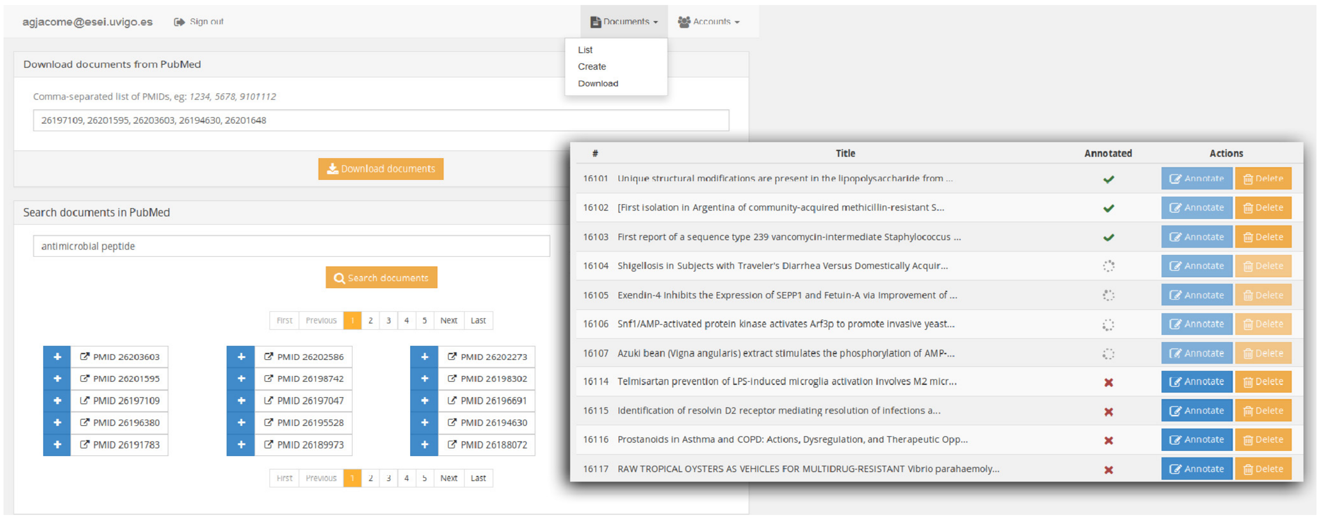
Behind the scene, once the SDS receives a query, the engine requests all the configured searchers in the searcher module to extract the relevant concepts and retrieve their database representation. SDS implements searchers for OSCAR4, Linnaeus and our dictionary-based disease and drug annotators. The ABNER tagger is not included as a searcher because it does not support term normalization. As alternative, the engine uses a common searcher that looks for the keywords directly in the index. This operation is done concurrently, i.e. all configured searchers work at the same time, and once all operations are completed the results are aggregated (removing duplicate terms).

The retrieved documents are ranked based on the aggregated TF-IDF score of the included concepts. This operation is done asynchronously with respect to the rest of the system, allowing multiple queries to be processed at the same time and without affecting the performance of any other operation on the system.

3.3. *Knowledge reasoning*

The most immediate result of a user search will be a list of documents, sorted by semantic relevance. However, given the potentially large number of results that a query may return, SDS relies on a semantic graph structure to provide improved means of navigation and analysis (Fig. 8).

At first, the graph view renders the concepts in the retrieved documents, linked by direct co-reference in text. Color denotes concept class type, and the size of nodes and the width of edges indicate the frequency of occurrence (Fig. 8-1). By toggling the concept class facets, the user may look into subgraphs (Fig. 8-2). Likewise, by clicking in the edge connecting two nodes, the user gets the documents supporting the reference or association (Fig. 8-3). Finally, the user may investigate the direct associations of a given node (Fig. 8-4), i.e. its neighbors, and the indirect associations between any given concepts (shortest path), i.e. look for concepts that do not have a direct link but are related by intermediary nodes (Fig. 8-5). Notably, the tool calculates the



Comma-separated list of PMIDs, eg: *1234, 5678, 9101112*

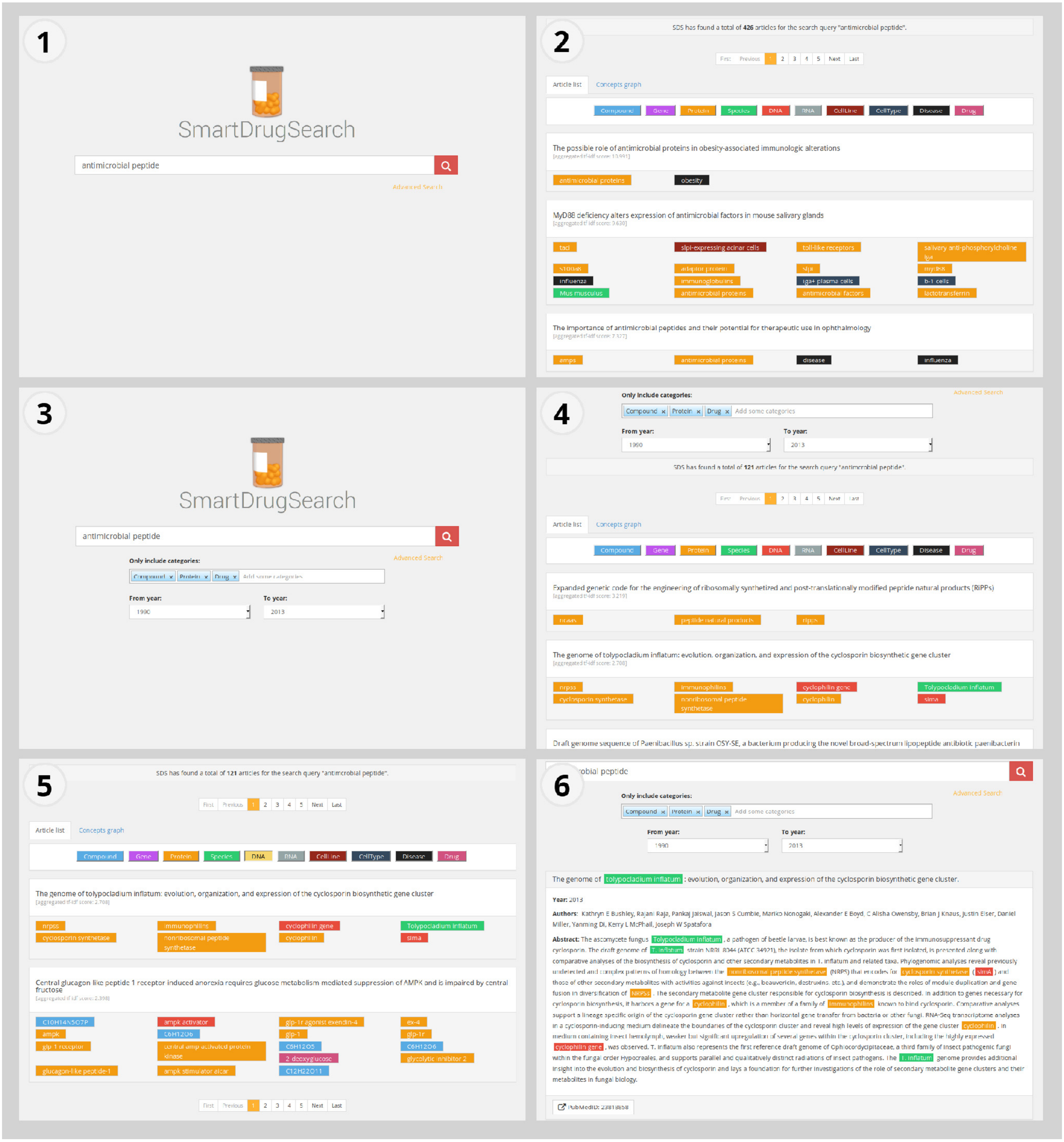
26197109, 26201595, 26203603, 26194630, 26201648

antimicrobial peptide

PMID 26203603	PMID 26202586	PMID 26202273
PMID 26201595	PMID 26198742	PMID 26198302
PMID 26197109	PMID 26197047	PMID 26196691
PMID 26196380	PMID 26195528	PMID 26194630
PMID 26191783	PMID 26189973	PMID 26188072

#	Title	Annotated	Actions
16101	Unique structural modifications are present in the lipopolysaccharide from ...	✓	Annotate Delete
16102	[First isolation in Argentina of community-acquired methicillin-resistant S...	✓	Annotate Delete
16103	First report of a sequence type 239 vancomycin-intermediate Staphylococcus ...	✓	Annotate Delete
16104	Shigellosis in Subjects with Traveler's Diarrhea Versus Domestically Acquir...	◌	Annotate Delete
16105	Exendin-4 Inhibits the Expression of SEPP1 and Fetuin-A via Improvement of ...	◌	Annotate Delete
16106	Snf1/AMP-activated protein kinase activates Arf3p to promote invasive yeast...	◌	Annotate Delete
16107	Azuki bean (Vigna angularis) extract stimulates the phosphorylation of AMP-...	◌	Annotate Delete
16114	Telmisartan prevention of LPS-induced microglia activation involves M2 micr...	✗	Annotate Delete
16115	Identification of resolvin D2 receptor mediating resolution of infections a...	✗	Annotate Delete
16116	Prostanoids in Asthma and COPD: Actions, Dysregulation, and Therapeutic Opp...	✗	Annotate Delete
16117	RAW TROPICAL OYSTERS AS VEHICLES FOR MULTIDRUG-RESISTANT Vibrio parahaemoly...	✗	Annotate Delete

Fig. 6 – SDS administrative perspective. Documents can be retrieved from PubMed or specified by the administrator. The administrator can easily manage document annotation and monitor the status of the documents.



Only include categories: Compound × Protein × Drug × Add some categories

From year: 1990 To year: 2013

SDS has found a total of **426** articles for the search query "antimicrobial peptide".

The possible role of antimicrobial proteins in obesity-associated immunologic alterations
[aggregated tf-idf score: 10.991]

MyD88 deficiency alters expression of antimicrobial factors in mouse salivary glands
[aggregated tf-idf score: 9.630]

The importance of antimicrobial peptides and their potential for therapeutic use in ophthalmology
[aggregated tf-idf score: 7.327]

SDS has found a total of **121** articles for the search query "antimicrobial peptide".

Expanded genetic code for the engineering of ribosomally synthetized and post-translationally modified peptide natural products (RiPPs)
[aggregated tf-idf score: 3.219]

The genome of tolypocladium inflatum: evolution, organization, and expression of the cyclosporin biosynthetic gene cluster
[aggregated tf-idf score: 2.708]

Draft genome sequence of Paenibacillus sp. strain OSY-SE, a bacterium producing the novel broad-spectrum lipopeptide antibiotic paenibacterin

Central glucagon like peptide 1 receptor induced anorexia requires glucose metabolism mediated suppression of AMPK and is impaired by central fructose
[aggregated tf-idf score: 2.398]

The genome of tolypocladium inflatum : evolution, organization, and expression of the cyclosporin biosynthetic gene cluster.

Year: 2013

Authors: Kathryn E Bushley, Rajani Raja, Pankaj Jaiswal, Jason S Cumbie, Mariko Nonogaki, Alexander E Boyd, C Alisha Owensby, Brian J Knaus, Justin Elser, Daniel Miller, Yanming Di, Kerry L McPhail, Joseph W Spatafora

Abstract: The ascomycete fungus Tolypocladium inflatum , a pathogen of beetle larvae, is best known as the producer of the immunosuppressant drug cyclosporin. The draft genome of T. inflatum strain NRRL 8044 (ATCC 34921), the isolate from which cyclosporin was first isolated, is presented along with comparative analyses of the biosynthesis of cyclosporin and other secondary metabolites in T. inflatum and related taxa. Phylogenomic analyses reveal previously undetected and complex patterns of homology between the nonribosomal peptide synthetase (NRPS) that encodes for cyclosporin synthetase (simA) and those of other secondary metabolites with activities against insects (e.g., beauvericin, destruxins, etc.), and demonstrate the roles of module duplication and gene fusion in diversification of NRPSs . The secondary metabolite gene cluster responsible for cyclosporin biosynthesis is described. In addition to genes necessary for cyclosporin biosynthesis, it harbors a gene for a cyclophilin , which is a member of a family of immunophilins known to bind cyclosporin. Comparative analyses support a lineage specific origin of the cyclosporin gene cluster rather than horizontal gene transfer from bacteria or other fungi. RNA-Seq transcriptome analyses in a cyclosporin-inducing medium delineate the boundaries of the cyclosporin cluster and reveal high levels of expression of the gene cluster cyclophilin . In medium containing insect hemolymph, weaker but significant upregulation of several genes within the cyclosporin cluster, including the highly expressed cyclophilin gene , was observed. T. inflatum also represents the first reference draft genome of Ophiocordycipitaceae, a third family of insect pathogenic fungi within the fungal order Hypocreales, and supports parallel and qualitatively distinct radiations of insect pathogens. The T. inflatum genome provides additional insight into the evolution and biosynthesis of cyclosporin and lays a foundation for further investigations of the role of secondary metabolite gene clusters and their metabolites in fungal biology.

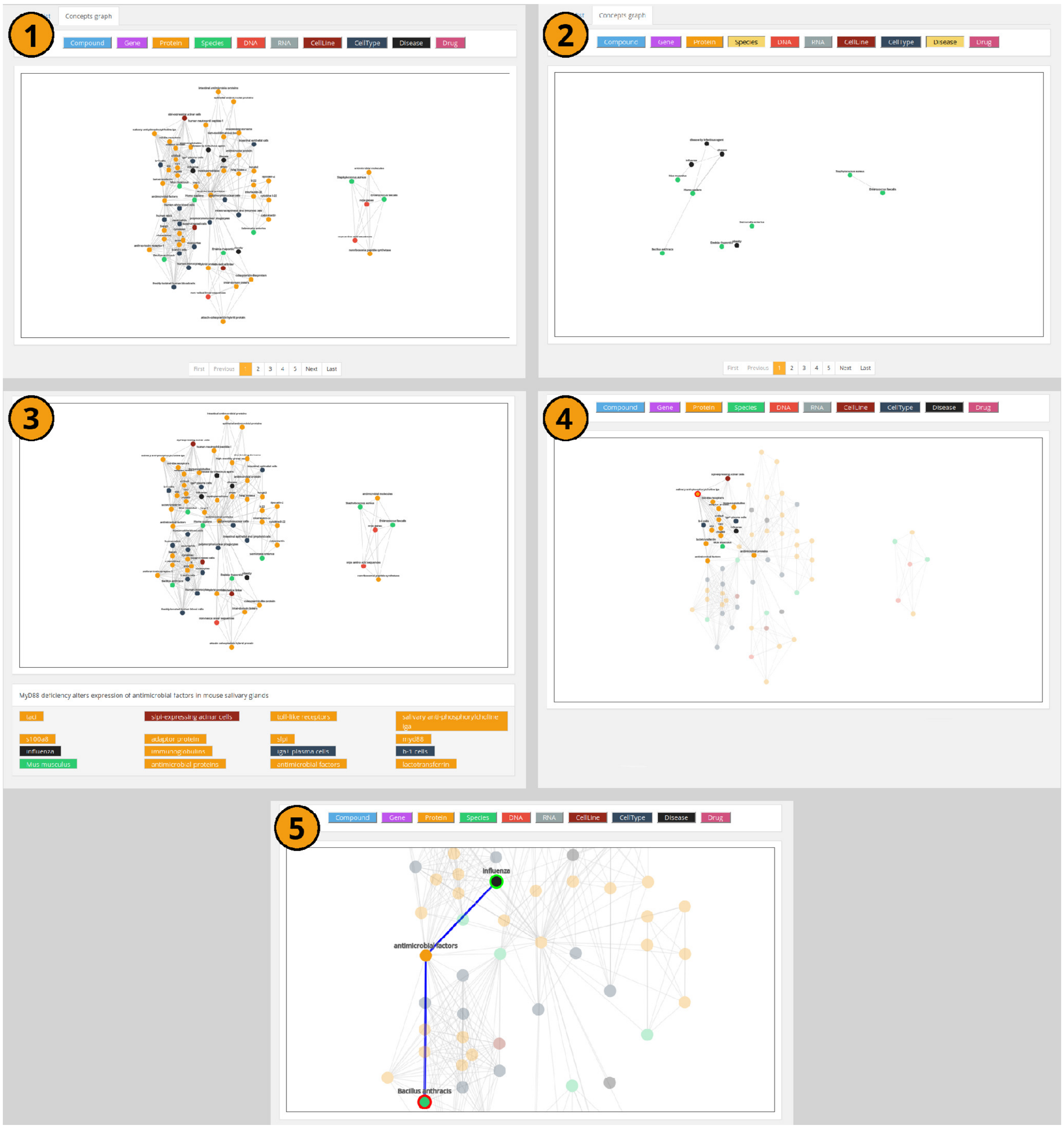
Fig. 7 – SDS search interface and simple results presentation. Users can issue keyword-based Boolean queries (here "antimicrobial peptide" is used as example) and/or use concept-based (e.g. only containing compounds, proteins and drugs) and document-based filters (e.g. publication date). Keywords are mapped to concepts and documents are ranked by semantic score. The list of retrieved documents can be narrowed by concepts of interest.

shortest path between the selected nodes, assuming that an association with fewer intermediary nodes is stronger.

3.4. *Application verification*

The different modules that compose the BIOMedical Search Engine Framework were tested for their respective execution times within the scope of the SDS application. These tests were performed with SDS running on a personal computer with a four-core Intel Core i7 processor, 8 GB of RAM, a Linux distribution executing the kernel's version 4.3.3 and the OpenJDK distribution of the Java Runtime Environment on version 8.u60.

The execution of the PubMed provider module was tested by keeping track of the download times of the documents



MyD88 deficiency alters expression of antimicrobial factors in mouse salivary glands

taci	slpi-expressing acinar cells	toll-like receptors	salivary anti-phosphorylcholine iga
s100a8	adaptor protein	slpi	myd88
influenza	immunoglobulins	iga1 plasma cells	b-1 cells
Mus musculus	antimicrobial proteins	antimicrobial factors	lactotransferrin

Fig. 8 – Exploring results in SDS. The graph representation enables a visually appealing presentation of the concepts in results. Concept-driven filters may be used to get specialized views as well as help find indirect associations.

currently stored in the database. The average execution time is 24 ms per document considering a variation in time between 6 ms per document (the minimum execution time observed) and 40 ms per document (the maximum execution time observed). As illustrated in Fig. 9, the execution time throughout the database population process, i.e. the download and storage of the 1,182,670 documents in the current SDS database, was mostly constant, with little deviations from the average time.

The population of the SDS database was performed by requesting page sizes of 200 abstracts from PubMed. While the average time to download a single document does not vary greatly with respect to page size, the aggregated time can be significantly affected. Although choosing the right page size depends mostly on network capacity, a very small page size (e.g. 1 document or a couple of documents) or a very large page size (e.g. 1000 documents) becomes ultimately slower than page sizes on the order of a few hundreds (e.g. 200 or 300 documents per page).

Next, we verified the annotation process in SDS. The annotation of the more than one million documents in the

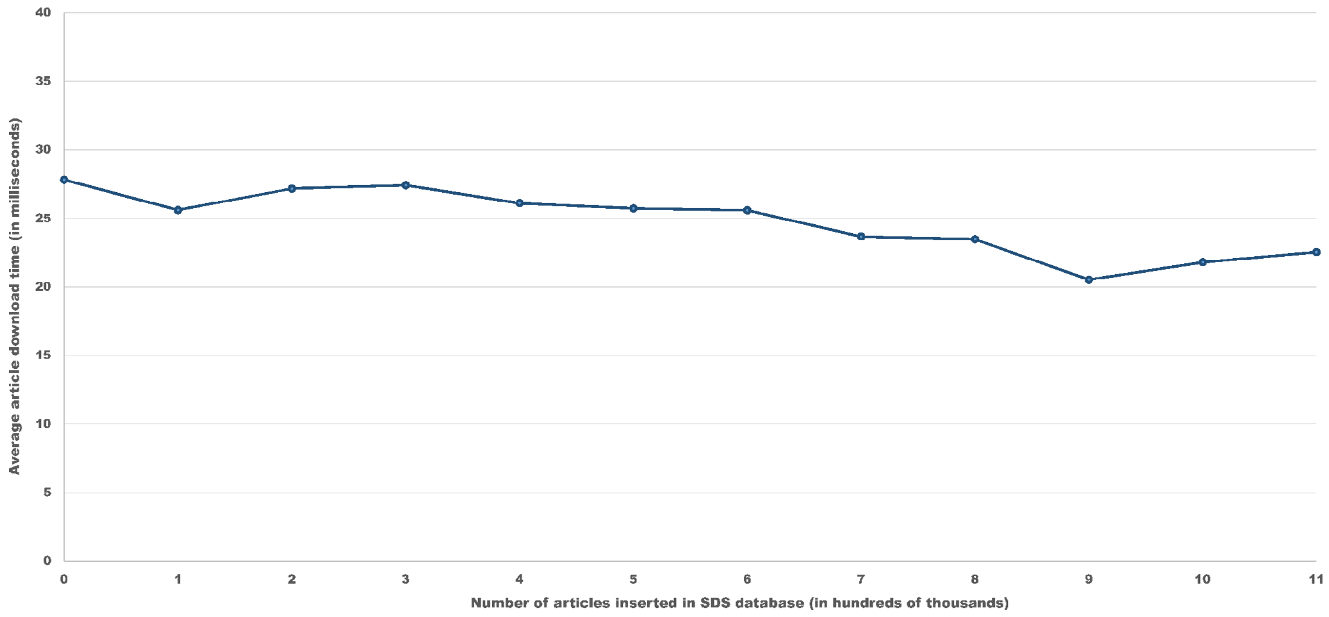


Fig. 9 – Execution of the PubMed provider module in the SDS application. Statistics are calculated for every hundred thousand documents downloaded and inserted in the database.

database led to an average annotation time of 374 ms per document. As illustrated in Fig. 10, the annotation time was quite variable, and while most of the documents took less than 50 ms to be annotated, a non-neglected fraction of the documents took from 100 ms to 350 ms to be annotated. This variability in the annotation time was expected, because this process depends mostly on the contents of the document being annotated and very little on the state of the actual system.

It is also noteworthy that the annotation execution time discussed here does not include the insertion of the annotations in the database, only the automatic recognition of the biomedical entities. The insertion of records in the database can impose some additional delays depending on how the database is configured to handle the insertion and retrieval of annotations in bulk.

Arguably, the indexer module is the most time-consuming of all the modules in the BIOMedical Search Engine Framework. Considering the current volume of documents and annotations managed by SDS, the indexation process took a total time of 8 hours. Under this scenario, it is highly

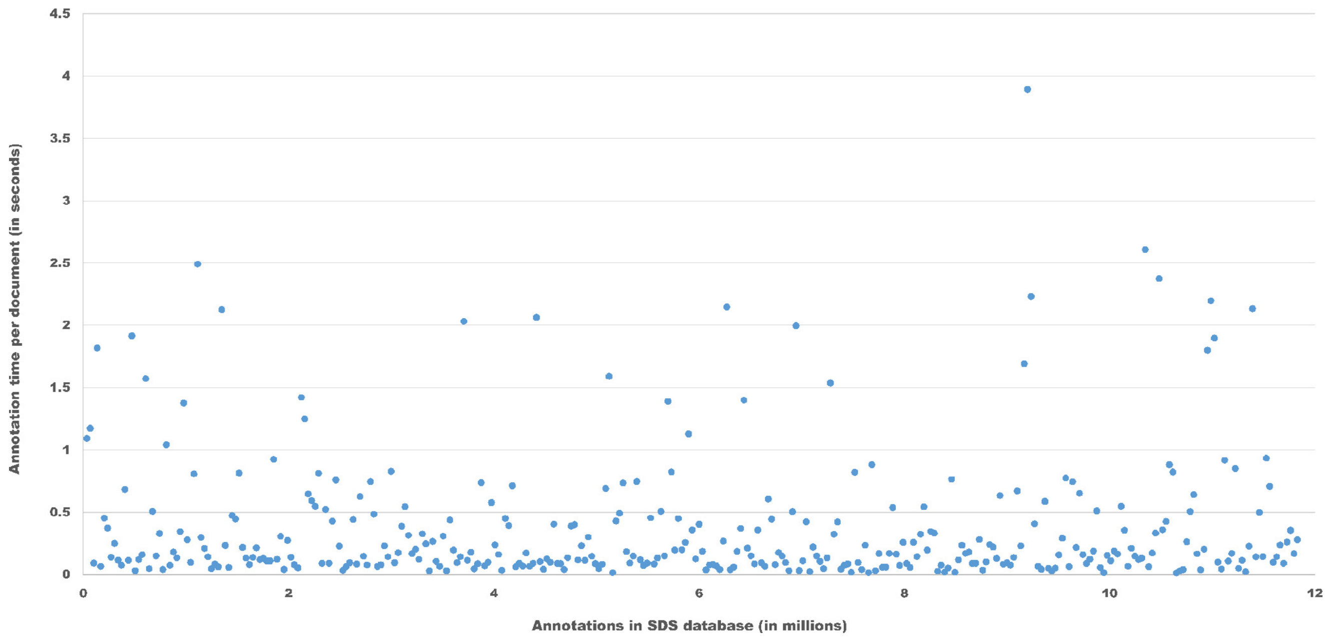


Fig. 10 – Execution of the annotation module in the SDS application. The execution time was calculated per document for the overall amount of documents currently in the database.

recommended that medium to large-scale applications, such as SDS, will be executed on a machine with high computing power. Conversely, if the data volume to be handled is relatively small, the application could be easily executed on a personal computer. For example, in an earlier version of SDS, the indexation of 30,000 documents and 360,000 annotations took only 40 minutes.

Finally, we tested the searcher module in order to determine the average response time to a search query. For this purpose, we executed 20 search queries with varying complexity (with respect to the total of articles retrieved, ranging from very few results to almost half the database) over various search indexes: 250,000 indexed documents, 500,000 indexed documents, 750,000 indexed documents and the complete SDS database of more than 1 million documents (Table 4). The documents included in each search index were selected randomly from the current SDS database.

As illustrated in Fig. 11, those queries that return a greater number of results (i.e. are more generic) tend to take longer to execute than those with less matching documents (i.e. are more specific). At the same time, one can observe that the volume of the search index is a key factor in the execution time, i.e. most of the execution time is spent on the retrieval of records from the database, and the translation of search terms into domain concepts is negligible.

In our worst scenario, i.e. a query that returns almost 500,000 records from the complete SDS index (Q9), the execution time is of 13 seconds approximately. We consider this time acceptable, in particular considering that a query that returns so many results is necessarily very generic, and it is unlikely that SDS users will issue such broad scope queries. Moreover, it is important to note that the query execution tests have been performed without memorization of query results, which will improve the execution time for subsequent equivalent queries in a real-world scenario. For example, if the user issues the query "human" and then searches for "homo sapiens", the second query will complete almost instantaneously, because the results for the former query will be already cached in memory.

The performance of the searcher module is the most noticeable to the end users of SDS. Therefore, the next releases of the BIOMedical Search Engine Framework and the SDS application will address the optimization of this module and, in particular, the mitigation of the time spent on the retrieval of records from the database.

4. Conclusions

This paper introduces the BIOMedical Search Engine Framework, which supports the construction of domain-specific literature search engines for biomedical applications. This framework integrates consolidated search engine technologies and state-of-the-art text mining methods and tools. Its primary aim is to take advantage of semantic analysis to enhance domain-specific search experience while minimizing the programming costs of elementary document retrieval and processing. Its modular architecture enables the customization and interchange of components to meet the

Table 4 – Number of documents retrieved for distinct queries and search indexes.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
250K	6,794	2,079	19,934	21	0	82	1,304	21	76,156	36,688	186	11,489	7,544	18,634	1,327	1,2761	87	374	524	144
500K	13,999	3,952	38,748	46	28	183	3,642	25	173,037	81,212	232	21,159	15,519	40,874	3,924	24,461	214	723	1,358	286
750K	20,447	5,489	58,504	167	59	403	5,860	290	279,743	103,173	255	35,017	26,647	60,861	6,072	33,945	461	1,587	2,589	620
1M	40,794	9,868	123,557	1,101	100	1,350	11,674	804	453,158	120,378	276	78,652	44,563	84,413	18,517	54,455	1316	2,532	3,376	1,233

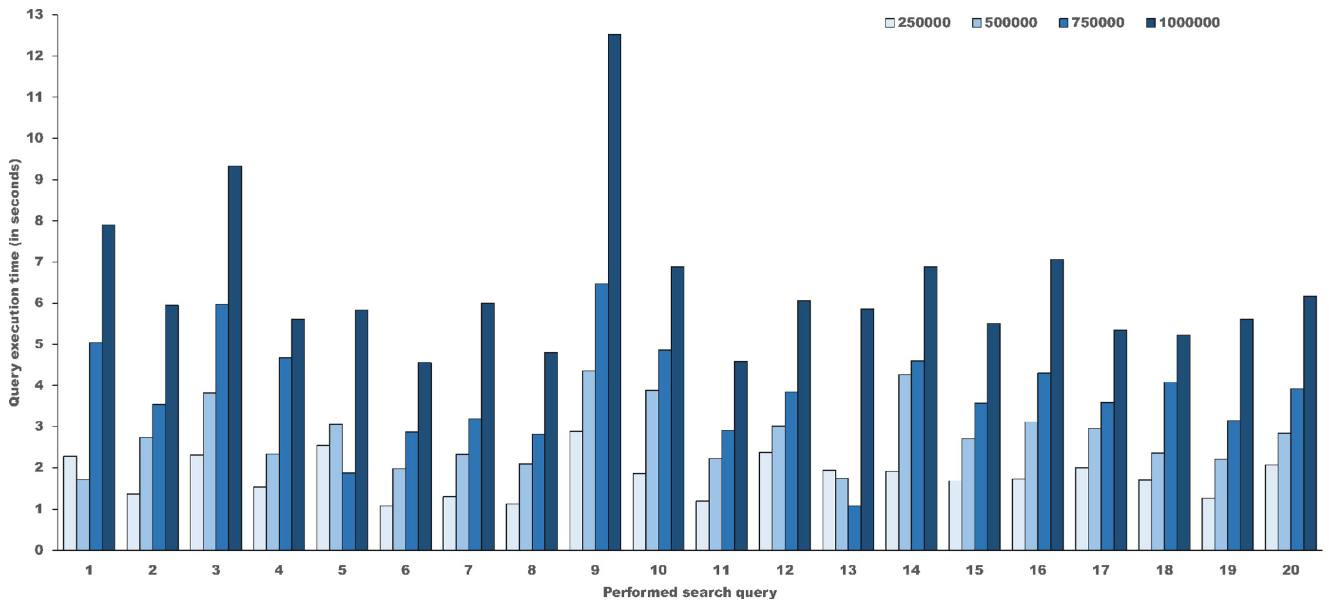


Fig. 11 – Execution of the indexation module in the SDS application. The execution time was calculated for distinct queries and considering different search indexes.

requirements of a particular domain of application. It also provides a RESTful API that enables the integration of the engine in existing systems and the complete personalization of the web interface.

The Smart Drug Search system was built on top of this framework and aims to assist researchers in the study of alternative and novel antimicrobial products. This engine uses NER tools to automatically recognize textual mentions to drugs/compounds, diseases, species, proteins and genes in the articles, and construct an efficient search index with those domain-specific concepts.

Administrator and user experience with SDS provided valuable insight on how to keep developing the framework. Future versions of the framework will provide an easier mechanism to include different document sources (e.g. patents or full-text collections) in the same way that different annotators and searchers can be already included. At the same time, the web-based interface is being enhanced to deliver a richer and more visually appealing experience to the end users. The graph view of search results is being explored in that direction, but more web-interface capabilities are already being planned.

5. Mode of availability

The code of the BIOMedical Search Engine Framework and its documentation are accessible at https://github.com/agjacome/biomsef and freely available under MIT License. The Smart Drug Search engine can be accessed at http://sing.ei.uvigo.es/sds/.

Acknowledgements

This work was partially funded by the [14VI05] Contract-Programme from the University of Vigo and the Agrupamento INBIOMED from DXPCTSUG-FEDER unha maneira de facer Europa (2012/273). The research leading to these results has also received funding from the European Union Seventh Framework Programme FP7/REGPOT-2012-2013.1 under grant agreement n° 316265, BIOCAPS. This document reflects only the author's views, and the European Union is not liable for any use that may be made of the information contained herein.

REFERENCES

[1] N.R. Coordinators, Database resources of the National Center for Biotechnology Information, Nucleic Acids Res. 44 (D1) (2014) D7–D19.
[2] Z. Lu, PubMed and beyond: a survey of web tools for searching biomedical literature, Database 2011 (2011) 1–13.
[3] I. Shemilt, A. Simon, G.J. Hollands, T.M. Marteau, D. Ogilvie, A. O'Mara-Eves, et al., Pinpointing needles in giant haystacks: use of text mining to reduce impractical screening workload in extremely large scoping reviews, Res. Synth. Methods 5 (1) (2014) 31–49.
[4] M. Lee, Z. Liu, R. Kelly, W. Tong, Of text and gene—using text mining methods to uncover hidden knowledge in toxicogenomics, 2014, 1–11.
[5] U. Consortium, UniProt: a hub for protein information, Nucleic Acids Res. 43 (Database issue) (2015) D204–D212.
[6] R. Caspi, T. Altman, R. Billington, K. Dreher, H. Foerster, C.A. Fulcher, et al., The MetaCyc database of metabolic pathways and enzymes and the BioCyc collection of Pathway/Genome Databases, Nucleic Acids Res. 42 (D1) (2014).
[7] A. Chang, I. Schomburg, S. Placzek, L. Jeske, M. Ulbrich, M. Xiao, et al., BRENDA in 2015: exciting developments in its 25th year of existence, Nucleic Acids Res. 43 (Database issue) (2014) D439–D446.
[8] M.Y. Galperin, D.J. Rigden, X.M. Fernández-Suárez, The 2015 Nucleic Acids Research Database Issue and molecular biology database collection, Nucleic Acids Res. 43 (Database issue) (2015) D1–D5.

[9] S. Kim, D. Kwon, S.-Y. Shin, W.J. Wilbur, PIE the search: searching PubMed literature for protein interaction information, Bioinformatics 28 (4) (2012) 597–598.
[10] T. Fujiwara, Y. Yamamoto, Colil: a database and search service for citation contexts in the life sciences domain, J. Biomed. Semantics. 6 (2015) 38.
[11] J.A. Gijón-Correas, M.A. Andrade-Navarro, J.F. Fontaine, Alkemio: association of chemicals with biomedical topics by text and data mining, Nucleic Acids Res. 42 (W1) (2014) 422–429.
[12] Y. Tsuruoka, M. Miwa, K. Hamamoto, J. Tsujii, S. Ananiadou, Discovering and visualizing indirect associations between biomedical concepts, Bioinformatics 27 (13) (2011) i111–i119.
[13] Y. Liu, Y. Liang, D. Wishart, PolySearch2: a significantly improved text-mining system for discovering associations between human diseases, genes, drugs, metabolites, toxins and more, Nucleic Acids Res. 43 (W1) (2015) W535–W542.
[14] J.Z. Wang, Y. Zhang, L. Dong, L. Li, P.K. Srimani, P.S. Yu, G-Bean: an ontology-graph based web tool for biomedical literature retrieval, BMC Bioinformatics 15 (Suppl. 1) (2014) S1.
[15] A. Malhotra, M. Gündel, A.M. Rajput, H.-T. Mevissen, A. Saiz, X. Pastor, et al., Knowledge retrieval from PubMed abstracts and electronic medical records with the Multiple Sclerosis Ontology, PLoS ONE 10 (2) (2015) e0116718.
[16] H.-J. Lee, T.C. Dang, H. Lee, J.C. Park, OncoSearch: cancer gene search engine with literature evidence, Nucleic Acids Res. 42 (Web Server issue) (2014) W416–W421.
[17] J. Kim, S. So, H.-J. Lee, J.C. Park, J.-J. Kim, H. Lee, DigSee: disease gene search engine with evidence sentences (version cancer), Nucleic Acids Res. 41 (Web Server issue) (2013) W510–W517.
[18] C. Knox, V. Law, T. Jewison, P. Liu, S. Ly, A. Frolkis, et al., DrugBank 3.0: a comprehensive resource for 'omics' research on drugs, Nucleic Acids Res. 39 (Database issue) (2011) D1035–D1041.
[19] O. Bodenreider, The Unified Medical Language System (UMLS): integrating biomedical terminology, Nucleic Acids Res. 32 (Database issue) (2004) D267–D270.
[20] R. Cornet, N. de Keizer, Forty years of SNOMED: a literature review, BMC Med. Inform. Decis. Mak. 8 (Suppl. 1) (2008) S2.
[21] R. Hoehndorf, P.N. Schofield, G.V. Gkoutos, The role of ontologies in biological and biomedical research: a functional perspective, Brief. Bioinform. 16 (6) (2015) 1069–1080.
[22] G. Salton, C. Buckley, Term-weighting approaches in automatic text retrieval, Inf. Process. Manag. 24 (5) (1988) 513–523.
[23] M. Odersky, L. Spoon, B. Venners, Programming in Scala: A Comprehensive Step-by-Step Guide, Artima Incorporation, 2011.
[24] P. Hilton, E. Bakker, Play for Scala, Manning Publications, 2013.
[25] J. Allen, Effective Akka, O'Reilly, 2013.
[26] D. Wyatt, Akka Concurrency, Artima Inc, 2013.
[27] NCBI Resource Coordinators, N.R. Coordinators, Database resources of the National Center for Biotechnology Information, Nucleic Acids Res. 41 (Database issue) (2014) D8–D20.
[28] B. Settles, ABNER: an open source tool for automatically tagging genes, proteins and other entity names in text, Bioinformatics 21 (14) (2005) 3191–3192.
[29] D.M. Jessop, S.E. Adams, E.L. Willighagen, L. Hawizy, P. Murray-Rust, OSCAR4: a flexible architecture for chemical text-mining, J. Cheminform. 3 (1) (2011) 41.
[30] M. Gerner, G. Nenadic, C.M. Bergman, LINNAEUS: a species name identification system for biomedical literature, BMC Bioinformatics 11 (2010) 85.
[31] L.M. Schriml, C. Arze, S. Nadendla, Y.-W.W. Chang, M. Mazaitis, V. Felix, et al., Disease ontology: a backbone for disease semantic integration, Nucleic Acids Res. 40 (Database issue) (2012) D940–D946.