Contents

1 CITREC Parser Documentation

1.1 PubMed Central Open Access Subset

The PubMed Central Open Access Subset (PMC OAS) is a collection of openly available articles from the life sciences. The collection represents a subset of the articles in PubMed Central, which is a digital archive of full texts maintained by the National Center for Biotechnology Information (NCBI). The NCBI is a subunit of the U.S. National Library of Medicine (NLM) [29]. The NLM and NCBI maintain a large number of data sources and information systems that are partially related to the PMC OAS. For clarity of the terminology used hereafter, Figure 1 gives an overview of related systems.

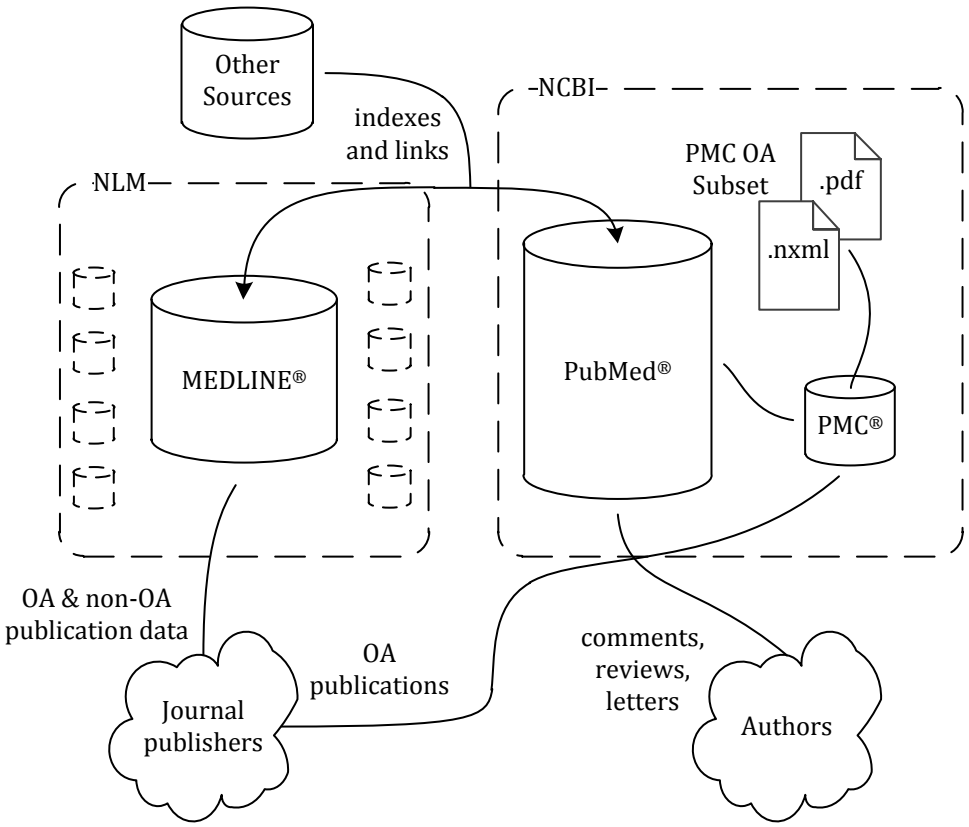

Figure 1: Data sets and information systems related to the evaluation corpus

MEDLINE is the most comprehensive index of literature in the life sciences. It does not contain full texts, but structured records of bibliographic data for articles published in peer-reviewed journals. Experts at the NLM manually assigned Medical Subject Headings (MeSH) to MEDLINE records. MeSH are a poly-hierarchical controlled vocabulary thesaurus comprising approx. 28,000 topic descriptors used to organize the respective documents.

PubMed is a free information system provided by the NCBI and the most prominent way of accessing data from MEDLINE. PubMed integrates nearly all data from MEDLINE and additional content that is still in the process of being formally included in MEDLINE or out of scope for MEDLINE.

PMC full texts mostly correspond to records in MEDLINE and/or PubMed. Some additional content, which does not necessarily meet the formal criteria for being included in MEDLINE or PubMed, such as editorial letters, comments, book reviews, conference summaries or non-journal manuscripts, are part of PMC as well [28]. Articles in the PMC OAS are provided in two document formats – as NXML texts, which is a XML dialect developed by the National Library of Medicine (NLM), and as PDF files. NXML files in the PMC OAS comply with the Journal Archiving and Interchange Tag Set (JAITS), which is a set of XML schema and DTD specifications provided by the NLM for digital archiving purposes.

We used the NXML texts in the PMC OAS for data extraction to the CITREC database, because opposed to PDF files, NXML texts provide well-defined, comparably easy to parse markup for most of the information that is needed in the CITREC framework. Yet, the NXML format does not encode information about word- and sentence boundaries that is used by some citation-based similarity measures in the CITREC framework. Therefore, we had to determine and extract this information in addition to the one encoded by means of the original NXML markup.

1.2 Word and Sentence Boundary Detection

The ambiguous nature of natural speech makes automated word and sentence boundary detection challenging. The grapheme of a period for instance is very ambiguous. For instance, a period can indicate the end of a sentence, an abbreviation, a decimal point, or delimitations within an email address. Specifics of life science domain contribute additional challenges. Articles in those fields frequently mention chemical or medical substances and structures, specialized abbreviations or other domain specific entities that are hard to match to common sentence structures and patterns. Given these challenges, we sought to incorporate a part-of-speech (POS) tagger that was specifically designed for the life sciences, thus can achieve a good detection performance.

We used the SPToolkit [18], which offers a lightweight and easy to integrate Java™-based sentence and paragraph tagger, called SentParDetector. The detector is based on heuristic rules designed for the life sciences. The developers of SentParDetector empirically evaluated its detection performance using the GENIA corpus, which is a set of 2,000 manually annotated MEDLINE® abstracts compiled for training and evaluation purposes [7]. The GENIA corpus resembles texts from the domain bat hand very well. SentParDetector achieved a precision of 0.997 and a recall of 0.9953 in this evaluation [18]. Due to its lightweight heuristic rules, SentParDetector is computationally far less demanding than full featured POS tagger such as the widely used systems OpenNLP [26] and StanfordCoreNLP [27]. The latter apply sophisticated machine learning approaches. We compared SentParDetector to the

aforementioned OpenNLP and StanfordCoreNLP concerning their detection and runtime performance. While all three delivered very accurate results, SentParDetector needed an average processing time of 30ms per document, while the other two systems needed ~1.5s per document.

Disadvantages of SentParDetector are its disability to process XML texts the missing functionality of word boundary detection. We overcame the inability to process XML format by substituting all XML tags in the original documents prior to performing any sentence or word detection. Unique placeholder strings that do not interfere with the detection heuristics of SentParDetector were used. The tag information was stored in an index for reinsertion after the sentence and word markup process. SentParDetector can be incorporated as a plain Java object that returns the output of the analysis as plain Java Strings. This feature allows to perform the substitution and reinsertion process as part of the calling application and therefore within main memory. The additional step causes an increase in the average processing time of 6 ms per document.

We develop own heuristic rules for word boundary detection using regular expressions based on known word segmentation heuristics. Because we ran SentParDetetcor for sentence annotation before word-boundary detection the heuristics have to consider the specific markup introduced by by the tagger. SentParDetector lists each sentence in a separate line and marks the beginning with a known XML style markup. This markup as all as the original XML markups according to the JAITS DTD were replaced with the following character sequences:

`*§S/§` - denoting the start of a sentences

`_Z*§000/§` - denoting an individual XML tag that was replaced with this placeholder string. The numbering 000 corresponds to an individual unique ascending number for each tag in the document. This way the original tags can be reinserted after the sentence and word markup process for retaining the original document structure information.

The following regular expression comprising alternative tests for 2 different main patterns that we found. In contrast to the actual source code, we display the expressions in multiple segments for giving additional comments.

The 1st pattern, which the input string is checked for, basically represents words separated by one or multiple whitespaces. It is matched by searching for the last alphanumeric character in a character sequence (word or numeric expressions) that is not part of a markup substitution mentioned above. This represents the ending of any term considered to be a word. Note that the heuristics treat numeric expressions or abbreviations as words.

```
“(?:(?:[a-zA-Z0-9](?!\\w|/§|([\\.,]?[0-9]+?)))”+
```

This first sub-pattern can optionally be followed by all non-alphanumeric characters except for white spaces or alphanumeric characters that are part of a markup substitution. In allowing this option e.g. punctuation marks or brackets between words are ignored.

```
“(?:[^\\w\\s]|(?:[S0-9]+(?=/§)))*”+
```

The distinguishing feature that needs to be given for the first pattern to be evaluated as fulfilled is a mandatory white space.

```
“(?:\\s)”+
```

Again, the whitespace can optionally be followed by all non-alphanumeric characters except whitespaces and alphanumeric characters that are part of a markup substitution, which can be thought of as allowing additional punctuation marks, brackets and alike before the second word starts.

```
“(?:[^\\w]|(?:S/§)|(?:[0-9]+(?=/§)))*”+
```

The last sub-pattern that mandatorily needs to be present is an alphanumeric character not being part of a markup substitution. This represents the start of the following word.

```
“(?:[a-zA-Z0-9](?!/§)))”+
```

If the regular expression engine fails to positively match the 1st main pattern it alternatively checks for a 2nd one.

```
“|”+
```

This 2nd pattern basically represents words in the original text that were separated by an XML tag, but no whitespaces. It is matched by looking for the last alphanumeric character in a sequence (word or numeric expression) that is not part of a markup substitution, but directly followed by such a substitution.

```
“(?:(?:[a-zA-Z0-9](?!(?:\\w)|(?:/§)|(?:[\\.,]?
[0-9]+?))[,\\.;\\?!\"'\\=/:&+\\-\\$%°]*(?=\\*§[0-9]+/§))” +
```

This first sub-pattern can optionally be followed by arbitrary characters, which allows for punctuation marks, brackets or sentence boundaries between words.

```
“(?:[^\\w]|(?:S/§)|(?:[0-9]+(?=/§)))*”+
```

The end of the 2nd main pattern must be an alphanumeric character not being part of a markup substitution, which represents the start of the following word.

```
“(?:[a-zA-Z0-9](?!(?:/§))))”
```

In a last step, the original XML markup is restored after sentence and word boundaries are detected and labeled using individual non-XML style markup. Using plain character annotations for indicating word and sentence boundaries instead of introducing additional XML tags had been chosen as a convenience solution. Additional integrity procedures would have been required when word and sentence tagging should have been done using XML for ensuring that original and additional markups form a valid XML structure.

For assuring the quality of the combined markup procedure, we manually analyzed a small, randomly chosen sample of 4 articles from 4 different journals. The word, sentence and paragraph markup of 3 paragraphs in each document that were taken from the respective first, second or last third of each text were analyzed. The results are depicted in Table 1. The notation in the table reflects common IR terminology stating true positives (tp), false positive (fp), true negatives (tn), false negatives (fn), precision $P = \frac{|tp|}{|tp|+|fp|}$ and recall $R = \frac{|tp|}{|tp|+|fn|}$. Five of the 6 false positives found originate from one document using place and tribe names in native African languages that comprise unusual combinations of diacritics and hyphens.

Words						Sentences						Paragraphs					
tp	**fp**	**tn**	**fn**	**P**	**R**	**tp**	**fp**	**tn**	**fn**	**P**	**R**	**tp**	**fp**	**tn**	**fn**	**P**	**R**
2,092	6	2,092	0	0.9971	1	84	0	0	0	1	1	12	0	0	0	1	1

Table 1: Accuracy evaluation for sentence and word tagging

This small sample cannot represent a solid statistical evaluation. Nevertheless, the markup accuracy of SentParDetector reported in earlier tests [18] seems to be reproducible for the texts in the PMC OAS. The defined rules for word markup can be assumed to yield a precision and recall being above 0.95, which is expected to be sufficient for the purpose of CITREC.

1.3 Document Data Parsing and Database Import

We developed a Java-based parser for extracting the desired data from the NXML-texts that had been enhanced with sentence and word markup. The Java programming language offers multiple frameworks for XML document processing, which offer individually different functional ranges and advantages. Widely known are the Java API for XML Processing (JAXP) [13], the Simple API for XML (SAX) [19] and the Streaming API for XML (StAX) [5].

JAXP represents a Java implementation of the Document Object Model (DOM) Specification [33]. DOM manages the logical structure of XML and other structured documents in form of a hierarchical tree. It offers random access operations on any content item in that document tree. In order to enable these free navigation, reading and manipulation functionalities the whole document needs to be read and transformed into the tree structure in advance. In turn, DOM implementations tend to

have higher memory consumption and processing times compared to other XML processing options [6,30], p. 34 f.].

The SAX Application Programming Interface (API) follows a so called push approach in accessing the information contained in XML documents. That is, a document is read sequentially in its entirety by a parser implementing the SAX API. Whenever the occurrence of an event from a predefined set of events is detected, e.g. the start or end of an XML tag, a notification is triggered and sent ("pushed") to the application invoking the parser. It is left to the application how to react on certain events. "Reactions" are defined within callback handlers, which represent special objects that contain the programming logic to be executed in case certain events occurred. Handlers are provided to the parser upon invocation. The SAX parsing process is read only, so XML documents cannot be manipulated by means of the SAX API. Furthermore, no navigation functionality is provided to the invoking application. The traversal of the XML document is strictly sequential, unidirectional (from start to end), completely controlled by the parser and not meant to be interrupted. Due to its simple nature, the SAX parsing approach features comparably low resource consumption and fast processing times [6,30], p. 36 f.].

The StAX API represents an event driven parsing approach similar to SAX. In difference to the SAX approach applications invoking a StAX parser are given full control over the parsing process. That is, they can determine the direction and progression of the iteration as well as define which events to check. This approach is commonly referred to as pull parsing, since application defined events are demanded ("pulled") from the parser. StAX parsers feature an equally efficient runtime performance as SAX parsers while being more versatile and flexible. However, they require comparably more development effort for designing and implementing suitable and efficient parsing procedures [6,30], p. 37 ff.].

The given parsing scenario requires the acquisition of data from a large number of individual documents. Data from all parts of the respective texts is of interest, e.g. publication metadata to be found at the beginning of a document, citation information throughout the text and references at the end of a document. Considering those characteristics, we chose a SAX parsing approach because manipulating functionality is not of relevance in the present situation. The documents need to be read in their entirety and processing them strictly in document order is acceptable. SAX parsers are able to fulfill such tasks with lowest possible resource consumption and high processing speed.

A variety of parsers implementing the SAX interface are available. For the given import task, we chose the open source parser Xerces [25], developed and maintained by the Apache Software Foundation. Xerces repeatedly ranged among the best performing candidates in benchmarks measuring data throughput and parsing speed [6,15]. Its active development by a large open source community is seen as another qualitative advantage.

The main component of the data import procedure is the content handler provided to the SAX Parser. The SAX interface defines 5 basic events reported by the parser to the content handler. These are the recognition of starting and ending tags for arbitrary elements and the special case of encountering starting and ending tags for the overall document. The 5th event is triggered when the parser iterates over literal character data. Content handlers are required to implement 5 corresponding methods for reacting on those events.

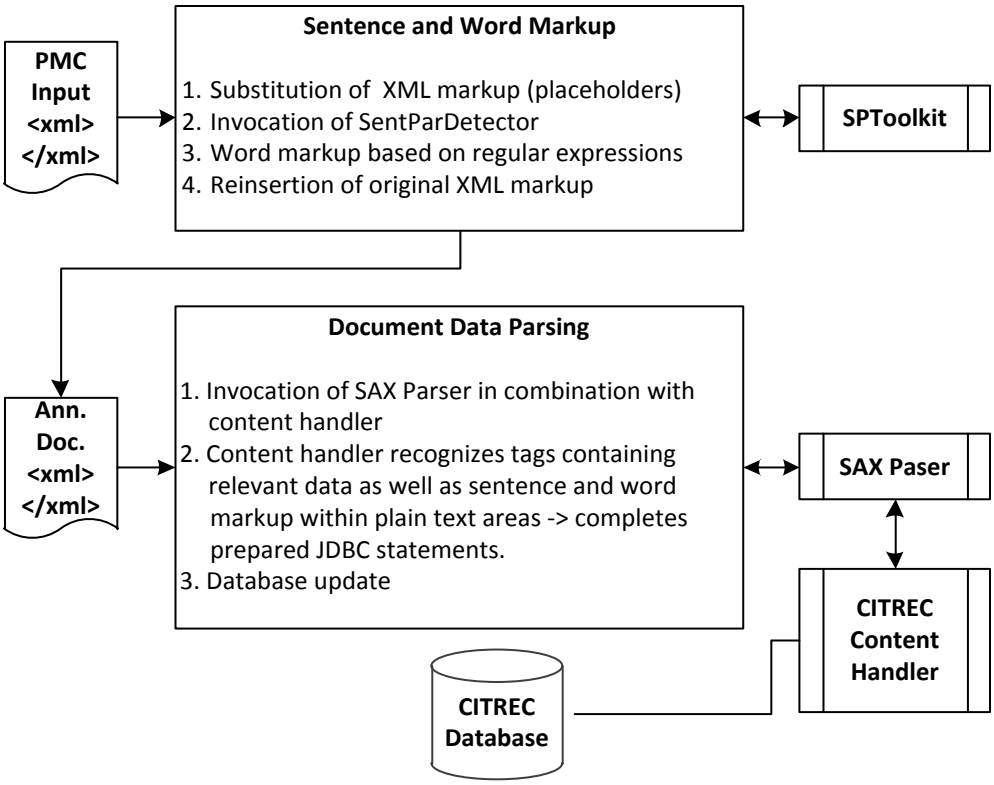

Figure 2: Two stage extraction process for document data

In the given case the methods for handling starting and ending tags were configured to extract data surrounded by tags of interest. It is used to create and/or complete prepared Java objects encapsulating the data to be extracted for the document itself, authors, citations and references. The method for handling literal character data has been extended with regular expressions that are able to recognize the sentence and word markup introduced to the document in the prior process step. Once all data for a certain entity is gathered, the encapsulating object is used to complete and execute prepared database statements, a feature of the Java Database Connectivity (JDBC) API [14], for inserting the relevant data into the CITREC database. The complete two stage data extraction process described so far is graphically summarized in Figure 2.

The average overall processing time for parsing a single document including file system operations and database updates using a single threaded application was 70ms.

1.4 Reference Consolidation

Different unique document identifiers, such as PubMed and MEDLINE identifiers (PMId, MEDId) or DOIs are commonly stated for references in the PMC OAS. Nevertheless, it cannot be assumed that identifiers are used consistently for all possible references. For instance, some authors might state no unique identifiers, some might use a PMId, some others a DOI or vice versa.

For identifying equal references best possible, which is a prerequisite for a citation-based analysis, consolidating available identifiers seems appropriate. The aim is to assign unique identifiers available in the corpus consistently to all references that can be assumed to refer to the actual document characterized by the identifier. To achieve this, valid relations between individual identifiers and documents need to be identified in order to update references missing a certain identifier.

By examining the dataset it has become obvious that given reference identifiers are subject to a certain error rate. This might be caused by human mistake, OCR recognition errors or similar. Errors have been verified with regard to every unique identifier, e.g. a stated PMId did not match the simultaneously given DOI or the document as such. Hence, care must be taken to identify trustworthy mappings of identifiers for preventing that incorrectly assigned identifiers or combinations of such are propagated.

For being able to identify and match references that do not state a unique identifier a key combination has been computed based on the authors and title stated for the referenced document. Examining Authors and titles in reference strings is a natural, nevertheless error prone, way of identifying referenced documents. Small differences in spelling e.g. caused by human mistake or technical reasons, such as varying reference templates or different character encodings, can prevent a positive identification. In the domain of life sciences the problem is worsened by the fact that a significant number of document titles contain special character sequences expressing highly domain-specific entities, e.g. gene and protein sequences, binding points and alike. For this reason, a maximum of 40 plain “ASCII characters” taken from author last names and title converted to lower case was chosen for computing an artificial author and title key.

For identifying equal references best possible, which is a prerequisite for a citation-based similarity analysis, consolidating available identifiers seems appropriate. The aim is to assign unique identifiers available in the corpus consistently to all references that can be assumed to refer to the actual document characterized by the identifier. To achieve this, valid relations between individual identifiers and documents need to be identified in order to update references missing a certain identifier.

For completing or correcting reference records not stating a certain identifier or probably stating an incorrect identifier, the following procedure has been applied.

A unique identifier (refDoc) was introduced to identify referenced documents in the CITREC database. First, all references whose records contain a PMId were processed. Each PMId was considered to identify a unique reference. Other identifiers assigned to the same references were stored as well for subsequent identification of documents that might not state the correct PMId, but another identifier. The mapping of unique refDoc identifiers to the document identifiers obtained from the NXML files is stored in the refdoc_id table of the CITREC database. The table refdoc_seq is used as an auxiliary table to create new reference identifiers for the CITREC database.

The first step is repeated for all documents that have a DOI, but not a PMId assigned to them. If a DOI is encountered that is already known from documents stating a PMID and a DOI, the existing refDoc identifier is assigned to the reference that only stated a DOI so far. Otherwise, a new identifier is generated. All identifiers that are newly discovered in this run are stored as well. In a subsequent analogous iteration documents stating only a MEDId, but no PMId or DOI are processed. After that, documents for which only author and/or title keys could be obtained are consolidated in the same way.

References

[1] M. Ahuja, A. Dhake, S. Sharma, and D. Majumdar. Topical Ocular Delivery of NSAIDs. *The AAPS Journal*, 10: 229–241, 2008. doi: 10.1208/s12248-008-9024-9. PMID18437583, PMC2751374.

[2] C. Alvaro, R. Lyons, G. Warner, S. Hobfoll, P. Martens, R. Labonte, and E. R. Brown. Conservation of resources theory and research use in health systems. *Implementation Science*, 5 (1): 79–98, 2010. ISSN 1748-5908. doi: 10.1186/1748-5908-5-79. PMID20961445, PMC2978118.

[3] J. E. Betancourt and J. M. Rivera. Hexadecameric Self-Assembled Dendrimers Built from 2'-Deoxyguanosine Derivatives. *Organic Letters*, 10 (11): 2287–2290, 2008. doi: 10.1021/ol800701j. PMID18452304, PMC2654094.

[4] E. Brewer. A peripatetic pediatrician's journey into pediatric rheumatology: part III. *Pediatric Rheumatology*, 5 (1): 17, 2007. ISSN 1546-0096. doi: 10.1186/1546-0096-5-17. URL http://www.ped-rheum.com/content/5/1/17. PMID17894862, PMC2045656.

[5] C. Fry and A. Slominski. The Streaming API for XML (StAX). Online Source, June 2006. Retrieved Feb. 12, 2011 from: http://stax.codehaus.org/Home.

[6] S. C. Haw and G. Krishna Rao. A Comparative Study and Benchmarking on XML Parsers. In *The 9th International Conference on Advanced Communication Technology Proceedings*, volume 1, pages 321 – 325, Feb. 2007. doi: 10.1109/ICACT.2007.358364.

[7] J.-D. Kim, T. Ohta, Y. Tateisi, and J. Tsujii. GENIA corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19 (suppl 1): i180–i182, 2003. doi: 10.1093/bioinformatics/btg1023.

[8] K. S. Kim. Occupational Hearing Loss in Korea. *J Korean Med Sci*, 25 (Suppl.): 62–69, 2010. URL http://-synapse.koreamed.org/DOIx.php?id=10.3346%2Fjkms.2010.25.S.S62. PMID21258593, PMC3023345.

[9] A. C. Lanteri. Skin Lesions in the Upper Lip/Nasal Region. *Eplasty*, 11 (ic:2): 1–4, 2011. PMID21253527, PMC3020792.

[10] R. Lionberger. FDA Critical Path Initiatives: Opportunities for Generic Drug Development. *The AAPS Journal*, 10: 103–109, 2008. URL 10.1208/s12248-008-9010-2. PMID18446510, PMC2751455.

[11] S. Monaco, M. Schuchert, and W. Khalbuss. Diagnostic difficulties and pitfalls in rapid on-site evaluation of endobronchial ultrasound guided fine needle aspiration. *CytoJournal*, 7 (1): 1–9, 2010. doi: 10.4103/1742-6413.64385. PMID20607094, PMC2895875.

[12] M. Nadar, C. Bartoli, and M. Kasdan. Lipomas of the Hand: A Review and 13 Patient Case Series. *Eplasty*, 10: 549–559, 2010. PMID 21045920, PMC2964102.

[13] Oracle Corporation. Project JAXP. Online Source, Feb. 2011. Retrieved Feb. 12, 2011 from: http://-jaxp.java.net/.

[14] Oracle Corporation. The Java Database Connectivity (JDBC) API. Online Source, 2011. Retrieved Feb. 12, 2011 from: http://download.oracle.com/javase/6/docs/technotes/guides/jdbc/.

[15] Y. Oren. SAX Parser Benchmarks. Online Source, 2002. Retrieved Feb. 12, 2011 from: http://-piccolo.sourceforge.net/bench.html.

[16] C. Payne, M. Petrovic, R. Roberts, A. Paul, E. Linnane, M. Walker, D. Kirby, A. Burgess, R. Smith, T. Cheasty, et al. Vero Cytotoxin-Producing Escherichia coli O157 Gastroenteritis in Farm Visitors, North Wales. *Emerging Infectious Diseases*, 9 (5): 526, 2003. PMID12737734, PMC2972679.

[17] S. Peddada, D. Umbach, and S. Harris. A response to information criterion-based clustering with order-restricted candidate profiles in short time-course microarray experiments. *BMC Bioinformatics*, 10 (1): 438, 2009. ISSN 1471-2105. doi: 10.1186/1471-2105-10-438. PMID20028515, PMC2813245.

[18] S. Piao and Y. Tsuruoka. A Highly Accurate Sentence and Paragraph Breaker. Online Source, June 2008. Retrieved Jan. 28, 2011 from: http://text0.mib.man.ac.uk:8080/scottpiao/sent_detector.

[19] Project SAX. Simple API for XML (SAX). Online Source, Apr. 2004. Retrieved Feb. 12, 2011 from: http://-www.saxproject.org/.

[20] R. Sakr, P. Marzouk, A. Bricou, F. Demaria, A. Cortez, and J.-L. Benifla. Uterine adenosarcoma associated to lymphovascular emboli: a case report. *Cases Journal*, 2 (1): 7515, 2009. ISSN 1757-1626. doi: 10.1186/1757-1626-2-7515. PMID9829987, PMC2740208.

[21] P. Shah, D.-J. Slebos, P. Cardoso, E. Cetti, G. Sybrecht, and J. Cooper. Design of the exhale airway stents for emphysema (EASE) trial: an endoscopic procedure for reducing hyperinflation. *BMC Pulmonary Medicine*, 11 (1): 1, 2011. ISSN 1471-2466. doi: 10.1186/1471-2466-11-1. URL http://-www.biomedcentral.com/1471-2466/11/1. PMID21214899, PMC3024872.

[22] Z. Sobani, S. Quadri, and S. Enam. Stem cells for spinal cord regeneration: Current status. *Surgical Neurology International*, 1 (1): 93–107, 2010. doi: 10.4103/2152-7806.74240. PM21246060, PMC3019362.

[23] T. W. Sturgill and M. N. Hall. Activating Mutations in TOR Are in Similar Structures As Oncogenic Mutations in PI3KCα. *ACS Chemical Biology*, 4 (12): 999–1015, 2009. doi: 10.1021/cb900193e. PMID19902965, PMC2796128.

[24] Y. Sun and D. Hu. The link between diabetes and atrial fibrillation: cause or correlation? *Journal of Cardiovascular Disease Research*, 1 (1): 10–11, Jan. 2010. doi: 10.4103/0975-3583.59978. PMID21188083, PMC3004163.

[25] The Apache Software Foundation. Xerces2 Java Parser Readme. Online Source, 2010. Retrieved Feb. 12, 2011 from: http://xerces.apache.org/xerces2-j/.

[26] The Apache Software Foundation. Apache OpenNLP. Online Source, 2010. Retrieved Feb. 12, 2011 from: http://incubator.apache.org/opennlp/.

[27] The Stanford Natural Language Processing Group . Stanford CoreNLP - A Suite of Core NLP Tools. Online Source, Nov. 2010. Retrieved Feb. 12, 2011 from: http://nlp.stanford.edu/software/corenlp.shtml.

[28] U.S. National Center for Biotechnology Information. PubMed Central,. Online Source, 2011. Retrieved Sept. 27, 2011 from: http://www.ncbi.nlm.nih.gov/pmc/.

[29] U.S. National Library of Medicine. MEDLINE®/PubMed® Resources Guide. Online Source, 2011. Retrieved Sept. 29, 2011 from: http://www.nlm.nih.gov/bsd/pmresources.html.

[30] A. Vohra and D. Vohra. *Pro XML Development with Java Technology*. Apress, Berkely, CA, USA, 2006. ISBN 1590597060.

[31] M. Wall and N. Dale. Activity-Dependent Release of Adenosine: A Critical Re-Evaluation of Mechanism. *Current Neuropharmacology*, 6: 329–337(9), December 2008. doi: 10.2174/157015908787386087. PMID19587854, PMC2701281.

[32] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0 (Fifth Edition). Open Standard, 2008.

[33] World Wide Web Consortium (W3C). Document Object Model (DOM). Online Source, Jan. 2009. Retrieved Feb. 12, 2011 from: http://www.w3.org/DOM/.