Get All, Filter Details - On the Use of Regular Expressions in SPARQL Queries

Saud Aljaloud, Markus Luczak-Rösch, Tim Chown and Nicholas Gibbins

{sza1g10, m.luczak-rosch, tjc, nmg}@ecs.soton.ac.uk
Electronics and Computer Science
University of Southampton, UK

Abstract. SPARQL is the standardised query language for the RDF data model. To process literal strings, filter expressions can be used with regexes. However, regex can be slow due to its computational complexity. As an initial step towards this area, we present an analysis of regex queries from a large real-world set of queries that have been posed on different SPARQL end-points that represent different domains. We report our findings and deliver some suggestions that can help the performance of regex queries within SPARQL.

Keywords: Regular Expressions, SPARQL, RDF, Triple Stores

1 Introduction

The SPARQL query language [1] allows queries on RDF data to be specified through Basic Graph Patterns (BGP). However, some use cases also require the ability to search within literal values; to this end, SPARQL provides the ability to filter values by matching against a Regular Expression (regex).

However, matching regexes against large corpora (for example, the literal values in a large RDF dataset such as DBpedia) can be computationally expensive; it may not be possible to hold all records in memory, which may result in expensive disk accesses. As a consequence, many SPARQL engines do not provide efficient regex queries, but instead provide conventional Full-Text Searching (FTS), often using existing text search engines as Lucene. however, SPARQL does not provide a standard syntax for querying FTS, so different SPARQL engines adopt different syntaxes/extensions.

In this study, we aim to explore the usage of regex within SPARQL. A new trend in SPARQL query mining is by analysing log files published by some dataset providers. These logs contain the actual queries that have been posed on their SPARQL endpoints; they may contain written SPARQL queries or queries generated by agents. DBpedia benchmark queries have been chosen based on a real query analysis from DBpedia logs [2]. This style of analysis follows in the condition of USEWOD[1], a dedicated workshop that was established in 2011 to study the use of the Semantic Web queries.

[1] `http://data.semanticweb.org/usewod/`

2 Preliminaries and Related Work

In the original SPARQL specifications [3], querying strings was done only through REGEX, whereas SPARQL 1.1 [1] introduces a number of additional functions that are mainly related to XPath functions, namely: STRSTARTS, STRENDS, CONTAINS, STRBEFORE and STRAFTER, which all can be seen as special cases of REGEX. Although FTS has been addressed within the working group[2], it has not been standardised so far; probably, there is no a common definition for the text search language. Moreover, implementing such a service may add technical constraints, whereas most triple stores usually reuse existing solutions as a preferred option, as Andy Seaborne, one of the SPARQL 1.1 editors, suggested[3]. For example, Jena[4], OWLIM-SE[5] and Sesame [4] all use Lucene. Others use their own implementations. For instance, BigData implements a B-tree full-text index[6]. Also, 4Store supports tokenising, double metaphones, and stemming as their FTS features by implementing a unique forward chaining that stores the FTS data in RDF format[7].

Regular expressions are a long-standing and well-studied topic in theoretical computer science; for an in-depth account of the state of the art and current research issues, see [5]. Most modern programming languages such as: java, perl, python and others implement different techniques that usually referred to as extended regex or backtracking. Backtracking-based engines (flavours) usually have more features than what Finite State Machine (FSM)-based engines offer. For example, back referencing, as one of the new features provided by extended regex engines, cannot be implemented within an FSM, as it is considered outside the regular languages [6]. Using extended regex engines, however, introduces new issues. For instance, consider the regex "(a+a+)+b"when run against the string "aaaaaaaab". This pathological regex, sometimes referred to as "Catastrophic Backtracking", can take an exponential time $O(2^n)$ and considered an NP-complete problem [6, 7]. Using FSM-based engines, however, can solve such a pathological regex at most linearly to the size of the input string as $O(n)$.

2.1 Usage Analysis and the Web of Data

Already in 2002 and again in 2004 Berendt et al. introduced Semantic Web mining [8, 9] as a trending research space. The authors describe how the two disciplines, namely the Semantic Web and Web mining, may converge. They present three perspectives from which the so-called Semantic Web usage mining describes best what our research is related to. In 2010 Möller et al. [10] published

[2] http://www.w3.org/2009/sparql/wiki/Feature:FullText

[3] http://mail-archives.apache.org/mod_mbox/jena-users/201306.mbox/%3C51C57EC0.2030601@apache.org%3E

[4] http://jena.apache.org/documentation/query/text-query.html

[5] http://owlim.ontotext.com/display/OWLIMv43/OWLIM-SE+Full-text+Search

[6] http://www.bigdata.com/bigdata/docs/api/com/bigdata/search/FullTextIndex.html

[7] http://4store.org/trac/wiki/TextIndexing

the next notable piece of work in this area. The authors explore reliability, peak-load, performance, usefulness, and attacks as challenges for mining the usage of Linked Data sets. Möller et al. address these challenges by analyzing raw logs in order to learn about user clients, requested content types, and the basic structure of SPARQL queries. Since 2011 the USEWOD workshop series promotes research on usage mining and analysis in the context of the Web of Data [11, 12] and, most impacting on our work, the reference Linked Data endpoint log files are the base of various studies of the real-world use of SPARQL queries amongst other research which is less related to what we are pursuing. Using an analysis of the syntactical and structural use of SPARQL to provide recommendations for index and store designers was subject of the work in [13] and [14]. Related to these studies the optimization of data caching and prefetching based on real-world SPARQL queries was presented in [15]. All these works deal with the optimization of the query performance but none of them inspecting the impact of regular expressions explicitly. A preprocessing algorithm for the in-depth analysis of SPARQL queries for statistical analysis and for data set quality assessment has been introduced in [16, 17]. In particular, this approach allows to derive insight into which parts and structural dependencies of the BGP of SPARQL queries are responsible for queries being unsuccessful in the context of a particular data set and how these problems could be resolved on the data set publisher's side [18]. SPARQL filters and consequently regular expressions are excluded in this approach but mentioned to be a useful next step to be investigated. While this puts emphasis on failing queries, in [19] the machine agent query behaviour on a single data set is studied with the goal to identify typical generic successful SPARQL query patterns applied.

3 Methodology

3.1 Research Data

Our study refers to the USEWOD research data set of the years 2013 and 2014. The collection comprises two formats: mainly, Common Log Format (CLF) logs, whereas the other small portion are in a plain text format. The collection contains queries that have been posed to different SPARQL endpoints. Table 1 shows a general description about each dataset; their triple store, the kind of regex engine, the number of triples[8] and the period covers those queries. Queries may be collected on different dates within the period.

3.2 Processing Method

To extract regex clauses from the log files, we use a SPARQL parser, namely Jena ARQ [20]. In theory, SPARQL is standardised and different triple stores are supposed to parse SPARQL equally. However, this is not always the case as

[8] `http://stats.lod2.eu/rdfdocs?sort=triples` - [accessed on: 01/02/2014]
[9] `http://www.arglist.com/regex/`

Table 1. Log files datasets and their general characteristics

Dataset	Domain	Triple store	Regex engine	Triples	Sample period
DBpedia	cross	Virtuoso	Henry Spencer[9]	$\approx 325m$	01/07/2009 - 27/01/2014
SWDF	Bibliographic	Sesame2	Java.util.regex	246510	01/11/2008 - 22/01/2013
LGD	Geo-spacial	Virtuoso	Henry Spencer	$\approx 226m$	23/05/2011 - 12/01/2014
Bio2rdf	Medical	Virtuoso	Henry Spencer	$\approx 371m$	16/04/2011 - 25/06/2011
OpenBioMed	Medical	Jena TDB	Java.util.regex/Xerces	883000	07/02/2011 - 19/11/2012
BioPortal	Medical	4Store	PCRE	$\approx 203m$	19/07/2012 - 12/12/2013

Table 2. General/regex statistics about the datasets

Dataset	Overall queries	Regex queries	Regex ratio	Regex length	
				Average	Maximum
DBpedia	48,648,011	1,623,710	%3.34	219	983
Bio2rdf	192,081	14	%0.01	144	319
SWDF	27,901,111	64250	%0.23	16	2,258
OpenBioMed	883,376	281	%0.03	12	43
BioPortal	26,375,686	18,293	%0.07	7	54
LGD	3,929,693	133,192	%3.39	20	296
Total	107,929,958	1,839,740	%1.70		

different SPARQL engines may have their own extensions to the language such as: FTS or spatial syntax. Table 2 shows the total number of queries for each dataset, how many of them contain regex filters, the ratio of regex queries against the overall queries and both the average and maximum length of regex patterns. There are 295,671 invalid queries that contain regex; those mainly have syntax errors and not included within Table 2. The Length of a regex pattern represents the entire pattern which includes both normal and meta-characters. For instance, the length of the pattern "$(type|class|subject|broader)$"is 28. Table 3 shows the most two frequent used patterns for each dataset and their occurrences.

4 Analysis

In this this section, we analyse the usage of regex from different perspectives. First, we analyse regex in terms of datasets usage of regexes. Then, we examine the actual regex patterns and their characteristics. Then, we also examine regex variables in the context of BGP and whether they are literals or URIs. Finally, a comparison between regex and FTS is presented within DBpedia dataset.

4.1 Datasets Usage of Regex Filters

In this section, we will go through datasets usage of regexes. First, it is notable that the overall queries from DBpedia comprises 45% leaving the rest for other datasets combined. On the other hand, regex queries are coming from DBpedia by 88%. Moreover, the average length of regex patterns within DBpedia is

Table 3. Most often used regexes for each dataset and their occurrences

Count	Ratio	Regex patterns	Dataset
616728	38%	`.+/((Company)|(Business)|(Company.*)|(CompaniesBasedIn.*)|(.*CompaniesOf.*))$`	DBpedia
199394	12.3%	`.+/((Place)|(PopulatedPlace)|(Town)|(City)|(.*CitiesIn.*))$`	DBpedia
8297	6.2%	`United States`	LGD
5342	4%	`Italy`	LGD
3619	5.6%	`(label|summary|name)$`	SWDF
1676	2.6%	`^http://data\.semanticweb\.org`	SWDF
7886	43.1%	`interaction`	BioPortal
991	5.4%	`rdfs:subClassOf`	BioPortal
197	70.1%	`^CG[0-9]*$`	OpenBioMed
29	10.3%	`concatenate`	OpenBioMed
2	14.3%	`drug`	Bio2rdf
1	7.1%	`el estring que quieras`	Bio2rdf

longer than other datasets, as shown in Table 2. This might be because of the diversity of what DBpedia contains within the literals or might be because of the structure of DBpedia itself as it contains a large portion of free text. For example, dbpedia:abstract property can take up to 500 words of free text [21]. This gives an advantage for DBpedia to be further analysed in terms of regexes.

Bibliographic (SWDF) and biomedical (Bio2rdf, OpenBioMed and BioPortal) datasets cover about 25% each for the overall queries, leaving only 4% for the geo-spacial (LGD) dataset. However, regexes from the latter dataset comprises 7%, whereas the Bibliographic dataset only combined 4%, and 1% for the other biomedical datasets.

By comparing each dataset's overall queries against their regex queries, we can see that DBpedia and LGD datasets have almost the same ratio of regexes by 3.3%. The others have lower ratio by 0.2% for the SWDF and 0.07% for the biomedical datasets combined. However, by looking at regex patterns from LGD, it is clear that most of regex queries were generated by an agent. Mostly, they are countries names corresponding to properties 8 and 9 from Table 5. As they are generated by an agent, they tend to have the same query structure. Specifically, regex patterns do not contain any meta-characters. Thus, analysing such dataset may not provide a rich usage of regex.

This leaves us to the only domain that contains more than one dataset within the log files; namely Bio2rdf, OpenBioMed and BioPortal datasets. Biomedical datasets, here, have shown a general low rate of regexes so far. Despite the fact that 25% of the overall queries come from these datasets, regex queries were only 4% and the ratio was 0.07%. Both Bio2rdf and OpenBiomed have only respectively 13 and 14 of regex queries after removing duplicates. In general, this might be the structure of such a domain that mainly their datasets consist of ontology classes rather than instances as in DBpedia. Another reason can be the constrained user interface for searching rather than writing queries through SPARQL end-points.

4.2 Regex Operators/Features Analysis

In this section, we analyse regex usage according to the use of regex operators/features. However, regex has many different flavours and features. We follow, in both definitions and terminologies, SPARQL specifications which relies on the XQuery 1.0 and XPath 2.0[10] regex syntax. XQuery and XPath also follow XML schema[11] with some additions such as: Start-with and End-with. The listed features are known to be supported by the investigated endpoints, see Table 1.

Table 4. Regex Stats/Features/Properties and their occurrences

Stats/Features/Properties	Syntax/Description	Total Count
Not-well formed regexes	Does not contain "normal characters"	10,533
Case insensitive	flag "i"	1,551,138
Quantifiers:	*, + or ?	1,082,558
*	* (only *, not preceded by dot)	90,692
+	+ (only +, not preceded by dot)	1,250
.+	.+ (longest possible match, one r more)	952,724
.*	.* (longest possible match, zero or more)	854,203
Restricted quantifiers	{n,m}	1,392
Reluctant quantifiers:	??, *? or +?	1,000
.+?	.+? (shortest possible match, one or more)	0
.*?	.*? (shortest possible match, zero or more)	0
Character class:	[characters]	11,849
Negation	[^characters]	2,560
Grouping:	(characters)	1,014,720
Back reference	(character) followed by \1 or \2, etc	0
Alternation	Ex: character|character	1,031,389
Start-with	^one or more character	68,610
End-with	One or more character $	969,035
URIs	regex that start with http	329,383
Non-ASCII	contains chat >= 128	12,455
Exact search	normal characters surounded with ^ and $	9,424
Meta-characters clear:	Patterns without any meta-characters	393,372
Just letters	only letters and/or white space	267,785
Just numbers	only numbers and/or white space	963

Simple search properties like (*Start-with* and *End-with*) comprise almost half of all regex clauses as Table. 4 indicates. These properties are more index-friendly in contrast to the other operations such as *Quantifiers*. Building an index for these kind can improve the speed of search by order of magnitudes, as proposed by [22, 23]. Moreover, SPARQL 1.1 has just added additional properties such as: STRSTARTS, STRENDS, CONTAINS, STRBEFORE and STRAFTER. SPARQL 1.1 differentiates between these properties/functions from regex. Building dedicated indices for such properties will then introduce a trade off between the space and time.

[10] `http://www.w3.org/TR/xpath-functions/#regex-syntax`
[11] `http://www.w3.org/TR/xmlschema-2/`

It is also notable that there are no back reference regex queries. This might be because back referencing is more expressive than needed by users within the data searched by SPARQL. Back referencing maybe the only feature that cannot be implemented using a FSM-based engines. It is probably that back reference is an advance regex feature that most users do not require for their filtering expressions. This also gives a hint about using FSM-based engines to be implemented, as classical engines are claimed to be more efficient [6].

Regex is independent from human languages. This makes it sometimes the only solution when a FTS does not support a particular language. In this analysis, there are 12,455 regexes that contain one or more Non-ASCII characters.

There are 9,424 exact search queries. Despite that this kind of regex, where a start-with is presented within the pattern, may yield a best case scenario in terms of checking each record, this still requires a full scanning of records. A better way of writing such a pattern may be by not using a regex filter; a direct query pattern that includes the full string can be more efficient, since SPARQL engines can benefit from their own data indices.

Just letters, just numbers and meta-character clean mainly refer to the case where a regex filter may not be the best way of writing a query. These may fall into the CONTAINS property introduced by SPARQL 1.1. Yet, these comprise around 20% of regex queries.

Regex is not just for searching string literal, but also can be applied on URIs. Searching URIs comprises 20% of all regex queries. For example, "http://dbpedia.org/ontology/"has been used 81390 times as a regex pattern. From the list of URIs we analysed, there is a common style for these queries usually associated with prefix queries such as "http://dbpedia.org/*"with 51851 occurrences. It is probably the case where the desired pattern is actually "http://dbpedia.org/ontology/.*". This is a common mistake users make by using "*"as a wildcard syntax, however, interpreted as a regex quantifier and therefore produces different results. Table 4 indicates that "*", not preceded by a dot, has been used 90,692 times.

4.3 Regex in Relation to BGP

In SPARQL specifications, regex is written within a filter expression. Moreover, there has to be a variable that is referred to from the BGP to be searched with regex. We analysed regex variables in relation to the BGP. There are 74,708 of subject variables, 41,636 predicate variables and 1,507,304 object variables that have regexed. Although it is expected that objects are the most used as regex variables, as they only can hold literals in terms of RDF semantics, subjects and predicates still can be used for searching URIs or blank nodes. In this special case where the referred regex variable was an object, we also analysed the predicates of that object. Examining these locations can help in determining whether the scanned variable is actually a literal or a different type such as: URI or a blank node that has been translated into a string literal (using STR function), for the purpose of searching these variables using regex. Moreover, identifying the predicates where their objects are more likely to be searched with regex, can

help in building indices for such predicates, as the case in Jena-Text[12] where a pre-defined predicates can speed up the search for FTS. Table 5 shows the top 10 predicates/properties that their objects have been searched within regex.

Table 5. Most often used predicates, where their objects are regexed

ID	Occurrence	Predicate URI	Range
1	1,024,422	`http://www.w3.org/1999/02/22-rdf-syntax-ns\#type`	rdfs:Class
2	123,684	`http://dbpedia.org/property/name`	rdfs:Literal
3	87,153	`http://www.w3.org/2000/01/rdf-schema\#label`	rdfs:Literal
4	69,257	`http://www.w3.org/2000/01/rdf-schema\#subClassOf`	rdfs:Class
5	39,249	`http://linkedgeodata.org/ontology/directType`	rdfs:Class
6	29,559	`http://xmlns.com/foaf/0.1/name`	rdfs:Literal
7	23,822	`http://dbpedia.org/property/reference`	rdfs:Literal
8	17,097	`http://linkedgeodata.org/property/is_in`	rdfs:Literal
9	12,959	`http://linkedgeodata.org/property/is_in\%3Acountry`	rdfs:Literal
10	12,515	`http://xmlns.com/foaf/0.1/page`	foaf:Document

4.4 Regex Vs FTS within DBpedia

DBpedia has shown a rich regex usage so far. Moreover, DBpedia has also maintained a FTS extension with their "bif:contains"property. We aim to examine how many FTS queries and compare them with the regex ones. Most importantly, how many queries contain both, and whether they are written optimally.

	Overall queries	FTS queries	Regex queries	FTS AND regex queries
DBpedia	48,648,011	169,204	1,672,890	18,554

In the case where both FTS and regex are within one SPARQL query, there are a large portion that have the same SPARQL structure. This most probably refers to the agent that submits these queries. For example, the SPARQL query below shows a good integration of FTS and regex. In this query, instead of scanning all records within regex, a FTS index will produce a subset of candidates to only be filtered with regex. However, there are some issues that can be addressed from this query; mainly regarding false-negative results. In the case of regex, a user should have an understanding of what results are expected. For example, a label like "East India Company"is not going to be returned as India is not at the start of the line. On the other hand, a user should also have an understanding of how FTS indexing works. For example, it is not always the case that the FTS tokeniser implements stemming; in this case, a label like "Indian Railways", for example, is also not going to be returned, as "Indian"would not be matched by the FTS, and therefore it will not be filtered with regex.

[12] `http://jena.apache.org/documentation/query/text-query.html`

```
SELECT DISTINCT ?s ?o WHERE {
        ?s  <http://www.w3.org/2000/01/rdf-schema#label> ?o .
        ?o bif:contains "India".
        FILTER (regex(str(?o), '^India')) .
        FILTER (!regex(str(?s), '^http://dbpedia.org/resource/Category:')).
        FILTER (!regex(str(?s), '^http://dbpedia.org/resource/List')).
        FILTER (!regex(str(?s), '^http://sw.opencyc.org/')).
        FILTER (lang(?o) = 'en').
}
Limit 10
```

5 Conclusion

Regular expressions is a very powerful technique that can be used for matching string data. Nevertheless, they may be slow, computationally expensive and sometimes more expressive than actually needed. In this paper, we studied this issue by analysing how users/agents are using regexes within SPARQL. Our analysis shows that there are issues of how regex being used from different bodies. In addition, it was expected that regex would be mostly applied on literal strings (human-readable labels, such as rdfs:lable), but was mostly on filtering URIs. Based on our findings, we deliver a number of suggestions that can help increasing the efficiency of regex queries within SPARQL:

- Users/SPARQL engine developers may take into account rewriting regex queries to be integrated with a FTS extension.
- W3C may consider adding additional specifications regarding Full-Text Search within SPARQL. This can help users to differentiate between FTS and regex, and use the appropriate one.
- SPARQL engine developers may build indices for some of the most used properties such as: START-WITH, END-WITH and CONTAINS, as they already have been syntactically added to SPARQL 1.1 specifications.
- SPARQL engine developers may consider adding a FSM-based engine to be used within simple patterns.

References

1. Seaborne, A., Prud'Hommeaux, E., Harris, S.: SPARQL 1.1 Query Language. W3C Recommendation (2013)
2. Morsey, M., Lehmann, J., Auer, S., Ngomo, A.C.N.: DBpedia SPARQL Benchmark - Performance Assessment with Real Queries on Real Data. In: 10th International Semantic Web Conference (ISWC'11). Volume 7031 of Lecture Notes in Computer Science., Bonn, Germany, Springer (2011) 454–469
3. Prud'Hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (2008)
4. Minack, E., Sauermann, L., Grimnes, G., Fluit, C., Broekstra, J.: The Sesame LuceneSail: RDF Queries with Full-Text Search. Technical report (2008)
5. Ellul, K., Krawetz, B., Shallit, J., Wang, M.w.: Regular Expressions: New Results and Open Problems. Journal of Automata, Languages and Combinatorics **9**(2-3) (2004) 233–256

6. Becchi, M., Crowley, P.: Extending Finite Automata to Efficiently Match Perl-Compatible Regular Expressions. In: 2008 ACM Conference on Emerging Network Experiment and Technology (CoNEXT'08), Madrid, Spain, ACM (2008) 25
7. Reidenbach, D., Schmid, M.L.: A Polynomial Time Match Test for Large Classes of Extended Regular Expressions. In: 15th International Conference on Implementation and Application of Automata (CIAA'10). Volume 6482 of Lecture Notes in Computer Science., Winnipeg, MB, Canada, Springer (2011) 241–250
8. Berendt, B., Hotho, A., Stumme, G.: Towards Semantic Web mining. In: In International Semantic Web Conference (ISWC, Springer (2002) 264–278
9. Berendt, B., et al.: A roadmap for web mining: From web to Semantic Web. In Berendt, B., et al., eds.: Web Mining: From Web to Semantic Web. Volume 3209., Heidelberg, Springer (2004) 1–22
10. Möller, K., Hausenblas, M., Cyganiak, R., Grimnes, G.A.: Learning from linked open data usage: Patterns & metrics. In: Proceedings of the WebSci10: Extending the Frontiers of Society On-Line. (2010)
11. Berendt, B., et al.: USEWOD2011: 1st international workshop on usage analysis and the Web of Data. In Srinivasan, S., et al., eds.: WWW, ACM (2011) 305–306
12. Berendt, B., Hollink, L., Hollink, V., Luczak-Rösch, M., Möller, K., Vallet, D.: Usage analysis and the Web of Data. SIGIR Forum **45**(1) (2011) 63–69
13. Arias, M., Fernández, J.D., Martínez-Prieto, M.A., de la Fuente, P.: An empirical study of real-world sparql queries. CoRR **abs/1103.5043** (2011) 1st International Workshop on Usage Analysis and the Web of Data (USEWOD2011).
14. Alvarez-Garcia, S., Brisaboa, N.R., Fernandez, J.D., Martinez-Prieto, M.A.: Compressed k2-triples for full-in-memory rdf engines. In Sambamurthy, V., Tanniru, M., eds.: AMCIS, Association for Information Systems (2011)
15. Lorey, J., Naumann, F.: Caching and prefetching strategies for SPARQL queries. In: Proceedings of the 3rd International Workshop on Usage Analysis and the Web of Data (USEWOD), Montpellier, France (2013)
16. Luczak-Rösch, M., Mühleisen, H.: Log File Analysis for Web of Data Endpoints . In: Proc. of the 8th Extended Semantic Web Conference (ESWC), Poster-Session, Springer LNCS 6643 (2011)
17. Luczak-Rösch, M., Bischoff, M.: Statistical analysis of web of data usage. In: Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn2011), CEUR WS (2004)
18. Luczak-Rösch, M.: Usage-dependent maintenance of structured Web data sets. Doctoral dissertation, Freie Universität Berlin (2014) retrieved 2014-03-12.
19. Raghuveer, A.: Characterizing machine agent behavior through SPARQL query mining. In: Proceedings of the International Workshop on Usage Analysis and the Web of Data, Lyon, France. (2012)
20. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations. In: 13th International Conference on World Wide Web (WWW'04), New York, USA, ACM (2004) 74–83
21. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia - a crystallization point for the web of data. Web Semantics: Science, Services and Agents on the World Wide Web **7**(3) (2009) 154–165
22. Cho, J., Rajagopalan, S.: A Fast Regular Expression Indexing Engine. In: 18th International Conference on Data Engineering (ICDE'02), San Jose, California, USA, IEEE Computer Society (2002) 419–430
23. Lee, J., Pham, M.D., Lee, J., Han, W.S., Cho, H., Yu, H., Lee, J.H.: Processing SPARQL Queries with Regular Expressions in RDF Databases. BMC Bioinformatics **12**(Suppl 2) (2011) S6–67