Access Control by Query Rewriting: the Case of KorAP

Piotr Bański, Nils Diewald, Michael Hanl, Marc Kupietz, Andreas Witt*
Institute for the German Language (IDS)
R5 6–13, 68161 Mannheim, Germany
{banski|diewald|hanl|kupietz|witt}@ids-mannheim.de

Abstract

We present an approach to an aspect of managing complex access scenarios to large and heterogeneous corpora that involves handling user queries that, intentionally or due to the complexity of the queried resource, target texts or annotations outside of the given user's permissions. We first outline the overall architecture of the corpus analysis platform KorAP, devoting some attention to the way in which it handles multiple query languages, by implementing ISO CQLF (*Corpus Query Lingua Franca*), which in turn constitutes a component crucial for the functionality discussed here. Next, we look at query rewriting as it is used by KorAP and zoom in on one kind of this procedure, namely the rewriting of queries that is forced by data access restrictions.†

Keywords: Access Control; Corpus Management; Query Rewriting

1. Introduction

The present article focuses on one aspect of managing complex access scenarios to large and heterogeneous corpora, namely on handling user queries that, intentionally or due to the complexity of the queried resource, target texts or annotations outside of the given user's permissions.

We first present the overall architecture of the corpus analysis platform KorAP, devoting some attention to the way in which it handles multiple query languages, by implementing ISO CQLF (*Corpus Query Lingua Franca*, see below), which in turn constitutes a component crucial for the functionality discussed here. Next, we look at query rewriting as it is used by KorAP and zoom in on one kind of this procedure, namely the rewriting of queries that is forced by data access restrictions.

2. KorAP: Architecture, Data Model, Query Languages

KorAP (Korpusanalyseplattform der nächsten Generation; Bański et al., 2012, 2013) is a new corpus analysis platform created at IDS Mannheim and designed primarily for the purpose of sustainably serving the German Reference Corpus (Deutsches Referenzkorpus, DeReKo; cf. Kupietz et al., 2010), the largest annotated archive of German texts, which is about to increase its size to 24 billion words in 2014 (Kupietz and Lüngen, 2014). In this role, it will succeed Cosmas II (Bodmer, 2005; Bodmer Mory, 2014).

This section looks at the overall architecture of the system, presents the fundamental data model assumed, and outlines the way in which KorAP approaches the assumptions of ISO TC37 SC4 Corpus Query Lingua Franca (CQLF).

2.1. KorAP: Architecture and Data Model

Figure 2 shows a schematic diagram of the architecture of KorAP. For the sake of sustainability, all components of KorAP are exchangeable, to make it possible for the maintainers of the system to follow future developments in data management. In addition to that, multiple backends are used for corpus management and multiple query languages are supported by the frontend, making the CQLF-Processor and the authentication-based Policy Service the pivot points for query processing.

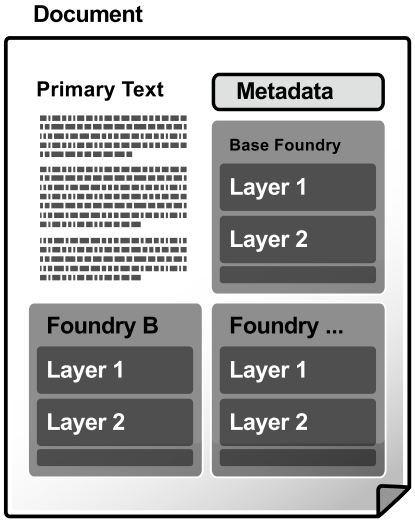

Figure 1: Rough structure of a KorAP document.

KorAP organizes documents into multi-dimensional *virtual collections*, created on the basis of internal or external features of texts and/or on the basis of the origin or the content of annotations (cf. Bański et al., 2013).

A KorAP document (cf. Fig. 1) consists of the document metadata (currently in the shape of a TEI-like header, taken over from DeReKo as encoded in the so-called I5 format, cf. Lüngen and Sperberg-McQueen, 2012), the raw text, and one or more so-called *foundries* (i.e., collections of annotations put together mostly due to originating from a single annotation tool).

Systems such as KorAP have to deal with, minimally, two kinds of data access restrictions: one set of such restrictions

*Andreas Witt: IDS Mannheim and University of Heidelberg

†We are grateful to the anonymous reviewers for their thorough, helpful and friendly remarks. They have led us to put stronger focus on the issue of query rewriting, and to modify the title of this contribution accordingly.

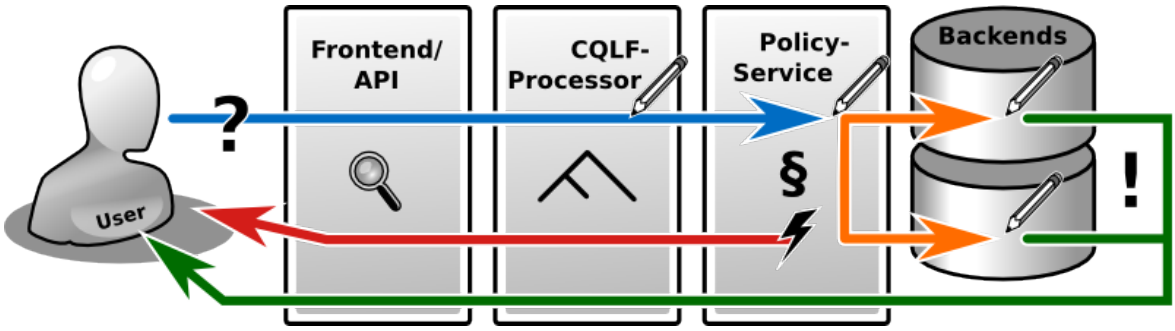

Figure 2: Architecture of KorAP and Query Flow.

comes from individual agreements with the data donors (e.g. publishers). Another comes from licensing conditions imposed by the creators of annotation tools[1]. Yet another set of restrictions handled by KorAP is user-imposed constraints on self-created annotations or self-defined virtual collections.

2.2. KorAP as a Reference Implementation of ISO CQLF

KorAP supports multiple query languages. Query strings sent to the system are serialized into a uniform JSON-LD representation (Sporny et al., 2014), which implements CQLF, a nascent corpus query language standard that is currently the subject of work of the ISO TC37 SC4 Working Group 6 (ISO/WD 24623-1). CQLF can best be thought of as a feature matrix onto which the individual query constructs can be mapped. In KorAP practice, query constructs thus mapped (serialized into JSON-LD) are then treated identically, irrespective of which query language interpreter they have been created by (see Fig. 3).[2] The JSON-LD representations are then deserialized into a form readable by the particular destination backend.

3. Access Control in KorAP

Data access control in KorAP is a complex task, because access restrictions may be imposed on either the texts or their annotations, or both. Authorization may depend on permissions that a research group grants to the user, but may also be more fine-grained (e.g. limited to the working place by filtering on IP-ranges). Authorization in KorAP needs to be dynamic – a user may join or leave a certain research group and thus gain or lose the related permissions. Virtual collections may be dynamically created, inheriting the licences of their source corpora. And these licences may change as well.

[1] In some specific cases, it may also happen that individual layers within some foundries are subject to different licensing conditions. For example, a software provider may decide to distribute the tokenization information freely, but impose restrictions on the output of the syntactic or semantic annotation process.

[2] KorAP's use of JSON-LD for the purpose of expressing CQLF construct was an intentional choice to complement the recommendations of ISO TC37 SC4 WG1 for linguistic data protocols (Ide, 2013). This is meant as a step towards ensuring the uniformity of the representation of language content standardized within ISO TC37 SC4.

When dealing with access control of data resources, implementers can follow multiple strategies with advantages and disadvantages specific to the application. In the following, we introduce four options theoretically available for KorAP, and discuss their consequences.

1. The *frontend* could prevent users from entering queries that target resources that the user is not allowed to access. This is especially applicable to query systems with closed sets of options, but less useful for a system supporting multiple query languages. In KorAP, this approach would have the additional disadvantage of implementing data access restriction code multiple times, because the web interface as well as the API serve as access points to the data, and it is not recommended to have multiple points for maintenance of security relevant code (cf. Rizvi et al., 2004).
2. The *backend* can be responsible for access control in either of the two following ways:
   (a) by limiting the results to the data that the user is authorized to access, or
   (b) by rejecting user queries targeting data that the user is not authorized to access.

   The first variant is mostly used in database systems by applying query rewriting and modifying `where` clauses in SQL statements before the query hits the database (cf. Oracle, 2014). The second variant is proposed by Rizvi et al. (2004). Using access control on the database level, however, imposes the same problem for KorAP as the implementation on the frontend level: for the sake of sustainability and delegation of labour, KorAP supports multiple, exchangeable backends, which means that access control would need to be implemented multiple times and re-implemented whenever a backend is exchanged. In addition to that, this strategy is not scalable with the dynamic nature of the data and of virtual collections (cf. Rizvi et al., 2004).
3. The results can be filtered after retrieval and before being presented to the user. This would allow for a single point of implementation, but at the same time may lead to potential complex computations on the backend side, even for users not allowed to access any data.
4. The query can be rewritten in a separated step before it is sent to the backends, limiting the request to the data that the user is authorized to access (as proposed in strategy 2a, but as a single point of implementation).

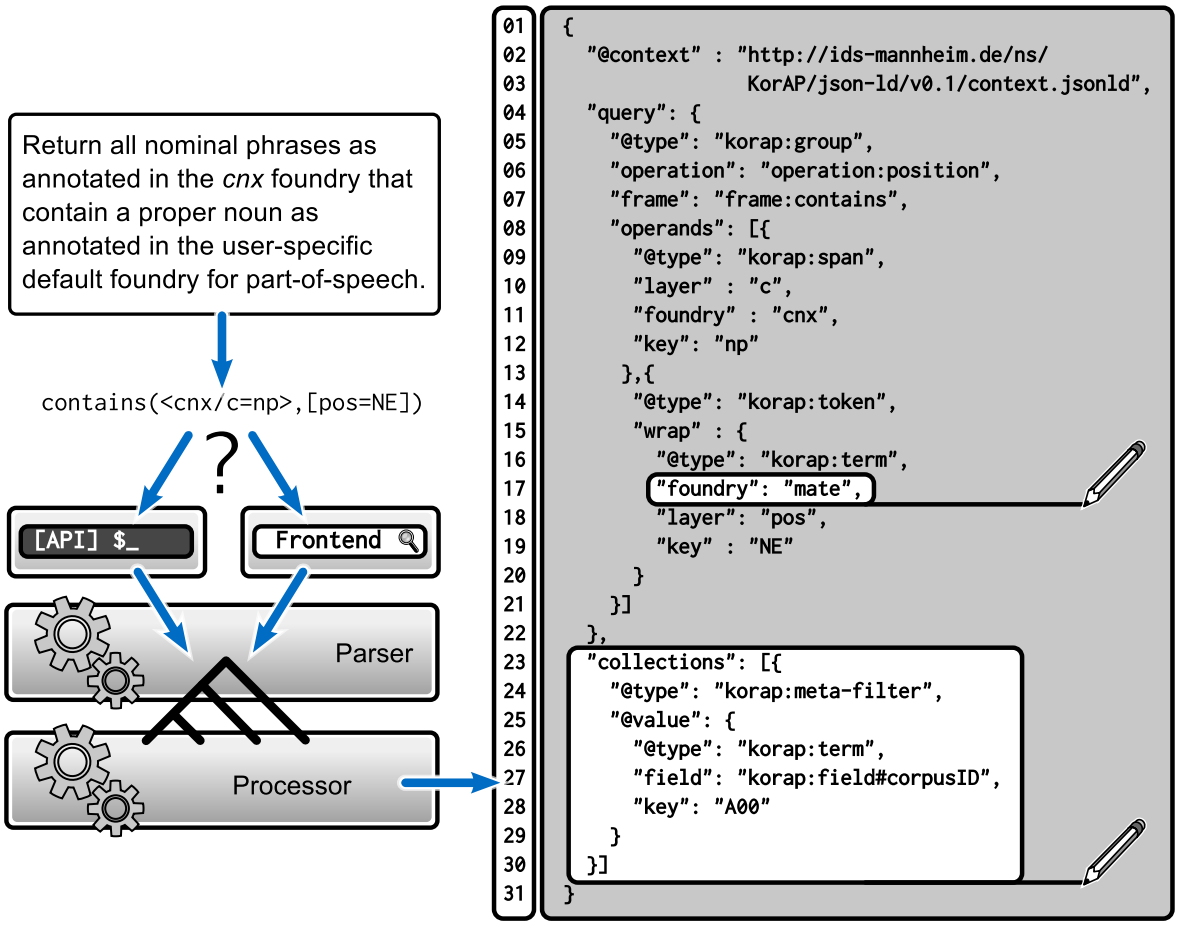

Figure 3: Serialization of a query with multiple rewrites.

KorAP takes the fourth approach and rewrites queries that in some way exceed permissions in a separated component for access control: the Policy Service.

4. Query Rewriting

In web search engines, query rewriting often takes place to improve recall, for example by deleting query terms (like stopwords; Jones and Fain, 2003) or by introducing alternative synonyms (Jones et al., 2006), often in scenarios of sponsored search results. Relational databases use query rewriting mostly in conjunction with materialized views, for "query optimization, maintenance of physical data independence, data integration and data warehouse design'' (Halevy, 2001, p. 270). Rizvi et al. (2004) suggest query rewriting as a method to allow for fine-grained data access control (see above).

Before focusing on the Policy Service component for data access control, we will look more closely at query serialization. By serializing queries in a standardized way, KorAP allows for multiple entry points for query rewriting steps (cf. the rewrite markers in Fig. 2), especially for query optimization. In what follows, we briefly look at two non-critical entry points for query rewriting (in the query interpreter and on the way to the backend), and then concentrate on the point critical for data access management.

4.1. Query Serialization Rewrite

KorAP supports multiple query languages by translating them in a unified form that instantiates ISO CQLF. Currently supported query languages include the Cosmas II QL (cf. Bodmer, 2005) and Poliqarp QL (cf. Janus and Przepiórkowski, 2007). First, the query strings are parsed using a language-specific grammar and translated in a language-specific abstract syntax tree (in the parser module in Fig. 3). This tree is afterwards serialized into our implementation of CQLF in the form of a JSON-LD document (in the processor module).

Although CQLF covers conceptually all possible constructs of corpus query languages, our implementation attempts to avoid verbosity by prohibiting distinct formulations of synonymous query constructs. For example, Cosmas II allows queries such as `Alice /+w1:1 Bob`, which means that the word "Bob'' has to immediately be preceded by the word "Alice''. On the other hand, it supports the query `Bob /-w1:1 Alice`, which means that the word "Alice'' has to immediately be followed by the word "Bob''. Because "Alice precedes Bob'' is a query construct equivalent to "Bob follows Alice'', the query interpreter will rewrite the query before serialization, only supporting unidirectional sequences.

4.2. Backend Rewrite

The serialized query output can, depending on the constructs present in it, be rewritten when getting deserialized

into the backend. Currently, KorAP supports two different backends: one using a Lucene search index[3], the other one using the graph database Neo4j[4]. The former backend deserializes the JSON-LD format into Lucene SpanQuery objects, whereas the latter deserializes it into Cypher queries. Depending on the destination query system, further optimizations by means of query rewriting can be applied. For example, the Cosmas II query `Bob /+w1:1 Alice` is equivalent to the Poliqarp query `[orth=Bob][orth=Alice]`, but will be serialized differently: the first query uses a general sequence operator including distance constraints in the JSON-LD serialization; the second query uses a direct sequence operator. The Lucene backend has optimized queries for direct sequences. When the deserializer parses a sequence operator with a defined distance constraint of zero words, the query is rewritten to a sequence without distance constraints, and thus optimized for the backend.

4.3. Access Control Rewrite

Apart from the most trivial operations performed at the level of the frontend alone, all user activity in KorAP involves the management of data access. Users' interaction with the broadly understood access management begins typically at the point of authentication (via a local or a Shibboleth-regulated distributed authentication system) and continues through defining virtual collections, retrieving information on them, et cetera. What we concentrate on here is a very specific point where a user's query needs to be examined for permissions and either rejected in full or rewritten and returned with an accompanying warning message.

4.3.1. Relevant Use-cases

Recall from the previous sections that one of the fundamental features implemented in KorAP is the ability to create virtual collections on the basis of various text-internal and external features. Depending on the choices made by the user, this may imply targeting a set of documents that is not uniform with respect to the license permissions. And thus, while the texts of the given collection may all be accessible to the user, some foundries may be restricted to (e.g., being used by the employees of the IDS as the tool-license-bearer). A somewhat similar scenario depends on another feature of KorAP, namely the ability to address, in a single query, annotation layers belonging to distinct foundries – it may happen that the queried layers are not uniform license-wise and thus, while part of the given query may return the appropriate results, other parts have to fail. Such a query is shown in Figure 3, where the user explicitly invokes the license-restricted Connexor[5] foundry and implicitly, the foundry set as a user default for part-of-speech information – in this case, Mate[6] (rewritten in line 17).

4.3.2. Access Policies

Determining whether a user is allowed to query a virtual collection or resource in KorAP depends on several conditions and on the type of information that is requested. We shall describe these types of information as different policies.

Whereas text publishers may specify which group of users is allowed to access their resources, other types of information units linked to these resources (i.e., annotations, residing in individual foundries) are not covered by these license conditions. They are instead influenced by the licensing conditions imposed on, or by, the annotation tools. To build on top of this premise of potentially multiple access conditions, in which access control for a resource is not only determined by one piece of information, but a variety of information units, KorAP's Policy Service allows the definition of rules for each unit. In order to derive an access control decision from a request, the Policy Service takes into account all units of information that are associated with the requested resource; this is referred to here as *horizontal consistency*. As a result, granting access to a linguistic resource constitutes the sum of the underlying access decisions.

In contrast to horizontal consistency of an access control decision, policies may also be structured vertically, as hierarchical dependencies between resources. Thus, access to, for example, a specific annotation segmentation is not only dependent on the policies defined for this particular resource, but also bound by conditions of the resource it is contained by (cf. Fig. 4).

Additionally, KorAP provides environmental parameters that can be attached to a resource policy. These environmental parameters determine if a policy is applicable during the request. This allows the Policy Service to apply certain policies on the basis of a location address or a time interval.

4.3.3. Role of the Policy Service

Before redirecting the query to the backend, an implementation of the Policy Service retrieves the JSON representation, extracts referenced resources, and locates the respective policies for the relevant portions of the query serialization.

In order for the query in Fig. 3 to validate, the user has to have read permission for the corpus archive referenced in the *collections* segment of the JSON-LD serialization that represents (see lines 23-30) the definition of a virtual collection as it is used by all components of the KorAP architecture. The Policy Service also extracts annotation foundries and layers specified by the user (lines 10-11 and 17-18) within the *query* segment. Access to these resources can only be granted if the Policy Service determines that the user possesses all the required conditions attached to the resource policy (cf. Fig. 4).

The application of the query rewrite process is triggered by the Policy Service in case the user cannot meet all the conditions specified in the resource policies or the query is missing parameters required to be validated. Fig. 4 illustrates that foundry level information can be inserted into the query by the Policy Service. To this purpose the Policy Service has access to user settings defining default values for annotation layer information (e.g. morphological, constituency, dependency layers), if the user has provided such properties in the user account settings.

On the one hand, as we have illustrated, a query request may be modified on the basis of the authorization level or of de-

[3] `https://lucene.apache.org/`
[4] `http://www.neo4j.org/`
[5] `http://www.connexor.com/nlplib/`
[6] `http://code.google.com/p/mate-tools/`

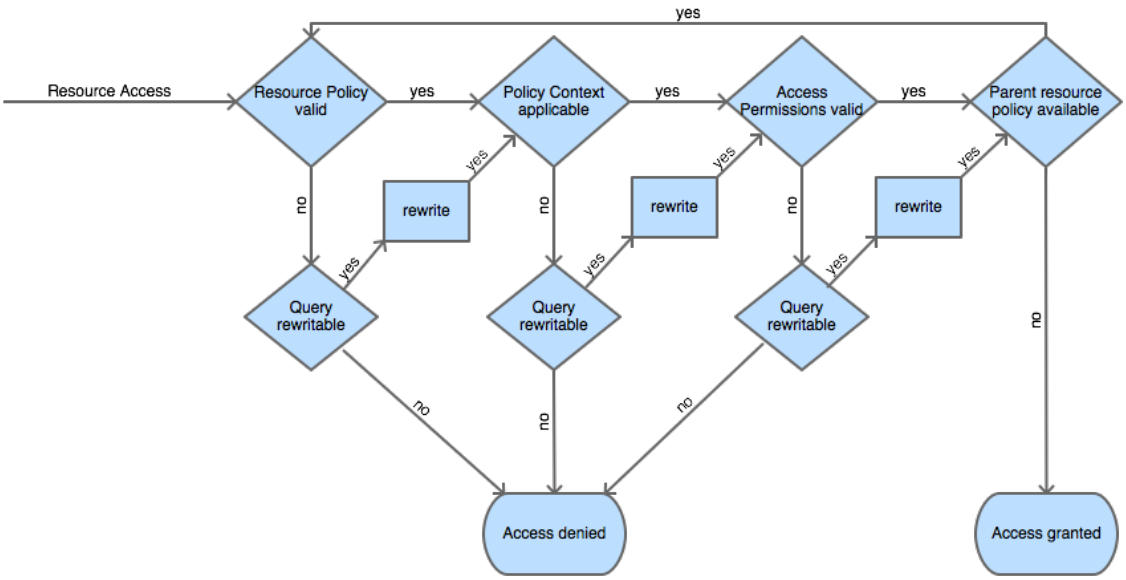

Figure 4: Policy architecture diagram.

fault properties of a user. On the other hand, queries on virtual collections sometimes have to be stable and reproducible with the exact same results for each request – and are not allowed to be altered by the query rewriter to prohibit inconsistencies. In that case, a query can be marked as *not rewritable* and access to the data will be rejected instead of modified if the user has no access to the whole dataset in the request (see the "Query rewritable'' decisions in Fig. 4). Rejection is also the result of a user targeting a restricted foundry – in such cases, the query can have no satisfactory result, because the information that the user explicitly requests cannot be provided.

5. Conclusion

We have presented a fragment of the functionality of the new corpus analysis platform KorAP, created at IDS Mannheim, in order to contribute to the store of best practices for dealing with access control issues in large, complex and dynamic language resources. Although the primary goal of current KorAP development is to produce a new engine for the DeReKo archive, the software will be released under an open license at `https://github.com/KorAP`. Information on the progress of our work is made available at `http://korap.ids-mannheim.de/`.

References

Bański, P., Fischer, P. M., Frick, E., Ketzan, E., Kupietz, M., Schnober, C., Schonefeld, O., and Witt, A. (2012). The New IDS Corpus Analysis Platform: Challenges and Prospects. In Calzolari, N., Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul. European Language Resources Association (ELRA).

Bański, P., Frick, E., Hanl, M., Kupietz, M., Schnober, C., and Witt, A. (2013). Robust corpus architecture: a new look at virtual collections and data access. In Hardie, A. and Love, R., editors, *Corpus Linguistics 2013 Abstract Book*, pages 23–25, Lancaster. UCREL. `http://ucrel.lancs.ac.uk/cl2013/doc/CL2013-ABSTRACT-BOOK.pdf`.

Bodmer, F. (2005). COSMAS II. Recherchieren in den Korpora des IDS. *Sprachreport*, 3/2005:2–5.

Bodmer Mory, F. (2014). Mit COSMAS II »in den Weiten der IDS-Korpora unterwegs«. In Steinle, M. and Berens, F. J., editors, *Ansichten und Einsichten. 50 Jahre Institut für Deutsche Sprache*, page 376–385. Institut für Deutsche Sprache, Mannheim.

Halevy, A. Y. (2001). Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294.

Ide, N. (2013). Web service exchange protocols: Preliminary proposal. Presentation given at ISO TC37 SC4 WG1 meeting in Pisa, September 2nd, 2013. Accessible at `http://www.anc.org/LAPPS/EP/Meeting-2013-09-26-Pisa/overview-ep-2013-09-26-pisa.pdf`.

Janus, D. and Przepiórkowski, A. (2007). Poliqarp: An open source corpus indexer and search engine with syntactic extensions. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 85–88. Association for Computational Linguistics.

Jones, R. and Fain, D. C. (2003). Query word deletion prediction. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 435–436, New York, NY, USA. ACM.

Jones, R., Rey, B., Madani, O., and Greiner, W. (2006). Generating query substitutions. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 387–396, New York, NY, USA. ACM.

Kupietz, M., Belica, C., Keibel, H., and Witt, A. (2010). The German Reference Corpus DeReKo: A Primordial Sample for Linguistic Research. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odjik, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, page 1848–1854, Valletta, Malta. European Language Resources Association (ELRA). `http://www.lrec-conf.org/proceedings/lrec2010/pdf/414_Paper.pdf` (25.5.2010).

Kupietz, M. and Lüngen, H. (2014). Recent Developments in DeReKo. In *Proceedings of LREC 2014*. European Language Resources Association (ELRA).

Lüngen, H. and Sperberg-McQueen, C. M. (2012). A TEI P5 Document Grammar for the IDS Text Model. *Journal of the Text Encoding Initiative*, 3:1 – 18.

Oracle (2014). Oracle database security guide 11g release 1 (11.1), 7. using oracle virtual private database to control data access. online.

Rizvi, S., Mendelzon, A., Sudarshan, S., and Roy, P. (2004). Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 551–562, New York, NY, USA. ACM.

Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., and Lindström, N. (2014). Json-ld 1.0. a json-based serialization for linked data.