Published in: Calzolari, Nicoletta; Choukri, Khalid; Declerck, Thierry; Goggi, Sara; Grobelnik, Marko; Maegaard, Bente et al. (Hg.) (2016): Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016), Portorož, Slovenia. Paris: European Language Resources Association (ELRA), pp. 3586-3591

KorAP Architecture – Diving in the Deep Sea of Corpus Data

Nils Diewald[1], **Michael Hanl**[1], **Eliza Margaretha**[1], **Joachim Bingel**[2], **Marc Kupietz**[1], **Piotr Bański**[1], **Andreas Witt**[1,3]

[1] Institut für Deutsche Sprache, Mannheim
{diewald,hanl,margaretha,kupietz,banski,witt}@ids-mannheim.de
[2] Centre for Language Technology, University of Copenhagen
bingel@hum.ku.dk
[3] Universität Heidelberg

Abstract

KorAP is a corpus search and analysis platform, developed at the Institute for the German Language (IDS). It supports very large corpora with multiple annotation layers, multiple query languages, and complex licensing scenarios. KorAP's design aims to be scalable, flexible, and sustainable to serve the German Reference Corpus DeReKo for at least the next decade. To meet these requirements, we have adopted a highly modular microservice-based architecture. This paper outlines our approach: An architecture consisting of small components that are easy to extend, replace, and maintain. The components include a search backend, a user and corpus license management system, and a web-based user frontend. We also describe a general corpus query protocol used by all microservices for internal communications. KorAP is open source, licensed under BSD-2, and available on GitHub.

Keywords: KorAP, microservices, large corpus data

1. Introduction

With DeReKo, the largest linguistically motivated corpus of contemporary German, growing beyond 25 billion words (Kupietz and Lüngen, 2014), new challenges for corpus management have arisen at the Institute for the German Language (IDS),[1] especially as the textual data is enriched by several layers of annotations, multiplying the quantity of data immensely. A platform capable of handling such large amounts of data, while providing a wide variety of features for searching, analysing, and legally protecting the corpus, needs an architecture focussing on scalability, flexibility, and sustainability.

In this paper, we present our approach to meet these requirements: a microservice-based architecture.

KorAP[2] ("Korpusanalyseplattform der nächsten Generation"; Bański et al., 2013) is a web-based corpus analysis platform, consisting of small independent components that are easy to extend, to replace, and to maintain.

All components communicate through *KoralQuery*, a general corpus query protocol. As we design the communication protocol to be easily extensible as well as backward compatible, our approach also addresses the fact that software may change or needs to be replaced over time. In this manner, we strive to ensure sustainability and flexibility in the maintenance of the system for the next 15-20 years, thus targeting the life span of *COSMAS II* (Bodmer, 1996), the corpus analysis platform that KorAP is designed to replace.

The present contribution outlines the architecture of KorAP, describes the necessary building blocks, and discusses the capabilities and caveats of searching, analysing and legally protecting corpus data against the requirements of a large annotated text corpus such as DeReKo.

[1] http://ids-mannheim.de/
[2] http://korap.ids-mannheim.de/

2. KorAP Infrastructure

Requirements typically associated with scalable, flexible, and sustainable systems were the most important criteria for architectural decisions in KorAP. These priorities led to a microservice-based design, "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms" (Lewis and Fowler, 2014), possibly distributed to multiple machines. The opposite of such a design is a single monolithic system. Newman (2015, ch. 1) lists key benefits for microservice-based architectures, including:

"scaling": As DeReKo grows, the capacity of KorAP has to enhance as well. However, scalability requirements differ depending on the component. While the search backend *Krill* (Sec. 2.3.) needs to scale whenever the corpus grows, the user management service *Kustvakt* (Sec. 2.2.) needs to scale whenever new users are registered. Multiple small services can be added on demand.

"optimizing for replaceability": It is impossible to provide a sustainable research platform and to foresee all necessary features for research questions from the beginning. In the scenario of publicly funded research projects such as KorAP, a microservice-based architecture makes it possible to replace parts of the system once new requirements emerge, without the need to replace a monolithic system as a whole. New advances in corpus technology can therefore easily be adopted. As a proof-of-concept, KorAP has developed two independent search backends (s. Sec. 2.3.1.), optimised for different research questions.

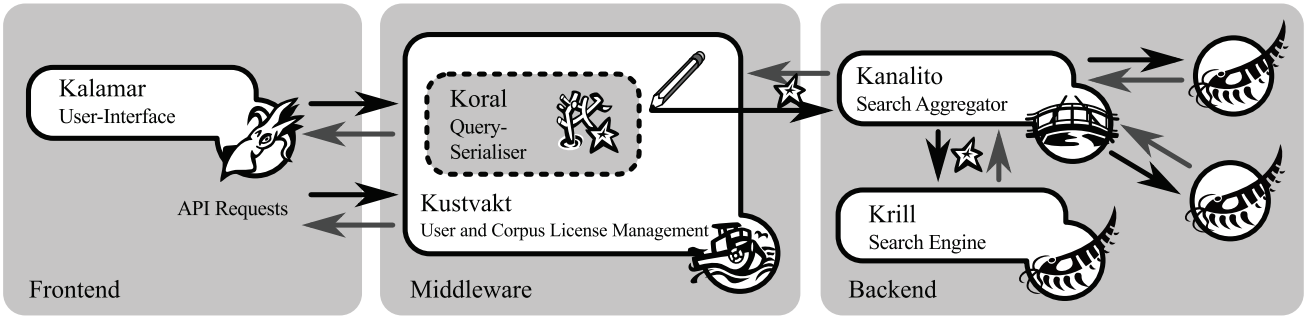

Figure 1: Interactions of KorAP components as microservices

"technology heterogeneity": KorAP is implemented using several programming languages (Java, JavaScript, Perl, and Python), and different data storages (including Lucene, MariaDB, Neo4j, and PostgreSQL). A microservice architecture makes it possible to choose the best technology for each task with flexibility, instead of one single software stack to fit all requirements.

"organizational alignment": Small components make it easier to develop in small teams, which can help to improve development speed. A small codebase is easier to adopt by new developers as well, as they need not have knowledge about the whole system. KorAP's components are open source, available from GitHub,[3]and will hopefully attract a community of developers for further improvements and sustainable development.

Although the listed benefits are not exclusive to microservices individually, they meet our requirements in total. Microservices, however, also have some drawbacks opposed to monolithic systems, including:

- more complex testing scenarios;
- an occasional slowdown of development and deployment speed, when new features require modifications of multiple interfaces or extensions to the communication protocol;
- overhead of remote API calls opposed to in-memory function calls.

For these reasons, KorAP's approach to microservices is not rigid: The decision whether a component will be implemented as a microservice or as an embedded library is based on the aforementioned advantages and disadvantages, and may change over time (i.e. any library may evolve to a microservice at a later stage of development or a microservice may become an embedded library). For example, the query serialisation component *Koral* (s. Sec. 2.1.) is implemented as a library rather than as a microservice, because it is used by only one component (Kustvakt) and would be negatively affected by the overhead of remote API calls.

To compensate for the slowdown of development and to speed up the evaluation of new features (especially regarding the user interface *Kalamar*; s. Sec. 2.4.), a rapid application development environment was created (*Rabbid*; Mell and Diewald, 2016).
Figure 1 outlines the interactions of KorAP components as microservices, from the application's frontend to the backend services. In the following sections, we describe the different components and their communication protocol.

2.1. KoralQuery

As the main access point to DeReKo, KorAP has to provide all search and analysis features common to more than 38 thousand registered users of COSMAS II.[4] This is achieved mainly through supporting its familiar query language. In an effort to accommodate a great number of additional search and analysis features without modifying this query language, Bingel and Diewald (2015) introduced the KoralQuery protocol, a JSON-LD (Sporny et al., 2014) based pivot format for corpus queries and the central webservice communication protocol of KorAP (see Fig. 2). The protocol distinguishes several types of query objects, pertaining to the search in arbitrary annotation layers (see Sec. 1 in Fig. 2) or the creation of virtual corpora (see Sec. 2 in Fig. 2), among others (for the specification draft, see Diewald and Bingel, 2015).
Its abstract and modular representation of queries strongly benefits the sustainability of KorAP in regard to the following aspects:

Stability. In using a common protocol, the components in KorAP live up to the same demands and specifications, which is crucial for the design of KorAP as a microservice-based system.

Backward compatibility. In abstracting away from concrete query languages, KoralQuery safeguards the support for existing languages. While this is of particular importance when a query engine is designed to replace an existing one (as in the case of KorAP and COSMAS II), it strongly simplifies development whenever additional query languages need to be supported by the system.

Forward compatibility. Additional search and analysis features can be integrated into KoralQuery through

[3] `https://github.com/KorAP`; licensed under BSD-2. Components not yet published are in preparation.

[4] `http://www.ids-mannheim.de/cosmas2/`

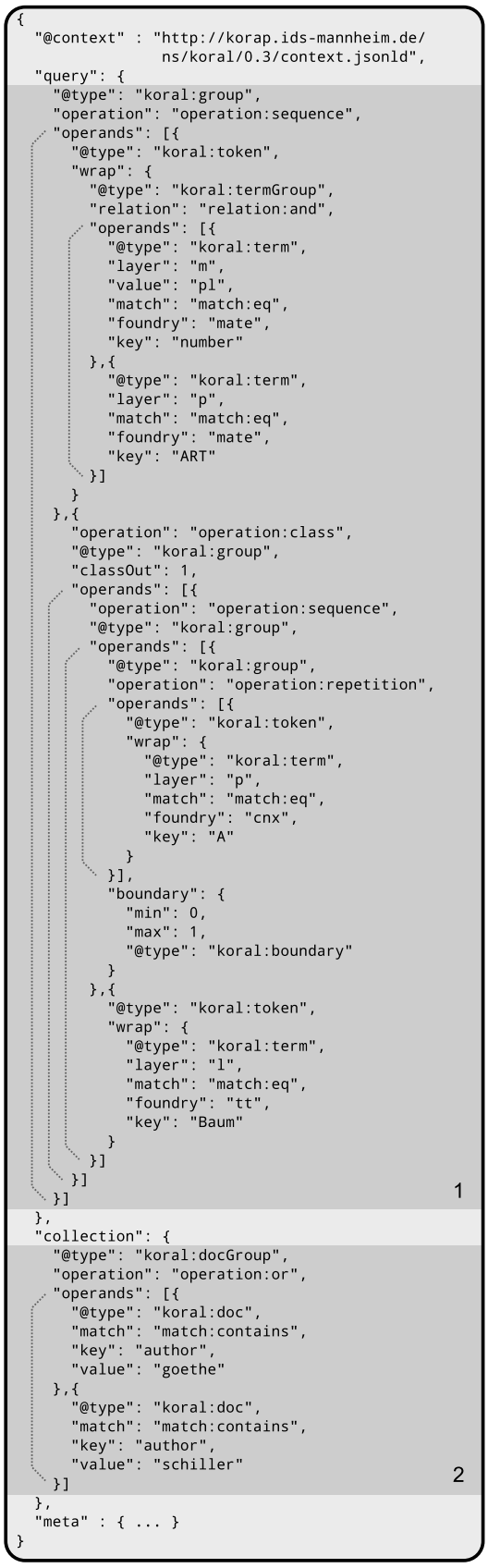

Figure 2: The query `[mate/m=number:pl & mate/p=ART] {1:[cnx/p=A]? [tt/l=Baum]}` (Poliqarp+) serialised as KoralQuery, with a virtual corpus defining all documents of the authors "Schiller" and "Goethe"

the definition of corresponding query object types (see below), thus extending the protocol to meet new demands when they arise. The protocol also reserves the possibility for components to transmit warnings or error messages upon being confronted with, for instance, novel constructs that cannot yet be processed by the respective component.

At its core, KoralQuery defines a set of linguistic types, for instance denoting tokens, syntactic constituents, or sequences of tokens. These types are defined as abstract structures that can flexibly address different annotation layers and sources at the same time, which ensures a high degree of independence from specific linguistic theories or annotation schemes. Theoretical insights into the desired set of supported features, as well as the scheme-independent design, initially came from a study of query language expressivity by Frick et al. (2012).

In addition to the serialisation of user-formulated queries, the protocol supports query rewrites. These may, for example, restrict the search space to specific documents that the user has permission to access (see Bański et al., 2014, and Sec. 2.2.). To make the user aware of any rewrites, the protocol provides a mechanism to report these modifications.

KoralQuery representations are generated from queries formulated in any of the supported query languages by the translation component *Koral* (Bingel, 2015), the reference implementation of KoralQuery's serialisation mechanism. Currently, these languages include COSMAS II QL, *Poliqarp* QL (a CQP variant; Przepiórkowski et al., 2004) *ANNIS* QL (Rosenfeld, 2010), and a subset of CQL, which is used in CLARIN Federated Content Search (CLARIN-FCS)[5]. Koral is used as a library by Kustvakt (see following Section) to translate queries to KoralQuery and use it as a common representation for different query languages.

2.2. Kustvakt

The scope of retrieval of individual texts is often restricted by intellectual property laws of various nature. Kustvakt's primary role, as the central user and corpus license management component, is to address this issue by examining queries and, if necessary, modifying the requested virtual corpus according to the permissions that the user has been granted, before the queries are passed on to the search backend (cf. Bański et al., 2014). An additional role that Kustvakt plays is to inject user-specific default values into queries (e.g. the preferred annotation sources for Part-Of-Speech). In cases where Kustvakt rewrites a query, the user gets notified about the nature of the rewrite and the reason for it.

Kustvakt provides a REST-like API and integrates with standardised authentication and authorisation schemes to allow multiple user clients to access KorAP.

[5] `https://www.clarin.eu/content/federated-content-search-clarin-fcs`; currently implementing the latest FCS 1.0 specification supporting basic CQL search such as term and phrase queries, accessible through the SRU/CQL protocol (`http://clarin.ids-mannheim.de/korapsru`) and the FCS Aggregator (`http://weblicht.sfs.uni-tuebingen.de/Aggregator/`).

2.3. Krill

Krill is the primary search and analysis backend for KorAP and the reference implementation for KoralQuery consumption. It is based on Apache Lucene[6] and provides support for efficient search on metadata, primary data, and multiple layers of annotation data.

To narrow down searches to documents that are part of virtual corpora, Lucene-based filters are applied, for instance to bundle documents by a certain author or from a certain range of time.

In view of the wide variety of query constructs supported by KoralQuery, Lucene's `SpanQuery` collection was vastly extended. Enhancing Lucene's native positional search, Krill supports distance search based on various distance units (e.g. word, sentence) and other complex constraints (e.g. minimum and maximum distance, occurrence order), for instance to find all adjectives within the distance of maximum 3 words from a certain noun. Besides, it can search for constituents embedded in larger phrase structures by position (e.g. find the preposition "out" at the end of a verb phrase). Both hierarchical and relational queries are applicable in Krill, for instance using the dominance and relation operators in ANNIS QL to specify hierarchy and dependency relations. All supported queries can be combined forming a complex nested query (e.g. find a plural pronoun within a verb phrase modifying a noun phrase).

2.3.1. Kanalito

Krill can be used either as a library or as a stand-alone microservice providing access to the search index via KoralQuery and a REST-like API. In the latter scenario, a Krill service can take part in a cluster of independent nodes, managed by a central service. In KorAP, the central management service for distributed Krill nodes is called *Kanalito*.

Kanalito accepts KoralQuery requests and passes them to registered Krill nodes in parallel. Each Krill node is responsible for just a subset of the corpus (a so-called "shard"; see Stonebraker, 2010). That makes the KorAP search backend scalable horizontally. Query results are afterwards aggregated, sorted and returned (see Fig. 3, top).

Kanalito is able to sort results by several criteria such as the frequency of matches in a document. It is also possible to retrieve statistical information on metadata, like how many matches were found in a certain range of time or in a certain genre. Long-running processes, such as conversion and indexation of new documents, are scheduled using a Job Queue and performed by Kanalito worker nodes (see Fig. 3, bottom). In the future, long-running processes may include complex analysis with non-realtime results as well. Kanalito is designed to be a thin orchestration tool for Krill nodes and will steadily adopt advances in Krill's capabilities. Tasks currently handled by Kanalito will be delegated to Krill nodes in case parallelisation is beneficial. Well-established distribution solutions for Lucene, like Apache Solr,[7] were investigated and may become part of Kanalito in the future.

[6] `https://lucene.apache.org/`

[7] `https://lucene.apache.org/solr/`

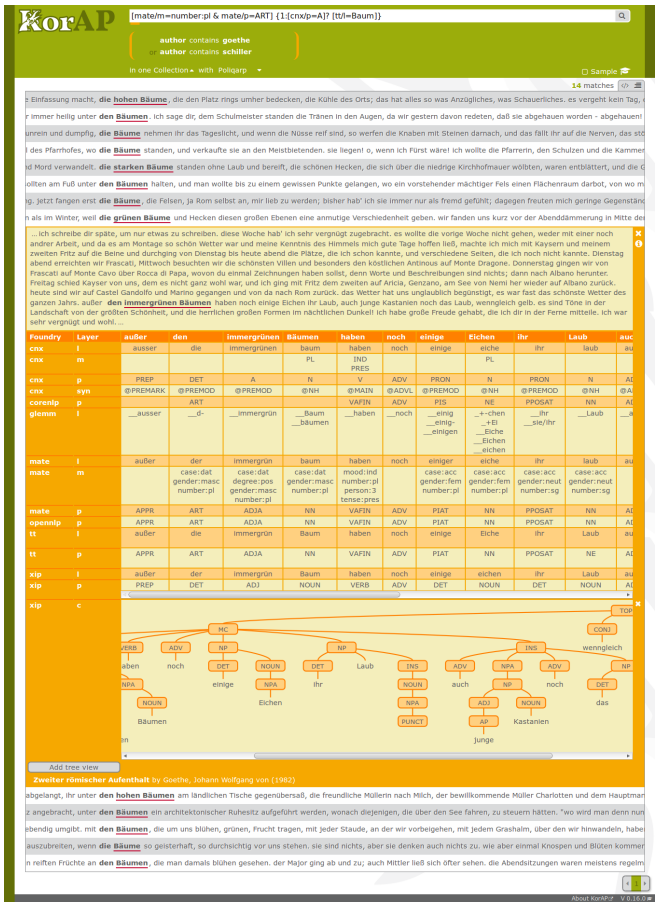

Figure 4: Screenshot of the Kalamar User Interface, with KWIC-, snippet-, table- and tree-views of search results

Following the concept of replaceability of microservices, KorAP supports multiple search backends. *Karang*, for example, is an alternative search and analysis backend based on the Neo4j[8] graph database, focussing on complex dependency queries and collocation extraction.

2.4. Kalamar

With Kustvakt providing a central Web API for all KorAP functionalities, a user interface component for search and result visualisation can be implemented as a simple microservice, too. One such user interface for KorAP is Kalamar.

Kalamar has a focus on usability, trying to represent the simplest frontend possible while providing full access to the functionalities of KorAP. Tools were implemented to assist the user in formulating research questions, like an annotation helper suggesting possible annotations to search for (e.g. by providing lists of Part-of-Speech tags associated to a certain annotation layer) and a virtual corpus helper assisting the user in creating complex virtual corpora (by means of nested combinations of metadata criteria).

After issuing a search request, results are presented in simple KWIC (Keywords in Context) views. Afterwards results can be expanded with a larger context and enriched by retrieving metadata, showing token-based annotations in a table view, and relational annotations in tree views (see

[8] `http://neo4j.com/`

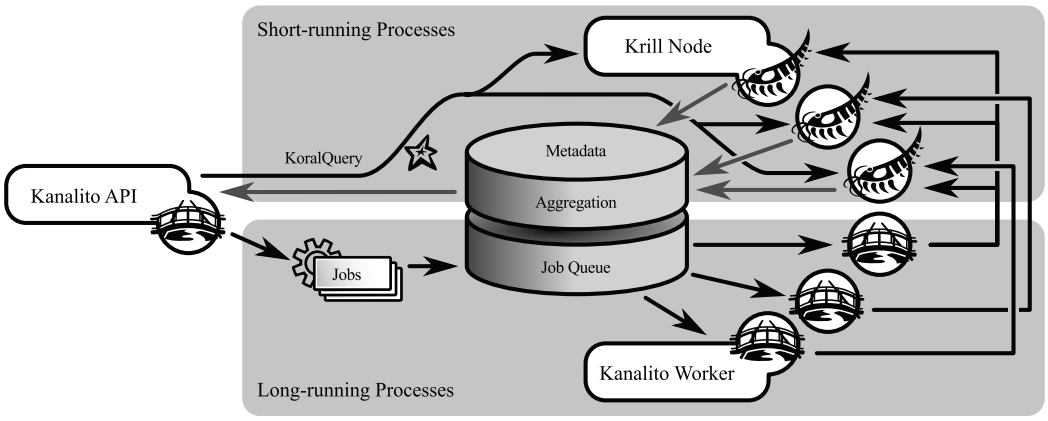

Figure 3: Short- and long-running processes in the Kanalito workflow

Fig. 4). These requests directly correspond to Kustvakt Web API calls, granting transparent access to the underlying corpus data.

3. Related Work

In the beginning of the project, various corpus analysis systems were evaluated for their aptitude to succeed COSMAS II (Bański et al., 2012). The *IMS Corpus Workbench* (CWB; Christ, 1994; Evert and Hardie, 2011) was considered as being a mature solution for most requirements, but was discarded due to its limitation of 2.1 billion tokens per corpus at that time (Evert and Hardie, 2011). Since then, corpus technologies have evolved: The next generation of CWB, for example, will not have the aforementioned limitation of corpus size (Evert and Hardie, 2015), and therefore can be considered a valid alternative to KorAP's search and analysis backend.

BlackLab,[9] actively developed at the Institute for Dutch Lexicology, is a corpus query engine that provides a similar set of features as Krill and uses the same underlying search engine. It handles multiple query languages and supports similar query constructs. Although it uses Lucene as well, it applies different indexing strategies. Thus, it can be a good candidate for performance comparison in the future and may serve as an alternative search backend. The same is true for *MTAS*, a similar search engine developed at the Nederlab,[10] focusing on a tight integration with Apache Solr.

Corpus query technologies based on relational databases (RDBMS) like ANNIS (Zeldes et al., 2009) focus on large sets of query features, but are rarely considered for large corpora, although recent evaluations have shown parallelisation to be a promising approach for reasonable performance using RDBMS for corpus analysis (Schneider, 2012). The design of KorAP architecture allows for straightforward adaptation of future corpus technologies.

For several years, transnational infrastructures have been implemented – in Europe this process is facilitated through the so-called ESFRI-roadmap. An infrastructure designed for dealing with language resources is CLARIN (Váradi et al., 2008). Following its API-first development approach, KorAP allows for an unobstructed integration into CLARIN. Kustvakt supports authentication mechanisms favored by CLARIN (i.e. Shibboleth) and provides OAuth-2.0-based authorization flows to grant access to the language resources of DeReKo (cf. Diewald and Kupietz, 2016). KorAP already participates in CLARIN-FCS and further support is in preparation (e.g. integration of CLARIN-Virtual-Collection-Registry and FCS 2.0).

4. Conclusion

KorAP is a corpus search and analysis platform for large corpora focussing on scalability, flexibility, and sustainability. A microservice-based architecture is a promising approach to fulfill these requirements, as described in this paper. The system consists of small components that are easy to extend, replace, and maintain. While KorAP is still under development, it is already in use for special corpora (cf. Fischer et al., 2016) and part of ongoing collaborations (Cosma et al., 2016). Being free and open source software, KorAP welcomes contributions from everyone.

Acknowledgements

KorAP is developed at the Institute for the German Language (IDS), member of the Leibniz-Community[11] and supported by the KobRA[12] project, funded by the Federal Ministry of Education and Research (BMBF).[13] The authors would like to thank Elena Frick, Peter Harders, Piotr Pęzik, and Carsten Schnober for their contributions to KorAP, and the anonymous reviewers for their constructive comments.

Bibliographical 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 *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, pages 2905–2911.

[9] http://inl.github.io/BlackLab/

[10] https://www.nederlab.nl/

[11] http://www.leibniz-gemeinschaft.de/en/about-us/leibniz-competition/projekte-2011/2011-funding-line-2/

[12] http://www.kobra.tu-dortmund.de

[13] http://www.bmbf.de/en/

Bański, P., Bingel, J., Diewald, N., Frick, E., Hanl, M., Kupietz, M., Pezik, P., Schnober, C., and Witt, A. (2013). KorAP: the new corpus analysis platform at IDS Mannheim. In Zygmunt Vetulani et al., editors, *Human Language Technologies as a Challenge for Computer Science and Linguistics. Proceedings of the 6th Language and Technology Conference*, Poznań. Fundacja Uniwersytetu im. A. Mickiewicza.

Bański, P., Diewald, N., Hanl, M., Kupietz, M., and Witt, A. (2014). Access Control by Query Rewriting: the Case of KorAP. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2014)*, Reykjavik, Iceland, May.

Bingel, J. and Diewald, N. (2015). KoralQuery – a General Corpus Query Protocol. In *Proceedings of the Workshop on Innovative Corpus Query and Visualization Tools at NODALIDA 2015*, Vilnius, Lithuania, May.

Bingel, J. (2015). Instantiation and Implementation of a Corpus Query Lingua Franca. Master's thesis, University of Heidelberg, Germany.

Bodmer, F. (1996). Aspekte der Abfragekomponente von COSMAS II. *LDV-INFO*, 8:142–155.

Christ, O. (1994). A modular and flexible architecture for an integrated corpus query system. In *Proceedings of COMPLEX '94*, pages 23–32, Budapest. `http://cwb.sourceforge.net/files/Christ1994.pdf`.

Cosma, R., Cristea, D., Kupietz, M., Tufiş, D., and Witt, A. (2016). DRuKoLA – towards contrastive germanromanian research based on comparable corpora. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Portorož, Slovenia, May. Accepted.

Diewald, N. and Bingel, J. (2015). KoralQuery 0.3. Technical report, Institut für Deutsche Sprache, Mannheim, Germany. `http://korap.github.io/Koral/`; Working Draft.

Diewald, N. and Kupietz, M. (2016). Integration der KobRA-Verfahren in die IDS-Infrastrukturen. Technical report, IDS Mannheim. `http://www.kobra.tu-dortmund.de/mediawiki/images/1/17/KobRA_Technischer_Bericht_IDS_Meilenstein_4c.pdf`; last accessed on March 3, 2016.

Evert, S. and Hardie, A. (2011). Twenty-first century corpus workbench: Updating a query architecture for the new millennium. In *Proceedings of the Corpus Linguistics 2011 Conference*, Birmingham, UK.

Evert, S. and Hardie, A. (2015). Ziggurat: A new data model and indexing format for large annotated text corpora. In Piotr Bański, et al., editors, *Proceedings of the 3rd Workshop on Challenges in the Management of Large Corpora (CMLC-3)*, pages 21–27, Mannheim. Institut für Deutsche Sprache.

Fischer, P. M., Diewald, N., Kupietz, M., and Witt, A. (2016). Aufbau einer Korpusinfrastruktur für die Beobachtung des Schreibgebrauchs. In *Modellierung – Vernetzung – Visualisierung: Die Digital Humanities als fächerübergreifendes Forschungsparadigma. Digital Humanities im deutschsprachigen Raum (DHd 2016)*, Leipzig, Germany, March.

Frick, E., Schnober, C., and Bański, P. (2012). Evaluating query languages for a corpus processing system. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, Istanbul, Turkey, May.

Kupietz, M. and Lüngen, H. (2014). Recent Developments in DeReKo. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2014)*, Reykjavik, Iceland, May.

Lewis, J. and Fowler, M. (2014). Microservices. Blog Post; `http://martinfowler.com/articles/microservices.html`; last accessed on October 12, 2015.

Mell, R. M. and Diewald, N. (2016). Korpusbasierte Diskurs-Recherche mit Rabbid. Talk. Accepted for Germanistentag 2016 as part of the panel "Erzählen in digitalen Diskursen: Die narrative Dimension der Neuen Medien".

Newman, S. (2015). *Building Microservices. Designing Fine-Grained Systems*. O'Reilly Media.

Przepiórkowski, A., Krynicki, Z., Debowski, L., Wolinski, M., Janus, D., and Bański, P. (2004). A search tool for corpora with positional tagsets and ambiguities. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*, pages 1235–1238.

Rosenfeld, V. (2010). An implementation of the Annis 2 query language. Technical report, Humboldt-Universität zu Berlin.

Schneider, R. (2012). Evaluating DBMS-based Access Strategies to Very Large Multi-layer Annotated Corpora. In *Proceedings of the Workshop on Challenges in the management of large corpora (CMLC-1) at the seventh International Conference on Language Resources and Evaluation (LREC 2012)*, Istanbul, Turkey, May.

Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., and Lindström, N. (2014). JSON-LD 1.0 – A JSON-based Serialization for Linked Data. Technical report, W3C. W3C Recommendation, `http://www.w3.org/TR/json-ld/`.

Stonebraker, M. (2010). SQL Databases v. NoSQL Databases. *Communications of the ACM*, 53(4):10–11.

Váradi, T., Wittenburg, P., Krauwer, S., Wynne, M., and Koskenniemi, K. (2008). CLARIN: Common Language Resources and Technology Infrastructure. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008)*, pages 1244–1248, Marrakech, Morocco, May.

Zeldes, A., Ritz, J., Lüdeling, A., and Chiarcos, C. (2009). ANNIS: A Search Tool for Multi-Layer Annotated Corpora. In *Proceedings of Corpus Linguistics 2009*, Liverpool, UK.