Mining and Analysing One Billion Requests to Linguistic Services

Marco Büchler, Greta Franzini, Emily Franzini
eTRAP Research Group
Georg-August-Universität Göttingen
Göttingen, Germany
mbuechler|gfranzini|efranzini@etrap.eu

Thomas Eckart
Natural Language Processing Group
Universität Leipzig
Leipzig, Germany
teckart@informatik.uni-leipzig.de

***Abstract*—The Leipzig Linguistic Services (LLS) are a SOAP-based cyberinfrastructure of atomic micro-services for the Wortschatz project, which covers different-sized textual corpora in more than 230 languages. The LLS were developed in 2004 and went live in 2005 in order to provide a webservice-based API to these corpus databases. In 2006, the LLS infrastructure began to systematically log and store requests made to the text collection, and in August 2016 the LLS were shut down. This article summarises the experience of the past ten years of running such a cyberinfrastructure with a total of nearly one billion requests. It includes an explanation of the technical decisions and limitations but also provides an overview of how the services were used.**

***Keywords*-natural language, web services, internet, collective intelligence, knowledge discovery, data processing, data analysis, software as a service**

I. Introduction

"What are the implications of Digital Transformation[1]?" This is the question we must ask since over the past two decades the huge amount of data gathered in galleries, libraries, archives and museums (GLAM institutions) is increasingly being turned into digital ecosystems. One immediate benefit of the transformation is that the interested public can now have access to an unprecedented amount of data. The drawback is that the extraordinarily high number of data-sets overwhelm users, creating a need for additional analysis tools for texts, objects, audio and video data.

The growing amount of digitally available data has raised user interest. The many ways in which users now wish to explore it have risen questions about the ways in which access to the data and the analysis tools is provided.

Granting everybody indiscriminate access to, for example, an Application Programming Interface (API) on a server hosting a few hundred gigabyte of data can mean that users with little knowledge may, for example, cause a user-initiated-denial-of-service attack that could crash the entire cyberinfrastructure.

In the mid 1990s, the Natural Language Processing Group at the University of Leipzig began work on the Wortschatz project (details in section III), which aims to provide corpora in hundreds of languages and in different size-normalisations, be that 100k, 300k or 1M sentences. As the resources grew in size, so did the number of requests for the data. In the early stages of the project a specific dump was created, parts of which even came with a small user-interface. The database dump was shared with interested researchers and partners in the business sector.

However, the personnel costs of the project became unsustainable. For this reason, a new plan was put into motion in 2004, consisting of the development of a SOAP-based API - the Leipzig Linguistic Services (LLS) - that enabled any interested person to access the data of the Wortschatz databases in any provided language. Overall 20 services were provided, delivering specific information such as baseform, category classifications, and thesaurus data. The aim of the LLS was to establish a Service Oriented Architecture (SOA) for linguistic resources based on small and atomic micro-services that could be combined by users for their particular needs. Users were then not only able to browse through the Wortschatz website, but also to integrate those services with their own existing digital ecosystems.

In 2005 these services went live and by September 2006 all requests were systematically logged. In July 2014 the number of logged requests reached one billion. While at the beginning the use was limited to academia, over time the services were increasingly used by the private and business sectors as well.

This article is structured as follows: we first describe infrastructure projects. This will be followed by two sections on the Wortschatz project and the LLS, as well as a section on the description of the log-file format. Next, we will present our experiences and results from the analysis of nearly one billion requests, stored in 189GB of uncompressed log-files. We will then describe which services were chained together by users on the client-side in order to identify potential aggregated server-side webservices and to deliver results on the dependency between number of requests per second and the response time. The intent of this article is to report on our experience for the improvement of load-balancing strategies in future projects.

[1]For a short description of Digital Transformation, see: https://goo.gl/8cRzVa (Accessed: 7 October 2016).

II. Related work

For the past ten-fifteen years, large (cross-)disciplinary Research Infrastructures (RIs), including Bamboo[2], CLARIN[3], DARIAH[4], CLARIAH[5] and Europeana[6], have been providing a variety of applications, materials and services to support the (Digital) Humanities, Social Sciences and related research communities [4]. A number of these RIs allow user access and participation via typically web-based interfaces, such as metadata search engines and data editors[7]. However, most of these infrastructure initiatives are primarily driven by the resource providers, and, among other shortcomings [4][5], occasionally lack the direct influence of their intended user groups.[8] As a consequence, the shared services that RIs provide can only be evaluated against the requirements and rare innovative contributions of their users. Yet, it is these requirements and contributions that ground and add value to these initiatives, acting as natural drivers for a constant adaptation and extension of infrastructure ecosystems.

In this context, the management of user feedback is largely based on static and time-consuming procedures (e.g. ticketing systems) and can only give a rough understanding of problems encountered. A more direct and representative feedback system is desirable to address the particularly problematic issues of data absence or misleading interfaces. Such a system would be especially important for strongly integrated infrastructures like CLARIN, which are designed as service-oriented architectures based on web-services. For those, the real interaction of end-users with provided services and their combination with complex service-chains also affect load-balancing efficiency or the usefulness of infrastructure under heavy-load.

III. The Wortschatz project

The *Deutscher Wortschatz* project or *Leipzig Corpora Collection* project (hereafter LCC)[9] began more than twenty years ago with the aim of creating corpus-based monolingual dictionaries for the German language [6]. Since then, it has expanded its scope to the collection and enrichment of digital texts in as many languages as possible. The enrichment process encompasses a variety of linguistic, and especially statistical, information. Amongst this type of information, there is a dictionary containing data about word frequency, sample sentences and statistically-significant word co-occurrences based on different window sizes [7].

The texts are acquired with different crawling strategies. Through a self-developed distributed crawling environment, called *FindLinks*[10], these processes are continuously revised and perform downloads of complete top-level domains as well as daily news in more than 80 languages via RSS feeds. Based on the materials collected, a variety of user-interfaces were developed to provide user-friendly access to the data and its associated applications.

To this day, the LCC provides textual materials in more than 230 languages, and the collection keeps growing. Table I shows the available amount of text material for a small selection of languages (measured in number of available sentences).

Language	*Number of sentences (in M)*	*Language*	*Number of sentences (in M)*
English	1,110	Georgian	30
German	1,023	Bokmål	27
Russian	456	Modern Greek	25
Spanish	244	Lithuanian	20
French	178	Catalan	16
...	...	...	...

Table I
Text material of the Leipzig Corpora Collection (excerpt)

IV. The Leipzig Linguistic Services (LLS)

The intention of the overall LLS architecture was to be as simple and generic as possible. A generic architecture can be reused in different scenarios but tends to have too many parameters and options, while a simple architecture claims usability and guarantees a faster learning curve. In the following, we describe the architecture, the editing workflow for adding a new webservice, the technical decisions and the usage of the LLS.

The LLS are entirely developed in Java. In 2004, when development began, `Java 1.4` was the most commonly used programming language to run under different operating systems, such as `Linux`, `MacOS` and `Windows`. Even though the plan was to run the services on a `Linux`-based server, their design made it possible to adapt to different and project-based operating systems.

The LLS run in an Apache `Tomcat 5.0.19`[11] with the Apache `Axis 1.3`[12] implementations. Later experiments with Apache `Axis 2` showed that, at least for the LLS, interoperability with clients from different programming languages worsened. This is the reason why the productive system retained the older version. In order to achieve a simple architecture, we used Apache `Ant 1.6.5`[13]. `Ant` was used as *the* central tool for generating the back-end services and deploying them in a `Tomcat` server (see red

[2]Link: https://goo.gl/Ldkicc (Accessed: 2 October 2016).

[3]Link: https://goo.gl/2iU4B4 (Accessed: 2 October 2016).

[4]Link: https://goo.gl/cilqED (Accessed: 2 October 2016).

[5]Link: https://goo.gl/x76TrW (Accessed: 2 October 2016).

[6]Link: https://goo.gl/JDpWTw (Accessed: 2 October 2016). While not strictly an RI, Europeana is integral to European research [4].

[7]For example, the CLARIN metadata repository at: https://goo.gl/k5sCS7 (Accessed: 5 October 2016).

[8]Research libraries for example have yet to strongly engage with RIs [4].

[9]Link: https://goo.gl/Mb6kT (Accessed: 2 October 2016).

[10]Link: https://goo.gl/n069bM (Accessed: 6 October 2016)

[11]Link: https://goo.gl/FiSm (Accessed: 7 October 2016).

[12]Link: https://goo.gl/t5EZo (Accessed: 7 October 2016).

[13]Link: https://goo.gl/U7ANa (Accessed: 7 October 2016).

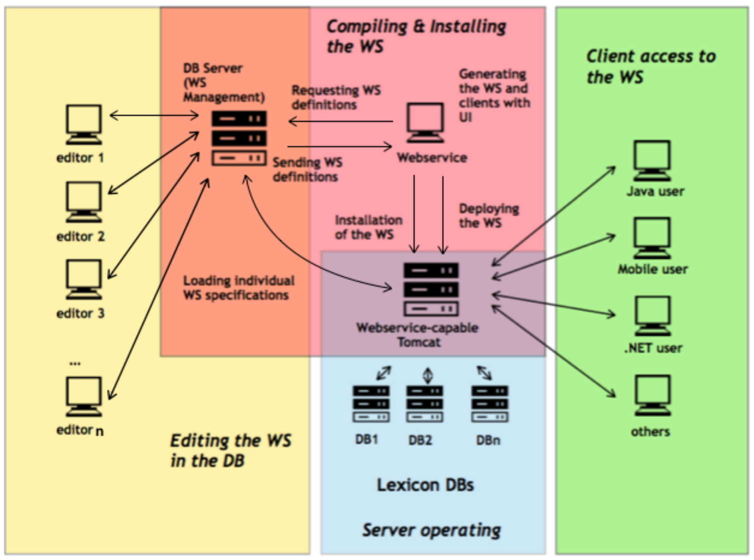

Figure 1. Four workflow modes with separation of concern: editing (yellow); managing, compiling and deploying (red); hosting and operating (blue); using the LLS infrastructure (green).

zone in figure 1). For this, different `Ant` tasks were written in order to create Java code out of a data-set description.

In order to create the server-side Java code for a specific webservice, a data-set needed to be added to the webservice management. This required the following information: `Id`, `WebServiceType`, `Algorithmus`, `InputFields`, `Name`, `Magnitude`, `StdLimit`, `Description`, `Status` and `AuthorizationLevel` (see yellow zone in Figure 1). `Id` describes a running and unique number that is provided by the system. `WebServiceType` expects a Java class that implements the `WSType` Java interface containing the two methods, `ping` and `execute`; while the latter contains the execution code for a service, the `ping` method serves to integrate the client-side with monitoring tools in order to check if the system is alive (i.e. by sending a ping and waiting for the corresponding pong). A `WSType`-implementing class can be any task to which it is possible to respond within an `HTTP` timeout of 60 seconds. Table II shows the selected types in the "Webservice Types" column. In the LLS production system only two Webservice types, `MySQLSelect` and `MARS`[14], were chosen. The first type implements a selection to a `MySQL` database, while the `MARS` type implements a collection of database requests as an aggregated service, which was originally built for a Max Planck Institute and later made available for others. `Algorithmus` expects values and parameters specific to a webservice type. `InputFields` describes the parameters that a webservice expects (see also "Input Fields" column in Table II). The `Name` field represents the name under which the service is deployed and made available. `Magnitude` is a cost measure for each request to facilitate the creation of a business model for commercial users. `StdLimit` sets a default value for limiting results in case a user does not submit a limit with a request. A value of 20 was chosen by default as lower limits significantly decrease the response time. `Description` contains a short description of what the webservice does in terms of displaying the information on the website. `Status` expects one of two values, `ACTIVE` or `INACTIVE`. Switching a service from active to inactive mode restricted its availability to users (i.e. no longer or temporarily available). This feature was especially useful when maintenance work had to be done on the databases. Upon reactivation, the service would be immediately restored for public use. `AuthorizationLevel` (see the "Access level" column in table II) indicates if access to the webservice is public or restricted to user accounts with special rights. This control measure was necessary as some services are time-consuming and thus not fit for public use.

Once an editor (see yellow zone in figure 1) had edited a webservice description, a dedicated person such as the system administrator would start the `Ant`-based compilation of the webservice. This included the server-side code generation and the deployment of the service in Apache `Tomcat`. Once the service was deployed, the Webservice Description Language file (WSDL) was automatically downloaded by the `Ant`-script and transformed into Java code. As `SOAP`-based webservices do not provide a user interface, the `Ant`-script generated a `Swing`-based user interface for each deployed webservice, which was automatically uploaded to `Tomcat` as well. Furthermore, the generated Java code is

[14]The values of `WebServiceType` expect a Java class including the package. To optimise space, table II does not display packages.

used to execute the `ping`-method in order to check if the service is ready. Once all installed services had successfully sent a pong back, the compilation and installation step was finished (see red zone in figure 1). The server operation (see blue zone in figure 1) of the Wortschatz databases are distributed on dedicated DB servers (see also section III).

With the webservices deployed, users were able to either download the pre-compiled Java client and integrate it into their own digital ecosystems or use the `Swing`-based graphical user interface of the clients. Additionally, the idea of `SOAP` is to provide a generic description of a webservice in the `WSDL`-file in order to generate the client source code in further programming languages, such as in `C#` as part of `.NET`[15] (see also [2]), `Perl`[16], `Python`[17], `Delphi`, `PHP` [18], `Ruby`[19] and `JavaScript`[20] (see green zone in figure 1). Independently from the underlying programming languages, over the past ten years we have observed different uses in research, business and in the private sector. In research, the LLS were used in the areas of text profiles and author classification [1]. The services were also used as data resources for sentiment analysis in research groups in Germany, Austria and South Africa. Users from the business field were mainly interested in using `Baseform` or `Synonym` services for improving internal search indexes. The LLS data was also used for information retrieval tasks in portals for weighting words in a word cloud or to display enriching information. Private users accessed the LLS to complete crossword puzzles. A dedicated service was installed upon request just for this purpose (see also table II), since it was possible to query a pattern of an incomplete word with a given word length limitation.

Another application for private and business users was the `Thesaurus` service, which could be used to request similar words given a specific context. OpenOffice, for example, does not perform as well in this task [2]. It was observed that the LLS infrastructure worked well for a user, because it avoided dealing directly with an unmanageable amount of data, while also enabling the integration of millions of data-sets accessible in users' environments.

From 2008 the SOA-based cyberinfrastructure LLS were used in Digital Humanities projects such as eAQUA[21] and eTRACES[22]. Their services were also integrated into environments such as OpenOffice in order to deliver services right within draft publications and let users interact with huge data collections. In this way, users could, for example, mark a word and look for specific occurrences of this word in Ancient Greek or Latin texts, and also add references to specific text-passages to their publication [3].

[15]Link: https://goo.gl/CpIdui (Accessed: 7 October 2016).
[16]Link: https://goo.gl/ecaavc (Accessed: 7 October 2016).
[17]Link: https://goo.gl/nxxqAw (Accessed: 7 October 2016).
[18]Link: https://goo.gl/1Lsh56 (Accessed: 7 October 2016).
[19]Link: https://goo.gl/jbvxWP (Accessed: 7 October 2016).
[20]Link: https://goo.gl/WCw8wN (Accessed: 7 October 2016).
[21]Link: https://goo.gl/Nwnsia (Accessed: 5 October 2016).
[22]Link: https://goo.gl/3MKcjs (Accessed: 5 October 2016).

V. Description of the log-file data

The LLS were designed to write one entry in the log-files for both the request and the response. As table II shows, LLS registered 965 million requests and responses forming 1.93 billion lines in the log-files and occupying 189GB of disc space. The decision to split the requests and responses into two lines per entry was based on the desire to display how many active requests were in the LLS infrastructure. Unlike requests made to a website, whose answers are typically delivered within a second, LLS responses could take up to 40 seconds if there was still some computation necessary, as was the case for the `Common Co-occurrence` service. The detection of the number of active requests can be easily achieved by computing the difference between logged requests and responses.

The LLS log-files provide daily snapshots of the 965 million requests made between 19th September 2006 and 29th July 2014, when the project stopped recording this information. A typical request looks like this[23]:

```
2006-09-19T08:43:32+01:00 - anonymous -
Baseform - 81.169.187.22 - IN - 0 - execute -
Wort=privilegium majus
```

where the hyphens separate data columns. The first column is the time-stamp in precise seconds; the second column contains the user account information; the third column represents the requested service (`Baseform`); the fourth column contains the client IP address; the fifth column indicates an incoming request; the sixth column indicates the status of the service, which can be either `0` (=ACTIVE) or `1` (=INACTIVE) (see also section IV); the seventh column indicates that the `execute`-method has been called. Alternatively, this field can contain `ping`. The last column contains the key-value-pair of the query (in this example the Latin multi-word unit *privilegium majus*) submitted to the service, be that *word*, *word length*, *significance threshold*, *pattern* or *limit* (see also table II).

The response to the above request is:

```
2006-09-19T08:43:32+01:00 - anonymous - Baseform -
81.169.187.22 - OUT - 0 - execute - (0, 0) - 0.03s
```

where the fifth column marks the response with `- OUT -`. The eighth column contains the resulting `String[][]`-match for the query (in this case `0,0`, i.e. none). A result of, for example, `(2,10)` would indicate that a data matrix is returned with two columns and ten rows. The second value is typically bound to the `Limit` parameter (see also table II). The ninth column displays the response time.

[23]All requests have been anonymised for privacy protection and publication. We plan to publicly release this data-set for the benefit of interested researchers and service providers.

Service	*Requests*	*Requests (%)*	*Non-empty responses*	*Coverage (%)*	*Input Fields*	*Webservice Type*	*Access level*	*Installation date*
Baseform	624,275,884	64.636%	315,724,185	50.57%	W	MySQLSelect	FREE	04/2005
Category	120,476,452	12.473%	43,276,840	35.92%	W	MySQLSelect	FREE	04/2005
Thesaurus	69,573,648	7.203%	37,151,565	53.39%	W, L	MySQLSelect	FREE	04/2005
Synonyms	60,745,973	6.289%	2,719,544	4.47%	W, L	MySQLSelect	FREE	04/2005
Sentences	60,087,714	6.221%	11,536,172	19.19%	W, L	MySQLSelect	FREE	04/2005
Wordforms	12,671,302	1.311%	4,309,791	34.01%	W, L	MySQLSelect	FREE	04/2005
Frequencies	11,932,213	1.235%	8,095,420	67.84%	W	MySQLSelect	FREE	04/2005
LeftCollocationFinder	1,416,001	0.146%	295,714	20.88%	W, PoS, L	MySQLSelect	FREE	10/2005
RightCollocationFinder	1,379,356	0.142%	235,323	17.06%	W, PoS, L	MySQLSelect	FREE	10/2005
Cooccurrences	1,057,722	0.109%	629,795	59.54%	W, ST, L	MySQLSelect	FREE	04/2005
RightNeighbours	959,560	0.099%	567,870	59.18%	W, L	MySQLSelect	FREE	04/2005
LeftNeighbours	731,449	0.075%	473,600	64.74%	W, L	MySQLSelect	FREE	04/2005
Similarity	467,809	0.048%	308,877	66.02%	W, L	MySQLSelect	FREE	10/2005
CooccurrencesAll	20,852	0.002%	20,848	99.98%	W, ST, L	MySQLSelect	INTERN	05/2009
ExperimentalSynonyms	20,779	0.002%	14,860	71.51%	W, L	MySQLSelect	FREE	12/2009
Crossword puzzling	2,902	$< 0.001\%$	1,306	45.00%	W, WL, L	MySQLSelect	FREE	10/2005
MARSService	616	$< 0.001\%$	616	100.00%	W, L	MARS	INTERN	10/2006
NGrams	564	$< 0.001\%$	149	26.41%	P, L	MySQLSelect	FREE	08/2011
NGramReferences	409	$< 0.001\%$	87	21.27%	P, L	MySQLSelect	FREE	08/2011
Common co-occurrence	55	$< 0.001\%$	43	78.18%	W1, W2, L	MySQLSelect	INTERN	10/2005
TOTAL	965,821,260		425,362,605					

Table II

OVERVIEW OF REQUESTS MADE TO LLS BETWEEN 2006-2014, IN DESCENDING ORDER. THE *Responses* COLUMNS ONLY LIST RESPONSES WHOSE VALUE WAS NOT EMPTY. FOR SPACE REASONS, THE VALUES IN THE *Input Fields* COLUMN ARE ABBREVIATED: *Word* (W.), *Limit* (L.), *Part of Speech pattern* (POS), *Significance Threshold* (ST), *Word length* (WL) AND *Pattern* (P)

VI. RESULTS

Table II provides an overview of the 20 services offered with a breakdown of the requests and the responses. Over half of the requests (64.6%) were made to the `Baseform` service. Similarly, services with high-quality and often manually-curated data, such as the `Thesaurus` and `Synonyms` services, were requested more often than the quantitatively-computed `Similarity` service, which provided similarly used words by assuming the distributional hypothesis [8] and thus compared the co-occurrence vectors of two words. Even if the coverage (see table II) for this service, 66.02%, is significantly higher than, for example, the `Category` (35.92%) or the `Synonyms` (4.47%) services, users appeared to prefer precision over recall for their end-user applications. Low coverage is also caused by requests to German language databases, especially by compound nouns that cannot all be included in a `Baseform` or `Category` service. Many multi-word units (MWU) were also requested. Out of all the requests, 84,760,875 (8.78%) were MWUs. With regard to the distribution of the webservice usage, only the two most frequently requested services, `Baseform` and `Category`, were queried more often than the total count of the MWU requests. This speaks to the impact of MWUs.

The less-used webservices in table II were primarily limited to internal uses, to newly installed services or, as was the case for the `Crossword Puzzling` service, to manual usage instead of automatic bulk-requests.

In the following subsections we investigate the data by attribution of requests, we detect user service chains and perform some load-balancing analyses.

A. Attribution of requests

As explained in section IV, a webservice in the LLS infrastructure could be requested for the `execute` and `ping` methods. Only 2,078,275 (0.22%) out of the total 965 million requests are pings. These will not be considered in this article beyond this point.

For every response there existed two different types of output: `- OUT -` or `- EXC -` . While the former represents a regular response, the latter expresses an exception that is caused by server communication issues. Out of all the log-files, we counted 674,830 exceptions, which cover about 0.07% of all responses thus guaranteeing an availability of $> 99.9\%$. Of these exceptions, 42.27% were caused by either an HTTP- or MySQL-timeout of 60 seconds each. 8.29% were caused by errors in the `Axis` webservice framework, which received invalid requests from clients. A further 5.39% and 4.52% were caused by service provider errors and service errors. In addition, 0.05% were out-of-memory errors that went so far as to cause a restart of the LLS framework. Finally, 7.65% of the exceptions were due to users submitting the wrong number of parameters (see also the "Input Fields" column in table II), and 0.24% of the requests did not specify any parameter, contrary to the expectation that users would specify at least one.

Overall, the LLS infrastructure shared data with 85 unique accounts. Additionally, and for practical reasons, an anonymous account was provided so that users could easily access all freely-available services (see also the "Access level" column in table II). The anonymous account sent a total 745,005,911 requests (77.14%) to the LLS infrastructure.

The following subsection details analyses pertaining to the geographical regions, the languages, the word cleanliness, as well as the temporal distribution of the requests.

1) By geographical regions: The open approach of the LLS, which allows anonymous access to the services, makes it difficult to get a thorough overview of the actual user-base and the users' geographical location. With the exception of specific logins by heavy users, the only traceable information is the IP address from which the request was sent. Thanks to this data we were able to extract the WHOIS protocol [15] information about the geographical provenance of the user. Table III contains the distribution of most frequent countries of origin for the 2006-2014 time-span. Requests where geographical information could not be unambiguously extracted from WHOIS responses (around 3.9%) were excluded.

Country	*Requests*	*Percentage*
Germany (DE)	921,184,562	99.29%
Ireland (IE)	2,003,348	0.22%
Swiss (CH)	1,957,431	0.21%
Austria (AT)	1,347,703	0.13%
Hungary (HU)	302,966	0.03%
Poland (PL)	212,357	0.02%
Japan (JP)	184,408	0.02%
Romania (RO)	90,140	0.01%
China (CN)	90,125	0.01%
France (FR)	82,969	< 0.01%

Table III
TOP-TEN LIST OF REQUESTS BY COUNTRY FOR THE YEARS 2006 - 2014

Table III shows that the vast majority of requests were sent from the same geographical region as the LCC, i.e. Germany or Central Europe, and specifically from countries where the most requested languages (German and English) are natively spoken. Moreover, it is not surprising that Ireland placed second. A deeper evaluation of associated IP addresses reveals that those specific requests were largely related to the popular cloud-computing platform Amazon Web Services, whose European servers are located in Ireland and Germany. But if we ignore this special case, the general pattern created over the nine-year period is clearly visible.

2) By language: As already described in section IV, the services provided text material based on various corpora for publicly shared languages. An analysis of requests can give an indication of the languages users were most interested in (see also table IV for the top-six requested language). As it could not be expected that all requests be issued to the correct database (i.e. with a matching language), which was the case for automatic text analyses without any client-side preprocessing, all queries were uniformly classified.

Given that the request types only require words or MWUs as input, it is hard to identify the correct languages in every case. In order to provide a well-founded estimate, an existing language identification tool developed at the LCC for the classification of complete sentences [9] was adapted. It was based on word lists containing the 100,000 most frequent words for all languages in question.

Language	*Requests (in %)*	*Language*	*Requests (in %)*
German	41.80%	Dutch	5.71%
English	36.65%	Italian	5.48%
French	5.82%	Spanish	4.53%

Table IV
PERCENTAGE OF REQUESTS IN PROVIDED LANGUAGES

A large number of the queries could not be classified (around 92%) as most of the requests are just one-word queries and are therefore difficult to automatically filter by language. But the remainder can be seen as a rough estimation for the complete data-set. Table IV shows this distribution of queried language material between 2006 and 2014. Despite the strong focus on German language material in the first years of the project, the LLS were later heavily queried for both German and English words. These languages combined account for the vast majority (around 78%) of all requests. This skewed distribution can be seen both as motivation to promote the existing services in other language communities and, in contrast, also as motivation to set future priorities on the more popular languages that seemingly cover an existing demand.

3) By cleanliness: An evaluation showed a significant amount of "broken" queries. This manifested itself in many encoding problems, including invalid lengths of queried terms, such as empty strings or verbose texts instead of single words or word groups. To get an insight into the extent of this problem, an existing tool was used to remove invalid input material based on blacklist patterns and simple language statistics [10]. These rules were adapted and extended to match the changed input type (i.e. short strings). Table V describes some of these rules and contains for each the number of queries that were regarded as "invalid".

Rule	*Matched requests (in % of all)*
Broken encoding	66,869,667 (6.920%)
Query too short	2,978,216 (0.310%)
URLs, HTML code, email addresses, etc.	189,895 (0.019%)
Query too long (more than 200 characters)	69,799 (0.007%)

Table V
APPLIED RULES FOR "CLEANLINESS" OF QUERIES (EXCERPT)

Overall, more than 71 million queries were regarded as "invalid" based on the applied rules (around 7.4% of all incoming queries). It is hard to determine why so many queries contain encoding errors, which in almost every case led to empty result-sets. One possible reason is again the automatic processing of large text collections without any preprocessing or encoding detection on the client-side.

4) By year: Figure 2 plots the yearly distribution of requests between 2006 and 2014, peaking in 2011 with

459,852,254 requests. The usage growth until 2011 was justified by strong dissemination and an efficient handling of the services. Between 2008 and 2011, the use of the LLS increased by three orders of magnitude, one order per year.

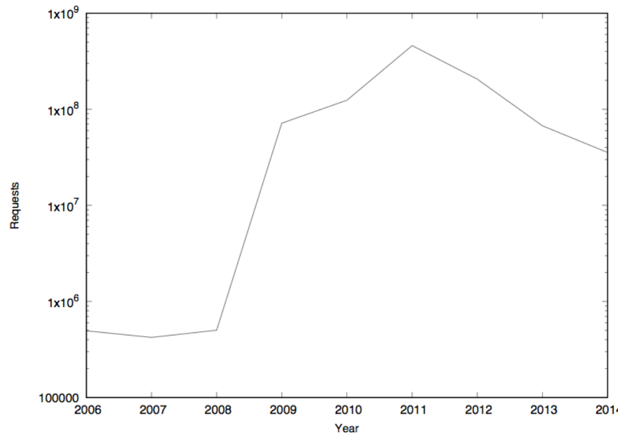

Figure 2. Number of requests per year between 09/2006 and 07/2014.

The decline from 2011 can also be explained. In 2011 two new services, `NGrams` and `NGramReferences`, were installed (see also table II) and Java was updated from `Java 1.4` to `Java 6`. Following these upgrades, the system encountered new memory leaks with loads as high as 60 requests per second. As a consequence, the incoming requests gradually filled the memory of the `Tomcat` server, producing `OutOfMemoryErrors`. Debugging became a complex task as it took roughly two months to discover an incompatibility between `Java 6` and the older `MySQL Connector 3.0.11`[24] driver.

Once the issue was finally fixed, a number of automatic scripts that used the LLS infrastructure were switched off. In the summer of 2011, based on the traffic and before the installation of the new services, we predicted that LLS would reach one billion requests in Q1/2012. However, due to the new installations and the consequent user-loss, LLS was only able to reach one billion requests in July 2014. This outcome serves to confirm the mantra "Never change a running system". At this time, the hot-deploy option had not been developed in the webservice framework `Axis` yet, but it was added later in version 2 (see also section VII).

B. Identification service chains

The LLS infrastructure was designed to provide atomic and loosely-coupled micro-services via the SOAP protocol. Each delivered service only provided a minimal core-function, which used alone often did not satisfy a sophisticated linguistic application or need. For this reason, we analysed the log-files to determine which chains of micro-services were observed on the client-side. The motivation for this analysis is twofold. First, to gain a deeper understanding of the sophisticated and aggregated applications that users built. Secondly, and from a service provider point of view, to reduce the number of requests: if a user sends requests from a self-built service chain (see table VI) of, for example, three involved services, then an aggregated service reduces the number of requests to one, concurrently reducing the total network latency.

For this reason, the top-six requested services `Baseform`, `Category`, `Thesaurus`, `Synonyms`, `Sentences`, `Wordforms` and `Frequencies` were selected for the analysis of service chains because they covered more than 99% of all requests (see table II). With the exception of the `Wordforms` service, which is not among the ten most frequently discovered chains, all other services occurred in at least two of the chains.

Unsurprisingly, the most popular service was the `Baseform`, which covered eight out of the ten most frequently discovered service chains, and seven out of those eight scenarios were the "openers" of a chain.

Rank	*Service chain*	*Percentage*
1	Baseform Frequencies	67.11%
2	Baseform Synonyms Sentences	26.32%
3	Synonym Sentences	3.00%
4	Baseform Synonyms	1.01%
5	Baseform Frequencies Synonyms	0.97%
6	Baseform Thesaurus	0.68%
7	Baseform Frequencies Category	0.24%
8	Baseform Category	0.24%
9	Frequencies Baseform Frequencies	0.23%
10	Thesaurus Similarity	0.20%

Table VI
LIST OF TOP-TEN MOST FREQUENTLY DISCOVERED SERVICE CHAINS

As table II shows, there were many re-appearing chain patterns, such as `Baseform Frequencies`, which occurred in the discovered chains with the ranks 1, 5, 7 and 9. It thus seemed reasonable to reuse the aggregated service (rank 1) in the service chains with ranks 5, 7, 9.

From an NLP angle, table VI displays six chains, represented by the ranks 2, 4, 5, 6, 7 and 8, following the `Baseform * [Synonym|Thesaurus|Category]*` pattern. This pattern implemented the traditional information retrieval task of query expansion, the most predictable type of service chain generation. Furthermore, the chain pattern `Frequencies Baseform Frequencies` implemented a linguistic frequency analysis for the comparison of the number of occurrences between any inflected word and its corresponding baseform.

To conclude the service chain discovery, an automatic analysis for such a detection seems possible and useful. By adding the `Similarity` service to the top-six requested services (see rank 10 in table II), the analysis also revealed service chains for research purposes only. For instance,

[24]Link: https://goo.gl/oW2LgT (Accessed: 7 October 2016).

while the `Thesaurus` service returned synonyms, the `Similarity` service delivered automatically-computed words that tend to occur in the same context and therefore have similar vectors of co-occurrences. The user(s) researched the difference between qualitative services with a high precision but a low recall and quantitative services with a lower precision but a high recall (see also the introduction to section VI). However, chains such as `Baseform Synonyms Sentences Baseform Synonyms Sentences` were more critical as they doubled one of the core chains from table VI. This discovery can be explained with the following example:

```
If I had had enough flour, I would have made more
brownies.
```

On the client-side, some users implemented a very pragmatic approach of requesting every occurring word in a sentence. In the example above, `had` occurs twice consecutively, causing the doubling of a chain with originally three services only. Although absent in table VI, requests of this kind were discovered in 24 more variants. For this reason, it is not recommended to automatically detect and install aggregated services, as this will contaminate the service registry with unwanted noise. However, chain discovery helps to provide candidates to be evaluated by human judgment.

C. Load balancing

One result of section VI-B is the observation that users tended to be pragmatic, which sometimes led to unwanted noise during the knowledge discovery process. An analysis of user behaviour is therefore justified. The goal is to investigate if users were using such an infrastructure more in the role of a "pragmatic user", who requests everything and often redundantly, or if they acted more as "smart users", who build a word list of unique words (word types) and request those instead of querying every running wordform (token). The reason for such an analysis is to provide an understanding of load-balancing for an infrastructure of linguistic resources based on LLS experience.

Figure 3 plots the power law distribution of the requested words with the rank r against their frequency f. As the figure shows, the query distribution does not follow the traditional Zipf law because both the range of the most frequent words and the characteristics of the long tail are not recognisable. Statistically, Zipf's law implies that a few hundred of the most frequently used words in any language roughly 500 in German already cover 50% of all tokens. Analysing the distribution of queries in figure 3 reveals that the 500 most frequently queried words cover just about 15.14% of the overall number of requests. Even the top-1000 most frequent words cover only 19.40%, implying that at least some users systematically removed function words from their analysis before using such a cyberinfrastructure. This observation is supported by a word class analysis for the four different part-of speech tags `N` (noun), `A` (adjective), `V` (verb) and `S` (function word). This analysis shows that the number of requests for all four word classes ranges between 23.84% and 25.19%, while the expectation is that roughly (and sometimes even more than) 50% of all tokens are function words. Furthermore, Zipf's law also implies that about 50% of all types (i.e. unique words) occur only once following the equation $p_f = \frac{1}{f \times (f+1)}$ with f as the word frequency. The data behind figure 3 shows that only 443,031 word types out of a complete list of 42,188,115 unique words occur only once, amounting to only 1.05% and thus producing significantly different results than the expected 50%.

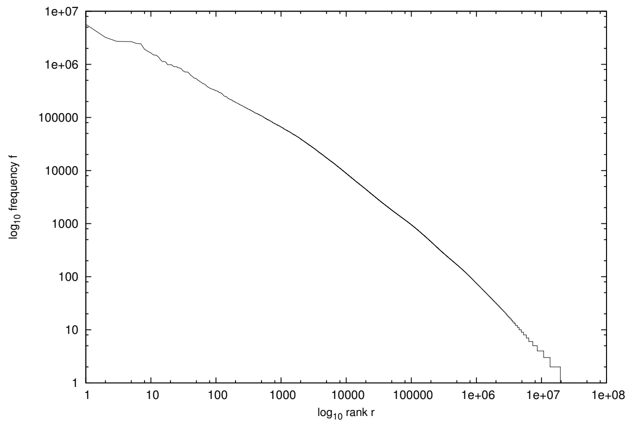

Figure 3. Log-log-scaled rank frequency plot for the LLS queries

To bring this query analysis for load-balancing to a conclusion, it was discovered that a global caching of function words reduced the expensive database queries by less than 20% only, since users partly removed function words for their analysis. This client-side preprocessing was appropriate seeing as a token-type-ratio (ttr) of of $ttr \approx 5$ for small texts and of $ttr \geq 20$ for bigger texts or text collections of could be computed. This implies that requesting an infrastructure such as LLS not for every token but for unique types reduces the number of requests for short documents to about 20% and for bigger documents or text collections to less than 5%. Nevertheless, "pragmatic users" can be identified. For this reason, based on our experience and log-file analyses, as a load-balancing strategy we recommend to track new users (e.g. by uniquely identifiable accounts) and to add an artificial delay to the natural response time so that those users are "trained" to request not every token but unique word types. This "punishment rule" does not only train users to use an infrastructure efficiently but also leads to the to the gradual disappearance of unwanted noise during the discovery of service chains. Besides, it helps to reduce temporal peaks when pragmatic users take over lots of server resources. This means that potential resource bottlenecks can be avoided without restricting content for the users.

VII. Lessons learnt

From a Natural Language Processing standpoint, the poor coverage of requests for some webservices (see table II) would need special attention beyond the above discussion on MWUs and users' incorrect language selection. The data shows that some requests resulted in an empty result-set `(0,0)` as the requested topic was either not covered or just barely covered in the relevant text corpus [11]. It also shows, for example, that pornographic queries were more popular at night (CEST). For instance, the query "penis length" was made to the `Synonym` service. Similarly, we detected 14,587 requests for the pattern `Penis*` with 775 unique queries. As the LLS were created by a publicly-funded university, topics like pornography were deleted as soon as identified by the web-crawlers, and are therefore unavailable even if requested. We can also observe that the text corpora are not representative of fashion, gymnastics, linguistics, and business language. While an analysis of such under-representation is beyond the scope of this article, these log-files can provide valuable support for corpus-building and corpus representativeness (see also [12], [13]).

From a technical viewpoint, the relation between the parameters and the results needs a deeper discussion. The idea behind SOAP- and REST-based webservices is to enable one computer to work with another regardless of the their operating systems and location. The underlying exchange format of both SOAP and REST is based on XML, which is, *per se*, interchangeable. Given that the services have different parameters (see the "Input Fields" column in table II), a static signature such as

```
public String[][] execute(String word)
```

is not possible. Even with the same source code underneath, the different signatures of the generated methods would need different implementations. For the submission of all key-value pairs needed to parameterise the webservice, we chose a `JavaBean` to encapsulate the data-structure of a `Map`. In the other implementations, the description in the Webservice Description Language file (WSDL) was interpreted differently. In some instances, the generated client-side data-structure was a sorted map, a tree map or a hash-based map. As the order of the key-value-pair did not matter, a change of order was not taken into account. The more critical interoperability issue was the serialisation and deserialisation of complex results. We found that the best data-structure is that of the `String[][]` data-type, because it is generic enough to deliver back any result independently from what is sent to the clients - whether it is just one value, one column or multi-columns in a matrix.

The problem with `String[][]`, however, is that it is not appropriately supported by webservice frameworks because these depend on the data-types defined in the XML Schema Definition (XSD)[25], which in turn only allow primitive data-types such as `xsd:string`, `xsd:time`, `xsd:double`, `xsd:decimal`, or `xsd:boolean`.

Arrays of primitive and complex data-types are also allowed in the definition of XSD. In order to better represent the semantics of a `String[][]` it was necessary to form an array of `String[]`, where the `String[]` represents a row and the encapsulating array stores the sum of the rows. Even if the representation of the semantics of the desired data-structure was not technically challenging, it led to different data-types in the source code.

While Java retained the semantics of an array of `String[]`, other implementations interpreted the WSDL-file with a `String[][]`. Alternatively, as was the case with `C#`, a so-called `JaggedArray` of the XML-based webservice description was created. Even if the generated source code eventually worked with different interpretations in different programming languages, it complicated the support of the LLS infrastructure as it was not possible to write up one document for the deserialisation of, and access to, the responded data.

In 2010, when webservice frameworks other than Apache `Axis 1.3`, namely Apache `2` and Metro, appeared on the market, we supervised a diploma thesis about benchmarking the performance of these frameworks [14]. There, an immediate result of a stress-test revealed that the Apache `Axis 2` and `Metro` frameworks can process 382% and 441% more requests and responses than Apache `Axis 1.3`. For this reason it was possible to measure a maximum throughput capacity of up to 280 requests per second as opposed to the maximum 60 requests per second in the current LLS infrastructure. Furthermore, [14] reports that REST-based webservices are around 228% faster at serialising and deserialising data than SOAP-based webservices when the result-set contains 100 data-sets. For result-sets of ca. 2500 entries, SOAP- and REST-based webservices have similar speeds. And when the result-set contains more than 2500 entries, REST performs better than SOAP.

Comparing REST-based webservices for inline- versus standoff-markup revealed that for a result-set between 100 and 10,000 entries standoff-markup performed 192% to 216% better than inline markup, while supporting stream-based webservices even better. With a responding result-set of more than 10,000 data-sets, standoff-markup performed gradually worse than inline markup. With a result-size of ca. 130,000 data-sets, standoff- and inline-markup performed equally well. Beyond this size, inline markup performed much better than standoff-markup[26].

[25]Link: https://goo.gl/xkUBDZ (Accessed: 7 October 2016).

[26]A response at a scale of 130,000 data-sets is somewhat unrealistic and ought to be avoided seeing as it is like dumping a database. Nevertheless, we wish to report this result because it was performed as an artificial test scenario in the lab under optimised circumstances (e.g. near-zero network latency and exclusion of other users).

VIII. Conclusion

Language-oriented resources have matured as a technology over the past few decades and are now gaining momentum in a wide range of areas within information- and knowledge-intensive applications. This trend is becoming more important with the increasing demand for automatic analysis of growing digital content. This article describes how the Leipzig Linguistic Services (LLS) addressed this demand and analyses the one billion log-files recorded by the project between 2006 and 2014. The LLS provided users with a means of querying the texts contained in the Leipzig Corpus Collection (LCC). The log-files yield statistics for languages, texts and words accessed, and in this article we supply details about the users' country of provenance, we elaborate on the web-services offered by the Leipzig-triad and on their usage.

In LLS' case, our opening question "What are the implications of Digital Transformation?" can be emphatically answered with the infrastructure mantra, "If you build it, they will come" (see also [5]). However, with regard to easy-to-integrate and atomic micro-services we found that users were generally very pragmatic and requested everything they had found in texts or on web-pages, such as RGB colour-sets, URLs and other meta-information. Based on the log-files, our inference is that it is easier to request a token and look for a match in LLC's database of millions of words than to invest time in conventional preprocessing and preselection on the client-side. Similarly, users repeatedly requested function words, sometimes only a few minutes apart. This user behaviour entailed a significant load and user control over the requests, as that type of recurring request on unchanged data could only be flagged as spam.

Moreover, we found that providing such an infrastructure over one decade challenges the compatibility of used software components. This article summarises experiences with interoperability issues in different programming languages.

From a Natural Language Processing standpoint, this paper contributes to existing conversations about the difficulty of building balanced and representative corpora. In fact, user interests detected in the LLS log-files can help to enrich corpora by adding further topics. This article also touches upon discussions about qualitative and manually-curated data versus automatically-computed and quantitatively-available results of language technology algorithms. Notwithstanding the improvement of Natural Language Processing algorithms, our results show that users prefer qualitative data and that they often request these services even if the domain and concept coverage is relatively low. The conclusion we draw from the user behaviour observed in almost one billion requests is that research fields, including the Digital Humanities, should share their data no matter how small through large infrastructure initiatives like DARIAH and CLARIN in order to increase the textual coverage of linguistic resources.

Acknowledgments

This work has been made available by the projects eAQUA (No. 01UA0803B), eTRACES (No. 01UA1101A) and CLARIN-D (No. 01UG1120c), and by the early career research group eTRAP (No. 01UG1409, 01UG1509) of the German Ministry of Education and Research.

References

[1] S. Borchardt, *Generierbarkeit einer XML Topic Map aus E-Mails unter Verwendung von Text-Mining-Methoden und Nutzung von Web Services* Bachelor thesis, 2005.

[2] O. Lau, *Wortschatz - Besserer OpenOffice-Thesaurus dank Web-Services*, c't 15/2005, vol. 15, pp. 214-219, 2005.

[3] M. Büchler, G. Heyer, S. Gründer, *Bringing Modern Text Mining Approaches to Two Thousand Years Old Ancient Texts* e-Humanities —an emerging discipline: Workshop in the 4th IEEE International Conference on e-Science, 2008.

[4] N. Lossau, *An Overview of Research Infrastructures in Europe - and Recommendations to LIBER*, LIBER Quarterly, vol. 21, no. 3-4, pp. 313-29, 2012.

[5] J. Van Zundert, *If You Build It, Will We Come? Large Scale Digital Infrastructures as a Dead End for Digital Humanities*, Historical Social Research / Historische Sozialforschung, vol. 37, no. 3, pp. 165-86, 2012.

[6] U. Quasthoff, M. Richter and C. Biemann, *Corpus Portal for Search in Monolingual Corpora* Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC), 2006.

[7] D. Goldhahn, T. Eckart and U. Quasthoff, *Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages* Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC), 2012.

[8] Z. Harris, *Distributional structure*, Word, 10, 2-3, pp. 146162, 1954.

[9] S. Teresniak, *Statistikbasierte Sprachenidentifikation auf Satzbasis* Bachelor thesis, 2005.

[10] T. Eckart, U. Quasthoff and D. Goldhahn, *Language Statistics-Based Quality Assurance for Large Corpora*, Proceedings of Asia Pacific Corpus Linguistics Conference, 2012.

[11] M. Büchler, G. Heyer, *Leipzig Linguistic Services - A 4 Years Summary of Linguistic Web Services*, TMS - Text Mining Services in conjuction with SABRE multi-conference, 2009.

[12] D. Biber, *Representativeness in Corpus Design* Literary and Linguistic Computing, 8, 4, pp. 243-257, 1993.

[13] S. Atkins, J. Clear, N. Ostler, *Corpus Design Criteria*, Literary and Linguistic Computing, 7, 1, pp. 1-16, 1992.

[14] S. Sander, *Performanceanalyse von SOAP- und REST-basierten Services in einer Linguistic Resources Umgebung*, Diploma thesis, University of Leipzig, 2010.

[15] IETF, *WHOIS Protocol Specification* Request for Comments 3912, 2004.