Technological Design of a State-of-the-art Digital Dictionary

Bojan Klemenc, Marko Robnik-Šikonja, Luka Fürst, Ciril Bohak and Simon Krek

Abstract
An important building block of a state-of-the-art digital Slovene language dictionary is its technological framework, which is briefly presented in this paper. We view the dictionary as a multi-tier architecture with a presentation tier, a middle application tier (a back-end application system with a component for semi-automatic data extraction), and a data tier. In their natural form, the language data are multidimensional. In a printed dictionary, there is just the presentation tier, and thus many relations contained in the underlying data are difficult to access or are even lost. By contrast, in electronic dictionaries there are no such restrictions. We can preserve the data in all its complexity and present it in various ways, since there is a distinction between the data and their presentation. This separation is the key factor in integrating various data sources (different corpora and external databases) into a unified database. Various users or programs can query different parts of the database based on their interests, and the presentation tier displays or returns the data at different levels of granularity. For each tier, we present the structure and review some of the technological considerations, which guarantee good extensibility, reliability, and adaptability of the final solution.

Keywords: digital dictionary, multi-tier software architecture, presentation layer, relational database, data extraction

1 INTRODUCTION

To create a modern digital dictionary of Slovene, technological considerations are no less important than lexicographical ones. This paper thus focuses on the technological aspects of such a dictionary. In particular, we first describe the core components of a modern digital dictionary, and then outline some ideas for its implementation. When designing a digital dictionary it is now crucial to consider the issues of sustainability, scalability, adaptability, and reliability.

Early implementations of digital (or rather digitized) dictionaries were, from a data-modelling perspective, a more or less direct mapping of the existing paper-based dictionaries to the digital form (cf. Urdang 1984; Boguraev and Briscoe 1989; Hajnšek-Holz 1993; Krek 2014b). Specifically, dictionary entries, together with their hierarchical organization and tags, were stored in formats such as XML (*eXtensible Markup Language*) files or, in case of web dictionaries, HTML (*Hyper-Text Markup Language*) files. In the latter case, the logical structure of a dictionary entry is intertwined with its presentation (appearance). By contrast, an XML dictionary entry specifies only the structure of the entry, whereas its presentation is generated using template-based transformations. Such templates may be defined by, for example, the CSS (*Cascading Style Sheets*) markup language. In the case of XML, we thus have a basic separation between the data and their presentation. The text of a dictionary entry may also contain references to other dictionary entries or their components.

The search queries supported by digitized versions of paper-based dictionaries are typically limited to headwords, a restricted set of elements (usually those specified in XML), and general text search. Search results are always presented in the same way: a dictionary entry (or perhaps several entries) that match(es) the query, possibly with highlighted portions of the matching text. Unfortunately, it is impossible to obtain a query-specific presentation of search results, since the organization of the dictionary data supports only a fixed number of predefined search result views. Such an organization of the dictionary data (and entries) is natural when dealing with a medium such as paper, where the data have to be organized and stored in their final, permanent form. However, dictionaries designed for digital media do not suffer from this physical limitation. Therefore, in designing a digital dictionary, we have to think beyond paper limitations and beyond static data structures, as the data have to be stored in their natural multidimensional form. Based on the desired queries, the data then have to be suitably filtered, rearranged, and presented.

It is therefore crucial to separate the presentation of the data from the data themselves when producing a digital dictionary. In this manner, the data can be stored in their entire complexity and presented from different viewpoints and at different levels of granularity. Technologically, it is thus important to separate the implementation of a digital dictionary into the *presentation tier* (or *front end*) and the *data tier* (or *back end*). The user does not have direct access to the data tier; he or she interacts with the data only through the presentation tier. The presentation tier presents the dictionary data to the user, intercepts the user's queries in the broad sense of the word (mouse clicks, search queries, etc.), and visualizes the results of the queries. The third component is the so-called *application tier* (or *intermediate tier*), whose role is to connect the data and presentation tiers. In particular, the application tier converts queries at the presentation tier into a form that can be used to retrieve the corresponding data from the data tier. The application tier then filters and reorganizes the retrieved data and forwards them to the presentation tier.

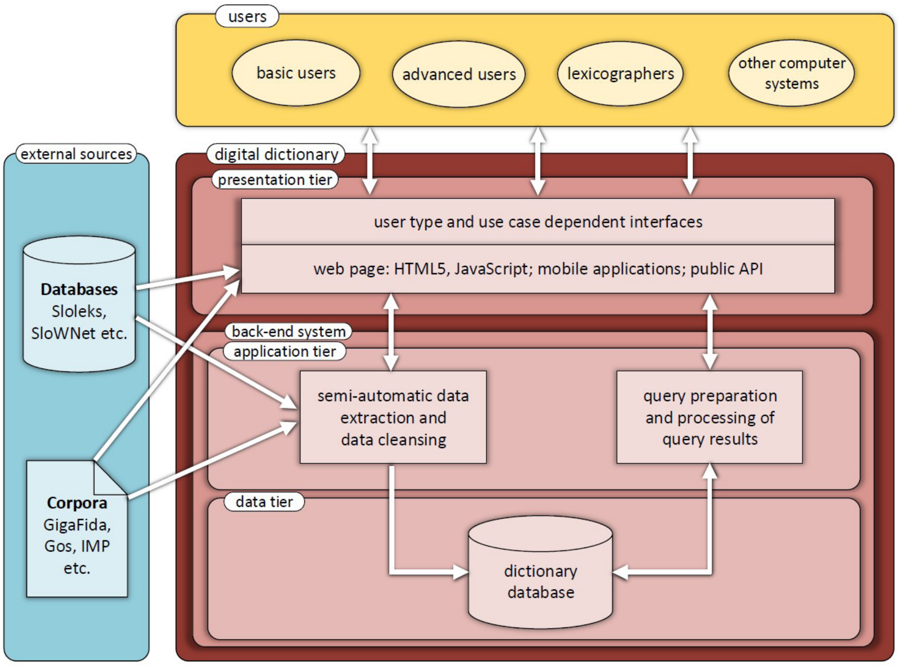

Figure 1: Architecturally, a digital dictionary is divided into three tiers: the presentation tier, the intermediate application tier, and the data tier. The user interacts solely with the presentation tier (through the webpage or mobile applications), which presents suitably selected and processed data from the data tier. The role of the intermediate tier is to connect the presentation tier and the data tier and to make it possible to fill the dictionary database from external sources.

We thus obtain a three-tier architecture (Figure 1), in which the user is only able to interact with the upper (presentation) tier, whereas the intermediate (application) tier and lower (data) tiers are invisible. Incidentally, the application and data tiers may be collectively called the *back end*. The fact that the architecture of the system is divided into multiple tiers makes it possible for individual parts to be relatively independent of each other, and higher tiers interact with lower ones through predefined programming interfaces. Consequently, a given tier can be replaced with another without having any negative effect on the other tiers. In addition, the separation of the presentation tier from the database makes it possible to integrate various dictionaries and sources. The idea is to have a single unified database and multiple "views" at the presentation tier, which can visualize different subsets of the database, e.g., written language, spoken language, modern language, archaic language, regional varieties, different combinations of criteria, and so on. At the presentation tier, we might also present different user interfaces to different types of users. For example, a high school student who uses the dictionary to write an essay might want to interact with a completely different interface (with different data and a different hierarchical structure of the data) than a linguist or a lexicographer. Although all users access the same database, there may thus be substantial differences in the level of granularity of the presented data and in the possibility of reading, writing, or modifying them. For instance, a lexicographer is allowed to modify the dictionary data, while other users are not.

The dictionary database may be updated both by the manual work of a lexicographer or by crowdsourcing (cf. Kosem et al. 2013a; 2013b). In addition, the system enables automatic extraction of the dictionary data from external sources, such as corpora. Data extraction is a repetitive rather than one-time process, since the language and hence the corpora constantly change. Therefore, in addition to serving as a connection between the presentation tier and data tier, the intermediate application tier also has to connect to external sources and make the initial data extraction process possible.

Technologically, the dictionary can be divided into four main components, which we briefly describe below:

1. **The database**, being the most important component of the data tier, is implemented as a unified relational database. Its role is to store the language data and the information extracted from the corpora.
2. **The back-end application system** (the intermediate application tier) integrates the entire solution and contains programming interfaces for interacting with the presentation modules (the web application and mobile applications) and programming code for interacting with the database.
3. **The automatic data extraction component** is, in fact, part of the intermediate application tier. However, because of its complexity, we will deal with it separately. Its role is to fill and update the database with

the data extracted from external corpora and databases. As part of the lexicographical process, the automatic extraction of data is presented in Gantar et al. (2015a).

4. **The presentation tier**, both in the form of a web portal and in that of applications for different mobile platforms (e.g., Android, Apple iOS, and Windows Phone), presents the lexicographical data to different types of users, makes it possible to search and browse the data, and facilitates data corrections and updates as part of the lexicographical process (ibid.). The presentation tier is not used only by people, and thus it also includes a programming interface through which other computer systems can interact with the dictionary.

It makes sense for the implementation of the dictionary to be based on open-source solutions to the greatest extent possible. This is because such solutions are now sufficiently powerful to support advanced operations and a high number of users. The division of the system into tiers enables us to select the most appropriate technology for each and then replace individual tiers if the need arises. The same principle holds for individual components. For example, the component for the automatic extraction of data is separated from other components at the application tier; if necessary, it communicates with them via programming interfaces.

The communication between individual tiers is based on the client-server paradigm. The client sends a request to the server, and the server replies with the appropriate response. This approach makes it possible for clients within the dictionary system to have comparatively modest demands for memory and processing power, since the data are mostly stored and processed on the server, whereas the client (at the presentation tier) merely displays the results of the user's query. Lower computational demands imply a lower energy consumption, which in turn enables the use of the dictionary on less powerful mobile devices, provided that they have a data connection to the server. Since the data in the database are regularly updated, the users always have access to the most up-to-date version. Such an architectural solution does not imply that the clients and servers have to be strictly separated; however, if they are installed on the same physical device, the database or a part thereof is replicated, and so we have to ensure that the individual copies of the database are synchronized (typically with one of the canonical copies of the database). To illustrate the usefulness of such a solution, let us note that (even on mobile devices) the dictionary can be used without an Internet connection.

The multi-tier and modular structure enables us to build, evaluate, and test individual components of the dictionary in parallel. However, the necessary prerequisite for such an approach is that the connections between the individual tiers, such as programming interfaces, are well defined in advance.

2 THE DATA MODEL AND THE DATABASE

A unified database and a separate presentation tier make it possible to integrate dictionaries and sources that were previously isolated. To build a suitable unified database, we first have to define an appropriate data model that will be able to store integrated data from various existing and newly-formed databases. Besides this, the data model has to support a broader set of queries, and has to cover those that were being executed on the existing databases, and enable additional queries on the integrated data. We also have to pay attention to the fact that the integration increases the quantity of the stored data that (still) has to be quickly accessible.

Table 1 shows the data sources, their inclusion into the unified database, and the existing format of individual lexicographical data that will be displayed in the user interface. The data can either be included directly in the database (YES in Table 1) or be accessible via a link to some external source, such as corpora (NO in Table 1). For a more detailed discussion on integrated dictionary sources and corpora, see Krek et al. (2013).

Table 1: Types of displayed data, their sources, their inclusion into the database, and their current format. The labels of formats are as follows: TEI (Text Encoding Initiative), LMF (Lexical Markup Framework), and LBS (Leksikalna baza za slovenščino – Slovene Lexical Database).

Displayed data	Source of the data	Inclusion into the database	Current format
phrases	extracted data	YES, as the lexicon	XML LBS
collocations - concordances	Gigafida (Slovene language corpus)	NO, a reference to the concordancer	-
parts of speech	Sloleks (Slovene morphological lexicon)	YES, as the lexicon	XML LMF
synonyms and translations into selected foreign languages	SloWNet (Slovene semantic lexicon)	YES, as the lexicon	XML DEBDIC
history, words	IMP (Corpus of the older Slovene language)	YES, as the lexicon	XML TEI
history - concordances	IMP (Corpus of the older Slovene language)	NO, a reference to the concordancer	-

Displayed data	Source of the data	Inclusion into the database	Current format
speech, words	Gos (Corpus of the spoken Slovene language)	YES, as the lexicon	(XML TEI - implementation in the project)
speech - concordances	Gos (Corpus of the spoken Slovene language)	NO, a reference to the concordancer	-
visualization of relationships	extracted data	YES	XML LBS
multimedia	WikiMedia, ...	YES, also as external sources	different multimedia formats
lexicographical statistics	Gigafida (Slovene language corpus)	YES	-

Sources in the textual form are usually stored in XML or plain text files. In addition to the contents, the XML files also store the structural data. Since different types of data have different structures (this is in part due to the type of contents they represent), it does not make sense to keep the XML structure in the database. (However, there are some exceptions where it is reasonable to keep smaller XML parts, such as emphases in descriptions.) Being a hierarchical form of data storage by its nature, XML is not very suitable for storing non-hierarchical data, such as dictionary data. However, owing to its hierarchical layout, XML is appropriate for serialization, and an XML file itself contains the data about the structure of the underlying data. For these two reasons, XML can be used for data interchange. (In the case of the dictionary, the data is interchanged with external sources and with external applications that interact with the dictionary through programming interfaces.)

It is important to consider relationships between individual records when organizing the data in the dictionary database. These relationships can be modelled by graph or relational databases. In terms of performance (Vicknair et al. 2010), both types of databases are able to handle large quantities of data that are typically associated with a dictionary. Several query languages have been defined for both graph and relational databases. For example, there are SPARQL (SPARQL Query Language for RDF) and several non-standard solutions (Wood 2012; Haase et al. 2004) for graph databases, and SQL and SQL/PSM for relational ones. Graph databases are highly flexible, since they do not have an explicitly defined structure, and are thus suitable for data with a variable structure. On the other hand, relational databases have an explicitly defined structure, which compels us to define the data model in advance. Besides that, we also have to consider which database queries are possible and which are not. Nevertheless, even the relational data model can be adapted in such a way that part of the structure is stored as data (Newman 2007).

Multimedia sources are stored as references in the database. To facilitate search queries, they are appropriately tagged.

Owing to the maturity of the corresponding technological solutions, the dictionary database is designed as a relational database. A simplified conceptual model of the database core is shown in Figure 2.

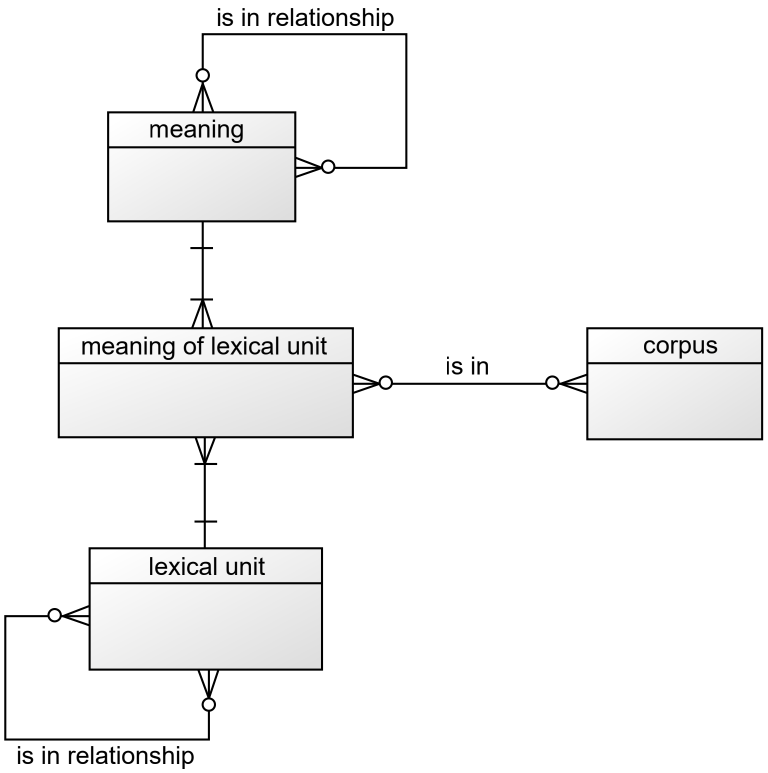

Figure 2: A simplified conceptual model of the database core, displayed in Martin's notation. This model serves as the starting point for the design of the entire database.

A lexical unit conveys a single meaning or several meanings, which can be in different relationships with one another. Lexical units can take the form of lexemes, phrases, phrasemes, or even parts of words, and can also be in different relationships with each other. For lexical units with a certain meaning, we store (aggregated) data about the sources in which they have been found.

The data model is designed in a sufficiently general way to enable the set of stored data to be extended to multiple language varieties and to treat these varieties equally. Moreover, when designing the data model we have to pay attention to the level of granularity of the data (a lower level of granularity means that we store a greater amount of aggregated data or lower-precision data, which in turn implies that it will not be possible to answer certain queries). Granularity is important both for data extraction and filling the database, since it determines what data have to be extracted and what extra amount of work will have to be carried out, e.g., in crowdsourcing or in the final processing performed by a lexicographer. For example, if, in the process of extracting data for lexical units, we do not record the time span during which individual lexical units occur, it will not be possible to restrict search queries to the lexicon from a given time span.

Several database management systems are available, and since relational databases are now well-established, there are a number of open-source solutions, although not all of these have the necessary functionalities. For our purposes, the database management system also has to support so-called recursive queries and SQL/PSM (procedures stored in the database). An example of such a system is PostgreSQL.[1]

3 THE BACK-END APPLICATION SYSTEM

The back-end application system serves as a link between the data and presentation tiers. Automatic data extraction is also part of the application system; however, owing to its complexity, we deal with it in a separate section. The role of the application system is to (re)format data requests received from the presentation level and to forward the requests to the database or external sources, such as corpora or external databases. Subsequently, the application system processes and filters the responses from the database or external sources and sends them back to the presentation level.

It is important to distinguish between the data themselves and additional restrictions and rules defined over the data, as these restrictions and rules can also change over time. For instance, collocations associated with individual lexical units can be recorded for a long time span (e.g., several centuries), but we might want to impose a rule to display only collocations occurring within, say, the last ten years. In this case, not only the data that match the rule but also the rule itself changes over time. The application tier has to make it possible to define such rules, and it has to formulate database queries based on the imposed rules and restrictions. This implies that we have to be restricted to time spans defined by

1 www.postgresql.org

the imposed rules and through the user interface, and we should be able to define the desired time span explicitly.

The application system provides its services in the form of a programming interface. An advantage of having separated tiers is that the source code of the application tier may be changed (completed, corrected, or improved) without affecting the programming interface, which means that the clients at the presentation level (the web and mobile applications) can still make use of the services without any modifications being needed. In addition to the clients at the presentation tier, the access to the programming interface has to be provided to other computer systems that would like to retrieve the data. We also have to enable connectivity in the sense of a semantic web (i.e., linked data).

Since the presentation and application tiers communicate according to the client-server paradigm, another important task of the application tier is to prepare the data in such a way that the clients receive only those data that they truly need, without any unnecessary data transfers.

4 AUTOMATIC DATA EXTRACTION

As shown in Table 1, the data in the dictionary database are extracted from different external sources. There are two main problems associated with data extraction: first, how to cope with the sheer quantity of the data in the external sources (for instance, the Gigafida corpus currently contains approximately 1.2 billion words), and second, how to ensure the quality of the extracted data. In addition, data extraction is not completed when the dictionary is published; rather, it is an ongoing process, since the language changes over time.

In the first stage, data are extracted automatically, and the results are then validated. Reliable data are written directly into the database, while those with a lower degree of reliability undergo a further filtering and manual processing stage.

To implement the automatic data extraction stage, we build upon the data extraction approaches developed for the purpose of creating the *Slovene Lexical Database* (*Leksikalna baza za slovenščino* in Slovene) within the project *Communication in Slovene* (*Sporazumevanje v slovenskem jeziku* in Slovene) (Gantar 2009; Gantar and Krek 2011), augmenting these approaches with more recent findings and technologically improved tools. For the entire lexicon that will be visualized, the following data can be automatically extracted: the headword in the base form (lemma), its part of speech, its frequency in the corpus, its grammatical relationships (which, in the database, are transformed into patterns), and the corresponding collocations together with their examples. As an important step in the process of automation, the so-called word sketch grammar within the Sketch Engine[2] tool has already been created. With the help of a designated software script that contains the descriptions of all relevant grammatical relationships for extracting collocations, we can retrieve a set of good candidates for usage examples of individual headwords within a realistic textual environment (Kosem et al. 2011). The software script makes use of the so-called GDEX (abbreviation for *good dictionary examples*) configuration, which defines the properties of such examples.

In the second stage of the data extraction process, the data are manually inspected before being included in the dictionary database. This work is carried out with the help of crowdsourcing, in the context of which the users label possible anomalies or errors in the data. Eventually, the data are formatted and confirmed by a lexicographer. The errors that have been confirmed to originate from the automatic extraction process are labelled and fed back to the data extraction system, which in turn learns from the errors using machine learning techniques, and thereby improves its performance.

Automatic data extraction belongs to the back-end system. Both the partially and completely processed data are written into the dictionary database. In the database, the data that have not yet been completely processed are appropriately tagged, which means that they may be either displayed or not displayed at the presentation tier. For example, both a lexicographer and general user access the same database, but the lexicographer will, besides interacting with a different user interface, also see the data that have not yet been completely processed and will be able to process them. The users participating in crowdsourcing have their own view of the data too. For the purpose of crowdsourcing, we can use existing platforms such as PyBossa,[3] which simplify creation of crowdsourcing applications (cf. Fišer et al. 2015).

5 THE PRESENTATION TIER: THE WEB PORTAL AND MOBILE APPLICATIONS

When designing the presentation tier, and consequently also the user interfaces for different applications, we have to focus primarily on user experience. The unified visual design of the applications is no less important. One of the goals of the presentation tier is to display the data on the web pages and popular mobile platforms in a consistent way.

2 http://www.sketchengine.co.uk/
3 http://pybossa.com/

When developing mobile applications it is advisable to take the so-called hybrid approach, which is the best way to port applications between different mobile platforms while ensuring the maximum reusability of individual components. A reasonable option to develop the basic functionality is to use the HTML5 and JavaScript technologies. The application core developed in this way can then be embedded into the application frameworks of the individual mobile platforms that have to be supported. Such a development is supported by numerous open-source tools, e.g., PhoneGap[4], which is based on the Apache Cordova[5] platform. The hybrid approach facilitates and accelerates the development of applications for all supported platforms. In addition, it ensures a unified presentation tier on all platforms and facilitates the upgrading of the applications. The core of a mobile application created in such a way may serve as a basis for developing a web portal.

For the purposes of achieving recognisability and a consistent user experience, it is advisable to design a unified visual identity for the entire user interface. It is important to follow the WCAG 2.0 (Web Content Accessibility Guidelines 2.0) standard and thereby ensure that the applications are also suitable for users with special needs.

6 CONCLUSION

In the technological design of a modern digital dictionary Slovene, a key concept is the separation of the presentation of the data from the data themselves. By following this route, the data can be stored in their entire complexity and presented from different viewpoints and at different levels of granularity. The dictionary is designed as a three-tier architecture, consisting of a presentation tier, intermediate application tier, and data tier. The task of the presentation tier is to retrieve the requested data from the data tier and display them to the user. Between the presentation and data tiers there is the intermediate application tier, which converts the user queries from the presentation tier into a form suitable for a direct execution in the data tier (on the database), and transforms the data retrieved from the data tier into the form required by the presentation tier. Another role of the intermediate application tier is the automated extraction of data from various corpora and external data sources. Since the language is constantly evolving, automated data extraction is an ongoing process that also involves lexicographers, who access the data through the suitable views at the presentation tier.

The separation between the data and their representation plays a key role in the integration of different sources (corpora and external data sources) into a unified

4 http://phonegap.com/
5 https://cordova.apache.org/

database. Different users, as well as external computer systems, may retrieve the desired data from the database using queries forwarded from the presentation tier. The presentation tier then also displays the retrieved data.

The main advantage of the multi-tier architecture is the independence of individual tiers, as long as the programming interfaces through which higher tiers interact with lower ones are appropriately defined. At each tier, we can therefore choose the most suitable implementation technologies, and a change at one tier does not affect others, as long as the programming interface remains intact.

We have followed the above-mentioned principles in our proposed implementation of a modern dictionary Slovene. In particular, we have divided its technological design into four components: a database (the data tier), a back-end application system with a component for partially automated data extraction (both belong to the intermediate application tier, but the data extraction component is treated separately because of its complexity and importance for the entire system), and a presentation component with the web portal and mobile applications (the presentation tier).

The technological design of the dictionary that we have described in this paper ensures that the solution to be built upon will serve as a central web-based language portal involving all levels of the Slovene language vocabulary. The key components, which enable the sustainable development of both the web portal and mobile applications, will be made available for further improvement under a free software license.