Axel Herold, Peter Meyer, and Frank Wiegand

4 Data Modelling

Fig. 4.1: Lego bricks.

A huge pile of Lego bricks: a great deal of material that makes it possible to construct numerous buildings. But what's the best way to start? Do you pick out individual bricks piece by piece in order to build a house? At the very latest, when you're struggling to find a red three for the third time, it might be worth thinking about whether you should have sorted the bricks first. But what's the best system to use? The reds in one box and the blues in another; the same for the yellows and whites? Or is it better to sort all the 2 × 1s, 2 × 2s, and 2 × 3s together, irrespective of colour? Whichever organising system you choose, after the bricks are sorted, you can "access" them in a more targeted way, i.e. building becomes no trouble at all.

Note: The German version of 2016 on which this chapter is based was written by Axel Herold, Lothar Lemnitzer, and Peter Meyer.

Axel Herold, Berlin-Brandenburgische Akademie der Wissenschaften, Jägerstraße 22–23, 10117 Berlin, Germany, e-mail: herold@bbaw.de
Peter Meyer, Leibniz-Institut für Deutsche Sprache, R5, 6–13, 68161 Mannheim, Germany, e-mail: meyer@ids-mannheim.de
Frank Wiegand, Berlin-Brandenburgische Akademie der Wissenschaften, Jägerstraße 22–23, 10117 Berlin, Germany, e-mail: wiegand@bbaw.de

Open Access. © 2024 the author(s), published by De Gruyter. This work is licensed under the Creative Commons Attribution 4.0 International License.
https://doi.org/10.1515/9783111233758-005

4.1 Introduction

Data modelling is also concerned with sorting and structuring, but of data rather than Lego bricks. In relation to lexicography, the task of data modelling is to structure the lexicographic content so that the computer can grasp it in a targeted way. Translated into the Lego example, it is possible to attach labels to the individually sorted Lego boxes that enable a machine to grab all of the red bricks or all of the blue ones in a targeted way, or – when sorted differently – to target only the 2 × 2 and 2 × 3 bricks. This programming is considerably easier than developing a machine that automatically distinguishes between the singles and doubles and can deliberately pick them out. It is exactly the same with lexicographic data: here we can also proceed more flexibly with the data when the content can be distinguished easily by a machine.

Just as different houses can be built out of the same Lego bricks, it is a normal requirement nowadays to present the same lexicographic content in different ways, e.g. in a print and an electronic dictionary. The prerequisite for this – and for so-called advanced searches in digital dictionaries, where users are able to enter complex combinations of search options (→ Chapter 5) – is appropriate data modelling.

In order to understand this process, we must first take a look at the different "levels" that have to be taken into account in the production of a print or electronic dictionary (→ Fig. 4.2 and → Chapter 3). The basis of any dictionary is, first and foremost, a lexicographic database. Various product-related excerpts can be derived from this database. A good example for this approach is the shared database called "Duden – Wissensnetz deutsche Sprache" (Alexa 2011) that serves as the basis for the content of various reference dictionaries on Standard High German where all of the dictionaries are collectively known under the brand name Duden. The "Wissensnetz" database includes the data contained in the dictionaries on German orthography, loanwords and synonyms, for example. To compile a dictionary, a specific excerpt is generated from this database targeted for a specific output, such as the 148,000 headwords in Duden 1 – Die deutsche Rechtschreibung (DDRS) with the associated information on spelling and grammar and brief explanations of meaning. Then, either a dictionary can be printed from this output-specific database or an app can be developed for smartphones and tablets or an Internet dictionary. The data modelling is done at the level of the database since all of the prerequisites for the following steps are created there. Returning to the Lego analogy, the individual bricks can be found at the database level. The finished outputs are located at the external presentational level, i.e. the houses in the Lego example or the individual dictionaries when applied to lexicography. The lexicographers work directly on the lexicographic database and the users interact with and read the outputs.

It may be worth stressing at this point that the underlying lexicographic database might not only include textual descriptions of linguistic phenomena. It might also store information such as images or even short video sequences that are used in the description of meaning within entries. This is most obviously the case for dedicated picture dic-

tionaries. Another common type of data stored in lexicographic databases are sound files that capture the pronunciation of headwords and possibly other parts of the entries such as related multi-word expressions or even citations. Thus, when we talk about lexicographical data throughout this chapter, we take it to mean data in a very broad sense.

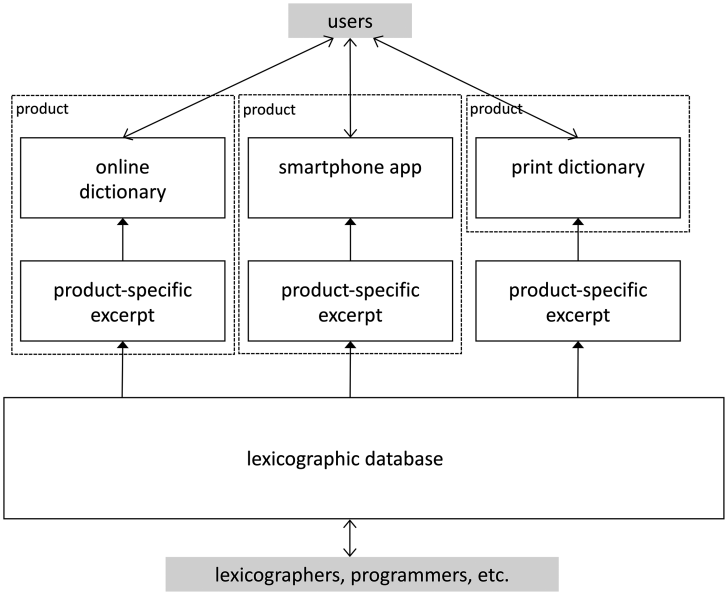

Fig. 4.2: Levels in the lexicographic process.

In order to be able to present the same lexicographic content in different ways, that content must be machine accessible in specific ways. The foundation for this is a suitable data model. Just like with the Lego bricks, an ordering principle has to be chosen: is the leading element the content of the information (e.g. whether it describes the word class or meaning of a headword), or only the part of the entry to which it belongs (the general information at the beginning of an entry or specifically a section on a particular meaning), or a different aspect altogether? It is important to have clear guidelines: as with the Lego bricks, choosing a new sorting system in the middle of the process is "expensive". For example, if we had already sorted half of the Lego bricks according to colour and then decided we wanted to sort them according to size, then all of the work done so far would have been in vain. It is no different when modelling data.

In this chapter, we will first discuss different data formats in which structured content can be formally represented, explaining their respective advantages and disadvantages, and how suitable query languages can be used to retrieve information from these data structures. The third section covers the core issues of data modelling – how to describe the structure of specific lexicographical content, e.g. which "boxes" the lexicographic content should be put in – both in abstract terms and with reference

to the data structures introduced before. put in, including the associated advantages and disadvantages. There are many lexicographic projects that face largely similar challenges. For this reason, initiatives have been launched oriented towards developing standardised solutions for modelling lexicographic data, similar to a set of guidelines for sorting Lego bricks. We report on these in the fourth section.

4.2 Data structures and representation formats

In this chapter, we pick up on concepts, explanations, and examples from → Chapter 1 on the technical foundations of Internet dictionaries. The starting point now is how the lexicographic information for a dictionary should be stored in a sensible way on a server. Let us revisit the HTML code already discussed in → Chapter 1 used to create a basic web page for a dictionary entry on the English lemma *disproof*, replicated here for convenience.

```
<html>
   <head>
     <meta charset="utf-8">
     <title>MyEnglishDict</title>
   </head>
   <body>
     <h1>disproof</h1>
     <p>[dɪs'pru:f] <i>n.</i></p>
     <ol>
       <li>facts that disprove something</li>
       <li>the act of disproving</li>
     </ol>
     <p><i>See also:</i> <a href="/entry/disprove">
        disprove</a></p>
   </body>
</html>
```

In our explanation of how web servers and web applications generally work in → Chapter 1, we deliberately left out the central issue of where exactly the web application obtains the HTML code for the relevant dictionary entry required by the user. One obvious and straightforward answer would be to simply integrate the code for the dictionary entry in the web application. A web application is a program that runs on the web server and responds to requests from the client. As such, the following instruction could, for example, be integrated into this program: “If the request reads ‘`GET /entry/disproof`’, then send the following HTML code back as the response:

(see code as shown above)". However, this is not a viable approach since any change in the appearance of the web page for an entry would require a change in the code of the computer program; this means that programming the web application and editing lexicographic content would become inextricably intertwined.

In order to separate the storage of data for lexicographic information from the programming and administration of the web application, an individual text file could, quite simply, be deposited for each dictionary entry and named according to the relevant lemma in a particular directory on the hard drive of the web server. Each text file would then contain the HTML code of the relevant web page. In this scenario, when the web application receives the request "`GET /entry/[LEMMA]`" (where `[LEMMA]` is a placeholder for the required lemma), it searches in the aforementioned data folder for a text file with the name `[LEMMA].txt`. If such a file is found, the program reads the content of the text file (which is the HTML code for the entry's web page) and sends it as an HTTP(S) response to the web browser making the request. The obvious advantage of this solution is that lexicographers can change or even delete the text files completely independently of the programmers and deposit new text files as the dictionary is extended; the web application itself remains unaffected by these changes and can simply continue to run since the program code contains no information at all about the content of the web pages.

In this approach – or, typically, a more sophisticated variant using a database, for example – the more "technical" aspects of compiling an Internet dictionary are somewhat decoupled from the more "content-related" aspects. However, the decoupling is neither as complete nor as far reaching as necessary: the lexicographic data that are made available to the program still consist of HTML code. Thus, these data are stored from the beginning in a specific format for Internet dictionaries and determine the appearance of individual dictionary entries. As such, lexicographers writing entries for an Internet dictionary like this find themselves at the level of data presentation from the beginning. Let us assume that the dictionary editors decide to make changes to the presentation of an entry at a later date:

- "Information about the word class must sit right next to the lemma and be written out in full, e.g. *noun* instead of *n*."
- "Information on pronunciation should appear between |vertical slashes| rather than between [square brackets]."
- . . .

In such a case, a simple change to some CSS code does not do the trick. All of the HTML pages have to be altered manually or with the help of suitable programming, even though the lexicographic content will not have changed at all. Therefore, what is also required is a separation between the *lexicographic data* proper and the properties of *data presentation*: the text files for individual dictionary entries on the web server should not contain any HTML code but rather just indicate the lexicographic information in a data format that makes sense for the lexicographic work and that is

abstracted as far as possible from the details of its final presentation. It is then the task of the web application to translate this data format into suitable HTML code. If the editors decide to make changes to the presentation of the entries at any point, only the program code for that process of translation has to be changed; the original data with the lexicographic information remain unchanged.

Yet what does a suitable format for representing lexicographic information look like? Answering this question brings us to the problems of data modelling and the data structures associated with it.

One obvious textual representation that is suited to machine processing is to separate the individual *types of information* from one another in a hierarchically structured form and to apply corresponding headings or "labels" to them. This can be done in many different ways, as in this sketch of a possible approach:

```
entry (id: MED.disproof):
  form:
    spelling: disproof
    pronunciation: dɪs'pru:f
  grammar:
    part-of-speech: noun
  senses:
    sense (numbering: 1):
      definition: facts that disprove something
    sense (numbering: 2):
      definition: the act of disproving
  cross-reference (refid: MED.disprove): disprove
```

As we can see, the content structure of the entry is represented here in blocks that are hierarchically nested inside one another and marked with "headings" and indents. Each block contains either a series of further subordinated blocks (the **senses** block contains two individual **sense** blocks and these each contain, in turn, a **definition** block) or simply text (the **pronunciation** block contains the text "dɪs'pru:f"). Some blocks have additional meta information that is recorded as what we will call *attributes* in parentheses after the block name, instead of in its own subordinate blocks. Thus, the whole block with the name **entry** is assigned the attribute **id**, here with a *value* that is supposed to specify a uniquely identifying ID character string for this particular entry; the block **cross-reference** contains the lemma of the referenced entry and has an attribute **refid**, whose value is the ID of the entry that is being referred to. The **sense** blocks feature an additional attribute **numbering** that indicates numbering labels such as "2.", "2.c", "(ii)", or similar. Such an attribute could be useful in the context of digitising a print dictionary. Note that the names of the blocks and attributes can, in principle, be chosen arbitrarily; the hierarchical structure could also have been designed in a different way. In our oversimplified example, it would

have been possible, for instance, to just have the blocks named **spelling**, **pronunciation**, **part of speech**, **definition**, and **cross-reference** as pieces of information directly subordinate to **entry**. Even the textual order of the data could have been different, one possible exception being the ordering of the **sense** blocks, which might reflect a lexicographical assessment of the frequency, importance, or relatedness of the senses. Finally, there is no logical necessity to introduce attributes as a separate syntactic device for marking meta information; it would have been sufficient to use ordinary blocks for that purpose. The particular way in which the data are structured in our toy example actually foreshadows standard ways of modelling and digitally encoding lexicographical content, which are to be discussed in what follows.

4.2.1 XML documents

Formally, hierarchical structures of this type can be described as trees and, accordingly, can be represented as tree diagrams, which do, in fact, look like a tree standing on its head. Such trees consist of individual positions (*nodes*) connected by "arrows" (*edges*) (→ Fig. 4.3).

The *root node* at the top of the "inverted" tree represents the entire structure, and the *child nodes* linked with it by edges represent the blocks of the highest structural level. From a formal and informatics perspective, trees are simple structures that are easy to describe and process. They have been used for a long time in metalexicography to systematically describe entry structures in print and digital dictionaries (Kunze/Lemnitzer 2007: 77–93; cf. also Wiegand 1989). Every node in the tree – each content block – has precisely one parent node – a block that contains it – with the exception of the root node. Accordingly, trees can be stored in a simple way in a computer in that each node is basically represented by referencing the storage addresses for its child nodes, with the exception of "childless" nodes or *leaves* at the bottom of the hierarchy, which are simply stored as text.

A very common way of encoding such tree structures in a textual form that is both human and machine readable is *XML* (Extensible Markup Language), which is strongly reminiscent of the HTML discussed in → Chapter 1 and which works "according" to the same basic principles. The individual content blocks are each enclosed in a start tag and a corresponding end tag and contain further "blocks", formally known in HTML and XML as *elements*, and/or plain text, that is, a sequence of characters, especially letters and numbers. XML is syntactically more rigid than HTML: for example, unlike HTML, elements always must have an end tag, even if they do not have any content at all. While the "vocabulary", i.e. the range of available element and attribute names, and the "grammar", i.e. the rules that govern where elements are allowed to occur in the document, are mostly predefined in HTML, in XML all these aspects can be defined individually for each concrete *application* – e.g. for encoding entries in a particular dictionary – in a so-called *schema*.

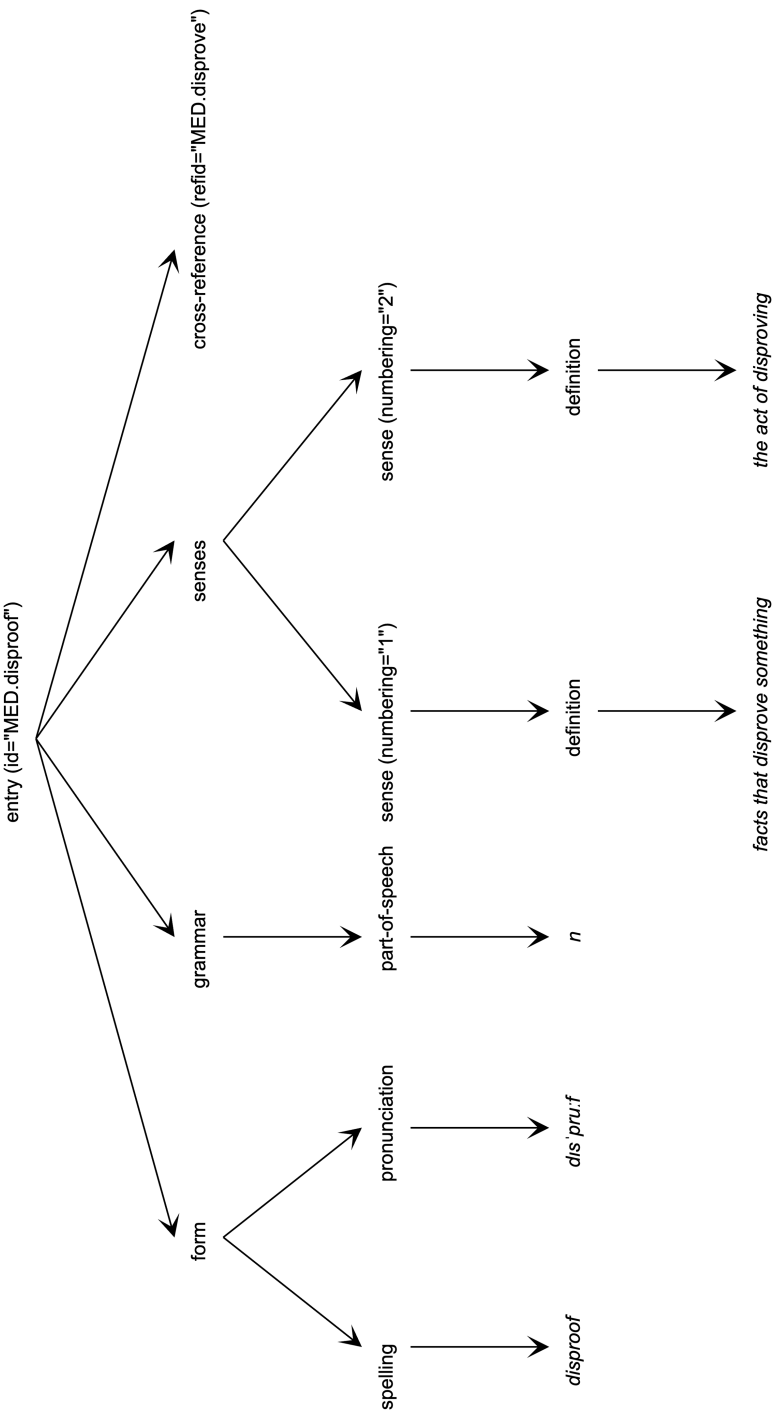

Fig. 4.3: Representation of the microstructure of an entry as a tree diagram.

Represented in XML, the toy entry could appear as:

```
<?xml version="1.0" encoding="UTF-8"?>
<entry id="MED.disproof">
  <form>
    <spelling>disproof</spelling>
    <pronunciation>dis'pru:f</pronunciation>
  </form>
  <grammar>
    <part-of-speech>n</part-of-speech>
   </grammar>
  <senses>
    <sense numbering="1">
      <definition>facts that disprove something
      </definition>
    </sense>
    <sense numbering="2">
      <definition>the act of disproving</definition>
    </sense>
  </senses>
  <cross-reference refid="MED.disprove">disprove</cross-
  reference>
</entry>
```

The first line is the *XML declaration*, which has a special syntax and specifies the character encoding used. (For more about this concept and the somewhat similar document type declaration in HTML, → Chapter 1) Just like with HTML, XML has the concept of attributes. Attributes are typically used to describe information that is in some sense descriptive of a specific element occurrence but is not regarded as part of its content. However, there are no clear-cut rules when to use attributes and when to use elements with text content. For example, the part of speech could also be coded as an attribute.

If the XML-based representation shown above is to be used for a web application, the application must contain program code capable of *parsing*, i.e. analysing the structure of this XML document and creating, in the simplest case, the HTML document shown at the beginning of → Section 4.2. This code can take advantage of the fact that the different types of lexicographic information in the XML document are each marked up semantically with their own tags. A widely used technology for generating other XML documents or HTML documents out of given XML documents is called an *XSL transformation*. As a side note, the program code of XSL transformations, often referred to as an XSLT *stylesheet*, is itself written using XML syntax. Using different stylesheets, an XML document can be "translated" into HTML pages in completely dif-

ferent ways, depending, for example, on the user's preference, or, alternatively, into other types of documents, such as a PDF file for printing. In the transformation process, any information contained in the XML document can be omitted or reorganised; in this way, the same XML document can be used to generate, for example, an overview presentation of the most important information as well as a complete, detailed view of a dictionary entry.

4.2.2 Relational databases

Another form of representation for lexicographic data, and one that has been around for much longer, are *relational databases*. By this, we mean a structured system of data tables, somewhat comparable to the tables in spreadsheet software. These data tables can be saved in an extremely efficient form on the hard drive of a server computer and be read, altered, and managed with great speed by a program known as a *database management system* (DBMS). Programs – e.g. web applications – that receive information from a relational database or wish to modify it "call" the database management system using *SQL*, a specialised query and data manipulation language. The database management system can run on the same computer as the web application or on another server, usually connected via a fast internal computer network. It is not possible here to go into the complex details of relational database technology; instead, by virtue of the miniature entry used as a toy example above, we shall demonstrate how dictionary entries can be described in a relational database. In order to keep the example simple, we initially assume that all of the entries in the dictionary include only one indication of pronunciation and, at most, one cross-reference to another entry. Only the number of word senses will vary. Then, the main table of dictionary entries might look as it does in → Table 4.1.

Table 4.1: Relational table ENTRYTABLE of dictionary entries.

ID	Spelling	Pronunciation	PartOfSpeech	CrossReference
. . .	. . .	. . .	. . .	. . .
MED.disproof	disproof	dɪs'pru:f	noun	MED.disprove
MED.disprove	disprove	dɪs'pru:v	verb	NULL
. . .	. . .	. . .	. . .	. . .

Since the number of word senses varies and, in theory, any number of meanings can belong to any one dictionary entry (the so-called *1:n-relation*), senses require their own table, which contains the sense definitions and the numbering label but also, crucially, the reference to the relevant entry. The references use the entry IDs, and the ID column acts as the so-called *key* to unambiguously identifying a table row (a specific

dataset or “record” as in → Table 4.2). The database management system can automatically guarantee the *referential integrity* of these references to datasets in other tables; that is, it prevents a dataset for an entry from being deleted in the entry table, if there is still a reference to the ID of this entry in the senses table.[1]

Table 4.2: Relational table SENSETABLE of word senses.

Entry	Numbering	Definition
. . .	. . .	. . .
MED.disproof	1	facts that disprove something
MED.disproof	2	the act of disproving
. . .	. . .	. . .

In a more realistic scenario, an entry can contain any number of cross-references to other entries. In that case, the “CrossReference” column is omitted from the main table, and a further table is needed for cross-references, as shown in → Table 4.3. Each row (record) in this table contains the ID of the source entry (which contains the reference) and the ID of the target entry (to which the reference refers). This is an m:n-relation: in theory, any number of target entries can belong to each source entry – each entry can refer to any number of others. At the same time, any entry can be referred to by any number of other entries. If the order in which multiple cross-references are to be presented in an entry matters lexicographically, it must be encoded as a separate column in the table because the rows of a relational table have no intrinsic ordering (→ Fn. 1).

Table 4.3: Relational table REFERENCETABLE of cross-references.

Source	Target	Position
. . .	. . .	. . .
MED.disproof	MED.disprove	1
MED.disproof	MED.proof	2
. . .	. . .	. . .

The web application can now make requests to the database management system in the aforementioned query language, SQL, in order to receive the lexicographic data

1 Note that the rows in a relational database table are not ordered in a technical sense (since, mathematically speaking, they are elements of a set), even if a specific order has to be chosen in a diagrammatic representation like the one in → Table 4.2. Thus, the order of word senses present in the XML document has no equivalent in the relational database although it might be recoverable from the **Numbering** column.

for the entry on *disproof* with the ID “MED.disproof”. The following SQL query returns all of the column values belonging to the row with the ID “MED.disproof” in the main entry table:

```
SELECT * FROM ENTRYTABLE WHERE ID="MED.disproof";
```

However, all of the corresponding rows in the other two tables must be retrieved as well:

```
SELECT * FROM SENSETABLE WHERE Entry="MED.disproof";
SELECT * FROM REFERENCETABLE WHERE Source="MED.disproof";
```

With the help of the data acquired in this way, the web application or web service can, in turn, construct its response to the client (e.g. an HTML page).

4.2.3 Other types of databases

While XML documents and relational databases continue to be the dominant representation forms for large lexical resources, other types of databases are being explored and used in different contexts. Conventionally, such non-relational databases are collectively referred to as NoSQL databases. Conceptually, one strain of development is focusing strongly on the notion of documents (resulting in a *document store*) and another strain is focusing on generalising the tree model underlying many traditional lexicographic databases to a graph model (resulting in a *graph database*).[2]

In a document store, entries are typically managed as individual documents by the database management system. For example, in an XML-based store, entries such as those described in → Section 4.2.1 are stored by the DBMS as separate and “individual” entities without mapping them explicitly onto a (complex) table structure. The indexing system of an XML database then allows for the direct retrieval of sets of documents or parts of documents using established query devices such as XPath (XML Path Language) or XQuery (XML Query Language).

XPath expressions allow the specification of the location of individual nodes within an XML document by specifying their properties in terms of their name (i.e. a *node test*), contextual constraints on the node (*predicates*), and an indication as to the *axes* that need to be followed when traversing the tree (e.g. following the edges vs.

2 There are other NoSQL approaches too, such as key-value stores, but we will not discuss them here because there has been little uptake so far in the domain of lexicography.

moving laterally across sibling nodes). Consider the following XPath expression and its application on the XML representation presented in → Section 4.2.1:

```
/child::entry/child::form/child::spelling
```

The forward slash separates individual steps of the path expression, the axis specification (`child::`) tells us to move along the line of descendants of the nodes (i.e. along the edges of the tree), and the node tests specify the names of the nodes to be expected along the path. There are no predicates in this example. When the expression is "processed" (evaluated), the result returned will be a set of all the nodes (*node set*) that are reached when traversing the tree as follows: start at the root node `<entry>`, from there proceed to a child node `<form>`, and from there to another child node `<spelling>`. Given the single document in the example, a node set containing the node `<spelling>disproof</spelling>` would be returned. As more and more entries are added to the database that are structured like the *disproof* entry, the resulting node set would grow accordingly, effectively providing a headword list derived from all of the spellings in all of the entries. The expression in our example can be abbreviated in two regards: syntactically and semantically. As the `child::` axis is considered the default, it can be omitted, resulting in the syntactically equivalent expression:

```
/entry/form/spelling
```

If we are only interested in creating a headword list from `<spelling>` nodes, it is not strictly necessary to specify that each such node needs to have a parent node called `<form>`. In this case we can change the axis that needs to be traversed from `child::` to the broader `descendant::`, resulting in the semantically equivalent expression:

```
/entry/descendant::spelling
```

or even:

```
/descendant::spelling
```

denoting all `<spelling>` nodes that can be reached when moving along the edges of the tree downwards from the root node.

As an example for the application of predicates, consider the following expression:

```
/entry[descendant::sense[@numbering]]
```

Here, the square brackets enclose predicates, i.e. constraints that the nodes in the path expression have to satisfy, with the `@` symbol denoting the name of an XML attri-

bute. Thus, the node set returned by this expression is the set of all `<entry>` nodes that have one or more descendant `<sense>` nodes, which, in turn, must meet the condition to carry an attribute called `numbering`. In this way, we can determine the set of all entries that describe polysemous words.

While XPath expressions allow for the selection of nodes that meet certain criteria so they can be retrieved and returned by the DBMS, they do not provide a means of modifying or storing data in the DBMS. For complex query, storage, and retrieval tasks, XML databases typically provide XQuery-based interfaces. XQuery uses XPath expressions to create node sets that can then be used in complex expressions. Consider the following example for such a query. It generates a fragment of HTML code consisting of an `<ol>` element that contains `<li>` child elements with all the entries' headwords, alphabetically sorted, as their text content. Note that the `collection('/db/dict')` part serves to illustrate a locator for the dictionary in the DBMS, which will often be stored as a collection of documents:

```
<ol>
   {
   for $headword in collection('/db/dict')/
   descendant::spelling
    order by $headword
    return <li>{ $headword} </li>
   }
</ol>
```

The evaluation of this XQuery expression goes beyond the application of XPath in two ways. First, it provides a template for HTML markups (`<ol>` for ordered lists and `<li>` for list items therein) that enables a direct rendering of the query result. "Second", much like with the SQL queries on relational databases, XQuery engines allow for further modifications of result sets, such as sorting (`order by $headword` in our example). In XQuery, users may define and invoke custom functions, and also the DBMS will provide its own interface via special functions so that the database can not only be queried but also modified, updated, and added to.

Let us conclude this section with a brief description of graph databases, which, at their core, rely on the mathematical concept of a graph. We will not go into the finer details of graph theory here but rather focus on the essence needed to understand its possible applications in the context of lexicography.

Generally, a graph consists of a set of nodes (called *vertices* in graph theory; we will use the term *node* here to underline the relation with the description of the trees above) and a set of edges, which are essentially pairs of nodes where the nodes are related to one another in a specific way. Graphs can be classified according to certain properties, such as whether the edges are directed or undirected, whether all nodes need to be connected or not, whether they may contain loops as opposed to only con-

taining a single path between any two given nodes, or whether the edges may carry additional information (such as weights), to name but a few.

Graph databases differ from each other in their restrictions on and assumptions about the features of nodes and edges. In the case of labelled-property graphs, nodes and/or edges may have explicitly specified, named (i.e. labelled) properties that can be used to store additional data directly without the need to model these properties as nodes and edges as well. As a consequence, nodes and edges in a labelled-property graph may have different "data types", i.e. sets of mandatory or optional properties. The opposite approach is adopted by the data model in the *Research Description Framework* (RDF). Here, the graph consists of a set of *triples* that each comprise two unlabelled nodes and an unlabelled edge connecting them. The edge (the *predicate*) always points from one node (the *subject*) to the other node (the *object*); thus the graph is directed. The triples constructed in this way can be considered statements about two *resources* for which the relation expressed by the predicate holds. The term *resource* is used very generically in RDF. In the domain of lexicography, a resource may be a single dictionary, an entry within a dictionary, or any constituent that entries are constructed from. To refer to resources, RDF relies on *uniform resource identifiers* (URIs) that unambiguously identify resources. While the subject and the predicate always need to be URIs, the object may be either a URI or a literal (i.e. a character string). Several notations are used for RDF triples, among them XML- and JSON-based serialisations as well as RDF specific formats such as N-Triples or Turtle. To provide a practical example, we use the easily readable N-Triple notation. The triple is given on a single line and terminated by a full stop:

```
<http://example.com/entry/disproof>
<http://example.com/has_headword> "disproof" .
```

This triple states that a dictionary entry referred to by its URI `http://example.com/entry/disproof` has a headword (`http://example.com/has_headword` – the relation is also referred to by its URI) that is given by the literal string "disproof". Statements regarding senses could be formalised accordingly:

```
<http://example.com/entry/disproof> <http://example.com/
means> <http://example.com/sense/disproof_facts_sense> .

<http://example.com/entry/disproof> <http://example.com/
means> <http://example.com/sense/disproof_act_sense> .
```

Note how the subject is identically referred to by its URI twice and how the object in each statement is also referred to by a URI this time. With a triple representation of our dictionary stored in a graph database (which in this case would be called a *triple store*), triples with subjects referred to by the URI `http://example.com/sense/disproof_facts_sense` would allow us to retrieve further information on the first sense of the

entry. To query an RDF triple store, SPARQL (the SPARQL Protocol and RDF Query Language) is used. The following SPARQL query retrieves the senses that are associated with the entry “disproof” (a line starting with `PREFIX` describes a prefix that is used to shorten the URIs):

```
PREFIX ex: <http://example.com/>
PREFIX entry: <http://example.com/entry/>
SELECT ?sense
WHERE
{
   entry:disproof ex:means ?sense .
}
```

For graph databases, many efficient algorithms have been described and implemented (cf. Robinson/Eifrem/Webber 2013), which makes it possible to quickly search for paths in graphs, that is, to locate routes from one node to another running along multiple edges. Especially in the context of Linked Open Data (LOD), graph databases have become hugely popular recently. The types of data considered in the LOD paradigm go far beyond lexicographic data. There is a strong focus on general knowledge bases such as Wikidata and DBpedia, two projects that automatically extract facts from WIKIPEDIA and convert them into *knowledge graphs*. Another common type of LOD resources are ontologies that model – often domain-specific – conceptual hierarchies. LOD resources form the basis for the *Semantic Web*, thus named to highlight its overarching goal, which is to provide the data and infrastructure needed to create semantic annotations for resources on the Internet. Early on, ideas were proposed to also include lexicographical resources (cf. Spohr 2012). Dictionaries that rely heavily on relations (such as the lexical-semantic wordnets discussed in → Section 4.4.2) are ideal for graph-based representations because of the close resemblance of their internal organisation and the modelling assumptions imposed by graph databases. Nevertheless, in principle, all lexical resources can be represented in graph databases.

4.3 Data modelling

4.3.1 Conceptual (semantic) data models

The discussion so far has shown how lexicographic information can be represented in very different data formats – textual or tabular – independently of presentational aspects, facilitating further machine processing and flexible presentation of the data. In the process, we also raised the problem that, when developing an Internet dictionary, it must first be decided in very general terms how the data will be structured that

need to be stored and processed. Particular questions that arise here are which types of lexicographic detail we need in our dictionary entries, which hierarchical relationships exist between them, which are obligatory, and which can occur more than once. As the example of cross-references between entries demonstrated above, these fundamental decisions about structure are necessary in the case of relational databases in order to determine the number and structure of data tables and their relationships to one another. But these decisions are also a prerequisite for determining which XML elements are needed for an XML-based dictionary and how they are to be nested inside each other; thus, it would make little sense to distinguish, as shown in the example in → Section 4.2.1, between a superordinate 'container' or 'wrapper' element <senses> and subordinate <sense> elements if there was a maximum of one meaning per entry.

If developing a dictionary involves specifying the required lexicographic indications and their relationships in an abstract way without already deciding, for example, on whether to use a relational database or XML, then we have entered the territory of *conceptual data modelling*. There are established and formalised diagrammatic formats for formulating conceptual data models, in particular the *entity-relationship model* and the *Unified Modelling Language* (UML). As an illustration, we shall present only a very simple example based on UML modelling applied to the toy dictionary entry discussed in → Section 4.2.

A large number of different types of diagrams are associated with UML. → Fig. 4.4 shows a *class diagram*. The rectangles represent *classes*, that is, types of *entities* that need to be modelled. This example sets out two types of entities, namely dictionary entries as a whole and word sense information within these entries. The names of *attributes* are located underneath the names of the classes, separated by a horizontal line. In UML, attributes are the properties that jointly characterise each entity (entry, word sense) of the relevant class. For actual entries, these properties in our example are an ID, the orthographic form of the lemma sign, its pronunciation, and its part of speech. Word senses have a definition and (assumed here for demonstration purposes) a numbering within the entry. In more detailed modelling, the *data type* of the individual attributes could also be given, for example, the pronunciation is a string of symbols of any length or the part-of-speech indication is one of several predetermined sets of symbols such as "n", "v", "prep", etc.

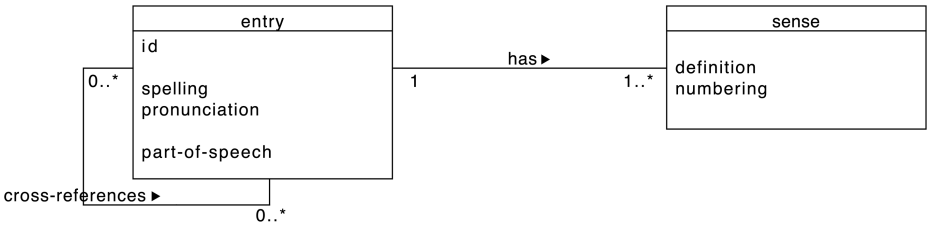

Fig. 4.4: Simple UML modelling of an example entry.

The relationships between the entities in classes are represented by associations, which are lines drawn between the relevant class rectangles to link them. The *multiplicities* of the association are given at each end of such a line. This is explained in the modelling requirements that the diagram above expresses (the asterisk symbol * designates in general terms an arbitrary non-negative integer number):

- A given entry has at least one but otherwise any number of senses (multiplicity 1..*).
- Conversely, each meaning only "belongs" to exactly one entry (multiplicity 1 or, in more detailed notation, 1..1). This is not a trivial point; one might wish to model relationships of synonymy explicitly in this way so that one and the same "meaning" can be assigned to multiple entries.[3]
- A given entry *cross-references* any number of other entries (multiplicity 0..*); since the entities that relate to one another are instances of the same class (entries), we speak of a *reflexive association*.
- Conversely, a given entry can be referred to by any number of other entries (multiplicity 0..*).

It is clear that the principles and terminology of conceptual data modelling sketched out here can generally be applied without alterations to describe the microstructure and mediostructure of print dictionaries. However, especially older print dictionaries tend not to have rigid, formalisable structures since they were conceived for human users rather than for machine searching and processing. As such, the fundamental difference between digital dictionaries and print dictionaries is not the manner in which information can be structured as such. Instead it is the necessity to actually store and prepare the data in some kind of rigorously structured way as well as the possibility of presenting this structured content in a flexible way and of making it possible for it to be searched accordingly. Here, the *granularity* of the data modelling can vary considerably. Especially in older dictionaries, there are often sections within entries that cannot be structured in a consistent way when they are retrospectively prepared in a digital form because of their narrative character; a typical example would be discursive explanations of etymology. In the most extreme case, this kind of section has to be modelled as an entity, the sole attribute of which is simply the whole text of the section as a non-structured series of characters that can only be accessed in a full text search. Freshly conceived digital dictionaries are the opposite extreme, since in this case it is possible to model the lexicographic data in a very granular way, that is, to store the individual types of information (indications) in a very fine-grained way, each as a different attribute of an entity. In the case of actively edited and maintained dictionaries, the modelling and overall lexicographic process must be flexible and ex-

3 In that case, the position numbering would have to be dealt with in a different way since the numbering assigned to a sense in one entry might differ from that in another entry that features the same word sense. The solution would basically be to encode the numbering as an attribute of the association itself, using what is called an association class.

tensible to guarantee that each entry may be revised at any time. This often makes it necessary for the conceptual modelling to adjust to new requirements that arise while the dictionary is already operational.

4.3.2 Logical data models

It is striking that the manner in which one entry is supposed to cross-reference another is not specified in the UML diagram in → Fig. 4.4. In the XML document shown in → Section 4.2.1, the cross-reference is achieved by providing the ID of the entry which is being referred to. However, implementing the actual "cross-referencing mechanism" assumes that the data(base) format is already known. On the conceptual modelling level, though, these issues are generally dealt with in abstract terms. The focus is essentially on content-related decisions such as the types and properties of entities that will be described and the types and properties of the relationships "between" these entities. Questions arising from the actual implementation of cross-referencing structures are addressed instead in the domain of *logical data modelling*, which involves "spelling out" the conceptual data model for a specific data format and the database system associated with it. The process of spelling out the data model is not a process that can be carried out mechanically since the conceptual and logical data models do not exist in a simple correspondence to one another. For example, the grouping of the **spelling** and **pronunciation** elements under the superordinate **form** element in the XML document in → Section 4.2.1 does not have any formal correspondence in the class diagram in → Fig. 4.4.

Logical data modelling with XML documents is again captured through suitable formal descriptions, so-called *schema languages*. There are several established formal schema languages for XML documents, including *DTD* (Document Type Definition), *XSD* (XML Schema Definition), and *RELAX NG* (REgular LAnguage for XML Next Generation). For illustrative purposes we show here a simple, almost self-explanatory RELAX-NG modelling applied to the toy example XML document in → Section 4.2.1:

```
element entry {
 attribute id {  text },
 element form {
  element spelling {  text },
  element pronunciation {  text }
 },
 element grammar {
   element part-of-speech {  string "n" | string "v" |
 string "adj" }
 },
 element senses {
```

```
  element sense {
   attribute numbering { text },
   element definition { text }
  }+
 },
 element cross-reference {
  attribute refid { text },
  text
 }*
}
```

The specified modelling determines exactly which elements are permitted to appear in a generic XML document for our fictional dictionary, with which attributes, in which position, and how many times. In the miniature modelling provided, the word class element **part-of-speech** can only contain one of the three labels “n”, “v”, or “adj”. In the example, the equivalents for the multiplicities from the conceptual modelling are the symbols “*” (corresponds to 0..* in UML; so “any number, including none”) and “+” (corresponds to 1..*, so “at least one”).

The various schema languages differ from one another in terms of their expressive power, i.e. they permit constraints to varying degrees and of varying complexity to be formulated. But their purpose is the same: to describe with precision the desired structure of a class of XML documents. Then a computer can check in a purely formal way whether or not a given XML document really matches this required structure. This process is called *validation*. The validity of XML documents is a fundamental prerequisite for any form of further machine processing of the documents. Thus, a program to translate any dictionary entry represented in XML into an HTML representation (e.g. an XSL transformation) can only be developed if it knows the structure of the XML documents and, therefore, where in these documents to find which indications.

Of course, there are also formal techniques for specifying the desired data structures for relational databases. A relational database schema determines which tables there are, which columns they have, which types of data can be entered into the different columns, which relationships exist between the tables, and which keys have to refer to a specific row in another table (→ Section 4.2.2). It is also possible to determine further restrictions in a database, so-called *constraints*, which prescribe, for example, the range of values allowed in a particular column or certain complex conditions for the permissibility of whole datasets (rows), the maintenance of which is automatically protected by the database management system.

4.3.3 Technical implications of logical data modelling

In principle, any given conceptual data modelling can be realised using any of the technological methods for representing and manipulating data introduced in this chapter. However, the choice of a representation format has far-reaching practical consequences, especially when it comes to the tools required for processing the data and the necessary technical equipment as well as the lexicographic work process and the compatibility of data with the output and requirements of other projects as well. A further criterion is the flexibility and expandability of the chosen form of representation in the event of new requirements for the lexicographic information represented in the dictionary concerned. Here, relational databases are often at a disadvantage since changes to the data modelling can bring about a complex reorganisation of the table structures. Finally, in certain circumstances a justifiable balance needs to be found between the desired complexity of data modelling and the speed of data retrieval.

In a relational database, the data are distributed across many tables in the optimum form for machine processing. In order for a human processor to be able to do anything with these data, a program designed for lexicographers to edit the data has to read the desired information from the various tables using queries to the database management system and then present it as readable text. Conversely, any changes or additions to the data input via the editing application must be "translated" again by this program into SQL commands to change the datasets in the various tables. Because the input program cannot randomly change the database schema (that is, the number or structure of tables, for example) and because the database management system itself systematically prevents formal inconsistencies in the data, adherence to the chosen conceptual data modelling and the integrity of the data are guaranteed, even if several people revise the lexicographic information in an entry at the same time.

At present, XML is still the de facto standard for representing lexicographic data. Unlike a relational database, where the lexicographic information for an entry is stored in a clever way so as to be dispersed across multiple tables, XML documents are initially nothing more than plain text documents that can be read by a human being, that contain all the lexicographic information for an entry in one place, and that can, in principle, be viewed and edited in any simple text editor or word processor.

However, in practice, specialised *XML editors* are used to edit XML documents. These automatically ensure, for example, that the documents are syntactically well formed. In other words, when changes are made, the editors prevent the general rules for constructing and structuring XML documents from being inadvertently contravened, such as the end tag being forgotten after its associated start tag. Professional XML editors can present XML documents in a way familiar from word processors so that a lexicographer working on the document shown in → Section 4.2.1 sees it in a similar way to that in → Section 4.2. Such a convenience view must first of all be configured for a given XML schema. During editing, one very important function of an XML editor is constant automatic validation of an XML document with respect to a

given XML schema. In this way, if an XML editor is set to use the schema from → Section 4.3.2, it can automatically prevent an additional **part-of-speech** element from being added in the XML source text of → Section 4.2.1. Nonetheless, in contrast to relational database systems, no standard solution exists for managing as well as simultaneously and collaboratively editing what might be a huge collection of large XML documents.

Numerous established technologies exist for the machine processing of XML documents. There are specialised query languages that make it possible to read information in a targeted way, leveraging the hierarchical structure of XML documents: the query language *XPath*, which makes it possible to systematically address elements and attributes, and the powerful programming language *XQuery*, which is built on the former (→ Section 4.2.3).

In view of the considerable technological differences between relational and XML representations, it is vitally important that the two formats can essentially be translated into one another. Some XML databases can even transform XML documents automatically into relational database tables with the help of a specified XML schema in order to efficiently store, search, and retrieve the data. Conversely, XML can be used as an easy textual conversion format if the content of a whole relational database (or just the lexicographic information of a single entry) has to be transferred from one system to another or has to be further processed in a different way.

Because of the extensive translatability of representation formats into one another, special data formats that are tailored to the workflows and existing, often historically developed, technical infrastructure are often used internally in lexicographic projects. Thus, it makes sense for collaborative or partially collaborative dictionaries (→ Chapter 2.2.3) to use a markup language for revising entries that is much simpler than XML or HTML. A well-known example is the markup languages used in Wiki systems like WIKIPEDIA; these systems are also used for extensive collaborative lexicographic projects (cf. Hämäläinen/Rueter 2018; also → Chapter 8). The disadvantage of using these formats is that they are often ill-suited for modelling complex and hierarchically structured information.

Data formats used internally are often not published systematically. If it is planned to transfer the data to other projects or institutions, they are typically “translated” into standardised data formats, as discussed in the following section.

4.4 Attempts at standardisation

Over time, typical forms of presentation have emerged for the contents of dictionaries so that users of a print dictionary can find the information they are looking for quickly and easily. Thus, pieces of information that belong together normally appear grouped next to one another and the headword to which the information refers is

usually highlighted by a particular font or by its position at the beginning of the grouping. Of course, lexical information does not have to be presented in this particular way, but a targeted search for specific content would be made substantially more difficult if a dictionary diverged from these conventions.

While conventionalised forms of presentation are sufficient for human dictionary users to search for and find the desired information, machine production and interpretation of lexical data require that a specific form of representation is identified and agreed as binding – that is, standardised. In particular, the exact specification of the formats used is a necessary precondition for the practical implementation of software tools. Work processes (for example, the entire lexicographic process, as described in → Chapter 3) can also be standardised. However, in this section we restrict ourselves to discussing the standardisation of lexicographic models and data formats.

Generally, there are many reasons for modelling and storing lexical data in a standardised form:

- Using standard formats ensures that different datasets are compatible with one another. Information of the same type appears in the same form of presentation. For example, a standard format can specify the precise form in which pieces of data have to be stored. In this way, it becomes possible to process, change, and present data from different sources with the same software tools. Above all, specific access to entries and individual bits of information within these entries is made easier in situations where lexicographic data from different sources are aggregated and merged according to users' preferences (→ Chapter 7).
- The lexicographic process is often supported by different software tools (→ Chapter 3 and Abel 2012). Using standard models and formats ensures that these tools are interoperable, both conceptually and technically. Here, the standard format represents a defined interface between the tools. The agreed output format of one tool serves as the input format for another tool. In this way, exchanging data becomes technically possible beyond the boundaries of individual work groups.
- Publicly documented standards are an important prerequisite for long-lasting and sustainable storage of lexical data, i.e. for their long-term archiving. They can be understood as explicit and detailed format documentation. On this basis, software tools can also be (re)implemented at a later date, even when the programs used originally cannot be used any more for technical reasons.
- Alongside the advantages already listed, which are primarily of a technical nature, the consistent use of standard formats also supports the internal consistency and coherence of a lexical resource. For dictionaries modelled according to a specific format, the rules specified in that format take on the role of the dictionary's grammar. With the help of corresponding schema descriptions, it is very easy to check whether an electronic version of a dictionary – an instance of this schema – corresponds to this grammar. Here, the specification of the format can be formulated in a very detailed way and can, for example, lay down exactly which values may be used for detailing specialist domains. Some grammars such as

Schematron also allow rules to be formulated that determine properties of elements that refer to properties of elements positioned elsewhere in a dictionary entry. One such rule might, for example, state that an entry containing a synonym reference must not contain an antonym reference to the same target reference as well while still allowing antonym references to other targets.

– The explicit and detailed modelling and storage of data structures and data elements that is required by most standard lexical formats – in particular, when using XML technologies – means that the storage volume of electronic "dictionaries" is often relatively large. Nevertheless, this effect is negligible considering the availability of increasingly inexpensive computer storage.

The idea of modelling lexical resources on a *common* model in order to ensure the compatibility and interoperability of electronic dictionaries is certainly not a new one. Indeed, Kanngießer (1996) already considered the question in relation to the growing range of electronic lexical resources at that time from the perspective of the (integrated) re-use of those resources. Starting from the observation that the challenge for standardisation consists in depicting very heterogeneous lexical models in a single representation, he sets out the central problem: "lexical re-use [. . .] is therefore possible to the precise extent that it is possible to unify grammars and their underlying theories" (p. 92). Because lexicographic description cannot proceed in a theory-neutral way and, at the same time, grammatical theories can take incompatible or contradictory basic assumptions as their starting points, any standardisation would necessarily lead to inconsistent forms of modelling within a particular theory. However, this does not apply equally to all lexicographic descriptions. Rather it is possible to identify invariant elements, that is, elements modelled in the same way independently of the grammatical theory underlying them, which can quite probably be modelled on one another (cf. also Romary/Wegstein (2012), who refer to these elements as *crystals* in relation to electronic dictionaries). If a model is restricted to these invariant descriptive parts and specified dynamically on the basis of the concrete resource to be modelled, then at least a valid partial model can be achieved. This approach has been supported more recently by the introduction of a standardised lexical metamodel, the Lexical Markup Framework (→ Section 4.4.3).

Standards for electronic dictionaries are often distinguished by a high degree of variability and modularity, meaning that the formats and guidelines for actual lexicographic processes can be adapted to project-specific needs. Therefore, they typically provide modelling frameworks rather than strictly fixed rules for lexicographic descriptions. Nonetheless, it can happen that – independently of the dictionary – there is no suitable model in a standard for a particular type of information. In particular, innovative dictionaries of contemporary language like ELEXIKO or the DWDS find themselves confronted with this problem. As a rule, project-specific data models are developed for these purposes that focus on the necessary types of information. Still, standard formats can be used to exchange lexicographic data with third parties, al-

though the transformation then necessarily involves some loss of lexicographic information. Another possibility, albeit one that is only practicable in the long term, would involve influencing the standardisation process, leading to more specialised data models that can be adopted in later versions of a standard.

The standardisation of lexical models and data formats does not have to be limited to the formal data structures themselves. In the ideal case, it also encompasses an explicit semantic description of these data structures and the elements from which they are constructed. One possible way of explicitly describing the semantics of data elements is by referring to an index of data categories and concepts that includes "definitions" for all of the elements (often in various languages), permissible values, and relationships between classes of elements. This is often achieved by referring to common ontologies.

Generally, broader technical standards underlie the modelling of the various lexicographic "standards" described in the following sections. For example, in many cases the individual letters and symbols that appear are coded using the UNICODE standard. In order to ultimately store abstract data models as data on the computer, they are often transformed according to a family of XML standards (https://www.w3.org/XML/; → Section 4.2.1). This process is known as *serialisation*. However, in what follows, we will not explore these kinds of base standards any further. Instead, we focus on the higher-level lexicographic standards.

Organisationally, attempts at standardisation can be located at different levels. The boundaries are never sharply defined, but it is possible to distinguish three prototypical organisational levels on which standards are located with differing degrees of obligatoriness.

In the simplest case, a standard only applies to a single dictionary project or work team. Initially, such *ad-hoc standards* have little relevance outside a relatively narrow project context. They are used exclusively for working and organisational processes within a specific project and often undergo changes and adaptations in relation to the specific requirements of the project.

Standardising models and formats in larger project contexts necessitates agreement between different actors, who may have different requirements. Because of their larger community of users and, in particular, when they continue to be actively developed, they emerge as a *de-facto standard* in the field in which they are employed. De-facto standards often establish themselves by being implemented in a wide range of computer programs. The Multi-Dictionary Formatter format (MDF; → Section 4.4.4) used by linguists in the Shoebox/Toolbox working environment when they are undertaking fieldwork to document endangered languages is one example of this kind of development.

Finally, attempts at standardisation can be pursued on an international level and can culminate, for example, in the adoption of an ISO standard. Multinational consortia like the Unicode Consortium or the Text Encoding Initiative (→ Section 4.4.1) play a role similar to that of the International Organization for Standardization (ISO). One

advantage of international standardisation lies in the associated convergence towards a stable standard. Models and formats are no longer subject to short-term changes because the standardisation process on this level takes a very long time. Another advantage is that the organisational structure of international standardisation bodies guarantees a reliable, long-term point of reference, which individual time-limited lexicographic projects are unable to provide in this form.

In the following sections, we present a selected number of lexicographic formats and models in more detail. In the process, we attempt to provide a cross-section of different types of dictionary, different groups of users, and different fields in which these dictionaries are used. Furthermore, the different standards are situated on different organisational levels. Nevertheless, we shall restrict ourselves to presenting formats and models for resources intended to be consulted by human users. We will not examine specialised dictionaries and lexical databases that are developed for automatic language processing applications.

4.4.1 Text Encoding Initiative

First formulated at the end of the 1980s and continuously developed since then, the guidelines for the Text Encoding Initiative (TEI) were conceived very generally from the outset, focusing on the standardised description of texts of any kind. These guidelines have detailed, ready-made ways of describing many different types of text. With their help, it is possible to model printed literature, handwritten texts, inscriptions on gravestones, transcribed dialogues, for example, with a high degree of accuracy. Thus, they can also be used to model dictionaries. Nowadays, the TEI guidelines are the most widely used text markup standard in the (digital) humanities, and there is a vast array of resources available in this form.

The main area where the TEI guidelines are applied in lexicography is in the retro-digitisation of existing print dictionaries from one of the three main perspectives identified in the guidelines: typographical, editorial, and lexical (cf. TEI 2023). Ideally, these different perspectives are modelled in such a way that they are cleanly separate from one another. However, the TEI model also allows for hybrid forms. Let us briefly elaborate on each of these three perspectives:

The *typographical perspective* reflects the surface form of a printed page that is determined by technical (typesetting) and typographical factors. It captures information on the fonts and emphasis used, on line breaks, and on the layout of areas of text on the page as well as further medium-specific properties of the actual two-dimensional representation.

The *editorial perspective* involves an abstraction from the two-dimensional positioning of textual symbols in that it constitutes a stream of letters, punctuation symbols, and possible processing instructions for a hypothetical typesetting process. Medium-specific artefacts from the typographical perspective (such as hyphenation at

the end of a line) no longer occur in this textual model. The lexicographic information is thus modelled conceptionally as a one-dimensional sequence of symbols.

Just like the editorial perspective, the *lexical perspective* is an abstraction from the two-dimensional typographical perspective. With the help of a semantically determined inventory of categories, the lexical information is assigned to specific lexical categories. This results in a semantic annotation for each piece of lexicographic information. Furthermore, the relationship of one piece of information to another models the scope of this information and what it addresses. For example, the lexical perspective makes it possible to indicate exactly which entry each sense description belongs to or which citation is an attestation for a specific sense.

Below, the (a) typographical and (b) lexical perspectives will be compared in detail using as a starting point a short entry on the lemma *nachtlied* (night song) from the first edition of Jacob and Wilhelm Grimm's Deutsches Wörterbuch (German Dictionary, DWB-Online). While the typographical perspective reproduces many technical typesetting details (the comma after the lemma, the colon after the meaning paraphrase, the indent at the beginning of the entry, the line breaks, hyphens, and so on), no information is provided about the lexicographic status of individual sections of text – even the boundaries between pieces of information are not clearly recognisable by virtue of the markup (for example, between the indication of gender – "n." – and the beginning of the definition – "abends oder nachts gesungenes . . . lied"). By contrast, the purely lexical perspective does not indicate how to present the lexicographical information. Punctuation marks that delimit pieces of information have to be derived in a hypothetical typesetting process following rules from the sequence of information ("in the event that further information follows, a colon follows the definition"; "authors' names are set in small caps"). A normalisation of values can also take place. For example, the indication of gender in the lexical perspective appears in the form "neuter", while – again following rules – the form "n." is used in the hypothetical typesetting process in order to shorten the text. Finally, the lexical perspective can encode information that does not appear in print at all, as is the case with the indication of the word class "noun".

```
(a)
<hi rend="capitalized indented">nachtlied</hi>,
<hi rend="italics">n. abends oder nachts gesungenes oder zu
  sin-
<lb/>gendes lied:</hi> nachtlieder dichten. <hi
rend="smallcaps">Petr.</hi>
40ª; wanderers nacht-<lb/>lied. <hi rend="smallcaps">Göthe
</hi> 1,109;
```

```
(b)
<entry>
  <form>
    <orth>nachtlied</orth>
    <gramGrp>
      <gen>neuter</gen>
      <pos>noun</pos>
    </gramGrp>
  </form>
  <sense>
    <def>abends oder nachts gesungenes
      oder zu singendes lied</def>
    <cit>
      <quote>nachtlieder dichten</quote>
      <bibl>
        <author>Petrarca</author>
        <biblScope>40ª</biblScope>
      </bibl>
    </cit>
    <cit>
      <quote>wanderers nachtlied</quote>
      <bibl>
        <author>Göthe</author>
        <biblScope>1,109</biblScope>
      </bibl>
    </cit>
```

The two perspectives each have their own specific fields of application. However, for lexicographic (and metalexicographic) work, only modelling from a lexical perspective is of use since it is capable of directly reflecting the inherent tree structure of dictionary entries (→ Section 4.2.1), which results from the relation between the entry's individual pieces of lexical information.

The inventory of concepts in the TEI is organised in a modular fashion. Because the TEI model can be adapted in very specific and far-reaching ways by those who use it and because it retains the option of subcategorisation, the inventory of categories can be extended practically at will. In the TEI world, such adaptations are called *customisations*.

One notable customisation that specifically aims at the representation of dictionaries is provided by the Lex-0 initiative (TEI Lex-0 2023). The main focus of TEI Lex-0 is on interoperability across different dictionaries and, thereby, on fostering tool reuse across lexical resources. This goal is pursued by streamlining the number of elements allowed in dictionary-specific contexts. For example, the different entry-like ob-

jects allowed in the general TEI framework (`entry` – general entry, `re` – related entry, `superEntry` – groups of entries, `entryFree` – unstructured entry, `hom` – homograph) are collapsed into a single `entry` object that may be used recursively and may be associated with a type attribute if needed. Other constraints introduced by the TEI Lex-0 guidelines concern attributes that are made mandatory as opposed to their optional status in the general framework of the TEI (e.g. the `id` attribute on `entry` and `sense` elements), or tighter restrictions for contexts in which certain elements may occur. We provide a serialisation in TEI Lex-0 for our toy example *disproof* below. Note the `xml:id` attribute on the `entry` and `sense` elements as well as the `type` attribute on the `gram` element – all of which are optional in the general framework but are obligatory in TEI Lex-0.

```
<entry xml:id="MED.disproof" xml:lang="en">
  <form type="lemma">
    <orth>disproof</orth>
    <pron>dɪs'pruːf</pron>
  </form>
  <gramGrp>
    <gram type="pos">n</gram>
  </gramGrp>
  <sense xml:id="MED.disproof.1" n="1">
    <def>facts that disprove something</def>
  </sense>
  <sense xml:id="MED.disproof.2" n="2">
    <def>the act of disproving</def>
  </sense>
  <xr type="related">
    <ref target="#MED.disprove" type="entry">disprove
    </ref>
  </xr>
</entry>
```

4.4.2 Lexical-semantic wordnets

The first large-scale lexical-semantic wordnet has been developed from the mid-1980s onwards at Princeton University under the name WordNet. It was originally conceived as a model of a section of linguistic knowledge inspired by psycholinguistics and cognitive science, namely the mental lexicon. Here, mental concepts that extend across individual languages are modelled (STONE, GO, RED), which are represented by *synsets* (collections of synonyms realised in individual languages: {rock, stone}, {go, go away, depart}, {red, reddish, ruddy, blood-red, . . . }). As such, wordnets belong

to the category of onomasiological dictionaries, that is, dictionaries that assign linguistic forms to lexical meanings.

A variety of lexical-semantic relationships exist between synsets. Formally, a wordnet represents a graph for which the synsets form the set of nodes. The lexical-semantic relationships of the synsets between one another are produced through a series of relations across the collection of nodes. They can thus be conceived as the set of vertices for the graph. Such a graph is not necessarily connected; nor does it have to be free of loops. Fellbaum (1998) provides a good overview of the construction and many early applications of the English-language WordNet.

Wordnets have enjoyed great popularity up to the present time, especially in the context of computational linguistic applications. A wordnet provides a good foundation for the automatic semantic annotation and analysis of texts. If wordnets in different languages are interoperably modelled or translated into a common form of representation, this approach can be extended to cover different languages. Human users employ wordnets first and foremost as thesauri or as synonym dictionaries. Princeton's WordNet exists in two storage formats: a proprietary text-based database version (*lexicographer files*, see below) and as a Prolog knowledge base, a way of representing knowledge that has traditionally been used in the research field of artificial intelligence. Many subsequent monolingual wordnet projects have also used text-based database representations as an exchange format or have made proprietary XML-based formats available.

```
{  [ rock1, adj.all:rough^rocky,+  ] [ stone, adj.all:
  rough^stony,+ verb.contact:stone,+  ] noun.Tops:
  natural_object, (a lump or mass of hard consolidated
  mineral matter; "he threw a rock at me") }
```

At the moment there is no single, standard format used by all wordnet projects. Nonetheless, WordNet-LMF does provide a suggested LMF model for wordnets and equivalence relations between synsets (→ Section 4.4.3; cf. also Soria et al. 2009). This suggested model was implemented as an example for a series of wordnet projects but has scarcely been adopted outside the original project context so far. It has therefore remained an example of an ad-hoc standard to date.

4.4.3 The Lexical Markup Framework – a model for all types of dictionaries

The Lexical Markup Framework (LMF) was adopted in 2008 as international standard ISO 24613:2008. This standard includes a modular metamodel to describe the actual models of a variety of types of lexical resources. The most important modelling princi-

ples are the consolidation of the elements on individual levels of linguistic description in modules (syntax, phonology, etc.) and the hierarchical arrangement of those units. In order to do justice to the issues discussed above concerning the theoretically informed genesis of a dictionary, LMF contains a reference mechanism which can be used to describe explicitly the semantics of lexical concepts. This is validated by reference to a further international standard (ISO 12620:2009), which describes a data category registry.

Using a range of examples, Romary/Wegstein (2012) demonstrate that, under certain prior assumptions, lexical modelling in the TEI framework can be understood as a realisation of the LMF model. Their core argument is the way the model is limited to "crystals", which form semantically autonomous units in an entry.

Since LMF has been available as an integrative, internationally standardised (meta)model, a series of specific formats derived from it have been proposed, for example: WordNet-LMF (→ Section 4.4.2), UBY-LMF (Eckle-Kohler et al. 2012), or the lemon lexicon model (McCrae et al. 2017). It remains to be seen whether one of these proposals does indeed develop into a de-facto standard format for the LMF model or whether reference to the common metamodel already suffices in order to represent lexical resources so that they are interoperable, i.e. they are able to communicate with one another. However, what we can conclude is that existing resources can demonstrably be modelled within the LMF model in many areas of electronic lexicography. Clearly LMF provides a sufficiently wide framework in order to model lexical resources of the most varied kinds (cf. Francopoulo 2013).

4.4.4 Toolbox and Multi-Dictionary Formatter

Toolbox is a computer program provided by SIL International for documenting and managing linguistic and, specifically, lexical data. It has been used especially by linguists working on the documentation of endangered languages for many years. Because of its widespread use in this group, the Multi-Dictionary Formatter (MDF) format used by Toolbox to store data represents a de-facto standard in this field of research. Users can employ a collection of around 100 lexicographic information types and also supplement this collection with custom types.

The MDF standard format is represented as an example below. The entry for the lemma *alabanja* is part of a dictionary of Iwaidja, an Australian language (presented in Ringersma/Drude/Kemp-Snijders 2010). Lexicographic information is introduced by field labels (in the example, among others, by: \lx – "lexeme", the form of the symbol for the lemma; \sn – "sense number", semantic classification mark; \ps – "part-of-

speech", word class; \de – "definition"). The lexical model underlying the MDF standard model is that of a semasiological dictionary, that is, a dictionary starting from lexical signs and assigning meanings to them.

```
\lx alabanja
\sn 1
\ps n
\de beach hibiscus. Rope for harpoons and tying up
  canoes is made from this tree species, and the
  timber is used to make \fv{larrwa} smoking pipes
\ge hibiscus
\re hibiscus, beach
\rfs 205,410; IE 84
\sd plant
\sd material
\rf Iwa05.Feb2
\xv alabanja alhurdu
\xe hibiscus string/rope
\sn 2
\ps n
\de short-finned batfish
...
```

The addressing of information remains, for the most part, implicit in the MDF format. Although the lexical categories are clearly identified, their relationships with one another are not. Individual conventionalised classification functions constitute an exception, such as those assigned to the \sn and \ps fields in the documentation. There is no explicit hierarchical categorisation of the entry. Using the different perspectives introduced above in our discussion of the TEI model (→ Section 4.4.1), the MDF format models a mixed form between the editorial and lexical perspectives. If we consider the main field in which the format is used, this becomes immediately clear. First, a typesetting process can be derived directly from the data since the information is already stored sequentially. The field labels then acquire the role of simple typographical processing instructions. Second, targeted access to lexicographic categories is made possible for linguists so that they can retrieve and analyse the data on the basis of specific linguistic phenomena that are addressable by the field labels.

4.5 Outlook

It is customary practice nowadays that standards are used in data modelling. For example, it is almost impossible to find a relatively large dictionary project that does not rely on the use of XML-based technology. However, the picture is slightly different when it comes to the application of the lexicographic standards or guidelines discussed in → Section 4.4. On the one hand, there are numerous initiatives and infrastructure projects working to promote and refine linguistic, lexicographic, and metadata standards, such as the European CLARIN and DARIAH consortia. On the other hand, the most important requirement for individual lexicographic projects is typically to develop a data model that best suits the needs of these projects. This often results in a data model tailored to a specific dictionary. Understandably, the applicability of the data model for everyday work within the project plays a crucial role – and is sometimes more important than data exchange and interoperability with other projects. It is always possible to transform a finely granulated, tailor-made data model into a representation using more general lexicographic standards, for example one that conforms to the TEI. Nonetheless, as discussed in → Section 4.3, this kind of conversion can, at times, be fraught with the loss of highly specific annotations due to generalisations imposed by the standard and also due to deviating interpretations of certain data categories. Thus, it remains to be seen to what extent international attempts at standardisation are embraced across the board.

The most compelling question for the future will be whether the highly granular markup of lexicographic content remains a prerequisite for data to be machine accessible in the first place. It is a long-standing belief in the lexicographic community that the granular, standard-based modelling of lexicographic data fosters their usefulness and applicability and ultimately leads to the data being much easier to process. Alas, for many tasks in the field of automatic natural language processing (NLP), the best results are often achieved by machine learning (ML) approaches based on manually annotated (lexical) data. After being trained on a high-quality standard-based dataset, the computer can then annotate, analyse, and retrieve unstructured, unannotated data (the supervised ML approach). Today, with the advent of Large Language Models (LLM), purely automatic, unsupervised ML approaches are gaining ground fast, i.e. algorithms that are based on current neural network techniques and trained on huge amounts of unstructured data. As such applications increasingly reduce the need for manually prepared data, lexicography might, in the long term, lose its significance in the field of the NLP. In fact, initial attempts to let LLMs create dictionary entries are already promising (Lew 2023). However, the central role of data modelling in “producing” innovative digital lexicographic tools and resources that can be analysed and understood by humans (as opposed to the black boxes that LLMs constitute) as well as its role in the sustainable archiving of lexicographic content remains unaffected by these developments for now.

To return to the Lego analogy from the beginning of this chapter: at the moment, it is (still) more effective to ensure that the red and blue bricks, and the 2 × 1s and 2 × 2s are labelled so that the computer can grab them in a targeted way. Perhaps at some point in the future, it will be more effective to either train the computer to identify the different types of bricks among the unsorted mass of Lego – or let the computer figure out the solution entirely on its own.

Bibliography

Further reading

DARIAH-Campus. Paris: DARIAH ERIC (Digital Research Infrastructure for the Arts and Humanities European Research Infrastructure Consortium). Online: https://campus.dariah.eu/. *Online open-source platform for learning resources on topics in the digital humanities*.

Lemnitzer, Lothar/Romary, Laurent/Witt, Andreas (2013): Representing human and machine dictionaries in Markup languages. In: Gouws, Rufus H., et al. (eds.): *Dictionaries. An International Encyclopedia of Lexicography. Supplementary Volume: Recent Developments with Focus on Computational Lexicography*. Berlin/Boston: De Gruyter, 1195–1208. *In-depth summary of XML-based lexicographic data modelling.*

Romary, Laurent (2011): Stabilizing knowledge through standards – A perspective for the humanities. In: Grandin, Karl (ed.): *Going Digital. Evolutionary and Revolutionary Aspects of Digitization*. New York: Science History Publications. *Good, accessible introduction to standardisation issues in relation to lexicography*. https://doi.org/10.48550/arXiv.1011.0519 [last access: April 27, 2024].

Literature

Academic literature

Abel, Andrea (2012): Dictionary writing systems and beyond. In: Granger, Sylviane/Paquot, Magali (eds.): *Electronic Lexicography*. Oxford: Oxford University Press, 81–106.

Alexa, Melina (2011): Modellierung eines semantischen Wissensnetzes für lexikographische Anwendungen am Beispiel der Duden-Ontologie. In: Klosa, Annette/Müller-Spitzer, Carolin (eds.): *Datenmodellierung für Internetwörterbücher. 1. Arbeitsbericht des wissenschaftlichen Netzwerks "Internetlexikografie"*. Mannheim, 61–70. https://pub.ids-mannheim.de/laufend/opal/pdf/opal2011-2.pdf [last access: April 27, 2024].

Eckle-Kohler, Judith, et al. (2012): UBY-LMF – A uniform model for standardizing heterogeneous lexical-semantic resources in ISO-LMF. In: *Proceedings of LREC 2012*. Istanbul, 275–282.

Fellbaum, Christiane (1998): *WordNet. An Electronic Lexical Database*. Cambridge, Mass.: MIT Press.

Francopoulo, Gil (2013) (ed.): *LMF Lexical Markup Framework*. Oxford: Wiley.

Hämäläinen, Mika/Rueter, Jack (2018): Advances in Synchronized XML-media Wiki Dictionary Development in the Context of Endangered Uralic Languages. In: Čibej, Jaka, et al. (eds.): *Proceedings of the XVIII EURALEX International Congress: Lexicography in Global Contexts: 17–21 July 2018, Ljubljana*. Ljubljana: Ljubljana University Press, 967–978.

Kanngießer, Siegfried (1996): Zwei Prinzipien des Lexikonimports und Lexikonexports. In: Hötker, Wilfried/Ludewig, Petra (eds.): *Lexikonimport, Lexikonexport. Studien zur Wiederverwertung lexikalischer Informationen*. Tübingen: Niemeyer.

Kunze, Claudia/Lemnitzer, Lothar (2007): *Computerlexikographie. Eine Einführung*. Tübingen: Narr.

Lew, Robert (2023): ChatGPT as a COBUILD lexicographer. In: *Humanities and Social Sciences Communications* 10, 705. https://doi.org/10.1057/s41599-023-02119-6 [last access: April 27, 2024].

McCrae, John Philip, et al. (2017). TheOntoLex-Lemon Model: development and applications. In: *Proceedings of eLex 2017*, 587–597.

Ringersma, Jacqueline/Drude, Sebastian/Kemp-Snijders, Marc (2010): *Lexicon standards: From de facto standard Toolbox MDF to ISO standard LMF. Talk presented at LRT standard workshop, LREC'2010*, Max Planck Institute for Psycholinguistics, Nijmegen/Goethe-Universität, Frankfurt. https://pure.mpg.de/rest/items/item_446072_8/component/file_446073/content [last access: April 27, 2024].

Robinson, Ian/Eifrem, Emil/Webber, Jim (2013): *Graph Databases*. Sebastopol, CA: O'Reilly & Associates.

Romary, Laurent/Wegstein, Werner (2012): Consistent Modeling of Heterogeneous Lexical Structures. In: *Journal of the Text Encoding Initiative [online]* 3. https://doi.org/10.4000/jtei.540 [last access: April 27, 2024].

Soria, Claudia/Monacchini, Monica/Vossen, Piek (2009): Wordnet-LMF: fleshing out a standardized format for wordnet interoperability. In: *Proceedings of IWIC*, Stanford.

Spohr, Dennis (2012): *Towards a Multifunctional Lexical Resource. Design and Implementation of a Graph-based Lexicon Model*. Berlin/Boston: De Gruyter.

Wiegand, Herbert Ernst (1989): Der Begriff der Mikrostruktur: Geschichte, Probleme, Perspektiven. In: Hausmann, Franz Josef/Reichmann, Oskar/Wiegand, Herbert Ernst (eds.): *Wörterbücher, Dictionaries, Dictionnaires. Ein internationales Handbuch zur Lexikographie. 1. Teilbd*. Berlin/New York: De Gruyter, 409–462.

Dictionaries

DDRS = *Duden – Die deutsche Rechtschreibung*: *Das umfassende Standardwerk auf der Grundlage der aktuellen amtlichen Regeln. 28*., *völlig neu bearbeitete und erweiterte Auflage*. Berlin 2020: Bibliographisches Institut.

DWB-Online = Deutsches Wörterbuch von Jacob und Wilhelm Grimm online. In: *Wörterbuchnetz des Trier Center for Digital Humanities/Kompentenzzentrum für elektronische Erschließungs- und Publikationsverfahrens in den Geisteswissenschaften an der Universität Trier*. https://woerterbuchnetz.de/DWB/ [last access: April 27, 2024].

DWDS = *Das Digitale Wörterbuch der deutschen Sprache*. Berlin-Brandenburgische Akademie der Wissenschaften. https://www.dwds.de/ [last access: April 27, 2024].

elexiko = Online-Wörterbuch zur deutschen Gegenwartssprache. In: *OWID – Online Wortschatz-Informationssystem Deutsch*. Mannheim: Institut für Deutsche Sprache. http://www.elexiko.de/ [last access: April 27, 2024].

WordNet = *WordNet*. Princeton, NJ: Princeton University. https://wordnet.princeton.edu/ [last access: April 27, 2024].

Internet Sources

CLARIN = *Common Language Resources and Technology Infrastructure*. https://www.clarin.eu/ [last access: April 27, 2024].
DARIAH = *Digital Research Infrastructure for the Arts and Humanities*. https://www.dariah.eu/ [last access: April 27, 2024].
DBPEDIA = *DBpedia Open Knowledge Graph*. https://www.dbpedia.org/ [last access: April 27, 2024].
SIL = *SIL International*. https://www.sil.org/ [last access: April 27, 2024].
TEI (2023) = TEI Consortium (2023) (eds.): *TEI P5: Guidelines for Electronic Text Encoding and Interchange. Version 4.7.0. Last updated November 16*, *2023*, *revision e5dd73ed0*. TEI Consortium. https://www.tei-c.org/release/doc/tei-p5-doc/en/html/index.html [last access: April 27, 2024].
TEI Lex-0 (2023) = Tasovac, Toma/Romary, Laurent, et al. (2023): *TEI Lex-0: A baseline encoding for lexicographic data. Version 0.9.2. DARIAH Working Group on Lexical Resources*. https://dariah-eric.github.io/lexicalresources/pages/TEILex0/TEILex0.html [last access: April 27, 2024].
Unicode = *The Unicode Consortium*. Online: https://www.unicode.org/.
Wikipedia = *Wikipedia*, *the free Encyclopaedia*. https://www.wikipedia.org/ [last access: April 27, 2024].
Wikidata = *Wikidata Knowledge Base*. https://www.wikidata.org/ [last access: April 27, 2024].

Images

Fig. 4.1 private.