vernetziko: A Cross-Reference Management Tool for the Lexicographer's Workbench

Peter Meyer
Institut für Deutsche Sprache
Mannheim
E-mail: meyer@ids-mannheim.de

Abstract

vernetziko is an assistive software tool primarily designed for managing cross-references in XML-based electronic dictionaries. In its current form it has been developed as an integral part of the lexicographic editing environment for the German monolingual dictionary *elexiko* developed and compiled at the Institut für Deutsche Sprache, Mannheim. This paper first briefly outlines how *vernetziko* fits into the XML-based dictionary editing technology of *elexiko*. Then *vernetziko*'s core functionality and some of the auxiliary tools integrated into the program are presented from both a practical and a technological point of view. The concluding sections discuss some software engineering aspects of extending the tool to handle cross-references between multiple resources and point out some of the advantages of *vernetziko* vis-à-vis corresponding features of proprietary dictionary writing systems. The software can be adapted to interconnect off-the-shelf components (database management systems and editors), thus providing a tailor-made lexicographical workbench for a wide range of XML-based dictionaries without vendor lock-in.

Keywords: electronic dictionaries; dictionary editing software; cross-references; XML; Java

1. Introduction

The proper technical handling of cross-references within and between articles in electronic dictionaries poses several well-known problems, cf. (Joffe et al., 2003); amongst other things, the editing process must enforce and preserve the validity and consistency of cross-references as well as any required bidirectionality (symmetry) of relations such as synonymy. Many contemporary electronic dictionary systems use a semistructured markup data representation, usually based on XML (Lemnitzer et al., [to appear]), which requires specific solutions for cross-reference modeling (Müller-Spitzer, 2007; 2010a).

This paper presents and discusses the conceptual underpinnings of a modular approach to handling cross-reference structures in XML-based dictionaries. In its current form, this approach has been implemented for the German monolingual online dictionary *elexiko* which forms part of an ongoing research project of the Institut für Deutsche Sprache (Institute for the German Language) (Haß, 2005; Klosa, 2011). *elexiko* is accessible free of charge under *www.elexiko.de*. For expositional purposes, we will focus on the specific implementation chosen for *elexiko*; its overall architecture as outlined in this paper is, however, easily adaptable to other dictionary writing systems.

Section 2 is a brief survey of the overall structure of *elexiko* XML entries and the technical interplay of various components of the dictionary writing technology in *elexiko*. Section 3 focuses on the core functionality and some implementational aspects of *vernetziko*, an assistive software tool primarily designed for managing cross-references in electronic dictionaries. Section 4 presents an overview of further assistive management tools built into the program and gives some background on the database design chosen for *elexiko*. Section 5 discusses several software engineering issues that arise when extending the tool to handle cross-references between multiple heterogeneous lexicographic resources in a dictionary portal. The concluding section 6 briefly summarizes the specific advantages of the approach presented in this paper vis-à-vis monolithic dictionary writing systems with built-in reference management.

2. Background: *vernetziko* as a part of the lexicographer's software environment in *elexiko*

The lexicographic information contained in each *elexiko* entry is encoded in a single standalone XML document. A cross-reference element inside a 'source' element of one article relates to a 'target' element in the same or another *elexiko* article, usually by specifying special ID attributes of the target article and target element. In this way, cross-references are stored in a strictly local and non-redundant fashion. An important implication of this design is that cross-references assumed to be bidirectional (e.g., links between synonymous senses of two lexemes) are simply represented as two references in two separate XML articles.

Every XML document – i.e. dictionary entry – is stored in an XML-enabled Large Object (LOB) together with some metadata as a separate record (row) in an Oracle database table (Müller-Spitzer & Schneider, 2009). In order to edit an article, authors use a Content Management System (CMS) that retrieves the corresponding XML file from the database and writes the altered version back later. XML files are edited locally by lexicographers using an off-the-shelf XML editor. *vernetziko* is a Java 6 SE application that interacts with all three of the aforementioned components:

- It 'remotely controls' the XML editor via the editor's API or plugin architecture; specifically, it can parse, analyze and modify the current document content.
- It has read-only access to the dictionary database via a standard JDBC interface.
- It interacts via HTTP with the CMS in order to check articles out and in. Strictly speaking, this third interdependency is not necessary; one could easily eliminate it by allowing *vernetziko* to update the database directly. This route has not been taken for the specific technical setup of *elexiko* in order to avoid duplicating code used in the CMS for authentication and data integrity verification, amongst other things.

Overall, a highly modular approach has been chosen for *vernetziko*, such that any of the three components enumerated above may easily be replaced by a different software component. On the implementation side this modularity is enforced by programming against Java interfaces that represent the functionality of the different components and abstract away from implementational details of database queries and calls to the XML editor's API.

3. Core Functionality

3.1 Cross-reference handling in *vernetziko*

vernetziko has primarily been developed as a software tool for the automated insertion, correction and checking of cross-references in an extensible set of XML-based electronic dictionaries. Cross-references in an *elexiko* 'source' article document – typically more than 20 – relate an 'address', i.e. a specific XML element of this document, to another address that usually belongs to another entry, possibly in a different dictionary. In this manner, cross-references are stored in a strictly 'local' and non-redundant way.

Most of the functionality of *vernetziko* is designed to overcome practical issues with this pragmatic approach, particularly with regard to referential integrity:

- Manually checking the consistency and validity of all outgoing cross-references encoded in an *elexiko* article would require far too much effort. *vernetziko* cross-checks all references in the presently edited document with the database and computes appropriate status information, automatically updating its displays when the document is modified in the XML editor.
- As said above, the target of a cross-reference is specified using ID strings, viz. the values of id attributes of the targeted article and XML element. In some cases, two nested elements – for a sense and its targeted subsense – must be specified in this way. When a new cross-reference is created, manually inserting such ID values is clumsy and error-prone. With *vernetziko*, lexicographers only have to specify a lemma and then select one of its (sub-)senses from a list to let the program fill in or correct all missing details of the desired reference, cf. Fig. 1.
- Incoming cross-references for a given dictionary article can only be found through complex database queries. *vernetziko* automatically performs all necessary queries and then enumerates and checks the status of all existing incoming references for the presently edited document.
- Bidirectional cross-references (e.g., links between synonymous senses of two lexemes that are required to be symmetrical in *elexiko*) are represented as two independent references in two separate XML articles. *vernetziko* matches the lists of outgoing and incoming references for the presently edited article in order to determine whether obligatory bidirectionality is already accounted for.
- Where an incoming cross-reference to the presently edited article is not yet complete or invalid, *vernetziko* can help to update the source document of the cross-reference in a few simple steps.

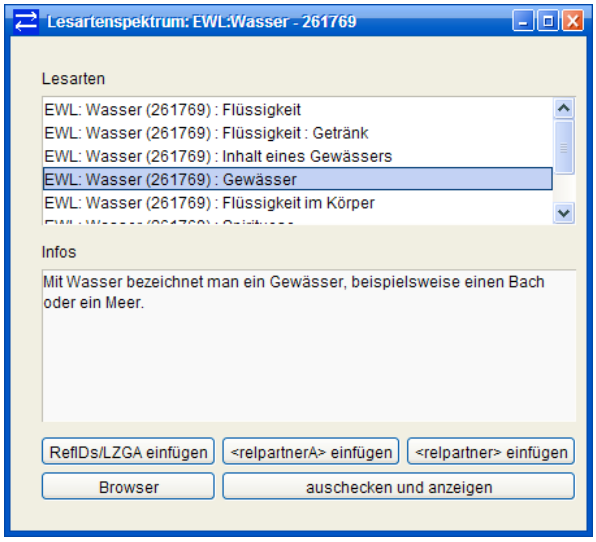

Figure 1: Selecting a word sense

Incoming and outgoing cross-references are listed in tabular form, cf. Fig. 2. References concerning sense-relations (synonymy, hyponymy etc.) are listed separately from all other kinds of references. The tables provide standard sorting and filtering functionality. *vernetziko* also offers a tree view of outgoing word sense references that displays the relevant parts of the XML structure (Fig. 3). Both these tables and the tree view can be used interactively for fast navigation, sorting and reference insertion.

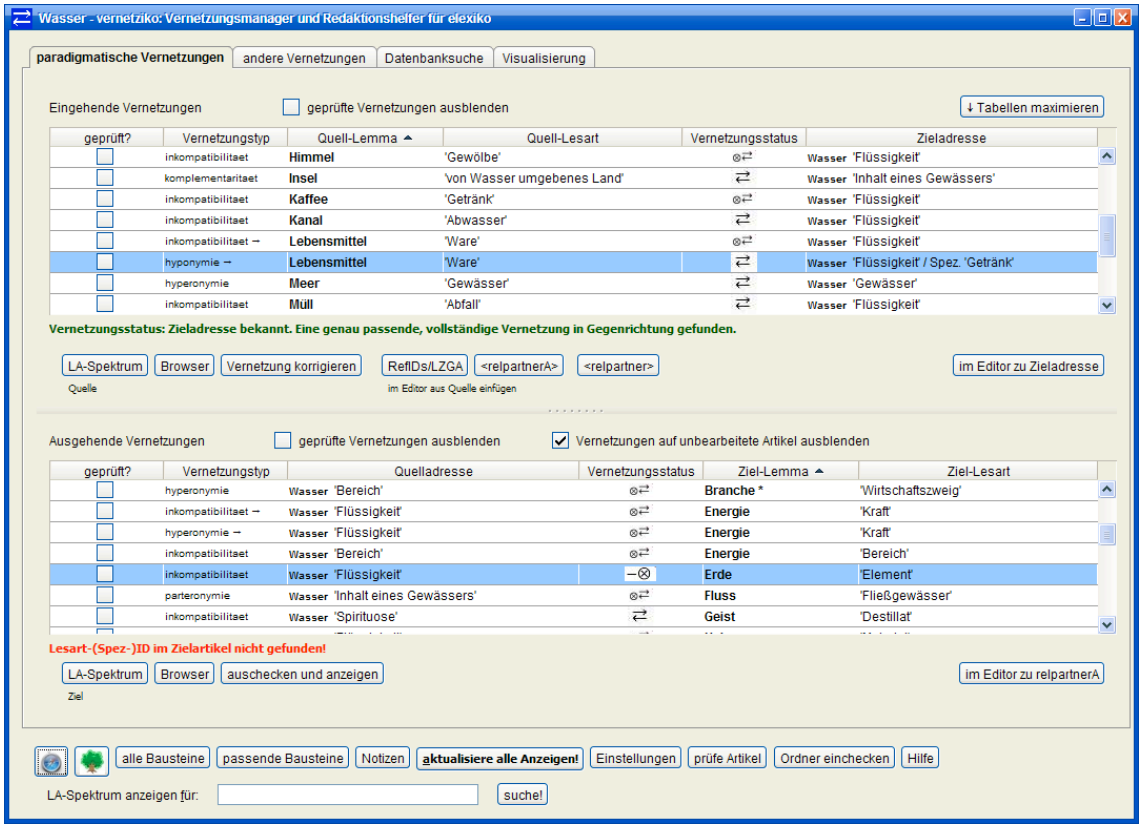

Figure 2: Tabular view of incoming (upper table) and outgoing (lower table) sense relations of *Wasser* ('water')

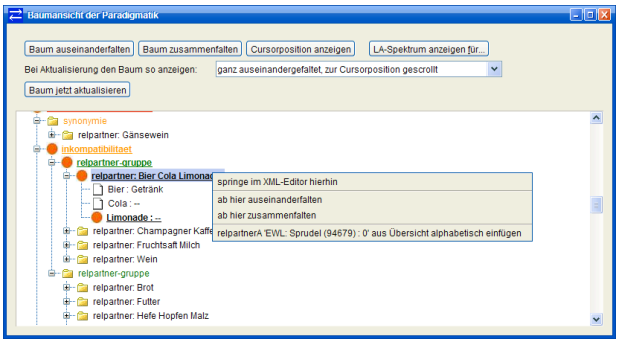

Figure 3: Using the tree view for sense relations

3.2 Cross-reference status

For the working lexicographer, the most relevant information in the tabular presentation is the *status* of the individual cross-references. The status is symbolized by various arrow icons that inform the user on the extent to which different requirements are met. In particular, cross-references should be *complete* and *well-formed*; more important, they must be *valid*, pointing to a target address that really exists in the lexicographic database, even if this address happens to be a preliminary reference to a still unedited article. Compulsory symmetry and transitivity in certain reference types such as synonymy can be an additional consistency requirement.

A cross-reference may fail to be valid or consistent in many different ways. The status icons are based on a systematic typology of possible cross-reference statuses that is exhaustive but still perspicuous and practical from the lexicographer's point of view.

In order to simplify the exposition of this typology, some terminology will be introduced first. A *unidirectional cross-reference*, or reference for short, is a labeled ordered pair consisting of a source and a target *address*. The label is the *relation type* encoded by the cross-reference, e.g. 'is a synonym of', 'is morphologically derived from'. An address is an identifiable subpart of a *resource*. Besides dictionary entries, examples for possible resources include files, Internet URLs and other digitally represented structured text documents. In a dictionary entry, sections pertaining to specific word senses are examples of addresses. If a dictionary entry is encoded as an XML document, any XML element within that document is a potential address, as long as it is systematically identifiable by an XPath expression. In many resources, different address types must be distinguished, such as word senses vs. sections on grammar in a dictionary. When no reference to a subpart of a resource is possible or necessary, this will be modeled as an address type with only one trivial address per resource. – Note that source and target address may belong to the same resource.

Simplifying somewhat, *vernetziko* distinguishes between the following statuses of unidirectional references:

a. The target resource does not exist or its specification is either formally inadmissible or factually inconsistent.
b. The specification of the target resource is incomplete.
c. The target resource is correctly specified, but the target address within that resource does not exist or its specification is either formally inadmissible or factually inconsistent.
d. The target resource is correctly specified, but the target address within that resource is not fully specified, possibly because the target resource is an as yet unedited entry.
e. The target address is correctly and fully specified.

If at least the target resource has been specified correctly in two different cross-references and there are no inconsistencies or other errors in both references (i.e., only cases d. and e. apply), these two references form a *possible bidirectional cross-reference* and are thus *possible reverse cross-references* to each other if and only if their relation types match (e.g. hyponymy vs. hyperonymy) and the target address of each reference is either equal to the source address of the other reference or contains this source address as a subpart.

For a given reference R this leads to the following panoply of possibilities regarding reverse references:

f. There is no possible reverse cross-reference for R, although the relation type of R admits of such references.
g. There is no possible reverse cross-reference for R, although this is considered compulsory (e.g. in case of synonymy, at least for *elexiko*).
h. R and exactly one of the potential reverse cross-references both have status e. above (target address correctly and fully specified). This is the case of a 'perfect' bidirectional reference.
i. There is more than one possible reverse reference for R, but none of these cases meets the requirements of h. above.
j. There is exactly one potential reverse reference, but at least one of the two references is not fully specified (in the sense of d. above).

In order to establish the status of cross-references, *vernetziko* uses Oracle's XML-enabled full text search capabilities to obtain all incoming cross-references, then reads in the XML data of all entries referencing and referenced by the presently edited one, parses all XML documents using a StAX parser and finally tries to match all cross-references with addresses in the respective entries and with possible reverse references. The user can start this process manually; a background task checking periodically for relevant changes in the currently edited XML document updates status information every five seconds.

3.3 Implementational aspects: Handling the interplay with the XML Editor

A fair amount of typical editing functions must be present in the XML editor's API, such as navigating the caret to arbitrary XML elements, inserting, deleting and modifying XML elements, opening and closing XML documents etc. As stated above, a Java interface represents all methods used to call editor functionality from within *vernetziko*. The editor-specific API calls themselves are encapsulated in a single class that comforms to this interface. For the *elexiko* project, two implementations of the interface have been developed so far, viz. for Corel XMetaL 3.1 and for the <oXygen/> XML editor (version 13). Any editor suitable for this kind of modular setup must either be usable as an application server to other standalone programs (for instance, through a COM mechanism in MS Windows operating systems; this is the case with XMetaL) or expose its API via some sort of plugin architecture (this is the technique chosen for <oXygen/>). These two scenarios have rather different technical implications, however; changing from one of them to the other is not a trivial task. In the first case, *vernetziko* is a standalone desktop application, in the second, it is provided as a bunch of plugin classes.

The most difficult aspect of a modular approach to remote-controlling the XML editor is that different editors use different, mostly proprietary, APIs to describe the structure of XML documents. Naturally, all of these APIs bear a certain similarity to, e.g., the Java DOM API. Since the editor-specific API classes representing XML nodes, elements, documents and attributes must be processed in many ways by *vernetziko*, it is necessary to devise editor-independent interfaces that represent the needed functionality of node/element/attribute/document classes. The editor-specific XML objects are then referenced in wrapper classes implementing these interfaces. This way, we obtain an editor-independent DOM-like representation of the editor's XML nodes; throughout *vernetziko*'s code, only the wrapper classes are used.

4. Further assistive management tools

4.1 Features of the user interface

vernetziko features a number of additional tools that help to speed up and simplify the editing process:

- Article-specific notes and XML snippets can easily be stored, retrieved and inserted into the edited document.
- An advanced database search tool allows complex Boolean combinations of search criteria including metadata and XPath expressions, cf. Fig. 4.
- Administrators may perform operations on large sets

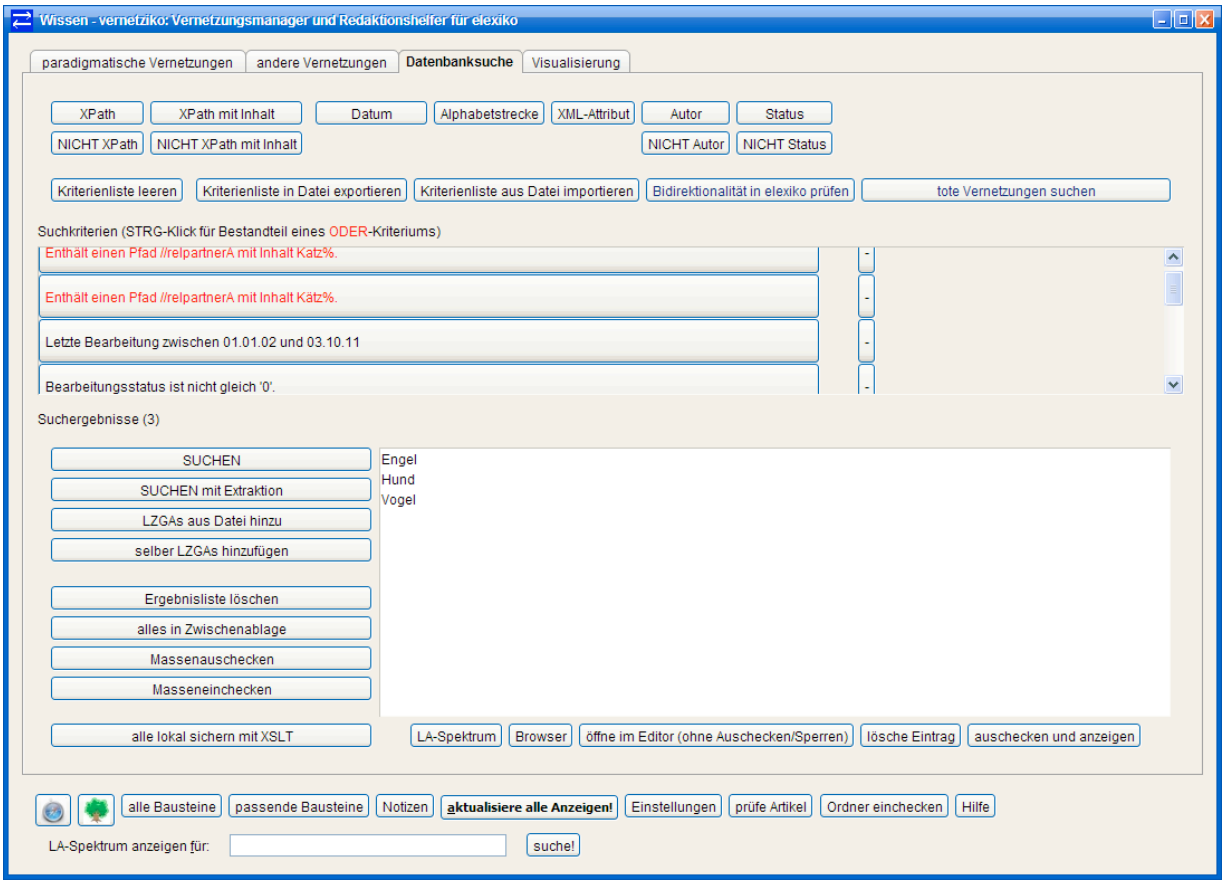

Figure 4: Extended database search and scanning options

of entries (alter user access rights, check in or out) that can be defined by search criteria or specified manually or from lemma list text files.

- Cross-references concerning sense relations such as synonymy and hyponymy can be visualized graphically. The visualization program traverses arbitrarily long chains of incoming or outgoing cross-references and can recursively construct graphs with very large numbers of nodes (word senses).

vernetziko not only helps to secure consistency of cross-references in individual dictionary entries, but also provides tools for scanning the entire lexicographic database of *elexiko* for problematic cross-references, viz.

- inconsistent references, in particular 'dead' references pointing to inexistent entries or word senses;
- unidirectional references for which a required reverse reference does not exist yet.

The results of these scans are output as UTF-8 text files.

4.2 General considerations on database design

There are several reasons why the seemingly obvious strategy of storing the cross-reference structure of a dictionary in a separate relational data structure in the database is not always feasible. For instance, the exact position of cross-references within the source element in an article's XML representation might vary depending on lexicographical considerations, which would necessitate the use of 'pointers' from within the XML document to the external link table. In such cases, a separate cross-reference table introduces new sources of possible inconsistencies and considerably complicates the editing process for dictionary entries since two database tables must be modified concurrently and kept in synch. It can be argued that an automatically updated relational 'cache' table that simply duplicates basic cross-reference data (addresses and reference type) is the right solution to meet performance requirements in these cases (cf. Joffe et al., 2003; Meyer & Müller-Spitzer, 2010). For the time being, even this solution is not used in the *elexiko* project since the database XML query technology is still fast enough to cope with real-time requirements.

The approach taken for *elexiko* therefore employs a maximally lean and redundancy-free database design and shifts the administrative burden to the external software tool *vernetziko*.

5. Managing Cross-References in a Dictionary Portal: Software Engineering Considerations

elexiko forms part of OWID, a web portal of German electronic dictionaries (Müller-Spitzer, 2010b). A tool such as *vernetziko* should be easily adaptable to the integration of new lexicographic resources into the portal,

in particular with respect to managing cross-references between different dictionaries of potentially heterogeneous structure, i.e. with widely differing DTDs/schemas.

The design of such management software has to address many challenges, if maximum generality, extensibility and reusability of software components are to be combined with a maximally perspicuous and parsimonious approach. These challenges include the following points:

- In an Internet portal, new online dictionaries may be added at any time. Entries in different dictionaries may be structured in various ways – conforming to widely differing DTDs/XML schemas – and contain disparate types of addresses.
- A specific type of address (e.g., word senses in a dictionary) may be encoded differently in different dictionaries. For example, the structure of the XPath associated with a word sense in an XML-based entry might vary according to the dictionary.
- The very same address type might be referred to in different ways depending on the referring resource, using, e.g., different XML attribute names.
- Sometimes the position where a reference is encoded in a document is relevant to the identification of the reference, sometimes not.
- Address types may differ to a great extent in the kind of informational structure associated with them; compare references to word senses with references to web URLs or citations.
- The programmer should be able to add new types of addresses or references in a modular way, if possible without touching existing classes.
- Different resources, address types, and reference types require different operations in a management tool. There is no set of common methods for all address or reference types pertaining to a certain resource. One and the same address type might even have to be treated differently, depending on the resource it occurs in. The implementation of methods that take references as input can depend on the resource and address type of both source and target entry.

Thus, from a software engineering perspective, *vernetziko* has to cope with a variant of what is often called the *expression problem*: New dictionaries and cross-reference types may require the addition of both new classes representing types of resources/addresses/references and new operations on objects of such classes. For *vernetziko*, a very simple solution based on parameterized types will be presented here. The solution is not strictly type-safe in that it uses type checks and subsequent casts, but given the lack of self types, multimethods, mixins etc. in Java, any completely type-safe solution produces an enormous overhead in static languages, cf. (Torgersen, 2004). In the pragmatic approach taken for *vernetziko*, there is one and only one class that is responsible for dispatching all method calls concerning resources, addresses and references. After adding new classes of any of these entity types, only the dispatcher class needs to be modified accordingly; type checks and casts are performed only in this class.

5.1 Domain entity classes

Entries. Since the different portal dictionaries are no suitable candidates for domain entities – no elementary operations are performed on dictionaries as a whole –, the notion of an entry belonging to a specific dictionary (or, more generally, that of a resource) is taken as the point of departure for the domain class model. All entry classes such as *Dictionary1Entry*, *Dictionary2Entry*, … derive from an abstract class *Entry* and store information that identifies the particular individual resource.

Addresses. Different address types are represented by subclasses (*WordSenseAddress*, *GrammarAddress*, …) of an abstract *Address* class that contain a reference to the *Entry* object the address object 'belongs' to. Different address types will require widely differing sets of fields for the information associated with them. One and the same address type may appear in entries of different resources; for instance, two dictionaries may each have dedicated sections for different word senses within every entry. On the other hand, a distinction between word senses in Dictionary1 and Dictionary2 is still needed, since they might have slightly differing formal representations, such as differing names of the relevant XML elements or attributes. Therefore, we parameterize the static address types on the type of the *Entry* field. In Java notation, the same sort of address, e.g., word senses, is reflected by different static types, e.g. *SenseAddress <Dict1Entry>* and *SenseAddress <Dict2Entry>*, according to the resource its entry belongs to.

References can be dealt with accordingly. In many scenarios, a single *Reference* class will suffice whose fields are references to the source and the target *Address* objects. Depending on the context, further fields will be used to represent classificatory or status-related information about a reference. Here, we parameterize on the types of both the source and the target address. The static type of a specific reference from a word sense in a dictionary entry to a paper in a specific volume of a linguistic journal may then look as follows in Java: *Reference <SenseAddress <DictionaryEntry>, PaperAddress <JournalEntry>>* (where *JournalEntry* objects model journal volumes).

5.2 Dispatcher class

Although objects of classes *AddressX <Dict1Entry>* and *AddressX <Dict2Entry>* share the same internal class makeup – representing the same sort of address in two different resources and therefore both being of type *AddressX<? extends Entry>* –, they must possibly be

handled differently, requiring, e.g. different code for navigating in the editor to the corresponding element. On the other hand, code duplication has to be avoided in the case where certain (but possibly not all!) methods pertaining to these both types can in fact be implemented identically.

In addition, not every address type is 'compatible' with a given resource (images don't have word senses); additionally, most combinations of a source and a target address type do not amount to a valid reference type. Many operations may only be relevant for a small subset of, say, address types (consider the task of printing information about an address). These many 'holes' in the matrices of actually existing type combinations and actually permitted parameterized types per operation cannot be accounted for in advance by the type system or some sort of inheritance hierarchy.

In typical scenarios, most methods don't change the state of entry, address and reference objects, the latter rather being used like 'passive' information containers. In addition, new functionality operating on addresses or references might be added at any time to the management application, which increases the danger of bloated and ever growing interfaces with empty implementations for many subclasses.

All considerations mentioned above point to a solution where domain entity objects are treated as mere data containers with minimal public interfaces. All public methods of the domain entity classes relay to the special dispatcher class mentioned above. As an example, a method call like *myAddress.moveXMLEditorCaretHere()* would be relayed by calling a static method, *Dispatcher.moveXMLEditorCaretHere(myAddress)*. The static method *moveXMLEditorCaretHere(Address<?> anAddress)* of the dispatcher class then type-checks the input parameter *anAddress* and, after a corresponding cast, calls the appropriate method of some service class in a type-safe manner. Note that though the Java compiler erases type information in generics, the parameter type can be obtained at runtime by getter methods: In our example, *myAddress* holds a reference to the resource (i.e. dictionary entry) it belongs to; the runtime type of this resource object is identical to the parameter type of *myAddress*.

6. Concluding Remarks

The software tool *vernetziko* adds advanced cross-reference management facilities and various helper tools to an already existing dictionary database system and editing environment. This is possible due to a modular software design that encapsulates the access to both other components of the IT environment, such as the XML editor, and the internal structural makeup of the lexicographic data involved. Support for new dictionary resources and new types of cross-references within and between dictionaries can easily be added in a plugin-like fashion. While most of the functionality provided by *vernetziko* is part and parcel of many commercial dictionary writing systems, the main advantage of the approach taken with *vernetziko* is that the software can be adapted to interconnect a wide variety of off-the-shelf components (database management systems and editors) and allows tailor-made access to and administration of almost arbitrary XML resources and legacy dictionary data, thus providing the 'glue' for a tailor-made lexicographical workbench without vendor lock-in – ideally suited to large-scale projects and to the management of cross-references between multiple dictionaries.

7. References

Gamma, E., Helm, R., Johnson, R.E., & Vlissides, J. (1995). *Design Patterns. Elements of Reusable Object-Oriented Software*. Amsterdam: Addison-Wesley Longman.

Haß, U. (ed.) (2005). *Grundfragen der Elektronischen Lexikographie: Elexiko - Das Online-informationssystem zum deutschen Wortschatz*. Berlin, New York: de Gruyter.

Joffe, D., de Schryver, G.-M. & Prinsloo, D.S. (2003). Computational features of the dictionary application "TshwaneLex". *Southern African Linguistics and Applied Language Studies* 21(4), pp. 239-250.

Klosa, A. (ed.) (2011). *elexiko - Erfahrungsberichte aus der lexikografischen Praxis eines Internetwörterbuchs*. Tübingen: Gunter Narr.

Lemnitzer, L., Romary, L. & Witt, A. (to appear). Representing Human and Machine Dictionaries in Markup Languages. In R.H. Gouws, U. Heid, W. Schweickhard & H.E. Wiegand (eds.) *Dictionaries. An International Encyclopedia of Lexicography. Supplementary Volume: Recent Developments with Special Focus on Computational Lexicography*. Berlin/New York: de Gruyter.

Martin, R.C. (1997). Acyclic Visitor. In R.C. Martin, D. Riehle & F. Buschmann (eds.) *Pattern languages of program design 3*. Boston, MA: Addison-Wesley Longman, pp. 93-103.

Meyer, P., Müller-Spitzer, C. (2010). Consistency of Sense Relations in a Lexicographic Context. Workshop on Semantic Relations, *International Conference on Language Resources and Evaluation (LREC) 2010*, May 18, Malta.

Müller-Spitzer, C. (2007). Vernetzungsstrukturen lexikografischer Daten und ihre XML-basierte Modellierung. *Hermes* 38, pp. 137-171.

Müller-Spitzer, C. (2010a). The Consistency of Sense-Related Items in Dictionaries. Current Status, Proposals for Modelling and Potential Applications in Lexicographic Practice. In P. Storjohann (ed.) *Lexical-Semantic Relations. Theoretical and Practical Perspectives*. Lingvisticæ Investigationes Supplementa. Amsterdam/New York: Benjamins, pp. 145-162.

Müller-Spitzer, C. (2010b). OWID – A dictionary net for

corpus-based lexicography of contemporary German. In A. Dykstra, T. Schoonheim (eds.) *Proceedings of the XIV Euralex International Congress. Leeuwarden, 6-10 July 2010*. Fryske Akademy: Leeuwarden, pp. 445-452.

Müller-Spitzer, C., Schneider, R. (2009). Ein XML-basiertes Datenbanksystem für digitale Wörterbücher – Ein Werkstattbericht aus dem Institut für Deutsche Sprache. *it – Information Technology* 51(4), pp. 197-206.

Torgersen, M. (2004). The Expression Problem Revisited. Four New Solutions Using Generics. In M. Odersky (ed.) *ECOOP 2004 - Object-Oriented Programming: 18th European Conference, Oslo, Norway, June 14-18, 2004. Proceedings* (Lecture Notes in Computer Science 3086). Berlin: Springer, pp. 123-146.