Southern African Linguistics and Applied Language Studies 2003 21(4): 239–250
Printed in South Africa — All rights reserved

SOUTHERN AFRICAN LINGUISTICS AND APPLIED LANGUAGE STUDIES
ISSN 1607–3614

Computational features of the dictionary application "TshwaneLex"

David Joffe[1,3]*, Gilles-Maurice de Schryver[2,3] and DJ Prinsloo[3]

[1] TshwaneDJe, PO Box 299, Wapadrand 0050, South Africa

[2] Department of African Languages and Cultures, Ghent University, Rozier 44, B-9000 Ghent, Belgium

[3] Department of African Languages, University of Pretoria, Pretoria 0002, South Africa

** Corresponding author, e-mail: david.joffe@tshwanedje.com*

Abstract: The aim of this article is to introduce and to elaborate on computational aspects of TshwaneLex, a new South African software application for dictionary compilation. A brief introduction to the lexicographic needs for such a computer program in South Africa will be given, followed by a description of the most salient features of the software package. Four key features of TshwaneLex will then be analysed, viz. (a) the data presentation in a tree structure, (b) the extendible input/output architecture, (c) database loading issues, and (d) the cross-reference system. For each of these the challenges encountered and/or the design decisions that have been made will be examined.

Introduction

It is reasonable to assume that sophisticated software should be available for the compilation of modern dictionaries in an era characterised by advanced programs for the manipulation of human language in general, such as software for word processing, corpus query and even voice recognition. This is however not the case. In nearly all instances publishing companies, such as Oxford University Press, Longman, HarperCollins, Cambridge University Press or Macmillan, as well as private dictionary compilers, use either in-house custom-made software, or they use general-purpose tools such as word processors.

Word processors offer the benefit of being user-friendly, but do not store dictionary data in a structured way. General data manipulation tools such as XMLSPY (2003) are more powerful, but require advanced technical knowledge of technologies such as XSL (Extensible Stylesheet Language, 2003) to perform tasks that should otherwise be simple for the lexicographer, such as quickly viewing a preview of a dictionary article. Custom-developed tools can solve these problems, but are generally far more expensive than off-the-shelf tools, as the same economies of scale do not exist to recover the development costs.

These are all problems facing the NLUs (National Lexicography Units) tasked with the compilation of monolingual and bilingual dictionaries for the nine official African languages of South Africa. Previously, the *Onoma Lexical Workbench* software was intended to fill this need. However, Onoma has since been discontinued.

Given this situation, the design of a new software tool for dictionary compilation was accelerated by its creator, David Joffe, supported by Gilles-Maurice de Schryver and DJ Prinsloo. The data of the in-house dictionary project *SeDiPro* at the University of Pretoria (De Schryver & Prinsloo, 2001: 384–385), as well as initial drafts of monolingual dictionary articles for Sesotho sa Leboa (Northern Sotho), were used in the development of the program. TshwaneLex (derived from *Tshwane* 'Pretoria' and *lexicography*) has been specifically designed with the needs of dictionary compilation for African languages in mind, but is not limited to African languages.

Primary features of TshwaneLex

TshwaneLex has been built against the background of certain fundamental assumptions in contemporary lexicography and a number of

specific general requirements and expectations of today's dictionary compiler. The first underlying assumption is the user-perspective in lexicography. Metalexicographers such as Hartmann and James (1998), Wiegand (1998), Prinsloo and De Schryver (1999), Gouws (2000), to name but a few, emphasise and formulate in great detail the importance of the user-perspective in modern reference works. From the viewpoint of *dictionary users* a dictionary compilation program should be designed in such a way that the output, dictionary articles, is structured and presented in a user-friendly way. However, at stake here is a second dimension of user-friendliness, namely one with specific reference to the users of the compilation program itself, i.e. the *dictionary compilers*.

A core focus of TshwaneLex has thus been user-friendliness. In this regard, a primary goal has been to, whenever possible, present a *familiar* abstraction to lexicographers, namely that of dictionary articles. Particular technical implementation details are, whenever possible, hidden from the lexicographer, for example with the generic input/output interface. Thus, lexicographers do not need to have an advanced level of computer literacy in order to perform the general day-to-day tasks of compiling a dictionary. In addition to making dictionary compilation more accessible to less technically literate lexicographers, this also has the benefit of lowering training time.

The second core functionality desired from TshwaneLex is to be able to create both monolingual and bilingual dictionaries, possibly even within the same database, which additionally allows for the production of semi-bilingual dictionaries. Thirdly, the program should provide a WYSIWYG (What you see is what you get) preview of how an article would appear on paper (for a printed dictionary) and/or onscreen (for an electronic dictionary). This preview should update immediately in response to changes made by the compilers. In the fourth instance it should include support for all relevant required fields specific to the indigenous African languages, such as noun classes or ga/sa/se fields (for the latter, cf. Prinsloo & Gouws, 1996). Finally, teams of lexicographers must be able to work simultaneously on the same dictionary database over a computer LAN (Local Area Network), which necessarily includes facilities such as preventing multiple lexicographers from trying to modify the same article simultaneously.

An overview of the functionality of TshwaneLex

The primary editing interface

The primary interface for editing the dictionary that the lexicographer is presented with is the language window. A language window is displayed for each language in the dictionary. For a bilingual dictionary, two language windows appear side by side, allowing the lexicographer to view and work on both sides of the dictionary simultaneously. Compare Figures 1 and 2 for the typical layout of a monolingual and a bilingual dictionary respectively.

Each language window (one in Figure 1, two in Figure 2) consists of four primary sections, viz. (a) lemma list, (b) tree view, (c) preview area, and (d) tools window.

The *lemma list* is a list of all lemmas in the dictionary, from which the lexicographer can select which lemmas to view or work on, and where newly created ones are added. A scroll bar enables rapid access to the entire lemma list.

The *tree view* shows the hierarchical relations of all senses, sub-senses, definitions, translation equivalents, usage examples, cross-references and combinations in the article of the selected lemma. The various constructs of the article appear in different colours, for example lemma signs are displayed in blue, translation equivalents in green, and combinations in red.

The *preview area* shows, in the same colour scheme as the tree view, a preview of the selected article. This allows the lexicographer to see what the article would look like in the final product. The preview area also displays articles that have cross-references to the currently selected article, as well as articles that are cross-referenced by the currently selected article, allowing the lexicographer to immediately check the validity of the cross-references.

The *tools window* displays a number of subwindows that contain various editing utilities, such as a text search facility and a lemma selection filter. Various text attributes are associated with the different types of constructs (lemma sign, definitions, usage examples, etc.) of an article. For example, a lemma sign contains text fields such as pronunciation particulars, deriva-

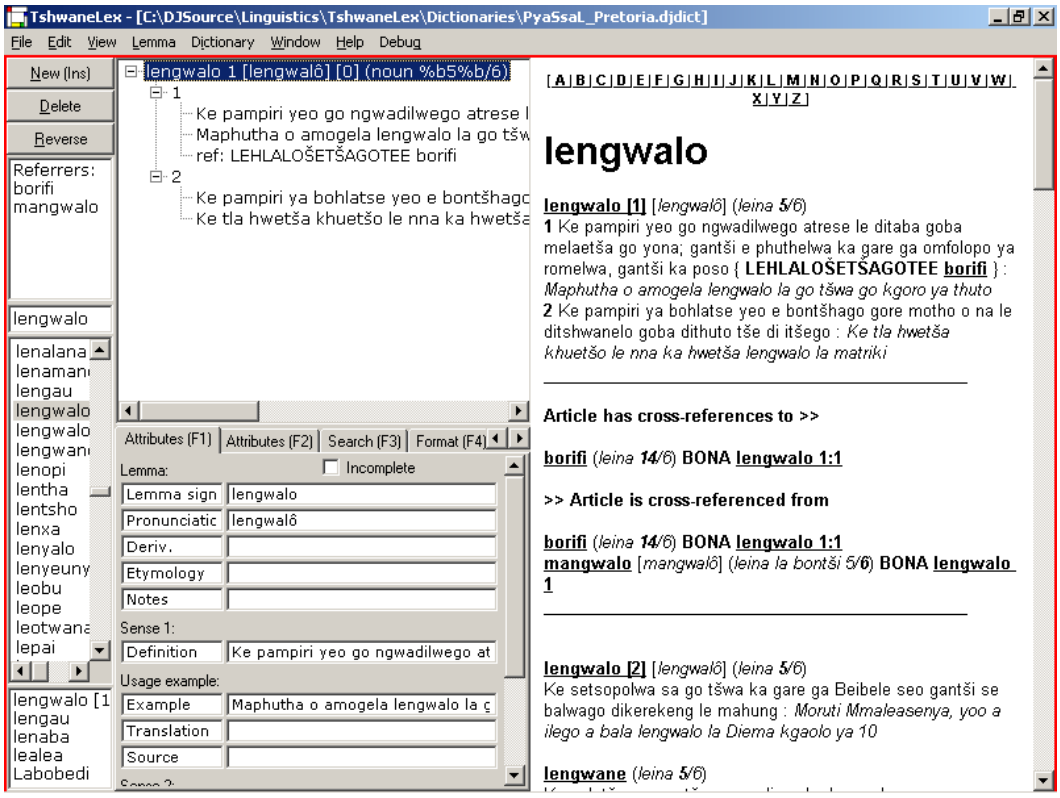

Figure 1: Screenshot of the TshwaneLex monolingual editing interface*

tion information, and an etymology field. The first sub-window, the *text attributes* window (F1), allows these fields to be modified by the lexicographer. The text attributes window also includes a checkbox for marking an article as being incomplete. The compiler can select this when unsure of some aspect of an article and wants to return to it later. Incomplete articles are marked with a question mark '?' and are automatically excluded from final dictionary output.

The second sub-window (F2) is used for inserting *usage labels and parts of speech* (including, for the African languages, noun class affiliations). The *search* window (F3) allows the entire dictionary to be searched for a text string. The *format and preview* window (F4) provides formatting preferences, and options for selecting the meta-language. Different sets of text labels can be created and selected by the compilers for usage labels and part of speech tags, thus allowing these to be displayed in the target user's own language. As a result, two different versions of, say, a Sesotho sa Leboa–English dictionary can be created from the same database — one for speakers of English, and the other for speakers of Sesotho sa Leboa. This will allow, for the first time in South Africa, major bilingual dictionaries to be produced specifically for mother-tongue speakers of African languages.

Finally, the *filter* function (F5) allows the lexicographer to define criteria for displaying a subset of lemmas only. With this function one can thus focus on lemmas with certain speci-

* Due to the nature of 'screenshots' Figures 1–4 are not of the usual standard. They are included because of the value they add to the article. To view these figures in their original format consult the online version of the journal available at: **http://www.ingentaselect.com**

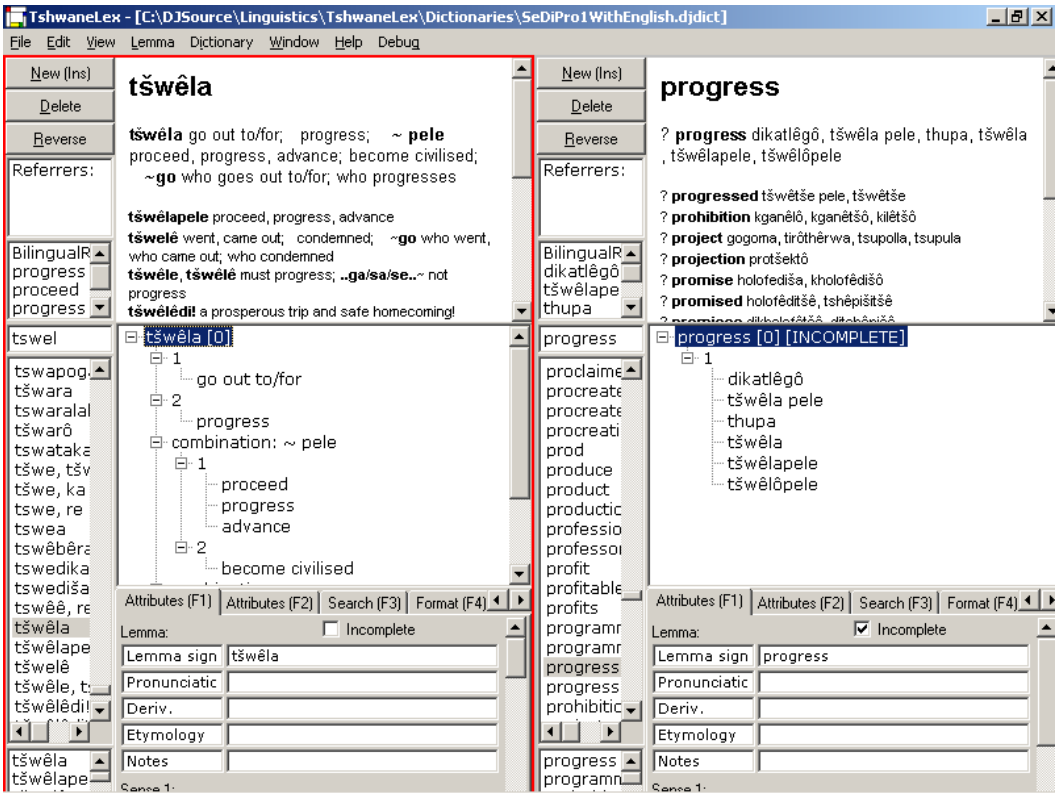

Figure 2: Screenshot of the TshwaneLex bilingual editing interface (note that the English articles on the right have been generated using automatic reversal)

fied characteristics. For example, the dictionary compiler can choose to display only those articles that are marked as incomplete, and then proceed to work on those. There are two basic filter operations, which may be combined. The first defines criteria for *inclusion* of articles, and the second defines criteria for *exclusion* of articles. The criteria within each operation may be combined with either a logical AND ("all of the following criteria") or a logical OR operation ("any of the following criteria"). A compiler might for instance define a filter to "display all articles which are marked as incomplete AND have usage examples", or a filter to "display all articles which are marked as incomplete OR have usage examples".

By combining the various filter operations, fairly complex filters may be defined. For example, in order to view all complete articles which have both a translation equivalent and a usage example, but which have no combinations, no cross-references and no ga/sa/se fields, the filter shown in Figure 3 can be created.

Bilingual editing features

TshwaneLex provides certain special bilingual editing features. These will only be mentioned briefly here. When in "linked view mode", the language window for the target language automatically displays only those lemmas that are related to the selected lemma in the source language. With this feature the consistency across the two sides of a bilingual dictionary can be safeguarded. Another bilingual editing feature is the "automatic reversal" function, which provides a basis for the creation of the other direction of the dictionary by automatically generating reversed lemmas. Although representing a flying start to reversing the dictionary (as can be seen from Figure 2), substantial manual lex-

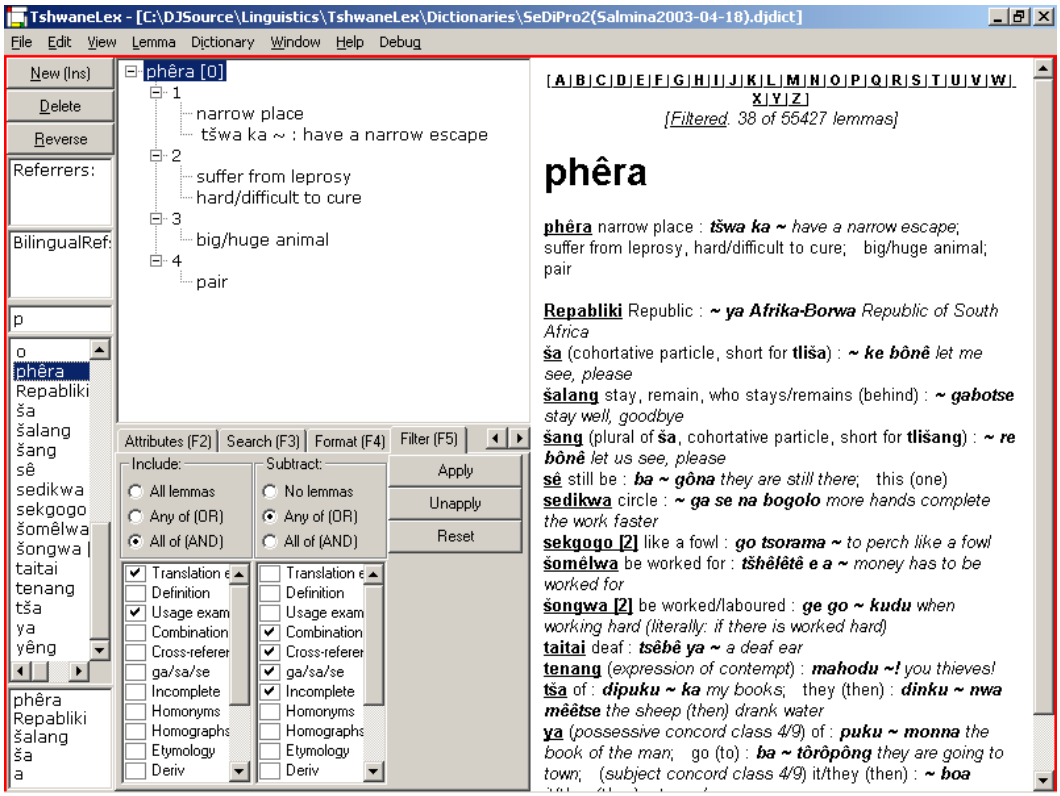

Figure 3: Screenshot demonstrating the lemma filter

icographic input will in most cases still be required.

Dictionary compare/merge editing feature

The dictionary compare/merge feature allows one version of a dictionary database to be compared against another. Articles that differ are shown side by side, allowing visual inspection of the differences. Differences may be resolved either by merging the two articles, or by replacing one with the other.

Apart from its evident use in version control and track keeping of progress, another useful application of this feature presents itself when a dictionary team is, for whatever reason, split up geographically over long distances. Indeed, it is not always feasible to have a high-speed network connection from remote sites back to the main dictionary database server. This is particularly true in developing countries. Lexicographers working remotely thus need to be able to continue working while not connected to the primary database server. The compare/merge function may then periodically be used to merge the changes into the main database, after which the newest version of the database can be distributed back to the compilers working remotely. Note that this is actually a reality for the Sesotho sa Leboa NLU, where the Head Office is based in Polokwane, while a Branch Office is located in Pretoria.

A discussion of four key features of TshwaneLex

The tree structure

From a shallow analysis, the structure of each lemma in a dictionary appears to be merely hierarchical: a lemma contains word senses, which may themselves have sub-senses. Word senses may in turn contain compounds (such as idioms), usage examples, translation

equivalents and so on. However, lemmas are not independent of one another — they can be highly inter-linked throughout the dictionary by cross-references. Computationally, the dictionary may thus be regarded as a directed graph, with each structural element of the dictionary being regarded as a node in that graph. However, if one excludes the cross-references, the resultant data structure is a tree. Using a tree allows various aspects of the software design to be simplified, such as synchronising network access to data, as each lemma may be treated as a single unit. Thus, the dictionary is implemented as a hierarchy of nodes, with the cross-references being stored in a separate structure. This is, however, essentially transparent to the user. There are different types of nodes for the different structural components of a dictionary. The top-level node corresponds to the dictionary itself. This dictionary node consists of one or more child nodes representing languages. Each language node contains a list of lemma-sign nodes, which may in turn have nodes for word senses, definitions, usage examples, cross-references and so on.

This hierarchy of nodes is considered so central in the design that the tree structure of a particular article is shown in the *tree view* whenever work is being done on that article. Such graphical representations can vary from relatively simple structures to much more complicated ones in the case of highly polysemous lemmas. Furthermore, the tree view is also the central window from which the lexicographer makes changes to the design of an article, such as adding new senses or usage examples. The internal tree structure of an article thus corresponds directly to the visual tree control in the language window's tree view, and each node in the tree view corresponds to a node in the article's structural hierarchy.

Each node type has various data attributes that may be associated with it; for example the lemma-sign node has, as pointed out above, pronunciation, derivation and etymology fields. Generally — and this is a crucial design feature — selecting a node in the tree view results in the *text attributes* window (F1) displaying the text attributes associated with the node type of the selected node. Reformulated, the text edit boxes that are shown in the sub-window F1 are *directly* linked to the tree view. So, if the lemma-sign node in the tree view is selected, then text edit boxes for the attributes of a lemma sign are shown in F1. If a usage-example node is selected, then text edit boxes for the various attributes of a usage example (i.e. example, translation and citation) are shown.

Extendible I/O (input/output) architecture

TshwaneLex has been designed with a modular, extendible architecture for loading and saving dictionaries from and to different data formats. The I/O interface between the TshwaneLex application itself and between the so-called "data source" has been abstracted, allowing interfaces for new types of data sources to be defined by inheriting from the general interface and implementing the specific behaviours required to interface with the specific data source, using standard OO (Object Oriented) methodology. Thus the application can load and work with dictionaries without directly "knowing" the implementation details of the system for storing and retrieving the dictionary data (e.g. a relational database or XML file). This design also allows TshwaneLex to potentially be extended, if necessary, with add-on modules for additional data sources. Such a module can be used to migrate an existing dictionary from an older system to TshwaneLex; for example, a module has been written to import the Onoma database format that was used (cf. De Schryver & Lepota, 2001: 3) by the Sesotho sa Leboa NLU.

A data source module may have different capabilities; a module that only exports to a particular format (but cannot be read in again), such as the HTML exporter, only has write capabilities. A customised importer for a third-party dictionary format may have only read capabilities, such as the Onoma importer.

Some data source types need to be able to support network editing, i.e. multiple lexicographers working on the same database over a LAN. For this purpose, TshwaneLex uses ODBC (Open Database Connectivity, 2002) and its query language, SQL (Structured Query Language). This theoretically allows TshwaneLex to interface with any ODBC-compatible database software.

Note that due to this requirement to have multiple lexicographers working simultaneously on the same data, XML is not considered prac-

tically suitable as a general storage format for the dictionary database. This is because the fundamental unit of logical storage in an XML database is the XML document itself. Thus, when a modification is made to the dictionary, the entire document needs to be retrieved, modified and re-stored (Staken, 2001). XML is, however, provided as an optional export format.

The import and export modules currently supported by TshwaneLex are shown in Table 1.

The generic I/O architecture also has an impact on the user-friendliness of TshwaneLex, as the lexicographer does not need to learn many technical details about each particular storage method. These details are, whenever possible, hidden from the lexicographer, who is thus presented with a consistent and predictable interface for all data storage types.

Database loading issues: 1. Loading the dictionary

One of the decisions in the design of TshwaneLex was whether or not to load the whole dictionary into memory at once. Some dictionary editing systems, such as Onoma, do not attempt to do so. Onoma only loads a small number of currently selected articles "on demand", and keeps them in memory only while those articles are being viewed or modified by the lexicographer. This was not perceived to be a desirable approach; it was felt that the lexicographers would greatly benefit from being able to view and browse the *entire* dictionary at once, as if scrolling through a word processor document. However, it would be too slow to do this if lemmas were loaded "on demand" every time they needed to be displayed. Thus, it is essentially necessary to load the entire dictionary database into the computer's memory at once when the dictionary database is opened. This is the approach followed by TshwaneLex.

The feasibility of this approach depends on the size of the dictionary, and the memory size and the speed of the computers used by the lexicographers. A moderately powerful Personal Computer today, such as a 2.4GHz Pentium 4 with 256MB RAM, should be capable of handling a dictionary the size of a typical commercial desktop dictionary. It should generally be possible to cope with larger dictionary databases by upgrading the computers used. However, this becomes impractical for extremely large dictionary databases, such as historical dictionaries, which may be hundreds of megabytes in size. The second edition of the OED (Oxford English Dictionary, 1992), for example, consists of over 600 megabytes of data. One possible partial solution to this is to divide a dictionary database into several subsections or "volumes". Nonetheless, future versions of TshwaneLex will be designed to be able to handle such large dictionary databases.

The relative advantages and disadvantages of the two approaches were considered, and it was felt that the benefits to the lexicographer of having the entire dictionary in memory at once outweighed the disadvantage of a few minutes wait at the start of each day's work. These advantages and disadvantages are summarised in Table 2.

Database loading issues: 2. Speed

The size of a dictionary database for a typical commercial desktop dictionary can be quite large, possibly in the order of several hundred thousand nodes and attributes. Unfortunately, the table-based structure of a relational database is not a very efficient method of storing and retrieving hierarchically structured data. A database table is a "flat", linear structure, and nodes read from the dictionary database must be resolved into a tree structure when loaded. Because all nodes are stored in a single table,

Table 1: Import and export modules currently supported by TshwaneLex

	Load (Import)	Save (Export)	Network
ODBC	Y	Y	Y
.tldict binary file format	Y	Y	
XML		Y	
Static HTML		Y	
RTF		Y	

Table 2: Advantages (+) and disadvantages (–) of loading the entire dictionary into memory at the start vs. loading articles "on demand"

	Load entire dictionary into memory	Load articles "on demand"
+	• Faster response whenever an article is selected for viewing or editing. • Lexicographer can browse all lemmas easily and quickly, and thus gains a more holistic view of the dictionary. • Allows for features such as immediate display of cross-references to and from the current article. • Allows for features such as rapid text search of the entire dictionary.	• No long wait when loading the dictionary at the start of the day's work.
–	• Relatively long wait when loading the dictionary at the start of the day's work, possibly up to several minutes, depending on the size of the dictionary and the speed of the network and computers used.	• Possibly perceptible delay whenever an article is selected while that article loads from the database. • Long delay whenever the dictionary needs to be exported, e.g. to XML or to RTF format, since the entire dictionary must be read from the database anyway.

attempting to load all nodes for a specific lemma would recursively require several linear searches (one for each "level" in the tree hierarchy) through this table by the database server, each of which, using big-O notation, has an O(n) completion time, where n is the total number of nodes in the dictionary. Thus, if there are m "layers" of nodes in the hierarchy of a dictionary article, the completion time of a query to fetch all nodes associated with that article would be O(mn). Although the value of m will usually be small, for larger dictionaries (i.e. large values of n) this operation can become noticeably slow, and since the operation must be performed frequently while working on the dictionary, the lexicographer will experience this slowness. Thus, it is worth improving the performance of this operation. In order to do so, redundant data has been added to the database, a not uncommon optimisation technique in database schema design. In this case, a redundant field has been added to the node table, which if non-NULL, contains the node ID of the root node of the article to which the given node belongs. This allows all nodes belonging to a particular article to be fetched with one single query on the node table, which can be completed in O(n) time.

Observe that it may be possible to quite significantly speed up the loading time of dictionary articles by using a local disk based write-through cache of articles in the dictionary. Whenever an article is selected, the "last-modified" timestamp of an article in the database could be checked against the timestamp of the version stored in the cache, and if the article in the database has not changed, the cached version could be used. This technique will be explored as a possible optimisation in future versions of TshwaneLex.

Other alternative techniques may also be investigated for storing the data in the database in order to speed up loading and saving of lemmas. One possibility is to depart from a strictly relational model, and to store each article as a single "chunk" of XML in a text field in a single table, rather than using many separate, linked tables. Compare Blaschke (2003: 61–62) in this regard.

Cross-references: 1. The cross-reference system

In terms of Wiegand (1996) the lexicographer uses cross-referencing to refer the dictionary user from one point in the dictionary (the *reference position*) to another (the *reference address*) for the main purpose of providing more lexicographic information. The *reference relation* itself is established by a *reference marker*, often text segments like 'see', 'compare', 'antonym', etc. or symbols such as →, ⇨, ▷, =, etc. For a detailed discussion of cross-

referencing as a lexicographic device, see Gouws and Prinsloo (1998).

Dictionaries are often marred by so-called *dead references* where the reference address does not exist. See Prinsloo (1996) for examples. The dictionary compiler is therefore engaged in a constant battle to maintain the cross-reference system and should be maximally supported by the dictionary compilation software in this regard. This firstly means that the dictionary compilation software should be designed in such a way as to ensure that there are no dead references, and secondly that cross-references should be adjusted whenever the locations of reference addresses change.

One of the important features of the cross-reference system in TshwaneLex is the ability to *automatically* update the homonym numbers and/or sense numbers at a reference position, whenever the homonym numbers and/or sense numbers of the reference address change. For example, if a lexicographer has created a cross-reference from the article **borifi** 'letter (for correspondence)' to "lengwalo 1:2" (**lengwalo** homonym 1, sense 2), and another lexicographer working on the article **lengwalo** later on decides to swap senses 1 and 2 around, thus implicitly changing the reference address, TshwaneLex will automatically update the cross-reference at the reference position to "lengwalo 1:1" (**lengwalo** homonym 1, sense 1).

Cross-references: 2. Representation of cross-references in the database

Two possible methods were considered for the storage of cross-references in the database. It should be kept in mind that these two approaches are not necessarily mutually exclusive, and a "hybrid approach" which allows both is under consideration.

The first method is to store cross-references as text strings. For example, if the lexicographer wishes to create a cross-reference from **borifi** to **lengwalo**, the text string "lengwalo" must be typed into a text field. In order to create cross-references to specific homonyms or word senses, a syntax is defined to allow the lexicographer to specify these in the cross-reference text field, e.g. "lengwalo 1:2" for "lengwalo homonym 1, sense 2". The Onoma dictionary compilation software uses such a scheme.

The second method is to store cross-references internally using the unique node IDs of the target article lemma or sense nodes. An actual structural link is thus created from the source lemma or sense (the reference position) to the target lemma or sense (the reference address). In TshwaneLex, the lexicographer uses a specific interface dialog, the cross-reference editor, to establish these links by clicking on the required cross-reference target lemmas or senses. An immediate preview is shown of both the source article and the destination article, allowing the lexicographer to make sure that the cross-reference is correct from a lexicographic standpoint. A screenshot of the cross-reference editor is shown in Figure 4.

It was decided that the ID-based cross-references were definitely a better option for TshwaneLex. It is a more user-friendly system, easier to learn, and most importantly leaves much less room for error. The advantages and disadvantages of the two approaches are summarised in Table 3.

Cross-references: 3. A hybrid approach

As a possible solution to some of the disadvantages of ID-based cross-references, a "hybrid" approach may be implemented, which would allow both ID-based and text-based cross-references to be employed. This would enable a lexicographer to create cross-references to articles that do not yet exist by using text-based cross-references. Later on, when the target article has been created, the software would have a function to automatically "resolve" the text-based cross-reference and convert it into an ID-based cross-reference. This function would inform the lexicographer if the text-based cross-reference was invalid and could not be resolved.

Cross-references: 4. The cross-reference cache

One of the design goals of TshwaneLex was to have an "instant" preview of the currently selected article. This preview, by default, also automatically displays all articles that have cross-references *to* the currently selected article. However, in order to achieve this, whenever an article is selected, the entire dictionary database effectively needs to be searched in order to locate all of the articles that have cross-references to the selected article. This poses a problem for the update speed of the

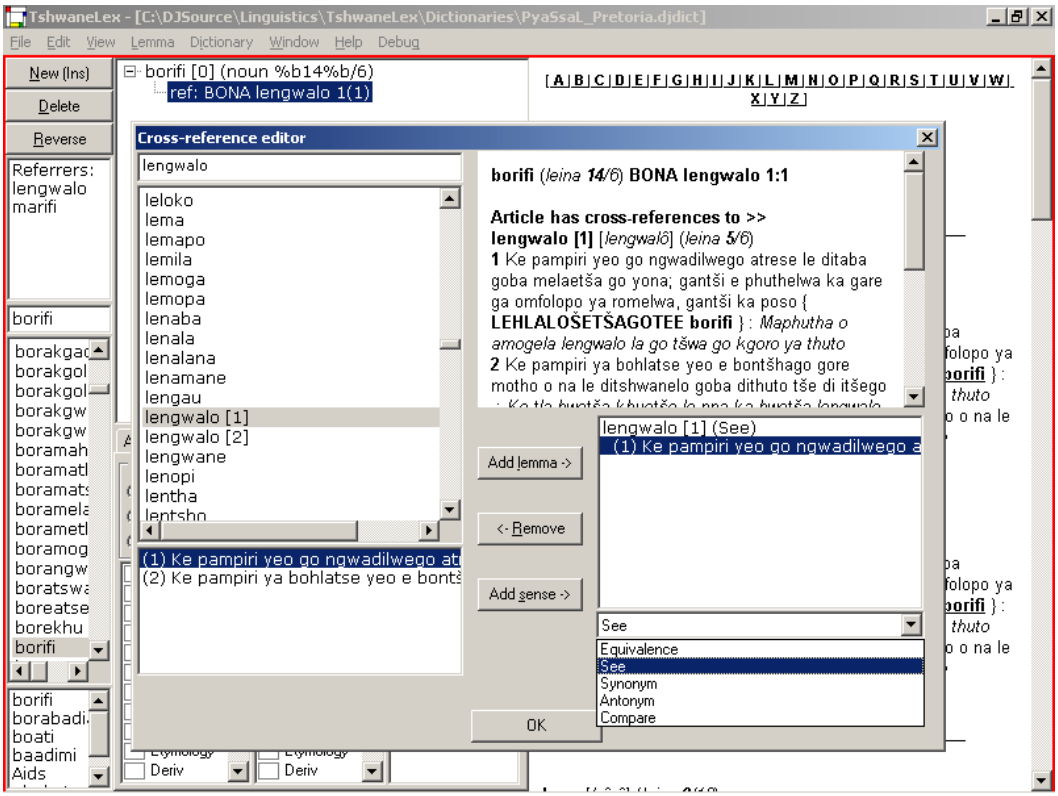

Figure 4: Screenshot of the cross-reference editor

Table 3: Advantages (+) and disadvantages (–) of ID-based cross-references vs. text-based cross-references

	ID-based cross-references	Text-based cross-references
+	• Ensures, from a structural perspective, the referential integrity of cross-references. • System effectively results in "automatic" updating of cross-references when cross-reference target homonym numbers, sense numbers or even lemma signs change.	• Simpler software design.
–	• Cannot create a cross-reference to an article if the target article does not yet exist. • Interfacing with the database becomes more complex; for example it may be problematic if one lexicographer creates a cross-reference to an article that another lexicographer is busy modifying. • Complicates various aspects of software design, such as network editing.	• Not user-friendly; lexicographers must learn a potentially confusing syntax for entering cross-references to particular homonyms and/or word senses. • Cross-references must be tracked manually, which results in a large likelihood of errors. Statistically, even the most experienced lexicographer is going to make some mistakes, or forget to update a cross-reference if a sense number or homonym number changes.

preview. For purposes of analysing the speed, a monolingual test dictionary with 50 000 randomised lemmas was created, and cross-references were created in 10% of these lemmas. The test dictionary contained 124 130 senses, 309 632 translation equivalents, 62 026 definitions, 37 121 usage examples and 5 000 (i.e. 10% of 50 000) cross-references. The total number of nodes in the test dictionary was 587 911. All operation timings were performed on a Pentium III 550MHz computer with 128MB RAM running the Microsoft Windows 2000 operating system. To search through all these nodes took on average 210ms on the test system. Although this seems fast, a 210ms delay each time an article is selected, while quickly scrolling through articles, would be "felt" by the compiler, and provide a hindrance to the "natural" process of editing the dictionary. Moreover, this delay would grow proportionally as the dictionary grows.

In order to solve this problem, advantage was taken of the fact that changes to the cross-referencing structure within the dictionary happen relatively seldom. A cross-reference "cache" has been created which contains a list of only the cross-reference nodes in the dictionary database. This cache is only updated when modifications are made to the cross-referencing structure of the dictionary, that is, whenever cross-references are created, deleted or modified. This cache then allows TshwaneLex to immediately search only the 5 000 cross-reference nodes in the dictionary, rather than searching through all 587 911 nodes in order to first find the cross-reference nodes. The effective number of nodes to search has thus been reduced from 587 911 to 5 000.

Using the cache, the search time for finding referrers is reduced from 210ms to, on average, 5ms. This is considered fast enough for "real-time" updating of the preview area.

Currently, a linear search is used to search the cross-reference cache, which has a completion time of O(n). For exceptionally large dictionaries (i.e. dictionaries with hundreds of thousands of lemmas), this may prove to be too slow. However, it should be possible to speed the search up even further, should it be necessary. Since the search key for finding cross-reference nodes is the cross-referenced lemma, one could keep entries in the cross-reference cache sorted by the lemma ID of the cross-reference target and use a binary search instead. The completion time of a binary search is $O(\log_2 n)$.

The future

In this article TshwaneLex, a novel dictionary compilation environment, was introduced. Special attention was given to four aspects which the creators and lexicographic advisors deem central. The first version of TshwaneLex will be focused primarily on the function of dictionary compilation, with specific priority being placed on the immediate needs of the South African National Lexicography Units. However, the software has been designed with a number of possible future applications, extensions and features in mind, such as electronic dictionary software, as well as a web interface for creating online versions of dictionaries. A first implementation of such an online dictionary interface created with TshwaneLex can already be found, for Sesotho sa Leboa, at http://africanlanguages.com/sdp/. Other possibilities include user-friendly interfaces to terminology management applications and concordance programs, and possibly even having such functionality built into TshwaneLex.

References

Blaschke C. 2003. Distributed terminology management: Modern technologies in client/server environments. In: De Schryver G-M (ed) *TAMA 2003 South Africa: Conference Proceedings*. Pretoria: $(SF)^2$ Press. pp. 59–64.

De Schryver G-M & Lepota B. 2001. The lexicographic treatment of days in Sepedi, or when mother-tongue intuition fails. *Lexikos* **11**: 1–37.

De Schryver G-M & Prinsloo DJ. 2001. Corpus-based activities versus intuition-based compilations by lexicographers, the Sepedi lemma-sign list as a case in point. *Nordic Journal of African Studies* **10**(3): 374–398.

Gouws RH. 2000. Toward the formulation of a metalexicographic founded model for national lexicography units in South Africa. In: Wiegand HE (ed) *Wörterbücher in der*

Diskussion IV. Vorträge aus dem Heidelberger Lexikographischen Kolloquium (Lexicographica Series Maior 100). Tübingen: Max Niemeyer Verlag. pp. 109–133.

Gouws RH & Prinsloo DJ. 1998. Cross-referencing as a lexicographic device. *Lexikos* **8**: 17–36.

Hartmann RRK & James G. 1998. *Dictionary of Lexicography.* London: Routledge.

HTML. 2003. HyperText Mark-up Language. Available at: http://www.w3.org/MarkUp/ [Accessed 12 April 2003].

ODBC. 2002. *Open Database Connectivity*. Available at: http://dmoz.org/Computers/Programming/Databases/ODBC/ [Accessed 12 April 2003].

OED. 1992. *Oxford English Dictionary, Second Edition on Compact Disk.* Oxford: Oxford University Press.

Prinsloo DJ. 1996. Review: Robert Botne and Andrew T. Kulemeka. 1995. A Learner's Chichewa and English Dictionary (Afrikawissenschaftliche Lehrbücher 9). *Journal of African Languages and Linguistics* **17**(2): 199–202.

Prinsloo DJ & De Schryver G-M. 1999. The lemmatization of nouns in African languages with special reference to Sepedi and Cilubà. *South African Journal of African Languages* **19**(4): 258–275.

Prinsloo DJ & Gouws RH. 1996. Formulating a new dictionary convention for the lemmatization of verbs in Northern Sotho. *South African Journal of African Languages* **16**(3): 100–107.

RTF. 1999. *Rich Text Format*. Available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnrtfspec/html/rtfspec.asp [Accessed 12 April 2003].

Staken K. 2001. *Introduction to Native XML Databases*. Available at: http://www.xml.com/pub/a/2001/10/31/nativexmldb.html [Accessed 12 April 2003].

Wiegand HE. 1996. Über die Mediostrukturen bei gedruckten Wörterbüchern. In: Zettersten A & Pedersen VH (eds) *Symposium on Lexicography VII. Proceedings of the Seventh International Symposium on Lexicography, May 5–6, 1994, at the University of Copenhagen* (Lexicographica Series Maior 76). Tübingen: Max Niemeyer Verlag. pp. 11–43.

Wiegand HE. 1998. *Wörterbuchforschung: Untersuchungen zur Wörterbuchbenutzung, zur Theorie, Geschichte, Kritik und Automatisierung der Lexikographie. 1. Teilband*. Berlin: Walter de Gruyter.

XML. 2003. *eXtensible Markup Language*. Available at: http://www.xml.org/ [Accessed 12 April 2003].

XMLSPY. 2003. *XMLSPY*. Available at: http://www.xmlspy.com/ [Accessed 4 August 2003].

XSL. 2003. *Extensible Stylesheet Language*. Available at: http://www.w3.org/Style/XSL/ [Accessed 4 August 2003].