

THE TKSESH DATABASE

AND INFORMATION EXCHANGE IN THE EGYPTOLOGICAL COMMUNITY

SERGE ROSMORDUC

Tksesh is an integrated philological system for the Egyptologist. It is a multiplatform editor, database and dictionary. The core of the system is a hieroglyphic editor. Edited texts constitute a searchable full text database, which can be referenced by a dictionary system, via hyperlinks. The dictionary can handle complex definitions by multiple authors. This communication briefly presents *Tksesh*-characteristics, and then discusses the issues it raises concerning electronic information exchange in Egyptology: data format, co-operative sharing of work, and referencing other's works.

Introduction

The object of this communication is a database system called *Tksesh*. It is intended as a full desktop for the philologist. The idea is that one should be able to use it to enter data associated with philological work, such as hieroglyphic transcriptions, transliterations, translations, lexicological work, notes, references about other works, etc.

Initial Motivation

The motivations which led to the realisation of *Tksesh* were numerous. First, we had a number of tantalizing examples of databases in the domain of classical studies, like the TLG or the Perseus Project.¹ The interest of these realisations as exploratory tools was self-evident. Second, from a computer science research point of view, we thought it was interesting to consider what improvement automated language processing could bring to such lan-

¹ TLG – *Thesaurus Linguae Graecae*: www.tlg.uci.edu/; PER – *Perseus Project*: www.perseus.tufts.edu/. Further articles: S. BILLET, *Apports à l'acquisition interactive de connaissances contextuelles*, Thèse de doctorat de l'Université Montpellier II (another approach of automated transliteration), Montpellier 1995; S. ROSMORDUC, *Analyse morpho-syntaxique de textes non ponctués*. Thèse de doctorat, École normale supérieure de Cachan, Cachan 1996; S. MEKNAVIN, P. CHAREONPORN SAWAT and B. KIJSIKUL, *Feature-based Thai Word Segmentation*, in: *Natural Language Processing Pacific Rim Symposium 1997*, Phuket 1997, 182-186.

guage databases. The third motive we had was that we were interested in document structure, from a computer science point of view. And philological works are very complex documents. In particular, referencing is an ubiquitous phenomenon. To say it in a few words, a typical philological study is full of links: links between the parts of the work, notably between the various texts and the corresponding notes, and links between different works: comments on other studies on the same subject, references to dictionaries, to parallel passages, etc. Computers are rather good at linking. Indeed, they are much more practical for this than paper, hence their potential interest.

Objectives of the System

Besides the goal of writing an interactive environment for philological work, we also want *Tksesh* to be a testbed for NLP (Natural Language Processing) techniques. Due to its architecture, *Tksesh* can be used as a toolbox for building new applications, provided we write a good technical documentation. *In fine*, our most important ambition is simply to increase the number of electronically available ancient Egyptian texts. Software has a rather limited life span; on the other hand, computer data can be adjusted for new needs.

Outline of Tksesh






The *Tksesh* software contains a number of elements, designed to function together. These elements are both quite independent, as far as they provide very different features, and intimately connected, as far as most elements refer to the others.

The central element of *Tksesh* is a *hieroglyphic editor*, which was designed with speed and ease of use as design principles. This editor is used for entering texts which then constitute a full text database. This database can be searched for words or sequences of hieroglyphs. Now, this is an interesting lexicological tool, but it is a bit coarse. If used alone, its interest is currently limited for two reasons. The first is that such a tool is really powerful if the text database is huge, and we haven't got enough data (yet ?). The second one is that words often need to be discussed, related to other words, and so on. In other terms, a plain database doesn't replace a good dictionary (the reverse is also true). For this reason, *Tksesh* also includes a dictionary editor, which allows one to search and enter structured dictionary entries.

The Hieroglyphic Editor

Tksesh hieroglyphic editor was primarily designed for entering texts in the database. This design decision has got a number of consequences. For instance, the system doesn't support printing (although the texts are in *Manuel de Codage* format and can be read in software like Winglyph). In general, priority was given to features relevant from a linguistic database point of view over presentation features. The text's purpose was not to be an electronic *facsimile* of the original documents. However, this approach has its limit, and as time goes by, *Tksesh*'s graphical possibilities are improving.

The editor system is based on the *Manuel de Codage*. Basically, one enters codes for the signs, or selects them in the menus. Grouping can be done from the menus, from shortcuts, or by typing Manuel codes like ":" or "*". However, the system ensures the final result is a correct *Manuel de Codage* file. For example, it's not possible to forget a parenthesis or the end of a cartouche.

The strong point of *Tksesh* as far as typing goes is that it is tolerant about codes. Transliterations are not limited to the list of the *Manuel de Codage*; instead, the system tries to do something reasonable. The signs from GARDINER list have been entered with their phonetic values. When the user types a transliteration, the system proposes a sign. If the resulting sign is not the expected hieroglyph, a press on the spacebar will propose a new one. For example, if one types "mr", he will get . If he wants the pyramid-sign (O24), he will press "space" a few times, and get . Next time "mr" is entered, the system will remember and propose O24 first. Even better, when the list of possible signs is exhausted, the system looks in the dictionary, for words having the said transliteration. Thus, typing "p3" and spacebar will first propose  then  and finally , fetching the word from the dictionary. If the user stops here, the following uses of "p3" will get him the definite article, which is a nice feature when you type Late Egyptian texts. The proposed list is cyclic, and further presses of the spacebar will go back to the first proposal.

The Dictionary Editor

Introduction

The next complex part of *Tksesh* is its dictionary system. It allows one to query already existing entries (currently looking for either a transliteration, a translation, or a hieroglyphic word), and to create new ones. The ultimate goal is to allow sharing of these dictionary entries between *Tksesh* users.

We started writing it as a simple lexicon, with three fields: "transliteration", "spelling", "translation", the latter being free text. But it soon came

out that a richer structure would be interesting. Considering the amount of work behind a dictionary's data, it's worth taking the time and pain to organize it. A real dictionary entry is something both complex and very structured. Entering it as free text is not a very good option, because its structure is lost to the computer. The human reader might be able to reconstruct it, the system won't. Much automated processing that would be possible with a well structured lexicon would be impossible. On the other hand, we wanted a versatile format, suitable for many users and many needs. The kind of information that can be found in real dictionaries ranges from simple translation to whole encyclopaedic articles on the words, including cultural information. Finally, we wanted an extensible structure, which would allow new kinds of data to be added later, while keeping the old files.

Structure of the Dictionary

The dictionary format is formal enough to be usable by a computer; yet it still allows a variety of definition styles. To achieve this, the entries are coded like tagged text, with a precise structure. We have tested its versatility by entering a few definitions from a number of different paper dictionaries, like FAULKNER, WILSON'S *Ptolemaic lexikon*, etc... The fields in the dictionary are of roughly three types: base fields, which contain one type and only one type of information (for example, a transliteration), complex fields, that can contain mixed information (e.g. text in transliteration and hieroglyphs at the same time), and finally organisational fields, like the group and comment fields.

Example Dictionary Entry

The figure below represents the entry for the word *3wl*. This entry corresponds to a number of meanings for the word: "extended", "long", "deceased", etc... The group fields are used to structure the entry. Groups are marked both by indentation and by « and » quotes. The whole entry is considered as a group. Groups can be nested, to represent sub-meaning of a words, derived words, and so on. The basic point is that if a group contains multiple fields of the same kind, let's say multiple transliterations, they are supposed to be variants. In the case of translations, this would mean near-synonymous meanings. In the above example, this can be seen at two places: the two translations "extend" and "stretch out" are taken as synonymous, and for the meaning "to be long", both spelling are supposed to be equivalent.

When some important information (transliteration, spelling, for example) is not available in a group, it is supposed to be *inherited* from its parent group. For example, the adjective-verb meaning *be long*, which has the

same transliteration as the head word, but different determinatives, inherits its transliteration, but changes the translation and the spelling.

Then comes the expression “*lb=f 3w*”. Expressions and composite words are a tricky problem, whose representation might need some improvement. At the time being, we have a number of tags that can be used in expressions to represent the currently defined word, an animate, or an inanimate. The other words might be free text, or explicitly transliterations of words in which cases they are indexed. (For instance, in the current case, a search for the word “*lb*” will retrieve the current definition).



Fig. 1 Dictionary Entry

References

In theory, it should be possible to link every occurrence of a word to its meaning in the dictionary. In practice, it would be interesting, but with two caveats. First, this linking would be an interpretative work. Hence, the need to allow more than one linking for a given occurrence. Conversely, the dictionary itself is an open work, and new entries can correspond to words which occur in previously analysed texts.

In fact, a real dictionary needs *selected* examples (while exhaustive search should remain a possibility). The *Tksesh* reference system is used for this. References are hypertext links to the text database. An important feature is that references are given in Egyptologist readable format. That is, it's possible in *Tksesh* to refer to a text passage. One of our projects is to allow more than one reference system for a text, when a text has multiple designations.

The Note Editor

As the dictionary editor is a structured text editor, we decided to use the same system for more general texts. This editor allows one to type short notes, with text references, hieroglyphs, etc. In the future, these notes should be searchable.

The Translation Editor

This part of *Tksesh* will be the central working area of the system. The current version is a bit crude: one can type a text's transliteration and translation. It's a reasonable system if one works on only one version of a text.

However, for texts which have variants, this is not sophisticated enough. Thus, the next version will be more complex.

Full Text Search Facilities

A full-text database function is meant to be searched. Searchable data include translations, transliterations, and of course the hieroglyphic texts. One can use any of these to search for a word. The hieroglyphic search is rather robust. It can cope with words which span more than one line, it knows about similar signs (like the rope and the quail chick for *w*), etc. As far as hieroglyphic texts are concerned, we intend to add improvements when time allows. In particular, we should be able to use the dictionary, which often contains variant spellings for one word. This is another example of the benefits of linking the dictionary to the rest of the system.

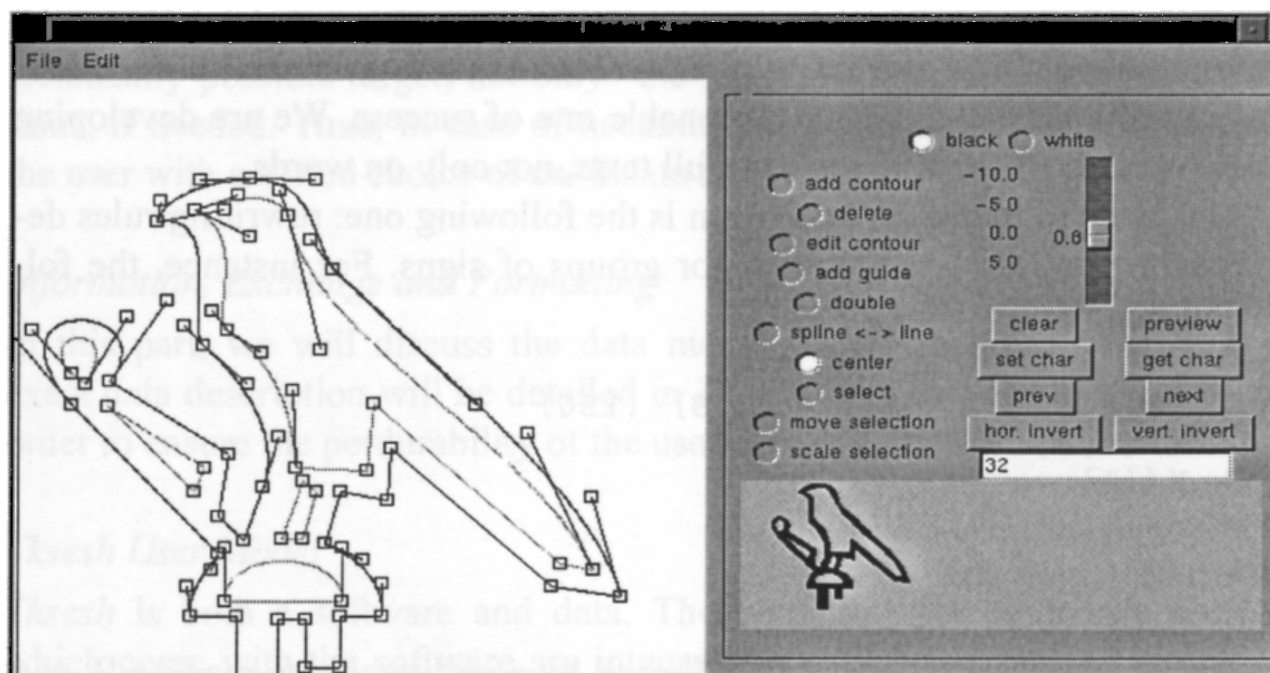


Fig. 2 Font Editor

The Font Editor

The current version of *TkseSh* includes a font editor, which allows one to create new signs or edit old ones.

TkseSh as a Toolkit

TkseSh is available in source form, and, even better, most of it is in *TCL*, which is an interpreted language. It means that it's relatively easy, and free, to use parts of *TkseSh* to build other programs. The facilities provided by the package include a simple database, and of course a hieroglyphic editor/display. *TkseSh* is also multiplatform, which means it could be used, for instance, to write a CD ROM interface.

TkseSh and Automated Transliteration

The automated transliteration system in *TkseSh* is a rule-based system (which is currently undergoing major improvements). The basic ideas for the system were already described in a previous article.²

² S. ROSMORDUC, *Traitement automatique du langage naturel en moyen égyptien*, in: R. VERGNIEUX (ed.), *Xième conférence Informatique et Égyptologie*, Bordeaux 1996, 97-103.

Ms F. KERBOUL³ developed a working model and a rule database which were promising. The current version, written in Prolog, is able to transliterate hieroglyphic words with a reasonable rate of success. We are developing a new version which will work on full texts, not only on words.

The basic principle of the system is the following one: rewriting rules describe the possible interpretation for groups of signs. For instance, the following three rules:

1. $X/[A, B], Y/[B] \rightarrow [A, B] \quad (150)$
2. $X/[A, B] \rightarrow [A, B] \quad (400)$
3. $X/[A] \rightarrow [A] \quad (380)$

state respectively that:

- a bilateral sign reading AB, followed by a uniliteral sign reading B, can be read AB,
- a bilateral sign reading AB can be read AB,
- a uniliteral sign reading A can be read A.

Of course, these rules are contradictory. If you find the chain $mn:n$, you can read it either by using rule 1, in which case you get mn , or by using rules 2 and 3, in which case you get mnn . The solution we choose here is to give each rule a cost (here 150, 400, and 380). The preferred solution is the one with the least cost. For instance, here, we will prefer mn (cost 150) to mnn (cost 780).

Yet, the rules has exceptions. For instance, $lr:r$ should be read lrr . In this case, it's sufficient to write a new rule:

4. $"lr"/[i, r], "r"/[r] \rightarrow [i, r, r] \quad (100)$

To ease determinatives analysis, word limits are explicitly represented. Thus, it's possible to write rules like:

5. $det, end \rightarrow [] \quad (500).$

which states that a sign which can be a determinative should be interpreted as one at word end.

³ F. KERBOUL, *Transliteration automatique des hiéroglyphes*. Rapport de stage de l'ENSTA, Paris 1997.

The system might of course make mistakes. A nice feature is that it's technically possible to get, not only "the" best solution, but the n best solutions, if needed. Thus, in case of necessity, it would be possible to present the user with a sorted choice of candidate answers.

Information, Exchange and Formatting

In this part, we will discuss the data model which underlies *Tksesh*. The exact data description will be detailed in *Tksesh* technical documentation, in order to ensure the perdurability of the user data.

Tksesh User Model

Tksesh is both a software and data. The texts and the dictionary entries which come with the software are integral part of *Tksesh*. One of the ideas we had when creating the software was that users could contribute data without too much effort. The most needed kind of data is of course hieroglyphic texts. But the possibility exists to share dictionary entries. Ultimately, it will be possible to distribute any kind of data entered into the base. Of course, distribution of data will be a user choice.

This kind of cooperative development leads to a number of problems. First, the authors of any contribution should be identified, just as in paper publications. Second, the existence of multiple contributors should not lead to data inconsistency in the base. This might happen, for example, if two users created the same entry, or if one user destroyed an entry used by another.

The two problems have solutions. First, any data submitted by a user is identified by the user key, which ensures both user identification *and* the impossibility for two users to create the same entry. Second, data, once submitted to the main database, can't be changed, *even by its author*. This is a bit annoying, but it's the only logical way to go. If data which has been distributed could be modified, inconsistencies could appear. Let's suppose user A has submitted an entry in the dictionary, and that this entry has then been referred to by user B. Now, if A could change his entry, B's reference could become false. This kind of problem is of course not unheard of in the database world, but the chosen solution seems to be the lesser evil.

Of course, if user A is unhappy about his previous entry, he can still propose a new entry on the same subject, as he would have done for a paper. Note that this rule only stands as far as structure is concerned. Very small changes which won't modify other entries might be done (for instance, spelling corrections).

Data Model

What Should there be in a Hieroglyphic Database ?

Most of the texts we entered in the database have a hieratic original. For these, the current encoding is fine. Truly, one loses a number of information, but a computer text is not a *facsimile*. However, for stelas, and illustrated documents in general, much information is lost. One would want, for example, to link utterances to the associated figure. The point is that a minimal encoding of iconography and of relations between iconography and text would be valuable. One also needs to add comments about the hieroglyphs themselves, in particular when the reading is dubious.

Dictionary Structure

The current dictionary structure has a nice property: it will be easy to translate into XML. However, we need to change it a little. By now, only a complete entry can be identified by a reference. But one sometimes needs to refer to one particular characteristic of a word: a precise meaning, a written form, etc. Thus, in the next version of *Tksesh*, everything will have an identity. In particular, every group, and anything in a group will have a unique ID.

Current Projects for Tksesh

Tksesh will be subject to a major rewrite in the near future. The current version is functional, but should be considered more as a model of things to come. The first “real” version will be numbered 1.0. The main changes will be that the data will be a lot more structured. In particular, it will be possible to reference explicitly most data in the base, in a sounder way. To give but one example, it is currently only possible to refer to a dictionary entry as a whole. The next version will allow to reference sub-definitions.

Another point for *Tksesh* 1.0 will be the use of XML as a representation and exchange language. Of course, translators from the previous data file formats will be provided.

Parallel Editor

A typical philological work has a rather complex structure, much more complicated than the simple text/transliteration/translation trilogy which is supported by the current translation editor.

We had first tried to use the hieroglyphic editor facilities, and to encode all parallel variants in the same hieroglyphic file. This is not a good solution: it isn't easy to add texts and it isn't clean, as it mixes more than one level of interpretation. More, it doesn't allow to distinguish between minor

variations and important ones. Plus, a philological work is of limited interest without its associated notes and comments.

So, we started designing a editor which would support multiple source texts. Text editions use two ways to mark variants. Some, like for instance the *Lesestücke*, indicate variants in footnotes, using either an « ideal » text or one of the sources as basis. Others give a synoptic edition of all texts (for instance DE BUCK's *Coffin Texts*).

The first solution is fine for noting small variants, mainly at sign level. It doesn't scale up well, and depends on the choice of the base text. The second solution is more expensive as parts of the texts are reduplicated. However, it fits better in the context of a text database, as all texts are considered equals. Another advantage is that its easier, from both a computer and ergonomic point of view to consider large variants than small ones. So, we will segment our texts in sentences, verses, or short paragraphs.

The next problem is to decide what should go in a segment, and how segments should be organised. We have chosen to allow one to associate more than one translation to a given hieroglyphic text, more than one hieroglyphic text to one translation, and so on. To distinguish between important and secondary variants, we have used the same idea as in the dictionary: grouping.

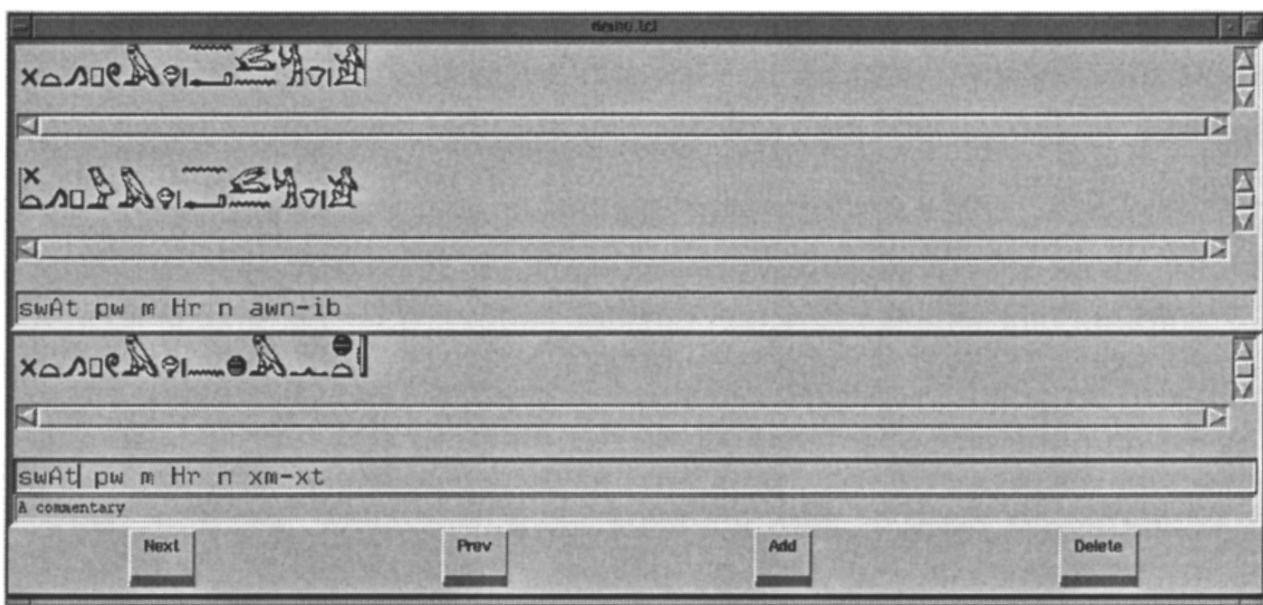


Fig. 3 Variant Editor

In figure 3, we show three variants of a passage. The first two differ by a small graphical detail, but the third contains a significant change, the word *'wn-ib* being replaced by *hm-ht*. So the first two variants are in one group, and the third in the other. The upper transliteration applies to the two texts from the first group, the lower transliteration and the comment to the last

text. Grouping can also be nested. The idea is that if a group contains two data of the same kind, they are more or less considered to be equivalent. If two data of the same kind are in different groups, they are considered significantly different. For instance, if we put two translations in the same group, they are supposed to be different *equivalent* wordings (perhaps in different languages). If one hesitates between two disagreeing translations, one should put them in different groups.

Another question is whether or not we should allow concurrent segmentations. This would be logical, but we have decided not to do it, for the sake of simplicity and ergonomics. So if one thinks of more than one possible segmentation for a text, one should either make segments big enough to encompass the whole problematic area, or write more than one analysis.

An important feature of this system is that hieroglyphic texts are not physically included in the document. Instead, we store references. The texts are simply kept in the text database.

The system should also allow one to add notes, about the whole segment or a small part of it, comments, lemmatisation, etc.

Conclusion

We hope this presentation will raise an interest in the software, and perhaps, who knows, will encourage people to contribute data to it. We hope to make it really productive very soon.

Appendix

TCL/TK

TCL/TK is a programming language developed by JOHN OUSTERHOUT at Cambridge University (USA) and then in the SUN research department. It is a simple, powerful, cross-platform language, specially designed to be embedable in programs written in compiled languages like C or Pascal. *Tksesh* is based on a number of extensions, written in C, to TCL/TK. It runs under UNIX and Windows 95. Due to the flexible nature of TCL, it is possible to use *Tksesh* to write little applications that would need to display hieroglyphs.

Availability

You can download *Tksesh* on <http://www.iut.univ-paris8.fr/~rosmord/TKSESH>. The system is available free of financial charges, as “textware”: if the system is of some use to you, please contribute some texts. It would be definitely better to ask which texts are needed before sending them.