A Language Research Workbench Software Architecture

47 Rooks

December 2015 revised December 2016

Introduction

I have used a number of high quality bible study software programs, Accordance, Logos, MySword and Olive Tree among them. And while they are very good there remain limitations that it would be nice to see removed. These shortcomings are very often the result not of a lack of willingness to provide support but a matter of priorities. Some suggestions while critically useful for one small group of users may simply be uneconomical for a commercial concern to implement given other requests for features.

This document attempts to organize ideas for an architecture for language study software that is extensible and open.. Whether it might ever be built in part or in full remains for the future to decide, but this preliminary investigation seeks to determine what might be desirable and possible.

As the title of this document might suggest I believe bible study software is too narrowly focussed. Many of the features in such software are equally applicable to textual and linguistic study of other texts, religious and secular.

For people wanting essentially a turbocharged e-reader current commerical bible software is very very good. It provides many great features for assistance in reading in multiple languages, looking up of lexica and research queries against many texts, reference books and commentaries. On the other hand it can be very difficult to import new data into these tools. When that is done the data is a sort of second-class citizen, supporting only a subset of the functionality available for other data. This very much inhibits research on such works.

Finally, beyond the mere integration of new data, the user also cannot extend function of the basic tool in useful but unthoughtof ways. New types of data cannot be introduced and searched, for example. All such requests must go back the software company and compete with other requests.

In contrast this architecture seeks to make all data equal under the tool, and permit the addition of new data, new data types and new functionality to the environment.

Basic Principles

The following things are simply considered essential items and are really stated here only so as to be clear and so that they might not be forgotten in later considerations.

> Unicode. There is no point in developing anything new in any other encoding at this time.
>
> Multi-platform. Whatever software is developed it ought to be possible to port it to a new platform with ease. Depending upon the platform, for example a cell phone, some features may not make sense to port but it should not be the case that the majority or even a large minority of

the code must be rewritten for the new platform.

User content. Users must be able to generate all content supported by the tool. No exceptions if at all possible.

Documented file formats. Every file format must be fully documented and where possible be open like XML or TEI or such like.

Extensible via a plugin model. It must be convenient, or at least possible, to add new functionality. It must be possible for users to develop plugins to support new content types. This allow the LRW to support new forms of research rather than merely supporting new modules conforming to a canned set of types.

Open source. The entire core and total plugin structure ought to be open source.

Primary Concepts

The Basic Model for Text

Essentially, all text should be treated equally. This means that if one can make notes on the biblical text, one can annotate the notes. If you can search the biblical text you can search the notes and annotations. Arbitrary distinctions between notes and biblical texts or between commentaries on works and the works themselves should be removed entirely or if that proves impossible they must be minimized.

In theory this also means that if a syntax model is developed for the Hebrew text of Genesis one might equally well be written for a commentary on Job written in English, or German or what have you. And if such a thing were to be done it should then be searchable in the same ways.

Layers

How then should text be viewed ? In layers. The simplest explanation is probably to take the biblical text but remember that this approach applies to all text. In fact it probably applies to images too.

If one imagines the biblical text as a simple stream of characters without any form of mark up, no chapters, no verse numbering, no morphology, no syntax structures, nothing, just the text, then we have the base layer. Over this then may be layered other information. The most basic these days would be the chapter and verse numbering. A layer of the chapter and verse information would be created separately from the text but anchored to it in places, those places being where the chapters and verses begin and end.[1]

Another layer may be added containing morphological tagging again anchored to the underlying text, this time on a morpheme by morpheme or word by word basis. And then perhaps other layers with different morphological models, or syntax or discourse tagging.

---

1 An immediately obvious use for this is to define a new layer of versification taking the default one supplied with the text module as a starting point. This would permit investigation of alternative versification which is sometimes suggested by a careful reading of the text.

A Layered Model for Searching

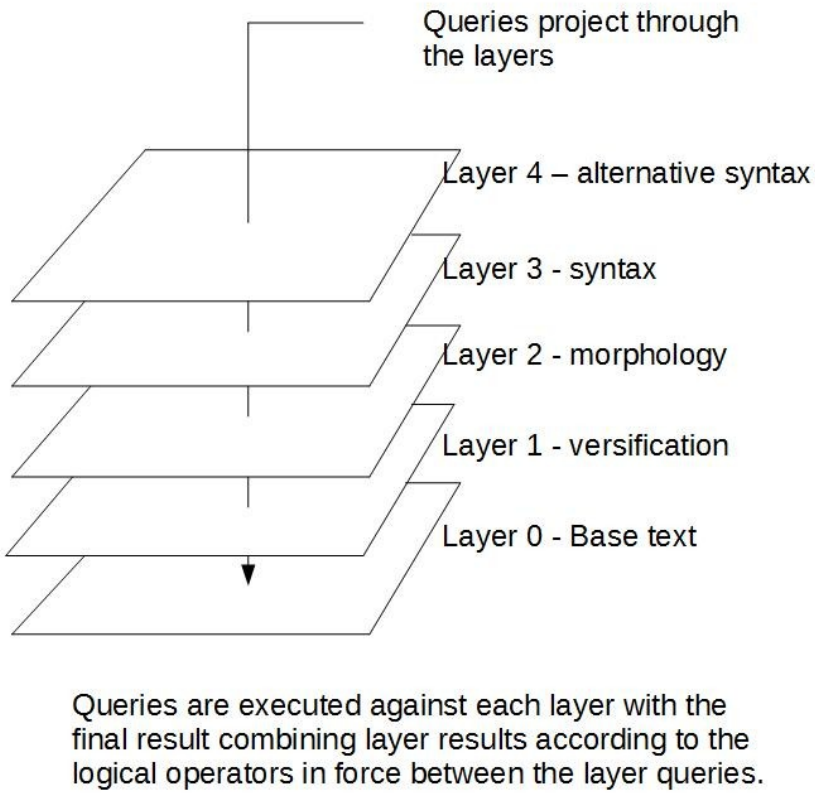

Illustration 1: A layered model for search

Searching in a layered model is done by search for characteristics in each layer and then requiring (AND logic) that they coincide at the same point or optionally coincide (OR logic), naturally with support for negation (NOT).

In addition there needs to be a way to specify search scope. This concept stems from searching for words or sequences of words appearing in a verse, or a paragraph, or a chapter and so on. In the layer model this is a little strange to visualize but as an example a predicate on the verse layer acts as a bucketing constraint on the other query predicates. All layers participating in the query must satisfy their predicates in the same bucket for a hit to be found. Thus one might issue a query saying that one wishes to find αγαπη and a SUBJECT in the same verse. This would query two layers, the words in the base layer, and the syntax layer for the SUBJECT, both constrained by the verse buckets derived from the versification layer. Of course, one might quite as easily use successive ranges of ten words length as

the bucketing criterion, or some other measure as might suit the purpose. Query support must also exist for saying that for example, a SUBJECT must be θεος. The specification of such a query language goes beyond the reach of the present document.

A Display Model for Layers

It is not necessary that all layers of data be displayed nor is it necessary that all displayed layers be displayed on top of one another. Some information should appear perhaps as popups, or mouseovers, or perhaps in parallel panes. In addition not all layer data is of the same type. Some information is graphical and must be displayed as a diagram, some is morphological data and might be displayed in a small table. Chapter and verse information though would likely be interleaved in the returned text. Some information might be displayed in multiple ways dependent upon user preference.

Layers do not have to be displayed in the output to be searched.

Search Results and Derivative Data

When searches are done various types of output may be produced. For example you may have a subset of the verses of a text, or perhaps a list of words from the text, or perhaps a count of words having a particular property. Some bible programs treat such result sets so differently from the original text that it is not possible to then take the results from one query and run another query on that data. This limits the flexibility and leads to the necessity to post-process the result set in another tool in order to obtain meaningful results for the particular research task. LRW by contrast treats the search results as a text type that may be processed again as though it were simply another module in the tool.

Queries produce data derived from the original, for example the verses in John containing the word “θεος”. Now this derived data set may be then be queried in its own right. But a user would likely find it useful to have full access to the same layers in the derivative as were available in the original, though restricted to the derived data's subset of the original text. This will be supported via a form of inheritence of the layers. Such inheritence may in some cases be performed by producing a derivative of all relevant layers.

Statistical Support

Queries against the layers may produce output of subsets of the queried layers or may produce statistics about the data, such as counts of words or averages and so on. Statistical output while different in terms of the data should be treated, in a general sense, no differently from the text data. It should be queriable like any regular data set or layer. The specific problem of this layer is that it is essentially tabular and mathematical functions may be usefully applied to some of the content. Thus this data has more than one appearance. Support for tabular behaviours will be accomplished by a plugin supporting this kind of data.

A Plugin Model

Associated with each type of layer will be a LRW plugin that knows how to create, search, output and

display it. Thus LRW is actually a core workbench program with a variety of plugins for processing the data. In keeping with the model of users being able to produce plugins and handle new kinds of data, even the original LRW layers would be supported by plugins. Done in this way no particular data type would be more primitive or built in than any other.

Module Creation Tools

All module creation tools used to produce the modules included in the distribution will also be provided to users. There will be no artificial restrictions leaving a user unable to create a module with features that exist in distributed modules.

What is a Document Type

I am not anxious to type documents but it seems unavoidable and after all people having been treating books, papers, essays, statistics, tables, poetry, speeches and countless other written forms as distinct. So here comes LRW's take on document typing. The principle objection to typing documents is that document type tends to drive what the software may do with a document. Unfortunately it may be the case that a user might want to perform operations on a document that are prohibited by its type. The document type should not limit operations that may be applied. Of course this cannot be one hundred percent the case as the type will also indicate what operations might apply. Thus the first principle of LRW is that, within reason, operations should be applicable to as broad a set of document types as possible.

Of course part of the problem of typing a document is that a document is not homogenous, but contains material appropriate to various types of operations. Thus the approach should likely be that one should permit operations on selections of a document that make sense. Thus if one selects a table in the midst of several paragraphs of text then one ought to be able to apply mathematical operations to the column data. Perhaps the way to think of this is that the selection creates a derivative of a different type to which different operations may be applied. This leads to an interesting thought; selection acts as a visual query. This is worth considering very carefully.

Are images documents ? Of course. What searching can one do with an image ? If robust optical character recognition (OCR) is available one could use visual selection and submit the selection to OCR. The output could be labelled as a derivative document and then subjected to subsequent operations.

Note also that document type is a multi-faceted.

> At the lowest level is the binary encoding, ASCII, Unicode and so on.
>
> Next JPEG, PNG, XML, TEI, EPUB, DOC, PDF and so on.
>
> Above that would be text, images, tabular data

At these different levels the data might be subjected to a variety of different processing algorithms.

Document Metadata

Each document will have metadata which includes information concerning authorship, relationships to other documents, and so on. This will be largely like that used in such tools as Zotero, Mendeley, DevonThink and the like. Metadata must support user defined metadata fields. Adding new fields must be supported and an appropriate handling must exist where fields are not present on a document.

Secondary Concepts

A number of secondary concerns or things not to be forgotten are noted here.

The Library

The library in LRW is the repository of research material. It will contain documents shipped with or downloaded to LRW. It will also contain any documents promoted to the library by the user. It should also be able to contain references to internet based resources. It may contain references to works not actually present in the library. These may contain a precis which may be searched to give an indication of relevance of the work to the task at hand.

Users familiar with research tools such as Zotero or DevonThink will see the similarity.

Library organization will be permitted on a number of dimensions. It should be possible to create views over the library according to various systems of organization. It should also be possible to create views of the library which are subsets of the works. Document metadata will be very important here as a way to view or subset the library.

Library Document Modification and Locking

One problem associated with permitting widespread modification of data is the possibility that a user may corrupt the basic data set leaving LRW, or a subset of it, in an unusable state. In order to protect against this, data sets and layers should be lockable to prevent modification. Generally speaking all layers and data sets shipped out with the product would be locked. In essence one might think of the library contents as rather like that of a regular brick and mortar library; they can be read, borrowed, notes may be taken and references made but they may not be written on, torn up, lost or stolen.

They may be unlocked for modification of course, which might be useful for corrections, but the preferred method of correcting text errors would be this; submit a correction to LRW and then create a layer as a copy of the faulty layer and make the correction there. Then until the corrected module is available from LRW the user may use their own corrected copy. When the corrected module is available install it and remove the temporary substitute. For this to work gracefully the same issues which must be faced for library document updates, discussed below, must be face here also. In fact I suspect this is solved by treating the private user corrected module as a module update, albeit a temporary one. For this to be relatively seamless it will require a way to substitute one module for another. Without this any user notes and query results would be invalidated by the change.

Updates to Library or Base Documents

Of course one must provide for changes to base document rather giving the lie to the library analogy; for while in a real library books are not updated but new editions collected, in a computer library updates are possible and in some cases justified.

However there are difficulties. When changes are made, all documents which annotate or relate to this document must be updated to ensure that they still related properly to the corrected text. The way this would work in LRW is to do a “diff” between the new and the existing version as is done by software source code control systems. As the new changes are examined their effect on annotations can be determined and the annotation references updated. This is a non-trival task but should be very possible. In order to do it though there must be a record of all layers which have references to the document being changed and whether they wish to be updated when the base document is revised. In addition it would be most helpful to have a way to avoid merge conflicts.

It should be noted though that in some cases it will be necessary to ask for user input on what to do in certain cases. This could potentially lead to very lengthy updates and notes long forgotten might be brought up for update which would only cause frustration and confusion. Suffice it to say that this support needs very serious and careful design and implementation.

Real Library Mode

There may be a use for a “real library” mode whereby a module update is treated as a new edition. The user may want to be able to decide how such updated modules are installed, either as updates or as new editions. Such support lies very much in the future for LRW, but this possibility is noted here to see if it garners any real interest.

Versioning

Coupled with issue of applying updates to files is the notion that files have versions. If for example one runs a query against a document, one produces a selection of verses as a new derived document. The provenance of this new derivative must be recorded along with the version information of the original file. Now if the original base document is updated it can be determined whether any changes to the base document affect the derivative. If they do the document may or may not then be updated depending upon the user's preference. It may be that the user wants to keep the first derivative document and then produce another from the revised original and compare them. Alternatively the user may wish to update the derivative immediately. This process can become arbitrarily complex if for example the user has modified the derivative in some way. It may be necessary to prevent updates to such a docment when the base document is revised. It is possible that one solution is that documents derived from immutable documents will themselves be marked as immutable so that while they may be annotated they may not be directly edited. If a user decides that they wish to edit the document it will then no longer be able to be automatically revised when the original base document is revised.

A Few Plugins to Begin With

Here are a few rough draft ideas for initial plugins that might be useful.

IO Plugins

Converting modules to the native format of a given bible software program is a perennial problem. It seems better to combine a readonly format with a read plugin for any file types of interest. Plugins for file types will expose the content of the file stripping out the metadata and presenting the data to the LRW APIs. An initial shortlist of formats of interest would include EPUB, TEI (particularly the variant used by the Codex Sinaiticus project), HTML formats such as that used by tanakh.us, and of course plain text.

Diagramming Plugins

Language analysis involves considerable numbers of different diagramming systems. These range from grammar diagramming, phrase structure trees, syntax diagrams, mind maps and so on. One or perhaps a number of diagramming plugins would be useful. In addition to ones conforming to one of the known diagram systems a simple tool which is less restricted in its approach yet would be useful.

Morphological Plugins

Just as there are variant diagramming systems there are various morphological systems. It is necessary for the research community to be able to add morphology to texts according to whatever method they might be researching. Thus a plugin capable of implementing a new morphological tagging and displaying any existing tagging would be very helpful. Any such plugin must produce layers which would be searchable by the standard LRW search infrastructure. They must also provide display overlay support.

Module Creation Plugins

Strictly I am not sure that this is a separate category really. Some plugins will create modules simply because that is what they do.

Optical Character Recognition

A complete solution to OCR for biblical and allied languages does not currently exist, at least not one that is relatively cheap and efficient. It would be possible to build a module, of a fair size to be sure, that could do this in LRW.