Mapping of the LRW Architecture onto Eclipse

47 Rooks

December 2015 revised February 2017

Introduction

It is assumed that readers of this document are familiar with the content of BibleStudySoftwareArchitecture.odt, hereafter [BSSA]. [BSSA] describes a fully plugable and extensible architecture for language study software. On reflection I realized that the Eclipse IDE (Integrated Development Environment) used for software development tasks already provides a full and robust plugin architecture. This document therefore examines implementing the [BSSA] architecture through the Eclipse plugin architecture. Readers should also be familiar with [PPDG] and it would be helpful to have some acquaintance with [PDE].

Background

Eclipse (www.eclipse.org) is primarily an Integrated Development Environment (IDE) for software development. It is opensource and free for most use, even for the creation of dependent products. It is designed from the start to permit wide-ranging and deep-going extension. It is written in Java making it multi-platform. This also means that it is conversant in Unicode. Thus the key elements of [BSSA]'s Basic Principles are already supported. Additionally it is full of editor support as that is a primary function in software development. Such editing support is an absolute necessity in software development as it is in writing research notes, experimenting with textual models and so on. Thus on a fairly superficial level it would seem that a properly configured Eclipse workbench would provide a very viable implementation of [BSSA].

We will now examine this possibility further.

Feature Comparison

A full explanation of each feature of the Eclipse plugin model is as beyond the scope of this paper as it is redundant. The reader should refer to [PPDG] for details on any features. Links provided are to the MARS release which is the latest available at the time of this writing.

Readers unfamiliar with Eclipse in use ought to familiarise themselves with Eclipse Workbench User Guide - Concepts which describes the fundamental concepts.

Workbench

Eclipse when first booted presents the user with a workbench window, unless new plugins have just been added in which case a welcome page providing access to help, tutorials and such appears. More on that later, but first the workbench. The workbench shows a number of divisions. On the left generally there will be a Navigator pane showing the resources in the *workspace*. In the centre is a large editor pane open to edit whatever file is currently being worked on. Below that pane is another with informational views in it. To the right is another pane with further views, these being things such as outliners and the like. All these panes support multiple tabs devoted to different *views* as Eclipse refers to them. All can be resized and opened or closed or reorganised at will. The basic layout of the workbench is governed by the *perspective* in effect.

This sort of layout arrangement is common to various desktop bible study tools, notably Accordance and Logos. It thus provides a good match for the task of working with text.

Workspace

Accordance users should not think of an Eclipse workspace as anything like an Accordance workspace.

An Eclipse workspace is the store of all the resources (loosely, source files and compiled output objects and metadata in a software development project). When Eclipse boots one workspace is chosen and it is used until the user switches workspace, which is generally a very rare occurrence. Within a workspace projects are created. Each is usually aimed at the development of a particular piece of software. Projects may depend upon other projects. The appearance of the workbench may change in switching from project to project if the new project's *nature* is different from the preceding one but otherwise not much will change. In fact resources from multiple projects may be opened at once in distinct tabs.

Current thinking on the workbench is this. It will serve as a fine way to work on a variety of research projects. A project might be created to examine the book of Jonah for example, another to study Greek in the Pauline letters, and yet another for working on a lexicon derived from Akkadian texts.

Projects

When one begins a new software project in Eclipse one creates a directory structure under the workspace which will contain all the files created. It will contain source files (anything from C, Java, HTML, XML and many many others), compilation output and metadata files and all manner of software objects that you require. And a project may depend upon other projects, perhaps to obtain software libraries from them.

When used by LRW, there would be a single fundamental project which would serve the purpose of a library of documents and texts. In all likelihood these items would be read only to prevent any data corruption by user activity. All other projects created by users would have at least this one as a dependency. This would give them access to various items they might need, like texts and lexica. Within the project itself would reside the user's own creations, which at time of writing would likely include, search results, notes, essays and so on, lexicons they might be creating, morphological tagging databases and so on. Anything that one might reasonably want to create.

Now if the user is creating something that might become a permanent reference they may wish to push it to the library project. This concept does not really exist in Eclipse fundamentally but could be added. In this way user generated content could be added to the library with essentially the same status as anything shipped with the product.

Perspectives

A perspective as Eclipse describes it does not really have a direct analogue in tools such as Accordance or Logos. Workspaces in Accordance and Layouts in Logos though end up providing the same sort of function. A perspective determines which views will be opened in the workbench by default for a particular kind of work and how they are laid out. So if one had a Text Study perspective it might contain a Navigator on the left, a central pane with an original text in the top perhaps with a translation next to it, and below a notes pane. On the right might be useful lexica. If instead one were developing a lexicon one might have a central lexicon editor with source texts in a pane beside or below it, probably with multiple tabs in it. User notes might be to the right and the resource navigator would remain on the left as usual.

If the user alters the layout in any way this is remembered the next time the tool starts, but there isn't really knowledge of this on a project by project basis. This may be somewhat involved to add. It is possible that project-specific perspectives could be created but that seems a heavy weight solution. It may not be but it would need to be investigated.

Views

Views are the smallest unit of display in the workbench. A view might be an editor, a status display of some kind, a table and many other things besides. They may be in tabs within panes and a suite of views constitutes a perspective. However you may open any view you see fit and move it whereever you like.

Resources

Resources are the files that contain all the source and supporting files of the software project. They are usually though not always within the workspace. In LRW it would seem likely that they would always be within the workspace. It is imaginable to put readonly library resources outside the workspace. At present however there does not seem to be a strong reason for doing that.

Resources have attributes which indicate the type among other things. This helps determine what editors should be used when working with the data. In the LRW case the this will control what IO plugins will be required to handle the data. This should permit the handling of EPUB, txt, HTML, TEI and whatever other file types LRW plugins support. This would include chart types as well, such as might support syntax charts of various kinds.

Finally, it should be noted that Eclipse supports multiple workspaces, not for concurrent use, but separating types of projects. Most of the time all projects are in the same workspace. However, it is possible that leveraging multiple workspaces might provide useful separation for some use cases.

Navigator

The Navigator view shows the user their projects and the resources in them. It does not have a concept of a library of read-only resources that might be common to all projects except as a dependent project or as external libraries. As suggested earlier a dependent project would likely serve as a document library. The navigator does support searching and double click opening in an appropriate editor.

Viewers and Editors

Eclipse defines the concept of a viewer but it is not restricted to merely viewing. A viewer might also be an editor. As envisaged currently plugins for LRW will be built to support any number of file types for reading, writing, searching and annotating. But pretty solid Unicode editors already exist in Eclipse though investigation of their exact capabilities would need to be conducted.

Comparison Support

Within the Team support in Eclipse there is considerable support for diffing files. This is a common thing in bible study software also. Conveniently there is also solid support for Git and Github which may well be helpful in publication of new resources.

Search

Search support that is very general and simple, such as search across all files with a regex is already possible. In additions searches by file type may be added. The Java Development Tools product does this so that search may be made sensitive to the language constructs of Java. Thus LRW should be able to provide support for morphological and syntactic searches.

Working Sets

A working set is a collection of files. These may be defined by the user and would likely provide benefit to LRW in allowing the user to define scopes of searches for particular investigations. A number of basic scopes would likely be worth adding like, lexicons, texts and so on. By way of comparison Working Sets could provide a good analog for the 'library group' concept in Accordance.

Plugins, Features and Products

Eclipse defines three basic component concepts. Plugins are the smallest unit of function that can be added to or removed from an Eclipse installation. One would write a plugin devoted to the handling of a particular file type for example, like EPUB. Multiple plugins constitute a Feature. A feature collects together multiple plugins that work on related function. One might imagine an Optical Character Recognition (OCR) feature which collects together various plugins associated with scanning documents, running OCR on them, and then cleaning them up, finally outputting them as resource files. Finally a product is a configuration of features that boots as the initial workbench configuration. In our example here LRW would be the product.

Help

There is extensive support for the user assistance in Eclipse plugins. This includes documentation, tutorials, cheat sheets, error messages and diagnostic logging. All of these will be leveraged by LRW plugins of one kind or another. The most significant thing to mention just here is that the entire documentation can be done as its own plugin. In something as large and complex as LRW, it is likely this will make sense.

Plugins for Plugins

Plugins themselves can provide extension points for further plugins. This feature might well be needed for certain functions that could be layered or which users might wish to layer on top of the LRW plugins.

Evaluation

Eclipse does provide a very extensible structure but it is not completely compatible with some of the features of major bible software programs. The key items for which investigation is required are the library and project visual layout preservation. Beyond these much is simply a matter of implementation. The user interactions will be different as befits a different tool, but the capabilities will be very similar or better.

References

BSSA A Language Research Workbench Software Architecture, 47 Rooks 2015

PPDG Eclipse Platform Plugin Developers Guide

PDE Eclipse Plugin Development Environment Guide

47 (2017)