ECLIPSE-BASED PHYSICIST WORK ENVIRONMENT

W.T.L.P. Lavrijsen*, S. Binet†, NERSC/LBNL, CA 94720, Berkeley, USA

Abstract

Eclipse is a popular, open source, development platform and application framework. It provides extensible tools and frameworks that span the complete software development life cycle. Plugins exist for all the major parts that today make up the physicist software toolkit in Atlas: programming environments/editors for C++ and python, browsers for cvs and svn, networking with ssh and sftp, etc. It is therefore a natural choice for a physicist work environment. This paper shows how the Atlas software environment and framework can be configured, debugged, build, and run with eclipse. It also presents plugins tailored for Atlas, which ease installation of the software, development and debugging of job configurations, and (interactive) analysis algorithms in a multi-language environment. Plugins for integrated tutorials and context-sensitive help are also provided, allowing people new to the Atlas software to quickly get started with their analyses.

ECLIPSE OVERVIEW

Eclipse is an open source community whose projects are focused on providing a vendor-neutral open development platform and application frameworks for building software. Eclipse is supported by the Eclipse Foundation: a not-for-profit corporation formed to advance the creation, evolution, promotion, and support of the Eclipse Platform and to cultivate both an open source community and an ecosystem of complementary products, capabilities, and services. [1]

Introduction

The best known face of eclipse, is its advanced, highly functional, graphical Integrated Development Environment (IDE). An example of a python [2] coding session is shown in Fig. 2. In this case, the IDE workspace is customized using the PyDev [3] perspective. A perspective, in eclipse, is a customization of the workspace, consisting of plugins and their layout, for a specific kind of development. There are many perspectives, several of them are part of the standard eclipse distribution (e.g. Java, Java Debugging, C/C++ Development, Code Sharing, etc.). PyDev provides python indexing, syntax coloring, on-the-fly code checking with PyLint [4], code completion, and display of documentation strings; as well as the normal package and file outline displays. The code in this example is cvs managed by means of the cvs plugin, and an interactive interpreter is available as an external tool (hence it can be either python or jython [5]). As (free) python IDEs go, eclipse is by far the best currently available.

There is, however, much more to eclipse than just an IDE. The Eclipse Foundation is supported by a variety of big-name industrial players: IBM, Intel, Borland, CA, BEA, Sybase, Nokia, Wind River, HP, and SAP. A couple of immediate things to note: this is not the “not-Microsoft” crowd, SUN is missing, and the industries represented by this group span the whole IT spectrum. What brings these companies together, is that eclipse is a platform for tools, the IDE being just one of them, and each company can provide extensions that support their business to the platform.

The abstractions provided by developing on a platform, as well as the ability of re-using all of the eclipse middleware, has the same effect on productivity for tool developers, as the availability of an operating system has on application developers. It is the middleware that makes it possible to turn eclipse in an effective environment, tailored and specific to Atlas users, and therefore should be explained here in some more detail.

Middleware

The more important parts that make up the eclipse middleware are illustrated in Fig. 1, and listed below. Developers can create extensions either as standalone applications, with their own interface (graphical and/or command line), packaging, and branding; or as tools that hook into the IDE or any other application that is extensible. The main distinguishable parts of the eclipse platform are:

* WLavrijsen@lbl.gov
† SBinet@lbl.gov

Figure 1: High-level view of the main eclipse elements and the place of the IDE relative to it. The grouping is somewhat arbitrary; arrows indicate use dependencies.
### OSGi
An implementation of the component model specification as designed by the Open Services Gateway initiative (version 4). [6]

### RunTime
Plugin support classes and plugin management. Historically, this contained the full eclipse component model, before the move to the OSGi implementation.

### SWT
The Standard Widget Toolkit: a thin, low-level, abstraction of widgets as provided by the local windowing system.

### JFace
Higher level GUI elements (dialogs, viewers, wizards) that are created with the low level widgets from the SWT.

### UI
Behavioral elements of the GUI, such as the workbench (which makes GUI elements cooperate with each other) and editors.

The IDE, then, as well as the various perspectives that extend the IDE, are based on these parts, which are collectively referred to as the Rich Client Platform (RCP).

---

**Open Services Gateway initiative**

The eclipse component model (Equinox) is best described as a “microkernel” in the operating system sense: plugins (called bundles in OSGi-speak) are both run- and lifetime (discovery, installation, and integration) managed; can connect only by extending extension points; and have their own “execution space,” achieved in Java by a careful control of the classpath and by having plugin specific class loaders. An extension point can be an abstract interface that needs to be implemented, a concrete instance that requires configuration and registration, or an API.

The essence of the OSGi model is that software complexity can be solved with module software, but only if you can keep the resulting complexity of the dependency management under control. The latter has therefore to be strict, i.e. 100% automatic, and there shall be no backdoors to communicate with a plugin other than through its extension points, which make dependencies explicit. By this measure, shared libraries and normal Java .class files are not modular: once loaded, their full public contents are available to any other code in the system.

Not all of the OSGi model is implemented: lifetime control begins at eclipse startup, not at plugin installation; and extensions are one-to-one, instead of through a broker.
ECLIPSE IN ATLAS

Clearly, eclipse can be a useful tool for individual Atlas users. However, given its extensibility, it could perhaps be made even more useful with Atlas-specific plugins.

Problems to solve

What, then, are the problems in the Atlas development environment that need solving? Or, where can a large gain in productivity be achieved?

- Software Life Cycle: there are several best practices in software development that are not adhered to because physicists have not heard of them, because the tools that support them have a steep learning curve, or are simply hard to find. Think of refactoring of code, writing unit tests, or simply code sharing.

- Tool Integration: Atlas has a multi-language environment (mainly C++ and python) which can complicate debugging: there are difficulties with mixing the runtime environments of individual tools; and setting up a work environment is a priori non-trivial.

- Code Generation: the requirements of several kinds of analysis are well known, as the physics objects to be used are well defined. Code generation (and automated removal, so as to provide a beginner’s sandbox) can help setup package dependencies, locate header files, provide example reader code, etc.

In addition, given that students and post-docs often stay only for a brief period of time with the experiment, they can benefit from the coherent presentation per se of the Atlas offline software and its tools. This will allow them to quickly familiarize themselves with the environment.

Athena Startup Kit

The python interpreter, too, is a good environment for tool integration and this is exploited by the Athena Startup Kit [7] (ASK). ASK consists of a set of modules that operate in user space and fill in voids left by the collection of tools used in Atlas software. It provides basic code generation for ATHENA packages and components, as well as site and CMT requirements file abstractions for runtime management. ASK comes with a CLI (the python interpreter), a GUI, and it is fully scriptable, as well as usable by selecting individual modules. By reusing ASK within eclipse, it is possible to quickly build a prototype to determine the feasibility and usefulness of a real, integrated product.

The integration of ASK modules (python) into eclipse (Java) is done with Java Embedded Python [8] (JEP). JEP integrates the C-python interpreter, allowing for cross calls from the two languages and sharing of object data. JEP is fully thread-safe.

Implementation

There are four different ways of integrating elements of the Atlas software into the eclipse environment, see Fig.3. Existing eclipse plugins and perspectives (e.g. the C/C++ perspective, CDT [9], and the CVS plugin) are used out-of-the-box, with only minor changes to their configuration to make them aware of the Atlas specific plugins and tools. Certain external tools (e.g. Pacman) are independent from other tools and are therefore implemented directly as IDE extensions. A large fraction of wizards go through the ASK plugin; these could have been split out further, but that wasn’t done for the prototype. Finally, ATHENA is run as an external tool on a process of its own.

The choice of integration is solely driven by the interaction needs of the tool: it can always be presented in such a way that the tool appears to be fully integrated from the end-user point of view. For example, ATHENA is safely run on a separate process, to make sure that all resources (memory, file handles, database connections, etc.) at the end of an ATHENA run are freed, and to prevent a faulty ATHENA component from taking down eclipse. But it is still started from the IDE menu and its output is presented on the integrated console, which is also the place to enter input for the interactive prompt.

EXAMPLE USE-CASE: WORKFLOW

A typical use case in Atlas is: create a new package, add some ATHENA algorithms, flesh out an analysis code, and put this work into the CVS repository. Except of course for the analysis code itself, which is the creative part added by the physicist, almost all of this process is boilerplate and therefore automatable with a few wizards.

First, a “New CMT Package” wizard is run, which allows the user to specify a package name, and any number of algorithms. A second wizard, consisting of a set of selectable pre-sets adds proper include files and basic example code. Then, the build process can be started, which will pick up CMT as an external tool (because of the configuration created by the new package wizard above). This is
shown in the screenshot in Fig. 4.

After fleshing out the analysis code, the final step, access to CVS, is handled through the CVS plugin. The plugin is pointed to the right place in the Atlas repository users area by means of the user name, and with a mouse click, the new package can be checked in.

**CONCLUSIONS AND OUTLOOK**

Eclipse, an open source platform for tools development, has been presented. Its IDE and the range of tools available for the IDE ease the day-to-day work of a software developer. We’ve extended, through eclipse’ plugin architecture, its capabilities to address specific Atlas end-user needs. This way, eclipse could become an ideal entry point for every Atlas newcomer as it would represent a completely managed environment consisting of tutorials, howto’s, and an out-of-the-box working environment.

Wizards to create, modify, and maintain (predefined) configurations for various kinds of analysis will allow the Atlas user to focus on writing and testing her analysis rather than on how to deal with Atlas software idiosyncrasies.

Moreover, with a few additional plugins and configuration of existing ones, eclipse can span the entire workflow of Atlas analysis tasks. For example, a Pacman plugin enables downloading a development kit and providing the necessary local setup, an Atlantis plugin allows visualization of analysis results. Because eclipse is an open platform, with wider use than just high energy physics, it is a rewarding task for students and post-docs to provide similar plugins. This way, eclipse truly becomes an Atlas Integrated Work Environment.

**REFERENCES**