Extending Eclipse with Languages Other Than Java

Position Paper for Eclipse Languages Workshop

Bjorn Freeman-Benson

October 3, 2005

I'm interested in writing Eclipse plug-ins (extensions) in languages other than Java, specifically, in dynamic languages like Lisp, Smalltalk, Python, Groovy, Perl, Ruby, PHP, etc. I'm motivated by this problem because I'm concerned that Eclipse's growth is limited by the "plug-ins are Java" requirement. There are a wide variety of developers, potential users of Eclipse, would are not happy with Java. Some will tolerate it, some will not even bother. So far, Eclipse has tapped into the "willing to write Java" community as is evidenced by the number of Eclipse-based environments for these languages: EPIC, STDT, PyDev, Groovy Eclipse, RDT, TruStudio, RDT, PHPEclipse, etc. Imagine the proliferation of Eclipse for these languages if developers could write these environments in their native language…

I see three ways to support these languages: shared virtual machine, embedded virtual machine, and separate virtual machines. In all cases, I'd like to see the plugin.xml extended in two ways:

1. An attribute on <plugin> that specifies which language extension this plug-in is written in. For example <plugin language="org.eclipse.ruby">.
2. All attributes and nodes that specific a class are extended so that anywhere a x.y.class class name is used, one could specific an alternate language and class, e.g., "perl:Foo".

Shared Virtual Machine. In the shared virtual machine model, Eclipse contains (through a plug-in) a compiler that translates the alternate language into Java byte-codes. Thus the Java code and the alternate language run on the same VM. Groovy using groovyc would work this way.

Embedded Virtual Machine. In the embedded virtual machine model, Eclipse contains (through a plug-in) an interpreter for the alternate language written in Java. Thus the alternate language VM is embedded in the JVM. Jython would work this way.

Separate Virtual Machine. In the separate virtual machine model, Eclipse would effectively run on two VMs: one the JVM and one the alternate language VM. The two VMs would communicate through some channel (TCP?) to handle calls/returns/exceptions/etc.

In order for any of these schemes to work effectively, the alternate language developer needs to use all the APIs and extension points of Eclipse including all the call-backs and event handlers. For the Java-derived languages, such as Jython, this is relatively simple. For non-Java-derived languages, the language environment will probably need to use reflection to build stubs or we need to set some conventions for class and method naming.