An Alien's Perspective

ActiveState Position

Background

ActiveState has been building IDEs for dynamic languages since 2000, both as part of Visual Studio .NET (specifically *Visual Python*, *Visual Perl* and *Visual XSLT*) and as part of our own IDE (*Komodo*) which we built on the Mozilla framework. We've recently been evaluating what products we could build on Eclipse, both from a technical perspective as well as what it would mean in terms of our business plan. Rather than go into our current thoughts about the future, which are speculative at present, I will limit this paper's scope to our experience building IDEs for dynamic languages, in hope that some of our choices, experiments, successes and failures allow for others to build better systems than the ones we worked with.

Belief: Dynamic language programmers resist the trappings of a heavy-duty IDE mental model

The most notable differences between the products built on VS.Net and the one built on the lower-level Mozilla toolkit from the user's point of view result from the capabilities of the framework and the cognitive load associated with the framework. The VS plugin model expects dynamic languages to be treated roughly the same as the first-class languages in the VS universe, and as a result all languages inherit large numbers of "features" (some of which of doubtful value to the average Perl or Python programmer). The notions of *Solutions*, *Projects*, *Configurations* are there for plugin vendors like us to take or leave, but not really challenge. When left to our own devices (as in the case of Komodo), we have chosen a much lighter weight approach when it comes to defining the mental model that the tool imposes on the user. As a result, projects in Komodo are completely optional, and users are free to translate their old "plain text editor" habits of "open file, edit, save, run" without having to learn any IDEish notions. While getting objective data is difficult, we believe that the light-weight approach is much less of an impedance mismatch than the VS.Net model for customers who are *dynamic language programmers* (it should be noted that our products are typically bought by the end users, not by managers). I am unclear of the extent to which Eclipse requires the adoption of the current Java-centric metaphors.

Technical Choice: Write business logic in dynamic languages when possible

Except when the infrastructure prevents it (e.g. parts of Visual Studio .NET), we strongly prefer to implement the business logic of most of the IDE in a dynamic language (Python and JavaScript in Mozilla, Python and Perl in parts of the plugins). For most of the code paths, the performance of the interpreters is not a bottleneck, and the resulting ability to rapidly build and evolve functionality is highly valuable. Komodo is developed by the equivalent of 3 full-time engineers, the visual plugins (under less active development) by a single person, part-time. For a company the size of ActiveState, writing large systems in lower-level languages feels unnecessary and inefficient (we have little experience with Java, and I make no claim as to Java's suitability for any purpose -- I'm mostly contrasting Python/JavaScript/Perl to C/C++).

Warning: Multiple VM systems cause severe pain

Komodo embeds a Python interpreter (the standard C runtime) in Mozilla, which combines a lot of C++ code and a JavaScript interpreter (with its own VM, GC, etc). Several very tricky bugs and memory leaks were found to be due to pathological cycles between the different memory spaces, thereby forcing one to add "old-fashioned" memory management hooks to dynamic language code, which is both error-prone and slow (to code and to execute). I would hope that a system based on an embedded or shared VM would be better, but I could imagine that Murphy's law would also kick in.

An Alien's Perspective, continued

Technical & Philosophical Choice: Leveraging dynamic language runtimes

Our approach in Komodo (and later in the Visual Plugins) has been to integrate the user-installed open source dynamic language runtime technologies into the IDE to support language-specific features of highly modularized subsystems, such as syntax checking, code analysis, debugging, etc. For example, the Python interpreter is used to do syntax-checking, not our own clone of the interpreter's parsing logic. This is an efficient use of manpower, but more importantly, it allows the IDE to use the version of the interpreter that the user is going to use when running the code (not our own "frozen" version). In some cases, such as syntax checking and debugging, the delays incurred by out-of-process operation are not problematic, although on some platforms (OS X in particular), process launches appear to be quite expensive. There are areas such as SCM integration, where we now wish we had invested in an in-process infrastructure, even though the risks of "taking the whole application down" are real (especially with third-party code). High performance lexical analysis for the purpose of code coloring/folding is done through highly tuned and error-resistant specialized code from the Scintilla open source project).

Most of the dynamic languages have very broad and deep libraries and tools available as non-GPL'ed open source, written either in C/C++ or in the dynamic languages themselves. If our experience with Komodo is anything to go by, integrating such non-Java code into Eclipse would seem to be a likely good step if one wanted to incorporate present and future dynamic language support codes into it. Incorporating a Java-based VM interpreter such as Jython would work for some Python libraries, but by far not most such libraries (at least given the current state of affairs when it comes to compatibility between Jython and mainstream libraries). I believe the situation would be worse for other dynamic languages.

Offer: The DBGP debugging protocol

As part of our streamlining of our internal code base, and to try and foster more of a collaborative ecosystem around dynamic language tools, we have developed in conjunction with Derrick Rethans a wire protocol that we use to debug Perl, Python, PHP, Tcl, XSLT, and Ruby (we'll probably add JavaScript support soon). It allows for most common debugging features, and is open to extension. The protocol is an open specification, and the interpreter-side engines are available under open source licenses. See http://aspn.activestate.com/ASPN/DBGP .We welcome feedback, extensions, and additional implementations.

Python Software Foundation Note

While I'm happy to represent the Python Software Foundation at this symposium, I don't wish to present a position on behalf of the PSF. First, because there's a clear conflict of interest, and second, because the PSF has traditionally taken no position on developer tools, other than to explicitly "let a thousand flowers bloom." The Python ecosystem currently supports a variety of commercial and open source development tools of a variety of scope, with no involvement from the PSF (although several IDE vendors are PSF sponsors).

David Ascher (david.ascher@gmail.com)

Chief Technologist and Managing Director
ActiveState — Dynamic Tools for Dynamic Languages
(ActiveState is a division of Sophos)

activestate.com

Director
Python Software Foundation
(The PSF is a 501 (c3))

python.org/psf