Mixed-language usage scenarios in Eclipse

Position Paper for the Eclipse Languages Symposium
Elias Volanakis <elias AT volanakis.de>
October 21, 2005

Abstract

The paper identifies some common use-cases for mixed-language solutions in software development and tries to find occurrences of those cases inside Eclipse. It argues that providing support for plugin-development in dynamically typed scripted languages, paired with a xml-based declarations of user interfaces, would be one way to make Eclipse accessible to a wider audience of developers and concludes by stating my positions related to this undertaking.

Mixed-language software is not uncommon

Extending existing software with parts written in other languages is not an unusual scenario. Some common tasks where mixed-language solutions are used include:

1. **Integration of libraries written in a different language into a larger piece of software.** This is done to avoid implementing specialized and often very complex algorithms. Instead a widely used, time-tried and optimized implementation from a different (often lower level) language is reused. Examples include libraries for communication, numerical computation or libraries that interface with native OS APIs.
2. **Integration of a new piece of software written in domain specific language (DSL),** which is a language with a very high level of abstraction, particularly suited to describe and solve a specific class of problems [1]. The domain specific language may depend on some kind of runtime (SQL running in a DB runtime; a DSL written in prolog) or may be compiled into executable code (a parser implementation generated from an AST [2] declaration).
3. **Extending a complex software system by means of a language better suited for a wider audience.** Popular examples include office suites that offer some kind of macro-programming facility (Office & VBScript), customizable business software (SAP & ABAP) and extendable software platforms (Mozilla & Xml User interface Language (XUL) paired with JavaScript).

The list above is not exhaustive. However the approaches included here share a common motive: to save implementation overhead, either by reusing existing code or using a language closer to the problem. The third approach has an additional aspect: to allow more people to utilize an existing and powerful software platform.

Mixed-language scenarios in Eclipse

The Eclipse platform itself also contains some occurrences of the usage scenarios described above:

1. **Integration of non-Java libraries:** For example SWT provides a thin wrapper of Java code together with JNI libraries to interface with the native operating system widgets.
2. **Integration of domain specific languages:** In my opinion, declaring a new extension point and providing an associated schema definition can be seen as creating a domain specific language. For example CDT's Managed Build Definition extension point (`org.eclipse.cdt.managedbuilder.core.buildDefinitions`) provides an xml-based DSL to declare project-types, configurations, tools, build toolchains and tool options. Eclipse and CDT also provide the „runtime" in which instances of that DSL (e.g. extension declarations of that extension point) are „executed". By declaring a tool and some tool options in the plugin.xml the „runtime" creates appropriate UI elements in the project preferences for setting those options.
3. **Complex software extended by a „simpler" language:** While I could not identify an occurrence of this usage scenario inside Eclipse, I believe this is something worth discussing, because it would provide an alternative way for non-Java developers to utilize the Eclipse platform for (rich client) application development. A way to facilitate this would be to support plugin-development in dynamically typed languages. Paired with a form of XML-based declarations of UI elements, this would provide a way for plugin development that is very similar to the approach taken by XUL [3] and XAML [4].

Considering XUL-like Eclipse development

In order to enable XUL-like Eclipse development two things would be necessary: Having a (xml-based) language for declaring UI elements and pairing it with support for plugin-development in a dynamically typed scripting language. While I cannot say what exactly it would take to make this a reality, there are some positions I would like to contribute to the discussion:

1. Good tooling is essential: Editing, launching and debugging the UI declarations and the scripted parts should be a comfortable user experience, on par with the support for statically typed languages. Having edited XUL and JavaScript files without tooling, I can say that this was not a particularly encouraging experience.
2. Avoid disruptions to the platform: Eclipse is now a very mature application development platform. If possible, enabling XUL-like Eclipse development should avoid the introduction of breaking API-changes, or it will be hard to gather support for such an undertaking.
3. Beware of the „language mismatch": Integrating a non-object-oriented scripted language will make it hard to implement existing object oriented interfaces. Concepts like object state and callback may be difficult to support [5].
4. It is not uncommon for market leaders in the software world, to try to segregate themselves from the rest of the market by technical means (for example by using proprietary data storage formats). Here we have a reverse situation: we are considering ways for an incredibly successful software platform to reach out to new market segments, for example by enabling plugin-development in dynamically typed languages.

References

[1] http://en.wikipedia.org/wiki/Domain-specific_language
[2] http://en.wikipedia.org/wiki/Abstract_syntax_tree
[3] http://www.mozilla.org/xpfe/xptoolkit/xulintro.html
[4] http://windowssdk.msdn.microsoft.com/library/default.asp?url=/library/en-us/wcp_conceptual/html/0ff5f36e-dd84-44d1-aa3e-5bb4f147b169.asp?frame=true
[5] „Extending Eclipse in Haskell", Leif Frenzel (Eclipse Languages Symposium)