Jython as a Scripting Environment

Richard Woolliscroft
Data Acquisition Group
Diamond Light Source Ltd.

The GDA acquisition software being developed jointly between Daresbury and Diamond uses Java as its main programming environment. Jython, a Java based language with analogous syntax to Python, has been selected as the scripting language for the system.

The discussion will include some ways that we are exploiting this fully featured Python like language with its close relationship to Java. A scripting environment can be created which has all the benefits of a full scripting language, but in which the syntax can be controlled. Emphasis will be placed on maximising code reuse in the software and minimising the amount of syntax that users would have to learn.

Introducing Jython

From the Jython web site:

Jython is an implementation of the high-level, dynamic, object-oriented language Python seamlessly integrated with the Java platform. Jython is freely available for both commercial and non-commercial use and is distributed with source code. Jython is complementary to Java and is especially suited for the following tasks:

- **Embedded scripting** - Java programmers can add the Jython libraries to their system to allow end users to write simple or complicated scripts that add functionality to the application.
- **Interactive experimentation** - Jython provides an interactive interpreter that can be used to interact with Java packages or with running Java applications. This allows programmers to experiment and debug any Java system using Jython.
- **Rapid application development** - Python programs are typically 2-10X shorter than the equivalent Java program. This translates directly to increased programmer productivity. The seamless interaction between Python and Java allows developers to freely mix the two languages both during development and in shipping products.

There are numerous alternative languages implemented for the Java VM. The following features help to separate Jython from the rest:

- **Dynamic compilation to Java bytecodes** - leads to highest possible performance without sacrificing interactivity.
- **Ability to extend existing Java classes in Jython** - allows effective use of abstract classes.

- **Optional static compilation** - allows creation of applets, servlets, beans, ...
- **Bean Properties** - make use of Java packages much easier.
- **Python Language** - combines remarkable power with very clear syntax. It also supports a full object-oriented programming model which makes it a natural fit for Java's OO design.

Jython and Java

Jython is completely written in Java. It could be viewed as a Jython program written to mimic the Python environment. As the two languages have this very close relationship, there are a number of benefits which can be exploited:

- Java classes may be called and used within Jython without any special coding. An instance of a Java class may be instantiated and used from within the Jython environment without any refactoring or special code in the class – thus any of the wide range of Java tools and utilities available may also be used from within the Jython environment.
- Jython classes may extend Java classes – allowing Java interfaces to be properly used by Jython classes.
- The Jython interpreter may be embedded into a Java program. So a GUI written in Java may have a command-interpreter inside to allow users to type commands or give it scripts. The Jython interpreter has its own namespace, separate from the encapsulating Java, but object references may be passed between the namespaces.
- If a Jython interpreter is embedded into a Java program, commands may be intercepted by the Java code before they reach the interpreter. This allows the opportunity for the Jython syntax to be extended within that program. Users may type commands using the extended syntax, which the Java may 'translate' into regular Jython syntax before passing it to the interpreter.

How Jython is being used by Diamond and Daresbury

Diamond Light Source and CCLRC Daresbury Laboratory are collaborating to produce a Java-based data acquisition system called the GDA (Generic Data Acquisition). This software has a client\server design, using CORBA technology as the middleware. It will be capable of operating EPICS devices.

The client process in the GDA is a GUI written using Java Swing. This will have visual controls for equipment movement and automated data collection (scanning). To allow users to have more control over experiments, a scripting environment is required.

Jython was chosen as the scripting environment because of its close relationship to Java and its simple syntax. As Jython can be embedded inside Java programs, and can create Java objects, very little extra code has needed to be written to add scripting functionality to the GUI.

Indeed, *nothing* has been written in Jython. A Jython interpreter has been embedded into the client GUI program; however, as all work is still performed inside Java classes,

the same Java classes that the GUI uses, the coding required has been minimal and it has been possible to perform debugging while using the Jython interpreter. This has reduced development time.

At runtime, all objects describing the beamline components (these would all be written in Java) are instantiated within the Java program. These objects are then shared in the namespace of the embedded Jython interpreter. Users may then control the beamline components through the Java GUI panels, or by typing commands into the Jython interpreter. As these are the same Java objects being accessed, there is no discrepancy between what the GUI and what the Jython environment observes. This also means that the GDA server program does not need to account for a command coming from either Java or Jython environment. Also, as these are full Java objects being accessed (rather than Jython objects inheriting from Java), during development these objects may be seen in a debugger, even if accessed via Jython.

The syntax available for scripting starts with the Jython language. When writing scripts to be passed to the embedded Jython interpreter, users have a full programming language. However the most common functions, such as moving and querying motor positions, have special commands to save the users typing. As, explained above, these commands would be translated into true Jython before being passed to the interpreter, this allows the possibility of different syntaxes between beamlines. Even though the same Java classes are being used, syntax could be developed for different beamline dependent of the user community and the emphasis which different experimental techniques have. For example, one beamline may wish to have a syntax which is as quick to type as possible with syntax similar to UNIX commands, whereas another beamline may have a community which is less familiar with programming and may wish for a pseudo-English syntax similar to SQL. It would be possible to develop both cases, with both still using the same underlying Java classes for equipment control.

Conclusion

Java and Jython make an attractive combination for data acquisition control software. Java allows for a flexible development environment, and its popularity makes useful open source third-party software easy to find. With Interactive Developmnet Enviornments (IDE's) such as Sun's NetBeans or Borland's JBuilder, speed of GUI development can now be comparable to Visual Basic.

The inclusion of Jython as a scripting environment within a Java program is simple and fast. Its close relationship to Java means that very little extra code needs to be written. It is simple to extend the Jython syntax used within the program to allow extra commands to be developed. All classes can be shared between the languages, and at runtime objects may be shared between the namespaces of the two environments. This means that the code base over both environments can be minimised, with benefits to development time and maintainability.