Smalltalk to Java Object Communication using XML

Dominic Blatchford

Introduction

This paper briefly outlines the findings of an integration prototype, produced to determine the feasibility of using XML to enable object communication between the Smalltalk and Java language environments.

The key goals were to ensure flexibility, ease of use and high performance of the developed solution. The prototype was considered successful when it was able to pass a Smalltalk object into the Java environment with all attributes (though not all methods) intact, and vice versa. The only methods that were translated were simple get/set access methods.

Integration Steps

To integrate the two language environments, the following steps were identified as being necessary. They ensure that the solution is not simply point to point, and that it will cope with diverse objects:

1. Provide a message/event driven interface to communications.
2. Write a simple TCP/IP socket based text transmission service.
3. Write/obtain XML parsers.
4. Write object serialisation mechanisms based on get/set methods and reflection.
5. Encode object attributes (serialised) in string format as XML nodes.
6. Implement a method for recognising and cross-referencing objects already encountered.

Details

Most of the steps followed to achieve object communication were fairly straightforward. The messaging/event infrastructure was based on a Singleton message manager, from which message listeners were able to register interest in certain *classes* of message; message producers were able to submit *instances* of particular messages. These were then delivered to all registered listeners. A remote *interface* was defined to mark certain messages as being transmittable, the actual socket based transmission system listened for remote messages.

The socket based communications modules and the XML parsers were much as expected in a system of this nature. The powerful reflection (or introspection) capabilities of the two OO languages enabled a bean-like strategy to be developed for object serialisation purposes. In this way, all ‘get’ methods may be executed recursively until a string or primitive value is reached. Any objects encountered along the way are serialised by stepping into their ‘get’ methods and repeating this function. By doing this, all attribute names become XML tag names and their attributes become either simple string values (including numbers) or more complex child nodes.

The only further problem that is associated with this recursive serialisation methodology is that XML is tree structured while objects often have circular relationships. It was, therefore, necessary to store references to all objects as they were encountered. Any duplicate encounters were trapped and marked as cross-references (using an <XREF> tag). These were (obviously) not stepped into, as infinite looping would have resulted.

To reconstitute serialised (XML) objects, the process can be reversed. It was found, however, that the programmatic overhead of duplicating object (skeleton) definitions in the partner language was unnecessary as the XML nodes were easily navigated in most data browsing activities.

Conclusions

The integration prototype was, as a whole, successful. Integration was achieved in a way that enabled any objects to be passed through. The system also presented an abstract and uncomplicated programming interface. Finally, the performance of the system was remarkably good, considering the complexity of processing. A large suite of objects, totalling about 40k of XML data when serialised, typically took around 0.6 seconds to pass one way.