E-BUSINESS JAVA IN THE CLASSROOM

DAVID PARSONS - THE OBJECT PEOPLE

A TUTORIAL FOR THE 4TH JAVA IN THE COMPUTING CURRICULUM CONFERENCE, SOUTH BANK UNIVERSITY, JANUARY 24TH 2000

dparsons@objectpeople.com

Introduction

The teaching of programming languages has long been context driven in the sense that certain languages gravitate towards certain application areas, and the way that we teach (examples, exercises, assignments etc.) follows that context. For example, COBOL teaching tends towards the batch processing of file based data, using data driven design techniques, whereas Ada teaching frequently reflects its use in control systems and design tends to follow functional decomposition. The context of Java is both wider than any previous language and more fluid. In 1996/7, educational institutions were adopting Java as perhaps a straight C++ replacement, or in some cases as a simple means of teaching distributed programming with client side processing (sockets or applets). Now, it may be reasonable to wonder what is the educational context of Java? Post version 1.0, Java can no longer be taught as a discrete subject because it has grown to encompass so many APIs that a greater focus has become necessary. Educators must now navigate through the alphabetti spaghetti of JDK, JRE, JVM, JDBC, RMI, EJB, SSI, JSP, CORBA, IDL etc., and find some way of deciding what is important. This tutorial asserts that the context of Java is rapidly becoming e-business, in particular the Java 2 Enterprise Edition model, and that this should provide us with the framework within which to teach the language.

An object and client oriented framework

Given the complex range of tools and APIs available for e-business Java, the educator has to make a number of choices when creating a teaching framework. The framework suggested in this tutorial focuses on the client-oriented elements of Sun's distributed e-business J2EE architecture rather than the persistence and middleware levels, with the intention of teaching Java using pure object models with an orthogonal distribution layer. It centres on the 'ideal' model of client and server based objects, distributed in a semantically cohesive way between applets and servlets, using remote objects where appropriate. Applets are included in this framework to allow a proper use of objects on the client. Servlets are included not ultimately for form rendering (a tedious process that is a useful introductory exercise but delivers little in terms of architectural challenge) but for server side processes and connecting to JavaServer Pages via Java Beans in session objects. The educational power of such a model is that it allows an object-oriented paradigm to be used cleanly across the web. No matter that as a commercial architecture it is probably unlikely and certainly incomplete - a developer who can build this kind of distributed object model will easily be able to adapt to the

compromises of the real world when necessary. Also the model can be built prior to the distribution and persistence layers, allowing an early focus on the domain objects without regard to their server based implementation. When the distribution layer is built we should have some concept of the Rational Unified Process 'executable architecture' that can support a full design process, i.e. we do not expect students to built complete systems but they can build 'iteration 1' of complete architectures.

The framework process

Of course such a framework must be developed incrementally in an educational context, providing a viable process as well as a product. To some extent we must explore avenues that will not form part of our ideal architecture, such as form rendering within servlets, in order to provide a shallow enough learning curve. The emphasis here is also on preserving object-oriented concepts in the underlying object model, which is a good reason for delaying introduction of EJBs until very late in the process, since they tend to wrap functions and database tables rather than represent fully fledged objects. Hence the suggested framework begins with a somewhat abstract object model, initially unconcerned with distribution and persistence. Then a front end can be built using the thin client architecture of HTML forms rendered by servlets. To develop a proper model/view separation, JavaServer pages can be introduced to render pages, supported by the processes of servlets. Beans and sessions can be used to manage the communication between them and allow proper distribution of responsibilities.

This architecture alone is enough for many e-business contexts, but it can be made more interesting by introducing a distributed object model using applets. There are a number of implementation strategies including sockets, HTTP and RMI, but all allow object distribution. On advantage of an RMI implementation is that it provides a real world context for an understanding of the role of interfaces. Finally we should address persistence, for which EJBs with container managed persistence are probably the easiest option. Again, RMI is useful in that it provides the foundation for an understanding of EJBs.

Summary

Java can be taught in a realistic and commercially useful way by focussing on the higher level aspects of the J2EE architecture. We can begin with pure object models and evolve them into e-commerce systems by adding client side views, distributed objects and persistence. Along the way we introduce the separation of model and view and the interface elements of distributed object models. The overall process can be seen as a first iteration within a managed cycle such as the Rational Unified Process (RUP), where an architectural prototype is developed during the elaboration phase. Depending on the time available and the levels of the students, the teaching framework might include only the object models, servlets and JSPs, with the more complex elements available for further courses or projects.