Creating Commercial Software with Jakarta
A Presentation for…
ApacheCon 2002, Las Vegas
By Derek Ferguson

Good afternoon and thank you for coming to my session today. The name of this session is "Creating Commercial Software with Jakarta". My name is Derek Ferguson, and I am the Chief Technology Evangelist for Expand Beyond Corporation, the worldwide leader in mobile software for enterprise management.

I've divided today's presentation into three main categories. To begin with, I'm going to explain to you a little bit about the tools we use at Expand Beyond – or EXPAND BEYOND as we refer to it – in crafting our Jakarta-based software. Then, in general terms, I'm going to look at some of the various techniques we have used in creating our PocketDBA, PocketAdmin and Wireless TSReorg products. Finally, I'm going to "zero in" on the procedural aspects that allow us to bring our Jakarta-based products to market with such a high degree of success.

Now, the first question that you might have is, "Who am I to be telling you all of this?" Well, in addition to being Chief Technology Evangelist for EXPAND BEYOND, I am also our Senior Product Developer. This means that I literally spend some of almost every day writing code that will ultimately go into one of our Jakarta-based software products.

In terms of Java media credentials, to begin with I have spoken at Java One, which is Sun's big annual Java conference – so certainly a kind of endorsement from the people who created Java in the first place. And I am also Editor-in-Chief for one of Sys-Con Media's (the company behind *Java Developers Journal)* biggest technology magazines. And, finally, I've written articles for O'Reilly's OnJava.com web site.

The final thing that I'd like to say about myself before we proceed is that I strongly encourage you to contact me with any questions or comments whatsoever at derek@XB.com. Really, I'm very serious about this – no one ever contacts me about anything after these presentations and I'm really quite excited to hear what you all are working on and to be of assistance in any way that I can.

Free, Non-Apache Tools

Some of the most important tools to the EXPAND BEYOND development process are created by organizations other than Apache. Whenever possible, we try to use free, open-sourced tools. However, unfortunately, sometimes we need functionality that is only available through a commercial, closed-source product.

Bugzilla is an example of a product which is not an Apache tool, but which is both open-source and free. We initially started using this product about a year ago strictly to track bugs that were reported to our organization by external customers. Shortly thereafter,

however, we decided that we liked its functionality so much that we wanted to use it to start tracking requests for enhancements (RFE's), also.

So, we made some enhancements to the code specifically to allow us to better track RFE's. For example, whenever we hear about a new idea for our product, we want to know where the idea came from – an external customer who actually needs it, or an internal user who just thinks it would be cool. We have also added some advanced reporting, so that we can keep track of all the entries in the database and monitor the progress our team makes in resolving them.

JUnit is also both open-source and completely free. We use it to automate our test-first programming standards. In short, this means that, before we write code to add any functionality to any of our products, we must first think of a good, automated way to test whether or not that functionality is working.

For example, there are tests in our system which actually connect to SSH and Telnet boxes on our internal network, specifically to ensure that the SSH and Telnet features within our products are still working. We run these tests via JUnit every night as a part of our nightly build. This allows us to spot instantly if anything has been broken when merging together all of the different developers' work.

The final open-source, free software product we use is called Wikki Web. We use it to keep track of all the little bits of information we accrue as a development team. These are the things like coding standards, schedules, and contact information that are really:

- Not appropriate for storage in Bugzilla
- Not appropriate for sharing outside of Development

Commercial Tools

The first commercial, non-Apache tool that we use to produce our software is InstallAnywhere. Originally, we were strictly an InstallShield shop. However, as we grew, we began to require more flexibility that we felt that we could get out of InstallShield. So, we migrated all of our installer scripts over for version 1.8 of our PocketAdmin product.

Since then, InstallShield has brought several major benefits to our product:

1. The ability to have our software easily install as a Windows service and to start as a service immediately after installation, if desired
2. Easy recognition of the main Java classes in our application as we are building our installers
3. Web-based installation via Java's Web Start technology

Clearcase was a tool that had been used by many of EXPAND BEYOND's initial development staff at their previous jobs. Many of them came from big organizations

such as Motorola and 3Com, and were eager to duplicate the development environments they'd had there as closely as possible when they came to work for us. Of course, what is good for a multi-million dollar corporation is not necessarily what is good for a startup, so we chose to economize a little.

Towards this end, we chose to buy Clearcase Lite instead of a full implementation of Clearcase. This has worked very well for us, as we really only use it for source control, anyhow. For the average programming task, a developer:

1. Accepts a report from Bugzilla
2. Checks out a view of the mainline from Clearcase
3. Writes a test for JUnit and watches it fail
4. Writes their code
5. Verifies that their own tests now pass
6. Checks their work back into their own view
7. Verifies that ALL of the tests still pass
8. Merges their view back into the mainline

db4o is the final piece of commercial software that we have chosen to use in our product. Unlike all of the previous tools, however, this one actually works for us at run-time, rather than at development time. As you will see in the next section, we use a highly metadata-centric application model to create all of our software. *db4o* is the object-oriented database that holds all of our metadata.

Apache Tools

Of all the Apache tools that we use in creating our Jakarta-based software, easily the most powerful and important of all of them is ANT. Before PocketDBA 1.4, our flagship product was built using the *make* utility. Unfortunately, this made it virtually impossible for most developers to build our products on their desktops – which were almost all Windows.

Now, it is true that you can run *make* by installing the CygWin tools that give you UNIX-style commands under DOS. However, in actual practice, we found that there were just too many configuration issues for this to be a practical solution going forward. So, instead, we migrated all of our build scripts over to ANT.

ANT solved our developer-workstation building problems. However, it also appealed to us through its integration into both JBuilder and IDEA and its highly-intuitive, XML-based configuration file format. We use it to build on demand as well as to do our automated nightly builds, which are so important to our process.

Tomcat is a close second in terms of its importance out of all the Apache tools that we use. In some ways, you could argue that it is the most important, insofar as it is the application server that houses our entire application.

However, in reality, the metadata-centric model of our application means that really very little of the processing in our applications is done by the application server at all. We basically just use it to serve up an initial Servlet, and everything after that is handled by custom EXPAND BEYOND logic.

LOG4J is a piece of Apache technology that is used even by our custom logic, however. Once again, in the olden days before PocketAdmin 1.8, all of our applications had their own methods for reporting errors and other status information to users and administrators. That changed right about at the start of 2002 when we adopted LOG4J as our corporate standard for logging any kind of information from our applications.

Our standards for debug level settings are as follows:

- **DEBUG** messages are of interest strictly to developers
- **INFO** messages might be of interest to particularly geeky users and administrators, but are otherwise hidden
- **WARN** users should always see these messages, because they mean something bad – but users can ignore them at their own peril
- **ERROR** this is definitely a problem and should be acted on, but at least the application is still working
- **FATAL** now, the application isn't even working

We have trained our technical support to set any of the required levels, so that they can provide us with as much information as is needed for us to troubleshoot issues.

Tomcat Threading

As I'm sure many of you probably realize: I can't go **too** far into telling you exactly how Expand Beyond achieves the magic within our products. However, there are some general concepts that I can share with you which illustrate some common problems and their workarounds when creating commercial software with Jakarta.

The first of these involves the nature of threads under Tomcat. As I'm sure many of you are aware, web traffic is, by its very nature, stateless. This means that, a web browser will connect to a web server, download the information on a page, and then disconnect. Under the hood of most web servers – Tomcat included – this means that the threads that are spawned by the processes occurring on any given page should **not** be allowed to survive, for fear that they will accumulate and eventually take down the entire server.

Unfortunately for us, much of the work that is done by our applications requires that we keep threads alive for more than a single page view. The way that we have achieved this is through a little concept we called "Thatched Threads". What this basically means is that we have written our own kind of garbage collector – only specifically targeted at threads – for the Tomcat application server.

A perfect example of this phenomenon is the relationship between our PocketAdmin product and our PocketDBA product. PocketAdmin is, essentially, a wireless SSH/Telnet client for handheld devices. As such, it has a very stateful execution model. PocketDBA on the other hand, is purely a web based application that allows Oracle, SQL Server, and DB2 databases to be administered from wireless handhelds.

The problem arose when our DBA customers for PocketDBA began telling us that they wanted to be able to perform some of the shell administration features for PocketAdmin using a web interface. How could we manage this? The answer lay in reworking some aspects of PocketAdmin to run as a subtask within the overall Tomcat architecture. Its threads are then managed by our "Thatched Thread" model, allowing us to use conventional web techniques such as Cookies to maintain state for PocketAdmin.

Application Architecture

Our meta-data based application model is another extremely important technique that we have leveraged in creating all of our best Jakarta-based wireless administration products. In short, by storing most of our application logic in "hot swappable" IQ files, we are able to easily distribute product updates to our customers with a very high frequency. All of our products are architected such that these IQ files can be swapped in-and-out and take effect immediately without even having to restart the server!

This kind of flexibility is made possible because the vast majority of logic in our applications is not actually stored in the Tomcat application server. Instead, the flow of logic when a client makes a request to one of our application is as follows:

1. The main EXPAND BEYOND servlet accepts the request and performs some basic authentication
2. The request is passed immediately to the metadata for further processing
3. The metadata eventually produces a response that is pure XML
4. One or more filters process this XML into markup before it is returned to the handheld device

The XML that is returned by our metadata is probably the most basic application of XML technology in any of our applications. EXPAND BEYOND stands 100% behind XML as a technology for enabling two very important things. The first of these is the free exchange of complex data between heterogeneous computing platforms. The other is as a way to serialize object oriented data without the overhead of persisting an implementation along with your data.

When XML is being processed via an EXPAND BEYOND application, we tend to use one of two techniques. In the case of simple searches for specific pieces of information, XPath is typically our technology of choice. For more complicated translations – for example, translating platform-neutral XML into markup that is specifically optimized for use on an HTML, WML, or cHTML device – we tend towards XSLT.

XSLT is an extremely powerful technology which is available within Apache's Xerces XML library, amongst other places. Using it, you can translate entire documents from one markup language to another quickly and efficiently. Unfortunately, XSLT has a very cryptic syntax and is also prone to giving very poor error messages. It is, therefore, essential that you allow plenty of time for any XSLT programming tasks that you might devise.

The most advanced use of XML in any of our Jakarta-based products, however, would have to be the XML Web Services that allow us to integrate with .NET. For more information on this, see my presentation, "Integrating Apache with .NET"

JDBC

The final technique that I would like to discuss with you involves our usage of JDBC within the framework of our Tomcat Web applications. As you know, PocketDBA is a product that allows DBA's to administer their databases from wireless handhelds. It is available in three flavors – Oracle, SQL Server, and DB2. In all three cases, administration is accomplished via a JDBC driver interface.

However, early on in our development process, we realized that there were some features sorely missing from many JDBC implementations. We had been spoiled because, honestly, Oracle's JDBC driver was really quite good – and that is what PocketDBA administered first. Unfortunately, when it came time for us to administer SQL Server, we soon discovered that Microsoft's JDBC driver had some serious flaws.

The most pressing of these flaws was its tendency to wrap every single operation in its own transaction – a practice which all but killed performance on most machines. To get around this limitation, we ultimately had to wind up licensing a commercial JDBC driver for SQL Server, rather than using Microsoft's free one.

DB2 presented an even worse obstacle. Administration of DB2 is typically **not** performed via SQL but, instead, is performed via a C-based API. In order to create the DB2 flavor of our PocketDBA product, then, we had to create a JDBC driver which "wrapped" our C API calls to make them look like SQL calls to the client code. This was, as you might imagine, no small feat.

Stand Up Meetings

One of the things that we value most at EXPAND BEYOND in creating our Jakarta-based software is the ability to reproduce the same results over-and-over again with a high degree of predictability. We feel that this is given to us through our Process – which is specifically designed to reduce risk and to produce products of the absolute highest quality possible.

The best way to describe our process in general terms is to say that it is a modified version of Extreme Programming. This is because, although most of the tenets of

Extreme Programming seem very logical to us and have been eagerly absorbed, some of them – as I will note below – do not quite fit our needs at Expand Beyond.

One of the first bits of XP that was adopted by EXPAND BEYOND was the concept of a stand-up meeting every morning. As our development team quickly grew, it became quickly apparent that many times people were working separately on tasks which were either identical or, worse yet, completely at odds with each other. For this reason, we quickly instituted this measure to allow everyone some insight into what everyone else is working on.

The idea is very simple. Everyone stands in some approximation of a circle. You go around the circle and, when it is your turn, you say what you worked on yesterday, and then what you plan to work on today. Once everyone has spoken, people who need partners find them and everyone goes about their business.

This has worked very well for us in terms of solving our communications problems. Unfortunately, what has happened over time is that it has become more and more of a development design meeting than a quick status report. This is probably true largely because people have stopped standing and are, thus, not urgent enough about getting the meeting over with.

Fortunately, the solution is simple. If people stop standing in your meetings, get them standing again. If standing isn't generating the appropriate degree of urgency, then have everyone jog in place. If that doesn't work, try jumping jacks! :-)

Ensuring Code Quality

As I mentioned above, at the end of the standing meeting, everyone has an option to pick partners. If this were 100% "Pure" XP, I would've said that everyone picks partners – without an option. This is one of the areas in which we have taken a bit of a break from XP, insofar as we allow people who don't wish to partner to submit their code to code reviews, instead.

The reason for this is that we have found that some people really do **not** work better in pair programming situations. In some cases, this is because they tend to approach problems from a very idiosyncratic point of view that makes it very difficult for other people to follow their patterns of thought. In other cases, it is because – although they are brilliant developers – they lack basic social skills. Whatever the case, we have found that it is better to let some developers simply work by themselves.

For those who do elect to work with partners, however, pair programming has proved to be an effective means of producing quality code. The process is simple:

1. Two developers sit at a single computer
2. One person "drives"
3. The other person "comments"

4. Whenever the person commenting has an idea, he takes the keyboard and "drives" while the other person "comments"

Variations on this have arisen very organically. For example, in the case of very difficult programming problems, it is not unusual to see 5 or more developers all huddled around the same computer at EXPAND BEYOND – working together as a team to find a solution.

Code reviews work in a very similar manner, except that the code is written and printed out – in *diff* format – before the driving and commenting ever begins. We use 3 person code reviews at EXPAND BEYOND. One of these people is the coder, who typically performs the job of walking the remaining two people – the reviewers – through his or her code. The reviewers make suggestions and criticisms, and the coder then goes away and fixes their code before merging it back into the mainline.

Project Planning

Our development cycles at EXPAND BEYOND are organized into 2 week segments called Iterations. At the start of every Iteration, we have an Iteration Planning Meeting. Before each of these meetings, our on-site customers choose which "Stories" they want to have included in the next build of our product. During the meeting, these stories are broken up into specific technical tasks and assigned estimated times to completion. If there are more stories than can be completed during this iteration, then some stories have to be pushed off to the next iteration. If there are more time and resources available than there are stories, then additional stories are accepted.

At the end of every Iteration, there is a period of Acceptance Testing. Acceptance testing happens before Q/A and is fundamentally different from Q/A. In Q/A, an engineer attempts to ensure that the product is performing according to technical specifications. In Acceptance testing, the on-site customer who initially wrote a story for implementation verifies that what has been produced actually matches what he or she was looking for.

Conclusion

This has been just a brief overview of some of the things that EXPAND BEYOND has found useful as we craft our products for the Jakarta platform. As I said at the opening to this presentation, I welcome all of your questions and comments to my email – derek@XB.com. Thank you!

URLS

- http://jakarta.apache.org/ant/index.html
- http://jakarta.apache.org/tomcat/index.html
- http://jakarta.apache.org/log4j/docs/index.html
- http://www.mozilla.org/bugs/
- http://www.zerog.com/products_ia_01.html
- http://www.junit.org/index.htm
- http://www.rational.com/products/clearcase/
- http://www.db4o.com/
- http://www.c2.com/cgi/wiki?WikiEngines
- http://www.XB.com