SOLUTIONS JOURNAL

The IDUG Solutions Journal
August 1998 - Volume 5, Number 2

Java, JDBC, and SQL
Are they ready for prime time?

The editors of IDUG Solutions Journal have invited two DB2 experts to duel peacefully on key issues facing DB2 today. In this regular column, Willie Favero of BMC Software and Craig Mullins of PLATINUM technology will face off. Sometimes they even agree.

By Willie Favero

So, what is the latest brewing in the DB2 Family? You got it, Java. Are you salivating, just waiting for the next Java announcement? What about JDBC? Have you read much about it yet? JDBC is the extra foam that makes all this Java stuff so much better. If you are into specialties, SQLJ might be the double mocha needed to make it all perfect. And, by the way, is anyone else tired of the coffee play on words?

Before going into detail about what is happening to the world because of Java, we should first separate the local supermarket coffee aisle jargon from the ideas being presented to the IT industry. If nothing else, Java, and object terminology in general, is assaulting us with an entirely new language, most of which simply means things with which we are already familiar. So here is a beginner's guide to the language of Java as seen through the eyes of an ex-systems programmer.

Java itself is fairly straightforward. The original idea from Sun Microsystems, Inc., was for an object oriented, and component based, language that allows someone to write a program one time and have it run anywhere, regardless of hardware or operating system. This promise comes accompanied by a language that resembles C/C++ and is relatively easy to learn. The number of verbs available in Java is a mere fraction of what you might find in C++. This makes Java easy to use, and gives us a language which avoids some of the problems inherent to C++. What else makes this three-year-old language so exciting? After a Java program is coded, it is compiled. Sound familiar so far?

At this point, things start to change dramatically. The output from the compiler is called Java bytecodes, which give Java its hardware independence. Enter now the Java Virtual Machine (JVM), which interprets the bytecodes for the specific platform on which it is

about to execute, regardless of whether that platform is OS/390, Unix, a television set, or anything else. The JVM is not the only thing that uses bytecodes. One alternative to the JVM is the Just in Time compiler (JIT), which interprets the bytecodes just in time to execute as machine language on the destination platform. It sounds pretty neat. But the JVM and JIT processes have one major disadvantage: they are both interpretive. And that makes it difficult to achieve the performance that many enterprises have grown to expect. More on that in a bit.

Another term in the Java lexicon is JavaBeans, which are simply object components _ reusable pieces of code. Some of us older mainframe types worked with a primitive form of JavaBeans in the COBOL days. We called them subroutines. Object oriented, and therefore Java, have taken component technology to a whole new level though. JavaBeans allow applications to be quickly assembled using prewritten components; this enables program prototyping to be used to produce a faster and higher quality code. End users will be able to get more use out of this program code because they were involved in the actual development of the applications. This technique is referred to as Rapid Application Development (RAD).

We'll soon be hearing more about something Sun Microsystems calls Enterprise JavaBeans (EJB) _ an effort to simplify the development of three tier applications. This effort is fully backed by JavaSoft, IBM, Oracle, and other major vendors. Not everyone, though, is behind EJB. Microsoft has already stated they're not interested in supporting EJB. They believe their Common Object Model (COM) does essentially the same thing.

Yet another hurdle Java must leap to be ready for prime time is database access. Much of the data used today rests in a relational database. Java offers no way to access that data. An additional software component must be added on top of Java to access relational data: Java Database Connectivity (JDBC) API. If you are already familiar with Object Database Connectivity (ODBC), you can think of JDBC as its offspring. JDBC is a subset of the functionality that has been delivered the last few years in a client environment. It allows for the same application portability while adding the same complexity ODBC adds. And it poses all the same challenges too. Like ODBC, JDBC uses dynamic SQL to work together with a relational database. JDBC will still face the same performance and security issues we have been facing with OBDC for years. Optimization will still be for SQL written to the least common denominator.

The question that must be asked is why is this better? Why not stick with what we now have? Even if Java gives us complete portability, do we actually need it? When was the last time you wrote a program to run in a Unix environment on a Sun box against an Oracle database, then suddenly decided you wanted to run that same completely unchanged application against DB2 on OS/390? Wouldn't you want to at least optimize the SQL for the platform you are now running?

What will make Java a prime time application development tool? Something called SQLJ, I believe. SQLJ is embedded static SQL for Java with full support for all DML, DDL, DCL, Commit, Rollback, and cursor processing. It will not, however, support

dynamic SQL, although you can code both SQLJ and JDBC calls in the same program. SQLJ has been proposed by Oracle, Tandem, IBM, and Microsoft (to name only a few) to ANSI for inclusion in the SQL standard. With SQLJ, a Java program is processed by a generic translator, or what we who use DB2 call the precompiler. All database vendors plan to use the same single generic translator. This adds actual Java code to the program, replacing the SQL calls. Now the entire program can be compiled into bytecodes, and a bind can be run to create a package for the SQL.

SQLJ has two significant advantages over JDBC. Foremost, it is static SQL. Because it is using static SQL and must be precompiled, it should give a Java application some improvement in performance. Because it is static, the SQLJ is also familiar. It has the appearance of the SQL we have been imbedding in applications for years. This will make SQLJ easier to learn and code than JDBC. Of course, you will still produce run anyplace bytecodes that use the JVM at the runtime destination. The bytecodes must still be interpreted at the server. How do you improve the performance of bytecodes? The latest innovation is the High Performance Java (HPJ) compiler. This compiler will take the Java bytecodes as input and produce an OS/390 load module. It will avoid the need to interpret the Java bytecodes using the JVM and, therefore, give you better performance by allowing Java programs to be used for applications where the performance of the program is critical to the application's success.

SQLJ should easily achieve widespread acceptance. Any vendor writing a package for multiple relational database systems can now use static SQL, via SQLJ, rather than ODBC, giving them improved performance while maintaining portability and database independence. IBM has announced support for SQLJ in DB2 UDB for OS/390 Version 6. But IBM is also considering adding support for SQLJ to the already available DB2 for OS/390 Version 5.

Anyone who still believes that Java is simply a web enablement tool may be missing one of the best development opportunities to come along in a long time. Java has all the potential. A key indicator of Java's apparent widespread acceptance, differentiating it from prior attempts by our industry to improve ways of producing applications, is the sheer number of vendors, their products, and the sophistication of those products that have appeared on the market in the last three years. Simply enter the word "Java" into your favorite web browser and watch the number of hits that are returned.

In just three short years, Java has accomplished what C++ still has not. The industry is at a stage where you cannot pick up an IT publication or browse a database or programming vendor's web page without reading something about Java or one of its components. I strongly recommend that you take time to learn more about Java. I think you will see Java become an integral part of not only DB2 UDB on the Unix and NT platforms, but also DB2 UDB for OS/390. Java will be one of the languages DB2 for OS/390 Version 6 uses for stored procedures and user defined functions. While JDBC and SQLJ are still in their developmental stages, we should see greater and more exciting things in the near future. As the old expression says, "They have the best people working on it."

If you are into the web (and well you should be if you are reading about Java), here are a few URLs that may help you get a little deeper into Java. The first URL is the announcement information for DB2 UDB for OS/390 Version 6 www.software.ibm.com/data/db2/os390/index.html . The announcement describes DB2's support for Java, JDBC, and SQLJ. The second URL points to detailed information from IBM about JDBC www.software.ibm.com/data/db2/os390/index.html . The final URL is the granddaddy of them all. It is the starting place and ultimate source for all to do with Java _ Sun Microsystems' JavaSoft web site www.javasoft.com/.

Willie Favaro has been a database professional for more than 20 years, the last 14 years primarily with DB2. He has been a software consultant for BMC Software, Inc. and a senior DB2 instructor for IBM. Favero is the author of numerous articles, a contributor to several IBM Redbooks, and a regular speaker at regional, national, and international conferences. He can be reached via email at willie_favero@compuserve.com.

JAVA: HYPE or HOPE?

By Craig S. Mullins

Unless you've been living under a rock, you've heard about Java and how it is going to transform the world of IT. But you know there's a lot of hype regarding anything that concerns the Internet. Is it all hype or is there hope for a brighter multi-platform world? Let's see what Java means to DB2.

What is Java?

First and foremost, Java is an object-oriented programming language. Developed by Sun Microsystems, Java was modeled after, and most closely resembles, C++. But it requires a smaller footprint and eliminates some of the more complex features of C and C++ (e. g., pointer management).

Java enables animation for and interaction with the World Wide Web (WWW). Although web interaction is Java's most touted feature, it is a fully functional programming language that can be used for developing general purpose programs (independent from the web).

Using HTML, developers can run Java programs, called applets, over the web. But Java is a completely different language than HTML _ and it does not replace HTML. Java applets are automatically downloaded and executed by users as they surf the web. The Java applet is run by the web browser.

What makes Java special is that it was designed to be multi-platform. In theory,

regardless of your machine and operating system, the Java program should be able to run. Many possible benefits accrue, because Java enables developers to write an application once and then distribute it to be run on any platform. Benefits may include reduced development and maintenance costs, lower systems management costs, and more flexible hardware and software configurations.

So, to summarize, the major qualities of Java are:

- its similarity to other popular languages
- its ability to enable web interaction
- its ability to enable executable web content
- its ability to run on multiple platforms.

Java Database Connectivity (JDBC) is an API that enables Java to access relational databases. Similar to ODBC, JDBC consists of a set of classes and interfaces that can be used to access relational data. Anyone familiar with application programming and ODBC (or any call-level interface) can get up and running quickly with JDBC.

Java, IBM, and DB2

Java for OS/390 is IBM's mainframe Java development environment. Using Java, developers can build web-based and general purpose business applications on the mainframe. Similarly, IBM provides Java support for its DB2 Universal Database (UDB) platforms, AIX and OS/2, (as do other operating systems suppliers such as Microsoft with Windows NT/95, Hewlett-Packard with HP-UX, and, of course, Sun with Solaris). With DB2, UDB developers can also code user-defined functions and stored procedures that run on the server.

In combination with the Java development environment, IBM also provides JDBC application support for Version 5 of DB2 for OS/390 and DB2 UDB. Using Java and JDBC, users can create applications that might otherwise be written in COBOL, PL/I, C, or C++.

The intended benefit of JDBC is to provide vendor-independent connections to relational databases from Java programs. IBM delivers this (as do most other RDBMS vendors), but from an industry-wide perspective, JDBC is still immature. This, coupled with other problems such as lack of support from Microsoft and slow performance, will probably limit JDBC acceptance and growth through the middle of 1999. But JDBC will most likely succeed and achieve widespread acceptance by the year 2000.

Hype or hope?

So, given this background, should we be wary of Java hype or full of Java hope? I think the best answer is "yes, both!"

Hype

Java is very simply the most hyped phenomenon since the Internet and the WWW. This does not mean it is without merit; just that its merits are oversold. There is simply no chance that 100 percent compatibility of Java across platforms will ever become reality. Differences will exist from platform to platform and system to system. Witness the recent events where HP delivered a Java Virtual Machine (JavaVM) for embedded systems that closely conforms to, but is independent from, the Sun standard. A JavaVM is required for each platform on which Java is to run. Key characteristics of the Java language require the presence of a substantial runtime environment, the JavaVM.

Differences will continue to occur because each vendor's offering is different and each requires differences to optimize their platforms. Sun has sued Microsoft to force it to remove the Java logo from its web site because its Java implementation does not conform to Sun's standard. Furthermore, vendors want differences; users are the only ones who want conformity.

There are other reasons why Java is not all it is cracked up to be. Java is an interpreted language. This makes it slower than compiled languages, so performance will be an issue. Many vendors (including IBM) are releasing just-in-time (JIT) compilers to react to this deficiency, but even a JIT compiler will be slower than a truly compiled and optimized language. Java is most useful for applications that require a high productivity development environment and high portability for the resultant programs. If instead the application requires maximum performance, platform specific processing, or the use of robust compiler technology, C++ (or another time-tested 3GL) will usually be preferable to Java.

Finally, there is a lack of infrastructure and tools available for the Java environment. Java is only three years old; organizations are not yet fully competent on how to implement, administer, secure, and maintain the Java environment. And vendors have not yet delivered robust tools needed to provide a robust application development lifecycle (testing, change management, debugging, implementation, etc.).

Hope

But none of these issues will sound the death knell for Java. Because of its many benefits, it is truly here to stay. Although 100 percent platform independence is unlikely, Java will get us closer than we have ever gotten before. At least there is a standard at the onset of the Java revolution that can be used as a gauge for successful or unsuccessful Java compliance.

And the list of current deficiencies will slowly evaporate over the next few years. Every vendor is working on Java tools to solve the infrastructure and development problems. This is true because more and more corporations are using Java to develop new and exciting applications. According to a 1997 survey by Forrester Research, 52 percent of

the Fortune 1000 firms surveyed were actively building applications with Java, and 81 percent of those plan to build Java mission-critical systems by 1999. The same survey also shows that using Java to drive database updates is increasing in popularity, with 54 percent versus 4 percent the year before. Java is being used currently and its use will continue to grow.

JIT Java compilers are getting faster and vendors are working on Java compilers that will resolve the speed issue once and for all.

One major benefit of Java applications is a lower cost of ownership compared with client/server applications. With client/server applications there are typically numerous DLLs scattered across machines that require modification, removal, or additional DLLs each time the application is updated. With Java, you have a better model because the code is downloaded when it needs to be run. Of course, this can impact performance when the code is being downloaded, but at least the application will be more likely to run correctly.

For DB2 users, Java is supported by Net.Data. With Net.Data and Java, developers can create Java applets to process the results of Net.Data applications and create graphs, charts, and other interactive elements. As Java matures, it will become the way you write stored procedures, user-defined functions, and triggers. Oracle, Sybase, and Informix stored procedures are based on proprietary SQL; DB2s rely on 3GL programs. Java stored procedures have the promise of running securely in any database _ this offers a potentially huge benefit.

Eventually, developers will be able to embed SQL directly into Java programs, call Java programs from SQL statements, and even store Java object instances into DB2 columns. All of these things will enable users to create active DB2 databases. This is good news, because it places the responsibility for taking the proper action in a single, central, secure place in the database.

Finally, Java's ability to make the web interactive will ensure the success of Java. Inasmuch as we as DB2 developers want to enable our users to access DB2 data over the web, we will rely on Java to help us do that.

Synopsis

There is enough hope for Java that we should all be able to put up with the hype that is out there. With time, patience, and a little luck we will have web-enabled, multi-platform, distributed Java applications accessing our active DB2 databases.

Craig S. Mullins is vice president of marketing and operations for the database tools division of PLATINUM technology, inc. He is also the author of the popular book, DB2 Developers Guide, now in its third edition; the book includes tips, techniques, and

guidelines for DB2 through Version 5. He can be reached via email at cmullins@platinum.com.

About IDUG | Conferences | DB2 Resources |
Regional User Groups | Solutions Journal | Vendor Directory

Home | Contact Us

International DB2 Users Group
401 N. Michigan Ave.
Chicago, IL 60611
(312) 644-6610