The Future of Notes and Java

by Bob Balaban

[Editor's note: This article resides in "Notes Today", the technical Webzine located on the http://www.notes.net Web site produced by Iris Associates, the developers of Domino/Notes.]

Notes is a Client-Server Development Tool

As our many business partners have already discovered, Notes is a fantastic tool for developing all types of distributed client-server applications. Lotus delivers the infrastructure software in the Notes box, together with some get-started and sample templates, while our partners develop workflow, customer tracking, sales lead processing, knowledge base, and other kinds of collaborative applications too numerous to mention here.

One of the keystones for supporting this kind of development effort is the ability to customize and add value to the basic Notes tools through programmability. Over the past few years, programmability has been extended from @functions to full-blown LotusScript with access to Notes objects.

This article describes the next generation of Notes programmability, scheduled for roll-out this year: Java application support for server-based and Notes client execution, to be followed by CORBA-based Java applet integration with all Web browsers.

In The Beginning: C Language Access and @Functions

Notes Releases 1 through 3 offered power users @functions to customize the Notes UI, write macros, and perform other data processing. In those days, if you hit the wall with @functions, you wrote C programs using the Notes C API toolkit.

While people were able to do truly amazing things with these tools, the @function language lacked some features common to friendlier programming environments, such as looping constructs, sophisticated branching and error recovery, and a debugger, and let's face it, not everyone wants to be a C programmer.

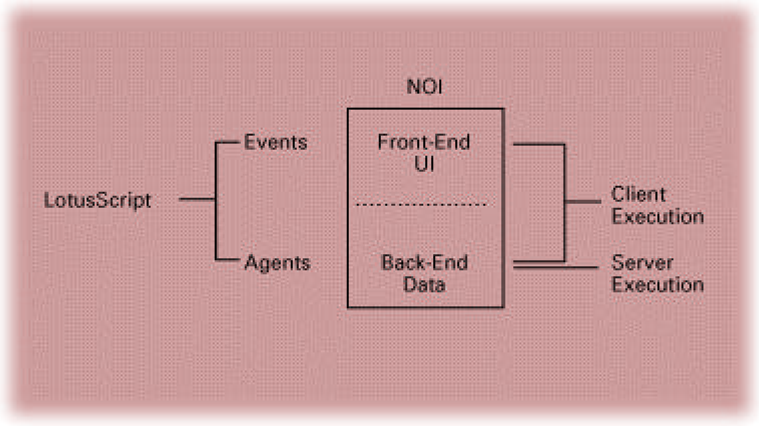
Then Came LotusScript and NOI

To fill some of these gaps for our developer community, we introduced LotusScript in Notes Release 4.0. Essentially an object-enhanced version of BASIC, LotusScript allows you to attach programs to events on databases, views and forms.

If you want some logic to execute whenever a document using a particular form is opened, you simply bring up the LotusScript Integrated Development Environment (IDE), locate the PostOpen event on the form, and enter your script code. The IDE includes a debugging facility as well, so you can single-step your program and test it out.

Furthermore, you can write "agent" programs. Agents are somewhat different from event-triggered programs in the user interface. Form and view-based, event-triggered script programs are generally used to manipulate the Notes UI. They run in the user's face for the most part.

While an event-triggered script can invoke an agent program, agents can also be set up to run on servers where no Notes client is present. Agents can be scheduled to run at particular intervals (hourly, daily, monthly), and when triggered by server events. This might include new mail arriving in a database, or documents being created or modified by other processes in a database.



Of course, offering a scripting language in Notes or other products doesn't support enhanced application development by itself: what can you do with it besides add numbers together or put up message boxes? Developers must be able to manipulate the entire product environment. The integration of scripting with the Notes Object Interface (NOI), beginning with release 4, meets this requirement

The NOI is conceptually divided between the UI, or "front-end" classes, which are used to manipulate the user interface, and the server, or "back-end" classes, which exhibit no UI at all. Of course, the back-end classes can be incorporated with front-end classes in UI scripts, but front-end (that is client-centric) classes can't be used in server-based agents.

The front-end classes were enhanced for version 4.5 and now include NotesUIDatabase, NotesUIView and NotesUIDocument. Each of these classes contains methods you can invoke, properties you can query or set, and events to which you can attach LotusScript code.

The back-end classes include most of the Notes objects already familiar to long-time users: NotesDatabase, NotesView, NotesDocument, NotesItem and NotesEmbeddedObject. Each of these classes also contains methods and properties (as do all others in the object hierarchy), but none supports events. Where the front-end classes are used primarily to manipulate the Notes client UI, the back-end classes are used primarily to manipulate data objects.

What Does All This Have To Do With Java?
Programmability is the name of the game with LotusScript and @functions. Programmability is how we at Iris and Lotus build the standard templates we ship with Notes, and also how the very large third party developer community adds value to the basic product.

Then along came the Internet and, especially, the World Wide Web. Suddenly, everyone (it seems) is using browsers as data viewers instead of proprietary client software. Browsers read a rich text format called HTML served up over TCP/IP connections via the HTTP protocol.

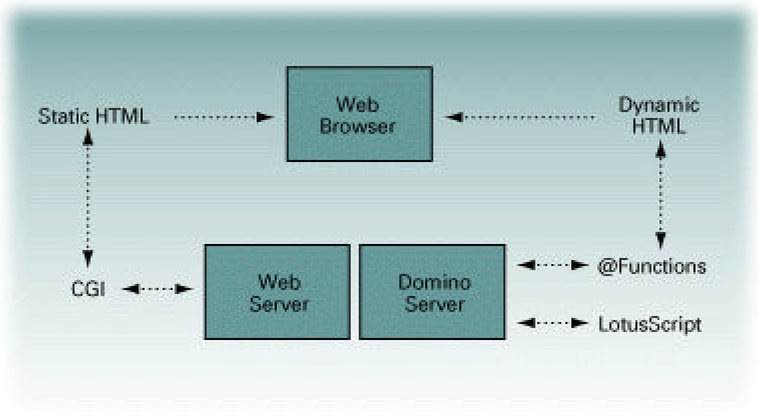
Those of us who have been spoiled by real client-server interactive application environments expect to customize the content of the data we publish based on workflow state, who the user is, and on business rules.

However, HTML by itself is little more than a rich text format specification. Consequently, most web sites today publish "brochureware", that is, static text. Documents just sit there. Everyone sees exactly the same data presentation. Or, at best, some simple animation, spinning heads or flickering logos offer some distraction.

When Sun announced Java a little over a year ago, it seemed at first blush to be little more than just another cute programming language. Suddenly it appeared as though everyone was animating web pages with Java snippets, excuse me, I should say, applets.

But with the explosion of interest in (and use of) the Internet, it quickly became apparent that Java had some real assets. First, it is a truly portable language and is available on almost all computer platforms. Second, Java comes with some very good class libraries which are tuned to making programming for the Internet easier. Finally, Java's networking and TCP/IP support is particularly strong.

Equally rapidly, it became clear that Java is a also natural fit with Domino/Notes. We already had an HTTP server as part of Notes. With Release 4.5, people realized the power of Notes for developing dynamic Web-based applications: @function formulas can generate HTML content based on who a user is, or based on what they're doing. Forms can be designed and programmed in the Notes environment, and the data submitted by the browser client can be automatically stored in a Notes database. LotusScript agents replace clumsy CGI scripts, and the agents are stored in Notes databases as well, with replication to new sites. Suddenly, a real, robust Web application development platform is available.



But what about Java? Wouldn't it be nice to use Java to turn dumb old Web browsers into real interactive clients? Sometimes it seems as though we have regressed back to the days of 3270 "dumb terminals", only they look a lot nicer now. Did we progress painfully to the point where we finally had really nice client-server applications, only to lose it to the World Wide Web? Nope.

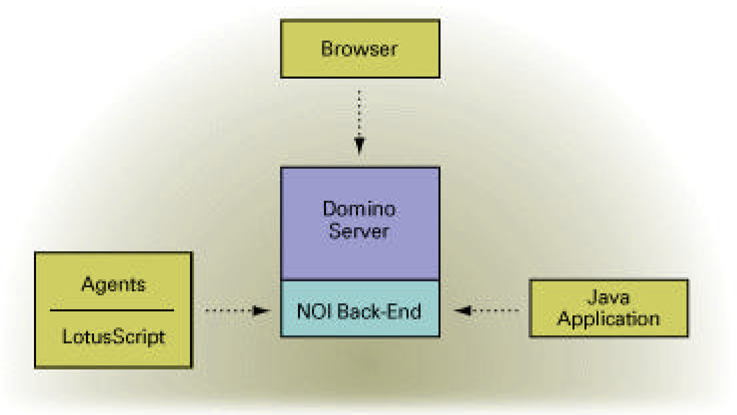
Unfortunately, neither Microsoft's Internet Explorer nor Netscape's Navigator browsers supports LotusScript. If that were the case, the Domino server could send out the LotusScript code we're already using, and scriptable forms would be immediately available in the two most popular browser products. But, it didn't happen that way. Neither Microsoft nor Netscape was interested in adopting LotusScript.

So, the basic reason why we care about Java is this: we can use it to turn dumb ol' browsers into smart Notes clients. With Java, our smart, server-based applications can reach out and tell generic browser clients what to do. Programmability.

How Do We Program Domino With Java?
The plan is to deliver Java programmability for Notes developers in two phases during 1997. Phase One will take the LotusScript back-end classes and provide a Java interface to them. You can think of this as enabling you to write a Notes API program using Java. The Java program will execute on a machine where Domino is installed.

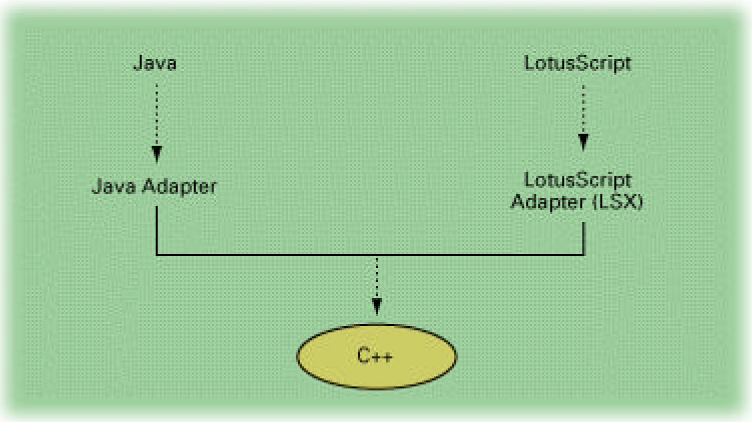
You will be able to write Java *applications* that manipulate Notes objects, but not Java *applets*. What is the difference? Applets run within browsers, with the Java code downloaded from the HTTP server in response to invocations and applet parameters embedded within an HTML page. Java applications, on the other hand, are standalone programs launched by pointing the Java interpreter at them. There are fewer security restrictions, since typically only trusted applications are installed on the user's machine.

Using this Phase One deliverable, you can write Java applications that make use of all the back-end objects to which you currently have access via LotusScript. You will also be able to import these Java apps into a Notes database as agents. Everything that can be currently performed with LotusScript agents (scheduled execution, appropriate access control, etc.) will be available using Java.



One important point to remember here is that the Java interface to the back-end classes is not a port of the LotusScript classes to Java. The classes are implemented in C++, and are exposed to LotusScript via the LotusScript Extension (LSX) architecture. This is essentially a LotusScript adapter interface that converts LotusScript calls into C++ invocations.

By simply modifying the Notes LSX containing the back-end classes to also include a Java "adapter", we are able to entirely reuse the existing C++ behaviors for Java. This means that there is no semantic difference between the way the LotusScript and Java classes behave, because internally the exact same code is being executed in both cases. It is only the syntax of the interfaces themselves that are different (necessarily so, because LotusScript and Java are two rather different languages).



But What About Browsers?

Phase One requires your Java program to run on a machine where Domino/Notes is installed. Consequently, you can't use the Java classes in an applet. But don't despair: Phase Two of our Java related product offerings (also targeted for delivery in 1997) will provide a remote interface to the Java classes delivered in Phase One.

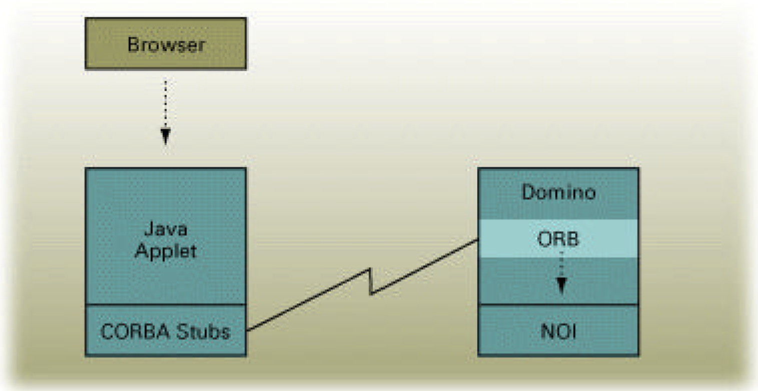
With a remote interface to a collection of Notes objects, you can write Java applets stored on a Domino server. Those applets can leverage the tremendous power of the Notes back-end classes to create, store, retrieve and otherwise manipulate data in Notes databases. These applets can be linked into HTML pages, downloaded to any Web browser, and launched there. This technology at last brings true client-server programmability to the Web, and also highlights Java's usefulness. An applet written in Java on any Web server will run in any browser on any other kind of computer.

How are we going to accomplish this? Writing a Java interface to some C++ classes on a Notes machine is straightforward, but how do we allow Java programs downloaded to a browser to talk to Notes/Domino objects that are really resident on that server, and request the needed services?

The answer is provided by the Common Object Request Broker Architecture, or CORBA. CORBA is not a product but a specification defined by the Object Management Group (OMG). OMG is a consortium of companies, including IBM, Lotus, Netscape and over 600 other members. The CORBA spec describes how to implement objects so that they can be shared across machines, even when the machines involved are running disparate operating systems.

Think of a CORBA-ized Domino server as containing a collection of Notes objects (the famous back-end classes), each of which implements an "interface" (that is, a set of methods that a Java program can call). The beauty of CORBA is that it tells you how to implement those interfaces to make them remoteable: they can be called from another machine. The objects "live" on the server, but a client program (an applet or application) can bind local Java variables to those objects and execute methods on them.

How Does CORBA Work With Domino?
A client-server conversation which complies with OMG's CORBA architecture will consist of three main pieces: the client Object Request Broker (ORB), the server ORB, and the protocol the two ORBs use to talk to each other.



Both the client and server ORBs are generated from the same object definition, though the client ORB might be implemented in one language and the server ORB in another. In order to define our objects so that they can be implemented with identical behavior in many different programming languages, we use the Interface Definition Language (IDL).

IDL is also part of the CORBA spec and provides a language-neutral description of a collection of objects. Given the appropriate IDL for the Notes back-end classes, an IDL compiler tool (usually provided by your ORB vendor) automatically generates a language binding for the object collection.

For example, since we want to implement the server-based Notes objects in C++, we use an IDL compiler that transforms our IDL descriptions into actual C++ code. Of course, the generated code only forms a skeleton implementation. We take that framework and write (or glue in) the code that actually performs the various pieces of functionality we care about.

For the client side, we can use a different IDL compiler to automatically generate the Java stubs which will be part of our applet. The stubs package method arguments in a standard way for each method on each class that we have defined in our IDL. They also use TCP/IP in a standard way to communicate method invocations to the corresponding server-based objects. Thus, each server object that you instantiate remotely has a unique, corresponding stub on the client machine that knows how to invoke methods on it, and receive the results.

This brings us to the third piece of the architecture: the conventions, or protocol, that the client stubs and server skeletons use to talk to each other over a TCP network. This protocol is called IIOP (Internet Inter-ORB Protocol). The IIOP spec defines the messages that pass back and forth between ORBs to invoke methods, pass arguments, and receive results.

The really cool thing about all this is that not only does this allow you to write Java applets that talk over the network to server objects implemented in C++, but the client and server ORBs can even come from different vendors. So long as both use IIOP according to the spec, they are able to cooperate effectively.

So, Now I Have to Learn IDL too?
No, not at all. The designer of an object model that will be exported via CORBA ORBs does need to know IDL to define the objects. But if, as with the Notes classes, you have Java-based client stubs, you simply write Java applets that make calls to the stubs. The stubs know how to deal with the corresponding server objects. The programmer doesn't have to know or care what language the server objects are implemented in.

If the object vendor publishes the IDL, you could even take it and use your own IDL compiler to generate new language bindings for the objects. You could also generate a set of server-side skeletons for the objects, but you wouldn't be getting anything for free: you would still have to write the code that actually implements the behavior that the objects are supposed to exhibit.

One More Time: Why We Care About Java
We at Iris and Lotus have become convinced that Java is central to our programmability strategy for Domino/Notes. Interest in Java has exploded along with interest in the World Wide Web. Java has revealed some of the exciting possibilities of the Web as an environment for truly interesting software applications.

Because of its wide acceptance and platform portability, we are convinced that the combination of the Notes Object Interface (for basic application building power) with a Java programming interface (for portability and ease of implementation) will yield amazing power and benefit for our developer community.

Add to that the ability to remote the Notes Object Interfaces through CORBA and IIOP and we have a package that introduces true client-server application development tools to the Internet. No more brochureware.

As we ship the Domino releases that support Java, you will be given the ability to use the power of the Notes infrastructure (structured object store, security, replication, full text searching and programmability) in combination with the ubiquity of the Internet to build all kinds of new workflow, collaborative and other extranet applications.

ABOUT THE AUTHOR

Bob has been working on NOI, LotusScript and agents in Notes for almost four years, initially as a Lotus employee, then for the past year as a member of the Iris team. Before Notes, he worked at Lotus on spreadsheets and other products, earning a U.S. patent for the Version Manager feature in 123/W. Bob holds a BA degree from Antioch College and an MA from the University of Michigan, both in Asian Studies, and was a recipient of the 1996 Lotus Commitment Award. Read more about **Bob in his "Iris Interview."**