The Need For Three-Tier Architecture

By Greg Grisham

Who needs a three-tier software architecture? You do, and maybe you don't. We tout the fact that our new product, BBj®, can support a three-tier design. But why is that, or should that be important to you? After all, Business BASIC developers have created superior applications for years without needing a third tier.

What Are The Three Tiers?

Let's start with a brief overview of the three-tier design. This is relatively new when you consider that it came into being in the mid-to-late 90's. Effectively, what this design constitutes is a separation of the following:

- User interface (thin client),
- Business processing (application logic)
- Database access

The three-tier architecture is used when an effective distributed client/server design is needed that provides increased flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. These characteristics have made three layer architectures a popular choice for Internet applications and net-centric information systems.

So, the concept of the three-tiers offers the developer a number of things:

- Distributed computing power
- Heterogenous operating systems
- Very light communication demands
- Overall scalable infrastructure

Should Everything Be Three Tier?

With all of those features, why isn't everyone jumping on this bandwagon? Because, it isn't always necessary or even desirable to have those features, due to the potential for increased complexity and decreased performance. In fact, what can be a feature in one application can be a flaw in another. Take a small, portable application that a developer has written for traveling salesmen. This application is self-contained, and has a "one-step" installation. To re-design it for a three-tier architecture, requiring a network connection and complex configuration options, would just not make sense. Furthermore, in this example, the application would be useless if the laptop upon which it was installed couldn't get access to the Internet.

Legacy Conversion

Typically, migrating a legacy system to a three-tier architecture can be done in a manner that is low-risk and cost-effective. BBj facilitates this by maintaining the old database and business logic rules, so that the old and new systems will run side by side until each application and data element or object is moved to the new design. But just getting the code operational may not be enough. There are some potential issues in this environment when coming from a legacy design.

Application Issues In A Three-Tier Design

If an application is executed in the immediate environment of the interpreter, a number of things are within the developer's reach, including any device connected to that system. In a three-tier architecture the developer needs to consider three environments. It starts with the thin client, or user interface, and what other applications might be involved on the desktop that could potentially affect performance. Next, comes the application server and where it is located. Last, is the data server. The concept of a detached data server is powerful, yet like the program issues, multiple data servers could exist in the network and the application must be sensitive to this. Generally speaking, in a thin client environment, the programmer shouldn't depend on access to the client's devices.

Device Locations

Printers, scanners, modems and additional hard drives are just some of the examples of hardware accessible to the developer, and therefore, to the application itself. Consequently, an OPEN on "/dev/lpt1" or a "Sysprint" device, can be expected to open a printer connected to that particular system. In a three-tier configuration, to open a printer, the application must be sensitive to the printer's location. A printer on "/dev/lpt1" could very well be a thousand miles away when you really wanted the printer sitting next to the user. Code can be added to print jobs specifically for "client-side" printing. In BBj, the printer is assumed to be connected to the server, but with the addition of "MODE=CLIENT" the developer can direct the output to the local printer.

System Calls (SCALL) From A Thin Client

BBx® allows for a direct system call from within a program via an SCALL. This has been used by developers for a number of operating system level functions. In a distributed environment, applications utilizing this functionality need to be revisited. If the particular function is reliant upon specific environmental conditions, it will be especially subject to problems in a three-tier configuration. In BBj, an SCALL can function as before, but it will communicate with the host operating system where the INTERPRETER is running (i.e. the application server host). This is fine, if that is where you anticipate it being executed. Much like with the printer control, BBj offers the ability to make an equivalent system call in the client environment through the BBjThinClient object. Below is an example of how a local execution of an external application could be executed:

```
REM Run an application on the client in a thin client environment
REM Obtain the instance of the BBjAPI object
LET myAPI!=BBjAPI()
REM Obtain the instance of the BBjThinClient object
LET myThinClient!=myAPI!.getThinClient()
REM Run notepad on the client's machine
myThinClient!.clientExec("notepad.exe")
```

It should also be noted that BBj will support synchronous as well as asynchronous execution (i.e. in asynchronous the interpreter will not wait for the completion of the task). A command can be made asynchronous by adding an "&" at the end of the command line.

Local Drive Access

Another consideration is any legacy code that made assumptions to access a local drive. A common coding technique would create a local work file, or possibly write to a backup drive. This can all work in BBj with a three-tier construct, but it will have to be reviewed and modified to ensure the operation is performed on the system for which it's meant. Some path syntax that could assume location are particularly suspect for review. A relative path entry like "..\data" is okay when you are sure of the system you're on and the directory you're in. Some logic may need to be added to confirm these details and potentially the data server as well.

Three-tier application design opens up a whole new world to Business BASIC developers. But, as with any new functionality, it comes with a list of considerations to weigh before "jumping in". BASIS has worked diligently to make the conversion to BBj as seamless as possible, but we realize that this is a new, and very dynamic dimension for existing application logic. We will continue to build functionality into the language to keep you competitive, and all the while alert to the nuances that new technology brings.