



Moving PowerBuilder® to EAServer—Design Considerations

A detailed look at design issues around moving PowerBuilder applications to a three-tier environment on EAServer.

So you're moving to the Web? Well, a move is always exciting and brings new opportunity. But a move has challenges as well (you don't want all your stuff to go flying out the back of the truck). You probably already know why you want to go to the Web. This document focuses more on how to move to the Web and what to watch out for. Taking a PowerBuilder application to the Web can be tricky business—even trickier than moving that sofa bed down the stairs. This document offers an overview of how to make your move successful and how to think like a Web programmer.

Guiding Principles

When moving an existing PowerBuilder application to the Web there are a few guiding principles that should be kept in mind. The first and foremost of these is that you need to be flexible. If you want your Web application to look exactly like your PowerBuilder application then you may as well not even begin. The reason for this is that the browser environment is a completely different platform and demands a different way of thinking. Be realistic in your expectations. For example, PowerBuilder applications usually contain numerous complex screens. During the conversion process, these screens will likely need to be broken down into simpler screens or at least have their functionality simplified. What this means is that you'll need to look at doing processes differently if your Web application is going to be successful. Go into the development process with an open mind and remember, your current interface does not need to be on the Web, your business functionality does.

Once the mantra, "Be flexible, be flexible" is sounding in your mind, consider another, more concrete guiding principle: the more separation that exists between the presentation logic and the business logic, the easier it will be to take an existing application to the Web. Specifically, if a PowerBuilder application's business logic is in non-visual objects, you are better positioned to move the application to the Web. If all the business logic is tied to data, you will have more difficulty. Generic business functions will move to the Web nicely but if business logic is intermingled with presentation logic, serious consideration should be given to reworking the application before migrating. A good place to start reworking an application is to move business logic into EAServer by creating PowerBuilder components. Migration to EAServer is more easily handled by moving code via the cut/copy/paste method as opposed to moving entire objects. This approach is much easier than a complete rewrite because, assuming the application has been in production, the code has been tested. Even if you can't use a whole object or function, use pieces of business logic from your current application wherever possible. By doing so, you'll reduce the number of bugs in the new application.

The third and final guiding principle is not required but deserves serious consideration: hire a consultant to serve as a mentor in getting an initial project up and running. By hiring a consultant, you'll be more likely to have one successful project up and running quickly. You can then model future projects off of that. Questions you might ask yourself in determining whether or not a mentoring consultant is a reasonable expenditure would be: How many developers will be on this project? How many developers have Web experience? How many of them feel comfortable with the new architecture? If you lack expertise, think about bringing in a mentor for the project. Even a mentor for phase one can greatly increase your chances for success and will give your in-house developers time to get familiar with the new Web environment.

Analyzing the Existing Application

Now that we've touched on some of the guiding principles of moving a PowerBuilder application to the Web, let's look at some of the nuts and bolts of making the move. The first thing that needs to happen is an analysis of the existing client/server application.

Analyzing the PowerBuilder Client

Start by looking at the rich features used by the existing client interface. Things such as tree views, tab pages, and other complicated graphical components can pose significant challenges when

moving to the Web. Tab pages and tree views can be rendered on the Web, but the application will be much more complicated. Look at using other methods to keep your initial project simple and straightforward.

Web programming is limited when it comes to the client interface. One place to consider revising your code is the `itemchanged` event. Client-side JavaScript™ and page refreshes are the two basic ways to process this type of logic. JavaScript can add complication to your Web application and should be kept manageable. You should not rely on JavaScript to handle business-critical processing. JavaScript should be used for simple tasks and simple validation and will save users from having to make a round-trip to the server to figure out that data they have entered is invalid. Validation can be duplicated on the server or at the database as well to ensure validation logic is enforced in all scenarios. But be aware that different browsers interpret JavaScript differently, so you have less control over the processing that occurs in JavaScript as contrasted with server-side code. Also, heavy usage of JavaScript will complicate your development cycle as new browser versions are released and additional testing is required. Server-side code, in contrast, runs in an environment over which you have complete control.

Page refreshes, like JavaScript, should also be used with restraint. From the user's perspective, page refreshes can be slow and annoying. For example, a user will become irritated if a page refreshes whenever they tab off of a data element. Only refresh pages at logical processing points, such as when a button or link is clicked.

As stated in the first section, presentation logic should remain on the client while business logic needs to be moved into EAServer so that it can be re-used on the Web. EAServer gives you the ability to use one set of code for both PowerBuilder clients and Web clients. A good rule of thumb for developing code in EAServer is to keep components generic. Components should do their processing and then return standard PowerBuilder data types. For example, when coding a retrieve of customer records and returning them to the client, develop a component that performs the retrieve and returns a datastore. Then, an interface component can be developed that will determine how the data is returned to the client. Using this approach, the business logic happens in one place, the generic component. As for the interface logic, it can often be re-used for many components. One interface component will handle calls from the Web by getting the datastore from the generic component and converting the datastore into a string of HTML. This conversion can be done using custom code or with EAServer's HTML DataWindow™ builder. If needed, a second interface component can be coded for PowerBuilder clients. This component will get the same datastore from the same generic component. However, instead of generating HTML, it will return a format designed specifically for the PowerBuilder client (i.e. a blob from a `getfullstate` call).

The end result of separating the business and presentation logic is that if a change is required to an application's business logic, the code can be maintained in one spot, the generic component. Likewise, if a change happens to the presentation logic of the Web interface, changes can be implemented easily without impacting other clients.

Analyzing PowerBuilder NVOs

After examining an existing application's client-side code, move on to an analysis of its non-visual user objects. What you're trying to determine is how well the existing code is set up for conversion into components. Are NVOs being used for business logic? The more that they are, the easier it will be to move code to EAServer. Is the application's business logic tied into the presentation logic? If so, start thinking of ways to separate them. The greater the separation, the easier it will be to move the application to the Web.

Analyzing PowerBuilder DataWindows

EAServer has a built-in HTML DataWindow generator. Using HTML DataWindows, traditional DataWindow functionality (retrieve, update, insert, delete, update) can be moved to the Web quickly and easily. Try migrating a few simple DataWindows. Once users are comfortable with the Web interface, add more functionality. Because HTML DataWindows look like basic client/server DataWindows, users will acclimate quickly to the new Web interface.

HTML DataWindows work with a dataobject and the HTMLGenerator component in EAServer. A simple Powerbuilder component in EAServer can use the HTMLGenerator to construct a string consisting of HTML and JavaScript to return to the client. The HTML DataWindow uses JavaScript to handle the client-side processing that is needed for basic DataWindow functionality. The JavaScript is handled by EAServer and does not need to be developed or maintained by a developer. Even basic item validation can be set up in the dataobject and will be converted to JavaScript. This allows for Web programming without having to learn Java or JavaScript. Developers can simply apply their current PowerBuilder knowledge while working in a familiar environment.

It's important to understand that HTML DataWindows function in a stateless environment. With the Web, having stateless components is essential. Since your application is open to the world, you must be prepared to handle an unpredictable number of users. Stateless components are the key to making this possible. HTML DataWindows use stateless components. The client-side JavaScript handles the changes by the user. When the DataWindow is refreshed via an update, insert, or delete, the current state of the DataWindow is passed back to the component as a string. This allows each call to the server to be independent and allows for stateless components.

Being able to migrate formatted DataWindows and the SQL that drives them is a huge benefit when migrating to the Web. Thanks to EAServer's HTML DataWindow generator, existing client/server DataWindows can be seen as an asset rather than a relic.

Analyzing PowerBuilder Reports

Reports are a great first step to the Web. The Internet was first designed to do exactly what reports do—display information. Reports can be converted to HTML either with an HTML DataWindow or with custom code. While the HTML DataWindow offers the fastest route to viewing reports on the Web, writing custom code provides greater flexibility for formatting how a report will appear in a browser. If your business rules dictate a very specific format, writing your own code might be best.

Writing custom HTML is best done in steps. First, the HTML template should be designed. Take some sample data for a report and come up with a static HTML page. This can be done using an HTML editor such as FrontPage® or Dreamweaver.® Once the HTML design is hashed out, the second step is to set up the data to come from an EAServer component. The HTML that needs to be dynamic can be cut and pasted into a function. Normally, the component will retrieve a datastore and then loop through the rows. As it loops, the data will be captured from each row and placed into a string variable that incorporates both the data and the HTML. Next, convert the HTML page so it can call the EAServer component. The page can be saved directly as a JSP page, since JSP pages support HTML. The dynamic pieces that have been removed will now be replaced with a call to EAServer.

This process will leave you with code in EAServer that is tied to HTML. If you are serving multiple types of clients, you will want to split this work. Have one component retrieve the data and return a datastore. Then a Web-interface will do the conversion to HTML. This will allow a PowerBuilder client (or another client) to use the same data retrieval logic. Designing reports in a systematic method such as this will allow you to create an eye-catching graphical Web page filled with valid dynamic content.

Analyzing PFC and Ancestor Class Usage

The PowerBuilder Foundation Class (PFC) is a feature-rich, heavy architecture that was designed to give lots of functionality to a PowerBuilder client. It was not designed to be run in EAServer and should not be used as an ancestor class for server-side components. It is good, however, to have an ancestor class for your server-side components. This ancestor class should be lightweight to run efficiently in a scalable, stateless environment. Having an ancestor class will allow you to maintain common code in one location. Logic that should go in an ancestor class includes:

- **Database Connection:** The base class should contain logic to pick up a connection from the EAServer connection cache. It should also contain logic to pick up the database parameters.

It is a good idea to put database parameters in the package properties of EAServer or another method that will allow you to move your code from development to testing to production without rebuilding your code.

- **Server Logging:** Logging is a critical part of developing and debugging EAServer applications. Developers will save a lot of time by setting up a logging function in the base component. When you set up the logging function, include the message severity as an argument. This will allow you to set up logic to turn debugging messages off when you move to production.
- **Common Business Logic:** Be careful with this one. You only want functionality that is truly universal to be placed into the base class. The size of the base component will affect the size of every single component, so it should be kept as small as possible. If you have some functionality that is used a lot, but is not universal, think about adding a layer of inheritance to the framework. Develop a base object with universal code. From that object, you can inherit and develop more specific base objects such as one for accounting and one for customer. These objects can then be used to store functionality that is common across accounting components or customer components, but not used across the entire application.

Developing your own ancestor class will give you flexibility to make efficient changes down the road.

Setting Expectations

Before you begin work on migrating a PowerBuilder application to the Web, it's important to set realistic expectations for the task before you. As stated in the Guiding Principles, you need to be flexible. Your Web application will not look and act the same way as your PowerBuilder application. Web development uses a different architecture and a completely different way of programming. If you end up with an application that looks exactly like your PowerBuilder client, then you haven't utilized the Web properly.

Work with the Web's strengths and stay away for the Web's weaknesses. The Web is well suited for widely distributing an application to a countless number of users. This takes away the terrible chore of maintaining the client. You can also release new code at any time and not have to worry about deploying the code to every client. Simply change the code on your server and it's done.

By opening your server to the in-house intranet or the Internet, your application will become available immediately. With this increased audience and the limitations of a browser, it is essential to keep your application simple. Check out other sites and get a feel for what is intuitive as a user and what is not. The best sites are not fancy—they stick with what's familiar to users.

There are many ways to make your application intuitive and easy to learn. Make sure that your screens are well documented to help the user. Don't assume that people will look at a screen the same way you do. In fact, with different browsers and varying screen sizes, a page might look completely different on another user's machine. Because of this, keep screens simple and straightforward.

When setting your expectations, keep in mind that you'll be giving up control of the PowerBuilder client and, therefore, most of the control you've grown accustomed to exercising over the client side of your application. All is not lost though. You can still control some significant aspects of your application. For example, you can restrict your site to certain users by having them enter through a log-on screen. You can require a browser to support JavaScript and/or frames in order to view your site. You can also write JavaScript to detect what environment you are currently running in and write code for specific browsers or operating systems.

The loss of client control is more than offset by the control you gain by having all your code on the server. Code for the entire application is now accessible as it never was before. No more updating clients and no more worries that someone is running an older version of the application. Every time a user runs the application, they are pulling down the latest code with each page request.

Designing a Solution

Your final Web solution can be a combination of existing EAServer components, new components and Web components. The new EAServer components will be made up of code from your existing application while the Web pieces will be new. JSP pages will be used to display the content to the user and a Java Bean can be used for code re-use, tracking session information and caching data. Moving to the Web is not an easy, straightforward process. PowerBuilder applications can be successfully migrated to the Web, but it is a complicated task. This sections talks about common ways to set up pieces of your Web application.

Thinking about Components

When developing on EAServer, the number and size of components is important. For example, if there are too many components, EAServer will expend too many resources maintaining the instance pool and searching for components when they are requested. On the other hand, if components are too large, EAServer will bog down loading them in and out of memory. If you have a large application that needs a lot of components, multiple servers can be used with only a subset of the components on each box. Or perhaps the application can be pulled apart into smaller ones, with a server or cluster of servers dedicated to each application. The technique of designing components is somewhere between art and science. There is no preset formula for what will work best. The only way to know for sure is to load test your code and adjust from there.

If there's any possibility an application may need to run on a server cluster, take time up-front to design the cluster architecture. If designed properly, one EAServer box can typically handle about 100 on-line users. Contact Sybase or a Sybase partner to get a better of idea of how many users a server will be able to support. This will depend on many variables, but having an idea before development begins will help in the creation of a realistic budget and ultimately lead to a better application.

EAServer is a powerful ally when developing applications for the Web and should be allowed to pull its own weight. Once you get your code running in EAServer, let EAServer do what it's made to do, namely, manage the connection cache, manage component lifecycle, and manage transaction support.

EAServer is good at maintaining connections to databases and serving them up to components as needed. You will need to set up the connection cache and set up your components to connect to it. Make sure your connection cache is set up to have enough connections to handle your load. After that, avoid the temptation of coding your own connection logic. Let EAServer manage those connections and maintain them.

EAServer was designed to handle the creation of components and perform component life-cycle management. To leverage this functionality, be sure to code for stateless components. To do this, each function should perform all the processing needed for a given task. You cannot, for example, call function #1 and then call function #2 and expect the component to remember the first call. In a stateless environment, you may not get the same instance of a component for each call. Functions must therefore be written as complete units of business logic.

Once components are written in this manner, a small number of them will be able to handle a relatively large number of clients. This is because a component will only be tied to a given client while it handles a request. Once the function returns, the component will be returned to the instance pool and will be made available to service another client. In this way, EAServer is saved from having to constantly create and destroy components. This enables EAServer to devote more resources to processing business logic.

Thinking about Beans

Java Beans are a powerful tool for managing state and session, as well as for invoking methods from EAServer components. The use of session Java Beans offers many advantages. The three biggest ones are code reuse, storage of session data and caching data.

Using a session Java Bean will give you the ability to reuse your Java code. Many tasks that happen from the JSP container (connecting to EAServer, verifying that the user is logged on, etc.) need to be stored in a common location for maintainability. Without a Bean, this code would need to be stored in every JSP page. If a change is made in your connection logic, you might need to change hundreds of JSP pages. By using a session Java Bean, common code can be stored and accessed easily and efficiently from any JSP page.

A session Bean also allows you to maintain state in a stateless environment. The Bean can be used to track users as they move through an application. Since EAServer components are stateless, a session Bean can be very handy. Has a particular user logged on? Have they logged out? Have they already viewed certain data? All of these pieces of information can be stored in the session Bean. The session Bean creates an instance for every user, so user-level information can be handled by the Bean. The Bean also allows sensitive data to be stored in one place rather than passed around from page to page. Remember, passing user IDs or passwords from page to page is a security risk.

Finally, a session Java Bean enables an application to cache data. If you have data that is relatively static, the Bean can perform the initial retrieval and then hold onto it for future requests. A good example of this would be a list of departments in an organization or a menu that is used on every page. By caching the data, an application will save itself from making redundant calls to a database. This, in turn, will allow a Web application to run faster and be more scalable.

Thinking about the Web client

As you've no doubt figured out by now, developing for the Web is different than developing for a PowerBuilder environment. Give yourself time to become comfortable with a new way of doing things. As you begin developing for this new environment, there will be many questions. Some of the larger, more obvious ones should be addressed before you begin:

Where are my events? The Web model has fewer events than PowerBuilder (what doesn't?). PowerBuilder programming is built around the event model. On the Web, there are far fewer events. Most of your processing will occur when the browser makes a request for a new page. Smaller events can be coded with JavaScript, such as the onClick event of a button or the onLoad event of the page. However, in Web programming, most of your processing happens between when a page is requested and when it is displayed to the user.

Where are my variables? Most JSP pages will have page variables that can be thought of as local variables. Instance variables don't make such an easy transition. Since the Web environment is basically stateless, instance variables are not used. There are relatives of the global variable though. Using the JSP session variables or a session Java Bean will give you a place to store session information such as user IDs and user-related information. In this way, "state" is maintained in a stateless environment. Global variables should not be used in EAServer—they should be moved into the Java Bean.

Where the heck does my HTML come from? Some HTML will come from the page itself, some from the Bean, and some from EAServer. Static HTML can reside directly in the JSP page. The Bean can be used to build HTML that is common to the application's JSP pages. For example, code for page redirects, menu, headers or footers can all go into the Bean. Finally, HTML that is tied to data will be built in an EAServer component.

International Contacts

Argentina +5411 4313 4488	Korea +82 2 3451 5200
Australia +612 9936 8800	Malaysia +603 2142 4218
Austria +43 1 504 8510	Mexico +52 5282 8000
Belgium +32 2 713 15 03	Netherlands +31 20 346 9290
Brazil +5511 3046 7388	New Zealand +64 4473 3661
Bulgaria +359 2 986 1287	Nigeria +234 12 62 5120
Canada +905 273 8500	Norway +47 231 621 45
Central America +506 204 7151	Panama +507 263 4349
Chile +56 2 330 6700	Peru +51 1 221 4190
China +8610 6856 8488	Philippines +632 750 2550
Colombia +57 1 218 8266	Poland +48 22 844 55 55
Croatia +385 42 33 1812	Portugal +351 21 424 6710
Czech Republic +420 2 24 31 08 08	Puerto Rico +787 289 7895
Denmark +45 3927 7913	Romania +40 1 231 08 70
Ecuador +59 322 508 593	Russian Federation +7 095 797 4774
El Salvador +503 245 1128	Singapore +65 370 5100
Finland +358 9 7250 200	Slovak Republic +421 26 478 2281
France +33 1 41 91 96 80	Slovenia +385 42 33 1812
Germany +49 69 9508 6182	South Africa +27 11 804 3740
Greece +30 1 98 89 300	South Korea +82 2 3451 5200
Guatemala +502 366 4348	Spain +34 91 749 7605
Honduras +504 239 5483	Sweden +46 8 587 70433
Hong Kong +852 2506 6000	Switzerland +41 1 800 9220
Hungary +36 22 517 631	Taiwan +886 2 2715 6000
India +91 22 655 0258	Thailand +662 618 8638
Indonesia +62 21 526 7690	Turkey +90 212 284 8339
Israel +972 3 548 3555	Ukraine +380 44 227 3230
Italy +39 02 696 820 64	United Arab Emirates +971 2 627 5911
Ivory Coast +225 22 43 73 73	United Kingdom +44 870 240 2255
Japan +81 3 5210 6000	Venezuela +58 212 267 5670
Kazakstan +7 3272 64 1566	Asian Solutions Center +852 2506 8700

For other Europe, Middle East, or Africa inquiries:
+33 1 41 90 41 64 (Sybase Europe)

For other Asia Pacific inquiries:
+852 2506 6000 (Hong Kong)

For other Latin America inquiries:
+305 671 1020 (Miami)



Sybase, Inc. Worldwide Headquarters

5000 Hacienda Drive
Dublin, CA 94568-7902 USA
Tel: +800 8 SYBASE
www.sybase.com

As mentioned earlier JavaScript should be used for simple tasks and simple validation. Some of the tasks for which JavaScript can be used are derived from the event model of an HTML document. Some of the more commonly used events are onLoad, onClick and onSubmit.

The onLoad event is part of the body tag and can be thought of as a constructor event. The onLoad event can be used to set up a page when it is first loaded. The onClick event is part of a button (or input) tag. The onClick event can be used to direct processing when a user clicks a button. It is most often used to perform some kind of processing, like validation, before a server-side call is made. The onSubmit event is coded as part of a form tag. The onSubmit event can be used to run processes before a form is submitted. Examples would be validation or manipulation of data before a form is submitted to a server.

Bringing It All Together

Now that you have an increased understanding of what's involved in moving a PowerBuilder application to the Web, try coding a small application for yourself. For example, create a page that posts a user ID and password to a Java Bean, then pass the values to an EAServer component, and finally, get the values to a database. To do this, start with an HTML page. Make sure your server is running and that it's able to serve the page you've created. Once you can serve HTML pages, create a simple JSP page. Start with a page that only has HTML in it. This will test whether you can serve JSP pages. Once you're able to do that, add some JSP script. This will ensure that you can process JSP script. After the JSP is running, you will want to test the Java Bean. Make sure that you can return a simple string from the Bean to your JSP page. Next, code the Bean to connect to EAServer. Test that you can return a simple string from EAServer, pass it through the Bean and display it on your JSP page. You're almost there. Now set up your component to access data from the database. Return this data through the component, then through the Bean, and finally display it on the JSP page.

These steps may sound simple, but performing them one at a time will ensure that your architecture is set up properly. Once you prove that the concept works, you won't need to waste development time wondering if your data is getting lost somewhere.

One final note in the spirit of using the right tool for the job, if you're staring at a big, ugly process that isn't used frequently or by many people, consider keeping it in its existing PowerBuilder form. The Internet is a great tool that allows business applications to be opened up to anyone, anywhere, anytime. However, it is not the solution to all your problems. If you have a process that has taken years to fine-tune in your application, consider keeping it there. Alternatively, consider ways to break that process down into smaller, more manageable pieces. The Web is an awesome environment, but it does not handle large, complex processes well. Don't force yourself to use a great tool for the wrong job.

About the Authors

PowerObjects is a Minneapolis-based software development and consulting firm that has provided IT solutions nationwide since 1993. We specialize in distributed application development and are recognized nationally for our expertise in moving systems from a client/server environment to an n-tier architecture. From our genesis as a two-man company that provided PowerBuilder development and consulting services, we have grown into a multi-million dollar company providing expertise in today's most significant technologies. Our commitment to uncompromising quality and integrity has earned PowerObjects a sterling reputation among companies of all sizes.

Contributors from PowerObjects to this article included Dean Jones, CPD, TeamSybase; Trent Nelson, Lead Consultant; and Bruce Coles, Consultant.