Using JSF Framework Development Standards for XPages Project (Part One: Rethinking the Approach to XPage Development)

May 20, 2013

In this multi-part series I am going to explore IBM Notes & Domino XPage development as a traditional Java based web application development framework, instead of the form & document model originally introduced in Lotus Notes. Over the series of articles, I will cover high level concepts, such as the underlying JavaServer Faces technology, and work my way through Java development for XPages with advanced concepts and sample code.

In this first article I am going to explore XPage as a Java web application framework, some core concepts for implementing it with Java, and how it will change the way you design your Domino data (for the better). There won't be example code or advanced technique discussions for accessing information, this time around. If you have ever written a Java web application using standard development methodologies, this particular article will probably be common knowledge to you (keep checking though for the more advanced articles covering data access objects and Java persistence techniques). But if you are developing XPages by placing the field of a form (or column of a view) directly on the page and connecting it to a control, then read on.

XPAGE – IT'S FAR MORE THAN JUST A FORM FOR THE WEB.

IBM has done an outstanding job integrating standard web technologies into XPages but they have also kept the ability for us to simply place a field control on a XPage for a user session with an open live connection to the database. From a business perspective, it is understandable why it was done. If IBM had simply created a web technology and not created a technology path for existing developers to make the migration, the change may have ended up being the undoing of Domino

But for our counterparts using other technology (Java, PHP, .NET) for enterprise application development, that concept of a field and an open connection to the database makes them cringe. And as a long-time Notes developer (since R3), this author also cringes when I see it happen in enterprise Domino applications today, especially because it does not have to.

The true power of XPages is not its ability to create a web application using a simpler version of form design for the web. Instead, by embracing the core technology behind XPages (which is now Java), you will open the door to a new world of capabilities for better enterprise application development, such as (to name a few): reduced duplication of Domino data across documents, external system and data integration, powerful third party and open source utilities, more development resources in terms of experienced developers, and most importantly, responsive applications with easily extendible and re-usable code.

XPAGES CAN FIX THIS PROBLEM

Of all the reasons that Java based XPage development will outperform the traditional Notes approach, one stands out in terms of the amount of code you have to write and general performance of your application. Think of this simple example. You have a customer database with a simple form which tracks the usual data (name, address, phone, etc…). Your customers make orders, so you have an order database. The chances are you have an order form with some basic fields (order date, order number, customer, etc…). Also in the order database are order line items. So you have a line item form. On that form you have quantity, product, price, extended price, and maybe discount information. This is a simple example and a real order system would have many relationships across data.

Now your boss comes to you and asks for the typical view: a list of customers and all of their orders. IBM Notes is not relational but you need to produce one view to show both the customer data and some basic order together. In the relational world, you would perform a join of the data. But in Notes, there is only one solution, duplicate the customer data on each order. This solution of repeating data is on the short list of reasons why traditional relational database developers steer clear of Notes & Domino. Everywhere we would like to "join" a bit of data to show it in one list we have to duplicate the data. Furthermore, as we update the parent data (the customer in this example) we have to write a ton of code to synchronize the changes to all of the related data.

And now think of the invoice form we may create. This is probably going to be a series of lookup fields on the form. Well, that's not duplicating data at least. Instead we are performing database round trip operations for every single field. In terms of performance, that is a huge hit. Our relational database counterparts do one simple select to get all the fields they need.

With Java based development for XPages, both data duplication and constant database lookups can go away. Yes, by changing the technology we use to access our data, we can actually change the way we store our data. But first, a bit of understanding for what's behind an XPage is required.

HOW IS XPAGES A STANDARD WEB TECHNOLOGY?

XPages contain many powerful features and technologies that make it a rapid application development environment. But at the core of XPages is the JavaServer Faces (JSF) technology. JSF is a Java specification for building component based web applications. JSF is part of the Java Enterprise Edition standard specification. I'll limit my discussion on "what is jsf" and focus the topic to the fact that JSF is a "Java" based web application framework and therefore XPage application development should follow certain conventions for web applications.

Java based web applications traditionally utilize the very popular design pattern called the "Model-View-Controller Pattern" (MVC). In short this means you develop your application with a separation of code into distinct layers. You would have code for just the data processing, code for just the UI, and code to control the interaction of the other two layers.

Implementation of the MVC approach to Java web application development is critical to achieving the database design results I mentioned earlier. If you decide to make the change to Java XPage development, then do everything within your power to fully understand the MVC approach. When thinking in terms of JSF & XPages/Domino, the three layers break down as follows:

Model: These are your Java objects that contain and manipulate the data from the database. These objects tend to fall into two categories: the data model and the service layer. The latter contains all of the business logic for working with the data. The work of reading the Domino Documents results in objects loaded with the underlying data. The Domino Document never makes it directly to the XPage. Instead, data is by the XPage from a Java object loaded with information from the Domino Document.

View: These are your presentation Java objects. Mostly (exceptions aside) there tends to be a one-to-one relationship between an XPage (graphical view) and a related view Bean. The bean will provide the XPage with methods to read data out of the Model objects. But the view will never talk to or know about the underlying database. In fact, the database and documents have long since been disconnected by the time the view shows you the results.

Controller: This middle layer is the JSF API. No code is written here by the developer of JSF applications (including XPages). This is the FacesServlet. This handles incoming requests and data streams and interacts with our View layer for us. Understanding that the FacesServlet is the controller is critical. Often, I have seen blog posts from Java developer attempting to tackle XPages where the person had no understanding of JSF and had spent much time trying to "invent" a controller layer. There may be times when you want to talk to the controller and ask it to take a different action then it wants to. That is supported through various methodologies in the JSF framework. But in all of that, you are still not writing a controller layer.

The separation of these three layers and how Java allows us to maintain that separation is the key to how XPages will allow you to create leaner faster database applications without duplicated data. In sticking with the MVC approach, we stop placing Domino document references on our page or connecting them to a control. Instead, controls are connected to our view layer Java objects which has references to model layer data objects.

In Java applications using the MVC approach, you never directly access the document from the view. In pure Java/Relational Database development, following a pattern such as MVC is nearly mandatory.

NON-RELATIONAL DOMINO DATA AS A RELATIONAL JAVA OBJECT MODEL

Domino is not a relational database. But it does not mean we can't program as if it is. To implement code that can treat your data as relational (so you don't have to duplicate it), you simply need to have a key field on your form.

For example, the Customer form may have a field called CustomerUNID. This would be the unique ID of the customer. Domino gives us a unique ID of every record you create. It's called the Universal ID. So, to capture the unique ID of the customer record, I would create a computed-when-composed text field with the following formula:

```
@Text(@DocumentUniqueID)
```

I capture it once per document as computed-when-composed so it is possible to make a copy (not replica) of the database and maintain all relationships.

Any child document of the Customer (ex. Order), will have a copy of the CustomerUNID as its “foreign key”. Then I can write code to pull together the relationship. That’s the only piece of data that is repeated across documents, as is the same in traditional RDMS development.

In this very simplistic example, I would have a relationship between my customer data and my order data, represented in the Java object as follows:

```
import java.util.List;

public class Customer {

	private String customerUNID;

	private String name;

	/*
	 * Here the orders for a customer are related in the same manner
	 * as a one to many relationship found in traditional RDMS based
	 * applications.
	 */
	private List orders;

	public String getCustomerUNID() {
		return customerUNID;
	}

	public void setCustomerUNID(String customerUNID) {
		this.customerUNID = customerUNID;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public List getOrders() {
		return orders;
	}

	public void setOrders(List orders) {
		this.orders = orders;
	}

}
```

If you look at field, “orders”, you will see that it refers to another Java class called “Order”. It is a list of “Order” objects. The code you build would load the customer’s orders into that list. Then on our XPage, you display the customer and iterate through the list of orders. Beyond some particular information about the Order, the Order class will only contain one piece of data from the Customer, the CustomerUNID value.

This was a very simple example and it shows no detail code on how to read the data, create the Java object relationships, and display it all on the XPage. The “how to” will be coming in a future article. Our purpose for this article was to help understand the “why” of stepping away from long-honored (but outdated) Notes & Domino development methodologies in favor of mainstream Java development frameworks.

In future articles, we will explore not just the “how” but also advanced topics that will allow you to leverage functionality in Java that will not only save time coding and make your application perform better, but also open up functionality for integrating advanced features into your systems.

Articles in this series:

1) Using JSF Framework Development Standards for XPages Project (Part One: Rethinking the Approach to XPage Development)
2) Using JSF Framework Development Standards for your XPages Project (Part Two: Designing Your Application Model for MVC)
3) Using JSF Framework Development Standards for your XPages Project (Part Three: Creating the DAO)
4) Using JSF Framework Development Standards for your XPages Project (Part Four: Finishing the Model Layer)
5) Using JSF Framework Development Standards for your XPages Project (Part Five: The View)

May 20, 2013 [http://www.pipalia.co.uk/rethinking-xpages-part-one/] | by Gary Glickman | in jsf, mvc, xpages .

7 Comments

7 thoughts on “Using JSF Framework Development Standards for XPages Project (Part One: Rethinking the Approach to XPage Development)”

Toby Samples
May 20, 2013 at 2:40 pm

Looking forward to this series.

Gary Glickman Post author
May 20, 2013 at 2:48 pm

Hi Toby. Thank you for taking the time to read the article. I hope you enjoy the rest of the series as it is published.

Mikael Grevsten
May 21, 2013 at 5:21 am

me 2

Gary Glickman Post author
May 22, 2013 at 12:27 pm

Thank you Mikael. I hope you enjoy the series over the coming weeks.

Cameron Gregor
May 22, 2013 at 3:44 am

Looking forward to this series as well, this is the route we are taking for our xpages development and will be great to learn from your experience.

I'm curious about what you say about the servlet alone is the controller layer and no additional code is written for the controller layer?
I thought that a backing bean would be considered part of the controller layer?
i.e. the backing bean has models as private members, and methods bound to buttons/actions on the view then act on those models.
Then aim to create fat models and thin controllers, with the goal that as much re-usable business logic is in the models.
However there is still some controller code specific to each scenario the models are used in?
Or do you consider this also part of the model layer?

Gary Glickman Post author
May 22, 2013 at 12:32 pm

Hi Cameron,

Thank you for taking the time to read the post. I hope you enjoy the series in the coming weeks.

Regarding your question "I thought the backing bean would be considered part of the controller?" The short answer is No…..and Yes. Confused yet?

Let me address the "No" part of my response first. At the highest architectural level of JSF, MVC breaks down as this:

Model: The business/data layer. Any code with rules for manipulating and/or storing data. This is where you see DAO, JPA, and EJB type code along with a service/facade layer to massage the information before the view gets a hold of it.

View: The JSF code you write in the form of beans interacting with the XPage (xhtml in non-Domino based JSF 2.0).

Controller: FacesServlet. That's it. There's no other code you write. There is some code you can write to interact with and influence this layer (see PhaseListner in JSF). But you don't write this layer.

As developers we will spend a lot of time writing view layer stuff as well as business/service layer stuff (unfortunately as Domino developers we also have to spend some extra time at the data layer because Domino does not support JPA/EJB… but that is a topic I cover later in the series). Spending all that time bogged down in writing the View and service code (and never having to deal with the true Controller because the FacesServlet does that work), we tend to get tunnel vision on what we are doing and begin to see our code as including the "C" part of MVC. I agree we write a lot of code that controls the interaction of the Model with the View. But, that is the Service layer (or Façade). It's an additional design pattern within the larger MVC architecture.

I don't want to dive too deep into the MVC conversation as the purpose of my posts is to help Domino developers implement true JSF into their applications. Maybe I'll post something about that down the road as it does continue to be a recurring topic. But for now, if you are interested you can find some great stuff about JSF and MVC on Oracle's site. Also, there are a few JSF bloggers (non-Domino) that have some great overviews as well; which leads me to the "Yes" part of my answer.

In my technical travels of searching Google, I came across a post by BalusC (a very popular Java blogger). He summed up my last paragraph very nicely. Here is what he said:

"In the smaller developer picture, the architectural V (of JSF MVC) can be divisible like this:

Model: Entity
View: JSP/XHTML (XPage for us)
Controller: Managed Bean"

But, that is just how we think of the code we write. The true architecture of JSF is MVC with the C being the FacesServlet and we are writing the V and often M part of the pattern.

Cameron Gregor
May 22, 2013 at 11:18 pm

Thanks for the explanation Gary, looking forward to the rest of the series!