The Middle Tier Manifesto: An Alternative Approach to Development with Microsoft SharePoint

Marc D Anderson
Sympraxis Consulting LLC
831 Beacon Street
Suite 118
Newton Centre, MA 02459

Web site: http://sympraxisconsulting.com
Phone: 617.633.2051
Blog: http://sympmarc.com
Twitter: @sympmarc

Introduction

This white paper outlines an alternative way to think about development for Microsoft SharePoint. In some ways, what's described is counter to the conventional wisdom, so if it doesn't get some people's hackles up, then it's not written right.

The premise is that the most common development approach -- using managed code which is developed in Visual Studio -- is more costly, time-consuming, and error prone than the approach using the Middle Tier which is described here. That's not saying that managed code is unnecessary; it's saying that much of the time it is overkill and that solutions can be developed, deployed, and managed more quickly and safely than with managed code.

For those of you who find that the estimates from your .NET providers (internal or external) for implementing a solution are prohibitively high, there's another option: the Middle Tier.

What Is the Middle Tier?

Implementation efforts in SharePoint can be segmented into three very clear layers where one can do the work:

First Tier: The SharePoint UI and Central Administration

This is where all of the basic setup and configuration for a SharePoint implementation happens. Generally there are two sets of people who work in this layer: End Users and Administrators.

Middle Tier: SharePoint Designer

This is the Middle Tier which this paper describes. Most documentation and training focuses on the other two layers, but there is much that can be done purely in the Middle Tier using Data View Web Parts (DVWPs), scripting (in the form of JavaScript or jQuery), and Cascading Style Sheets (CSS).

Third Tier: Visual Studio

This is where traditional SharePoint development occurs. Developers write custom code in C# or Visual Basic which interacts with SharePoint through the Object Model. These custom solutions are generally deployed to the SharePoint farm using packaged Features which deploy files into the file system on the server and/or executables in the form of DLLs into the Global Assembly Cache (GAC).

The Middle Tier moniker makes sense because it essentially sits between the other two in terms of effort and complexity and because it truly is a middle ground which draws from both of the two extremes.

What Is Development?

This may seem like a rather silly question to ask, but when it comes to SharePoint, there are multiple schools of thought on what development actually is. Cutting through the marketing and advertising, SharePoint is basically an application and development platform. What this means is that it can be used right out of the box (quite well, in fact) or be used as a platform upon which to build other applications.

The term development can mean many things. Interestingly enough, looking at the dictionary definitions, it is hard to find anything specific to the way most computing professionals think about it. What is there is pretty basic:

> *The process of analysis, design, coding and testing software*[1].

Note that the definition doesn't say anything about what the "software" is. There's a prejudice on the part of Microsoft and the general .NET development community which says that if it isn't .NET code, then it isn't "true" development. This is nonsense. Development is development is development. If you have to write code, whether it is C# or CSS or script (or Logo or FORTRAN or COBOL or…), you are developing something.

Historical Context

Techies used to be called programmers because that's what they generally did: wrote free-standing "programs" which accomplished a small set of tasks. Somewhere along the way the term "programmer" became derogatory in favor of the more high-falutin' term "developer".

It doesn't matter what language or toolset you are using to write software: you need discipline and organization, the ability to break complex problems down into their piece parts, spy repeatable actions, and create code which performs the exact same tasks the exact same way over and over and over again.

Computers these days do much of the work for us, offering us rich IDEs and things like Intellisense which take huge parts of the thinking off the plate. (Punch cards anyone? Teletype terminals?)

Never before in the history of technology have the available tools been so richly capable of taking our hands and walking us down the path, whether it be the path to riches or the path to failure. But the tools don't make the solutions good. People make the solutions good. People still need to decide which tools are appropriate for each given set of tasks. Sure, the tools are going to get better and better over time (look at Visual Studio 2010 versus VB2), but good tools don't provide an excuse for bad thinking.

---

[1] Dictionary.com, "development," in *The Free On-line Dictionary of Computing*. Source location: Denis Howe. http://dictionary.reference.com/browse/development. Available: http://dictionary.reference.com. Accessed: April 11, 2010.

So what is a "SharePoint Developer"? One can find advertisements for positions with this title in the newspaper and online all the time. The most common definition of a SharePoint Developer is probably the wrong one: a .NET coder who knows how to spell SharePoint. Just because Microsoft slid SharePoint over on top of .NET doesn't mean that the only way to develop on top of SharePoint is with .NET tools. Some of this is probably rooted in Microsoft's early dismissal of the Web in favor of continued desktop toolsets. That schism still exists for Microsoft today, though of course they have fully embraced the Web as a platform for years now.

In many cases .NET is a sledge hammer when all is needed is a flyswatter, dynamite instead of a Kleenex. You get the idea.

What Makes the Middle Tier Approach Different?

Developing solutions in the Middle Tier means using some different tools and methods than the other two layers, though there is some commonality. The primary tools in the Middle Tier are as follows.

Data View Web Parts (DVWPs)

DVWPs are XML- and XSL-driven objects which are interpreted by the server at runtime. Many people have called DVWPs the "Swiss Army Knife" of SharePoint Web Parts, but it may also be the duct tape and WD-40. It's important to note that in SharePoint 2003, these Web Parts were called Data View Web Parts (DVWPs); in SharePoint 2007, Data Form Web Parts (DFWPs); and in SharePoint 2010, they are called XSLT List View Web Parts (XLVs). These are all fundamentally the same concept, albeit along an evolutionary path. Here, the term Data View Web Part, or DVWP, is used to refer to all three.

DVWPs basically consume XML and output it in a different way using XSL to define the output format. DVWPs can work with the following types of Data Sources:

- SharePoint Lists
- SharePoint Libraries
- Database Connections
- XML files
- Server-side scripts
- XML Web services
- Linked Sources (basically combinations of the above)

Many people rapidly get frustrated with DVWP development in SharePoint Designer because they attempt to do everything with the common dialogs SharePoint Designer provides. The possibilities for a DVWP are pretty much limitless once changes are made directly in the XSL. Once this has been done, the common dialogs will generally no longer be helpful, so it's always a best practice to get as far as possible with the common dialogs and then make customizations to the XSL to finish up.

Scripting

Scripting allows developers to create functionality which executes on the client machine in the browser rather than on the server side. Many parts of SharePoint utilize scripting out of the box, such as ECB (Edit Control Block) menu controls. All of the existing scripting in SharePoint 2007 is JavaScript-based.

With Microsoft's stated support of jQuery going forward, the possibilities for scripting have opened up considerably. jQuery is simply an abstraction on top of JavaScript which allows things to be accomplished faster and more easily. jQuery actually **is** JavaScript. Once the core jQuery library is referenced, the whole world of plugins and libraries which sit on top of it is available as well.

References to the jQuery library and any needed plugins can be included in Content Editor Web Parts (CEWPs) on specific pages, in the pages themselves (generally the best practice rather than CEWPs, for consistency), in page layouts, or in the master page. It all depends on the scope of use planned in the solution.

CSS

Yes, CSS can be used in all three tiers, but in the Middle Tier it becomes even more important. Many of the best scripting tricks alter the CSS on the page during the page's life in the browser. CSS becomes a fluid definition of format, not one that is predetermined by the code running on the server, where the user is stuck with the look and feel as it originally arrives to the browser.

SharePoint Designer-based Workflows

The fixed set of out-of-the-box workflows is fairly simplistic. SharePoint Designer-based workflows provide capabilities to implement much more complex business processes. In many cases, what can be done with SharePoint Designer-based workflows will be more than adequate. While Visual Studio (especially the 2010 version) gives far more robust development tools for workflow, the SharePoint Designer dialog-based configuration approach provides power as well, even though there is not true coding involved.

Because the artifacts of the tools in the Middle Tier are interpreted at runtime, there is no compiled code which must be deployed to the server. This makes Middle Tier development extremely "low touch" on the server. Many major issues which occur on SharePoint server farms are caused by changes made by developers, even well-trained ones, on the server.

None of the Middle Tier tools can cause harm to the server back end. This is by design; Microsoft has ensured that scripting and interpreted code cannot impact the server file system. This is appropriate and absolutely supportive of full-featured Web-based solutions.

Many times people will say that since the components of Middle Tier solutions are stored "in the database" that these solutions can't possibly be efficient or perform well under load. This is just not true. Frequently, managed code developers struggle to get their code to behave in such a way that the server load is manageable? While it is certainly possible to write code in the Middle Tier that is inefficient, it is no more or less likely than developing inefficient managed code. As stated earlier, good developers write good code, regardless of the toolset.

Benefits to the Middle Tier Approach

Common Web Development Skills

The primary tools in Middle Tier development are XML, XSL, CSS; scripting languages like JavaScript and jQuery; and Web Services. These toolsets are applicable to many Web platforms besides those which are based on Microsoft's offerings. This makes the knowledge far more transferable within a development team than working purely with .NET. Yes, .NET is more and more pervasive every day, but it can't be as pervasive as the Middle Tier toolsets, which are used in almost all Web development environments.

Lower Cost Tools

The primary Middle Tier development tool is SharePoint Designer, which is available for free from Microsoft. Third Tier tools like Visual Studio are much more expensive and a license is required for every developer.

Faster Development Lifecycles

Whether the Middle Tier is used simply for prototyping or for full-blown solutions, a good Middle Tier developer can many times get a solution in place faster, and almost always iterate more quickly, than a managed code developer. This isn't because the developers are any better or worse, but because of the way the toolsets allow each developer to work and deploy their solutions.

Reduced Testing Cycles

Because nothing is deployed to the server, testing can be done in a much more "unit testing" mode. If a given page functions properly, taking the correct inputs and creating the correct outputs, chances are very good that the page is performing correctly. Obviously, integration testing is also important, but the Middle Tier toolset mitigates negative impacts on the server side due to undisposed objects and the like.

Adaptability in Corporate Environments

Any major company or enterprise-scoped business is likely to have robust IT system management procedures and policies. These policies often prevent server level changes without seemingly infinite wickets of planning, communicating, and testing. During the development phase of a system the use of managed code isn't as big of an issue; but what happens a month down the road, six months down the road, or a year down the road when

that code needs to be updated, and that update requires interruption of service to the entire enterprise? Middle Tier solutions are far more adaptable in a large corporate environment than the deployment of managed code solutions.

Stable Servers

With the "low touch" approach to development, virtually nothing is loaded onto the SharePoint server farm, if at all. This greatly reduces the possibility of causing any harm to the servers and ensures that patches and upgrades cannot have an impact on the customizations directly. Then there are the problems of maintaining Code Access Security, accessing web.config, the GAC or bin deployment dilemma, etc. All of these issues have in common the fact that they require administrative access on the Sharepoint servers and can potentially cause them harm.

Backups Capture Full Solutions

Because all of the code and files which make up a solution is stored within the Site Collection, a Site Collection backup will fully contain the solution. With managed code which is deployed server-side, various files are deployed to the file system which are not captured in a Site Collection backup. Off course, a good Disaster Recovery Plan will cover those files and contain clear instructions on how to rebuild the environment, but it will be a separate set of actions required in the case of a disaster situation.

Drawbacks to the Middle Tier Approach

The Middle Tier approach shouldn't replace all other methods. There are some drawbacks to it.

SharePoint Designer

If your IT department has seen fit to block the use of SharePoint Designer, then the middle Tier is just plain not available to you. This short-sighted decision on the part of many IT groups is generally based on a lack of knowledge about how SharePoint Designer works and what can be accomplished with it. If the Middle Tier approach is an appropriate development path, then it's time to open up a dialog about it with IT and to find out what in the Governance Model is driving the decision. (If there is no documented Governance Model, then there's a bigger problem.)

Deployment

The Middle Tier was clearly not the area where Microsoft spent most of their thinking when it comes to deployment. Components like SharePoint Designer-based workflows can be a bear to deploy. They are primarily intended for "one off" or single use situations. Because the components of the Middle Tier are deployed primarily in aspx pages, there are not good tools to replicate these components from one environment to the next. In cases where a formal Software Development Lifecycle (SDLC) is being adhered to, these deployment challenges can be almost a showstopper.

Repeatability

Again, due to the "one off" nature of some of the Middle Tier components, it is somewhat more difficult, though far from impossible, to develop repeatable, reusable solutions. However, there are techniques which can mitigate these challenges. If it is important to be able to deploy a high number of instances of customized solutions regularly, then the Middle tier toolset may not be appropriate.

Performance

There are different schools of thought on this, and there are also many variables which come into play. Each solution is unique and has a particular set of requirements around Service Level Agreements (SLAs), disaster planning, etc. In some cases, Middle Tier solutions may perform more slowly than managed code solutions, but in other cases there is no noticeable difference. The important thing is to make the decisions based on real data rather than speculations.

Middle Tier Best Practices

Modular Code

Whether it is XSL, CSS, or script, the same basic concepts of modularity can be applied to development. This speaks to the skill of breaking things down into their piece-parts. By developing XSL templates that can provide general purpose functionality or script functions, the small amount of additional work required pays off, just as it does with good .NET component-driven development.

In fact, all the same tenets apply to development in the Middle Tier which apply to any development platform. Code should be simple, well-documented, componentized, and reusable. It should be efficient based on the parameters of the toolsets.

It is important to acknowledge that, for each piece of functionality, it will be either a one-off solution (meaning that it will not be, nor will it ever be, reused anywhere else) or something which needs to be highly generalized and reentrant.

As an example, the script which is built into the jQuery Library for SharePoint Web Services could have easily been just a quickly dashed off cascading dropdown function which worked in a particular page but nowhere else. That was in fact the germ of the library's SPCascadeDropdowns function, but it made more sense as a generalized piece of functionality which could be and used again and again and again. Once more, this is not different than the mindset one must apply to .NET development. Good developers write good code, regardless of the platform.

Reusable Code Objects

Storing code objects in Document Libraries facilitates reuse and provides "change once, fix many" benefits. The most critical thing is to be consistent. Because code objects are stored

in Document Libraries, they can be treated as content, with Approval and Check in/Check out processes as needed. This won't be a true code repository, but will work just fine for smaller teams, even distributed ones.

Everyone on the development team needs to know the rules and everything should be documented as part of the Governance Plan. The file locations below are based on some best practices, but different naming conventions are fine as long as they are documented and consistently used by everyone on the team.

CSS

Store CSS either in a Document Library in the root site of the Site Collection called something like "CSS" (WSS) or in the /Style Library/XSL Style Sheets (MOSS)

Script Libraries

Store script in a Document Library at the root of their Site Collection called "JavaScript" or "jQuery Libraries".

Images

Store images using in the branding in the /images folder (WSS) or in /SiteCollectionImages (MOSS).

Decision Criteria

So when is each of the three approaches warranted? The questions below may help to determine which tier is best suited for specific development projects. Each solution will have a different set of answers to these questions. Each project in the portfolio of solutions may have a unique profile and therefore each solution may fit best into its own tier.

- What skill set does the development team have? If it's a tried and true .NET shop, then the Middle Tier may not be right, even though it may provide great benefits.
- How much of the solution involves many (more than a few dozen) instances?
- What type of server access is allowable? If none or very little, then the Middle Tier is an excellent option.
- How frequently will the solution need to be updated?
- What are the security requirements?
- What types of volumes are expected for list items?

The most important consideration is that time, effort, and expense will increase moving from the First Tier to the Middle Tier to the Third Tier. All things being equal, it is best to work with the lowest tier that can provide a robust and reliable solution. Just because managed code is needed for a complex integration point doesn't mean that it is needed to change a calculation in a single Web Part in a SharePoint site.

Conclusion

The goal of this white paper has been to provide some things to think about in planning SharePoint development approaches and tasks. The main point is that the Middle Tier provides a significantly important set of approaches which are often dismissed or misunderstood due to a lack of knowledge or a "not invented here" mentality. The Middle Tier uses ubiquitous Web development tools and greatly diminishes the likelihood of making an adverse impact on the SharePoint server farm. And, the Middle Tier approach can enable full-featured functionality which can be rapidly iterated and deployed for quick ROI.