Pleasing Bosses and Customers

A compelling case for ASP.NET MVC

By Matthew DeMeritt, ESRI Writer

Although it was called "RESTful Apps with Microsoft ASP.NET MVC," the first user presentation at an ESRI Developer Summit (DevSummit) could have just as easily been titled "Pleasing Your Bosses and Customers." Given the importance of bosses and customers in dictating developer decisions, it's little wonder that more than half the slides in DTSAgile developer Brian Noyle's presentation at the 2009 DevSummit contained images of, you guessed it, bosses and customers.

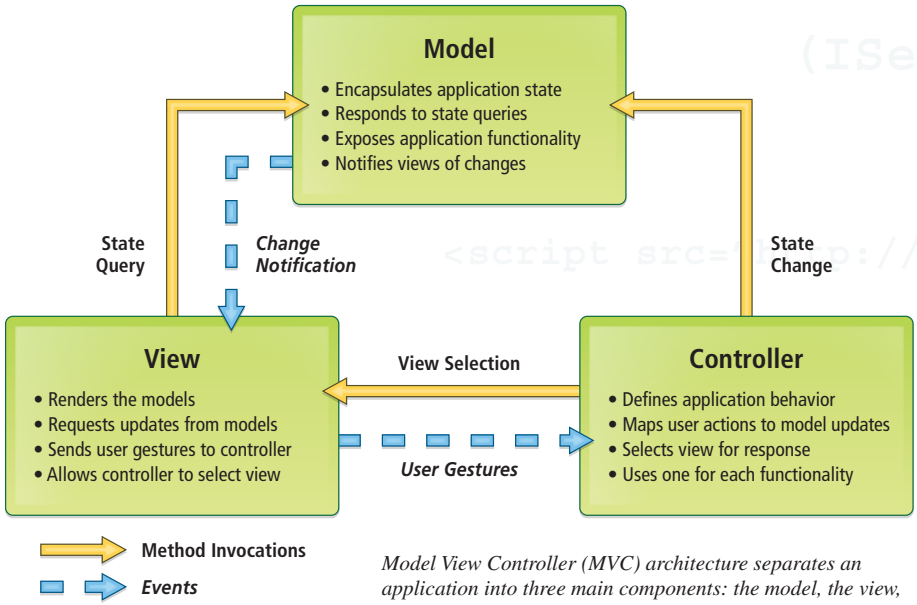
Noyle's presentation demonstrated DTSAgile's efforts to increase the quality of its GeoWeb applications using a RESTful architecture with ASP.NET MVC. *[MVC refers to Model View Controller, an architectural pattern that separates an application into three main components: the model, the view, and the controller.]* According to Noyle, the Web 2.0 tools that ESRI and Microsoft have produced eliminate much of the noise that can complicate development and the user's Web mapping experience. In the last year, ArcGIS Server, ASP.NET MVC, and a handful of other advances in API technology have combined to facilitate the creation of fast, focused Web mapping applications, simplifying the developer's toolbox and greatly speeding the development process.

> "The beauty of a Model View Controller architecture in general is that it enforces from project inception a compartmentalization of data, logic, and UI without compromising the functionality end users require."

A preconfigured development approach eliminates common (and ultimately unnecessary) design decisions from the outset. "ESRI and Microsoft have created technology that allows us to develop very rapidly in a predefined architectural paradigm that makes a lot of the decisions for us," said Noyle. "This helps move development forward very quickly."

Solving a Classic Problem

To demonstrate the clarity that an MVC architecture offers developers, Noyle compared the classic Active Server Pages (ASP) and ASP.NET Web Forms programming models to the ASP.NET MVC model. Classic ASP and Web Forms applications tend to violate the programming paradigm known as the "separation of concerns." As a code base evolves in classic ASP programming or Web Forms, business logic becomes mixed with presentation logic in the user interface, data access bleeds into the business layer, and so forth. Markup for these applications can get messy and noncompliant in a hurry since the developer does not exercise full control over the emitted HTML. There are as many flavors of browsers as flavors of ice cream these days, so what's the likelihood that page will render on everyone's desktop? Not likely.



Model View Controller (MVC) architecture separates an application into three main components: the model, the view, and the controller, and upholds the "separation of concerns" programming paradigm.

"The beauty of a Model View Controller architecture in general is that it enforces from project inception a compartmentalization of data, logic, and UI without compromising the functionality end users require. MVC separates typical developer concerns, isolating them into buckets of information within the application where they should be," Noyle said. Application data provided by a data access layer stays within the domain layer (or the M of MVC). User interface elements for rendering and presentation are explicitly isolated in one to many views (the V). And a suite of controller classes orchestrates the trip from database to view and back again (the C). Because humans like HTML, but machines like binary, JSON (JavaScript Object Notation), XML, or code on demand supporting multiple representations of data or features is no problem with ASP.NET MVC because it separates concerns while giving the developer full control over the HTML.

Noyle acknowledged that not all projects benefit from an ASP.NET MVC solution (a static content site with simple mapping, for instance). However, many applications are data driven and require advanced mapping functionality. Tackling those applications in ASP.NET MVC makes the most sense.

Better Bombproofing

According to Noyle, MVC facilitates much more robust and reliable unit testing than Web Forms or classic ASP programming typically allow. "Plain and simple, ASP.NET MVC was designed and built to produce testable software. If Microsoft thinks it's important, then developers should think it's important."

Typically, DTSAgile will create points of entry (separate C# projects) in its code base for the automated testing of application components. Application models, controllers, and all supporting services get tested a lot. MVC allows developers to set up automation

in which a simple mouse click can run nightly tests that answer questions such as

Does the application code function as it is supposed to today?

Did I fix something that broke something else?

"If I run it at five o'clock again tomorrow and something fails, I know something that I have written today has broken the application and I need to hunt it down," Noyle said. "Automated monitoring and regression practically ensures code quality from start to finish."

What About the Maps?

While this was a GIS developer conference, Noyle's message was not primarily mapcentric, and with good reason. "DTSAgile tries to follow the KISS *[Keep It Simple Stupid]* principle when choosing a map canvas for its applications," said Noyle. "What is the simplest technology I can use to satisfy the user's requirements? DTSAgile builds very focused, high-performance applications for clients so the map canvas, and nearly all associated code that drives it, usually winds up in an ASP.NET MVC view, specifically as JavaScript code.

"Very rarely will I use server controls when building mapping applications," continued Noyle. "We code against the ArcGIS Server REST API wherever possible, and if we need to build advanced spatial functionality, we will typically surface it as a REST resource—not surprisingly as an ASP.NET MVC Controller method—that calls a custom server object extension or COM utility." Near the end of his presentation, Noyle showed an abundance of mapping applications that used ArcGIS Server APIs for REST and JavaScript, and the ArcGIS Extension for Microsoft Virtual Earth API, and OpenLayers. "ESRI and other vendors have made mapping so easy for us that we can now focus on tailoring applications to customers' specific needs."

Should Noyle's customers care about the MVC architecture, compliant HTML, RESTful access to mapping resources, or all the unit testing that go into producing one of his applications? Not according to Noyle.

"The bottom line is that we have seen a paradigm shift in expectations since Google Maps shipped in 2005. As an industry collective, we need to realize that our bosses, our clients, and our users expect fast, robust, focused apps that provide relevant information right now. They don't care about the bits and bytes that do it." Perhaps it is appropriate, then, that Noyle's presentation, focusing on technology examples and architectural patterns and practices to increase development velocity and code quality, brought developer focus back where it belongs: to users and clients.