BUNGEElabs™

Bungee Connect and Model-View-Adapter (MVA)

The architectural pattern for next generation interactive cloud-based web applications

BUNGEEconnect™

www.bungeeconnect.com

Introduction

Over the past 10 years the web has evolved from simple web pages to complex, interactive web applications. As business use of the web has increased, the desire for greater productivity has required new technologies that can deliver higher levels of integration and interactivity.

A growing set of languages, protocols and data standards (AJAX, XML, ASP, JSP, Javascript) have come on the scene to meet higher user demands, but the methodology of building the application hasn't materially changed since the client-server days. As a result, these new requirements and emerging technologies have pushed existing development models beyond their original design goals. New approaches are required to meet the increasing complexity of developing multi-tiered applications for the cloud.

Model-View-Adapter (MVA) is a new architectural pattern for the development of next generation web applications. Bungee Connect, through its cloud language Bungee Sky, is the industry's first implementation of the MVA architectural pattern for the Cloud.

Bungee Connect enables the creation of the next generation of cloud-based applications that can better integrate with data and have a far more interactive user experience. Bungee Connect employs the MVA architectural pattern to more efficiently develop this new class of web application, enabling more business applications to take advantage of cloud-based delivery.

What is Model-View-Adapter (MVA)

Similar to the Model-View-Controller (MVC) pattern, MVA separates business logic (the Model) from the presentation or data layer (the View). A key principle of MVA is an absolute separation of the Model and View, avoiding the pitfalls of co-mingling of special case logic, common in MVC implementations.

This strict separation between Model and View is enabled through Adapters. Adapters work as controller agents without requiring the developer to specify the controller instance logic each time Models and Views are required to interact. Adapters manage all interactions, updates and flow of control and can be combined to facilitate a broad range of user interactions, updates, data transformations, and flow of control scenarios. Adapters connect the Model to different interactive elements within the View such that they are managed, monitored and updated in a just-in-time, granular fashion without creating any explicit interdependency between Model and View. In addition, this same ability to view and adapt to an application model applies to any external source including data and services.

How MVA is different from MVC:

Like its predecessor, MVC, the MVA pattern helps developers manage the complexity of applications which have multiple ways to access and/or update a centralized business process or data source.

The MVC pattern was conceived when "page-at-a-time" interaction for web applications was the norm. When designing traditional, full-page-refresh web applications of this type, controller logic was relatively simple. But this simplicity has given way to complexity as AJAX has become more popular and end-users have come to expect more interactive interfaces delivered through the browser. With richer user web interfaces, the number of interrelationships that need to be maintained, synchronized and reflected in user interface presentation and feedback multiply, making the interaction between View and Model much more difficult to implement and manage.

Similarly, managing connections to external services and data compounds the complexity. Existing MVC frameworks offer no help because MVC is typically applied for user interface interaction alone.

In web-based applications there may be a wide range of interactions and views available to the end user representing the same piece of data. For example, in a web-based calendar application, the user can expect to see the same data presented in multiple views, i.e., day, week, month. In a traditional MVC pattern, the developers would have to design, code and manage the interaction specifically for each instance of the control as well as the connection to each data source. In addition, because MVC does not typically deal with state management, developers would have to create the facilities to keep track of the global implications of all permutations of Views through the course of application use. Continuing the calendar example, the MVC developer would need to code the facilities to immediately reflect updates to an appointment for every instance on the screen.

Bungee Connect and MVA

In Bungee's MVA implementation these connections and state information flows are managed by Bungee Connect Adapters. Because Adapters themselves may be reused across the application, MVA dramatically reduces and even eliminates entirely the typical controller logic required by traditional MVC implementations. MVA obviates the un-manageable “glue” code that is not easily encapsulated for reuse in other parts of the application. With regard to the calendar application example, using Bungee Connect, the developer need only bind the different day, week and month controls to the same appointment object and when that object is changed all controls bound to it are automatically updated and reflected in the user interface.

Generally in MVC implementations, the controller logic and the overall communication protocol between Model, View and Controller (sometimes called the "wire-up") are largely left to the developer to design and implement. The developer is responsible for defining and managing the ongoing enforcement of separation boundaries, adding to the development effort and design overhead as well as making it difficult to evolve the code over time. The policies defining the Model/View separation are easily compromised by developers both during initial development and over the life-cycle of an MVC-based application, resulting in blurred distinctions between Model, View and Controller code areas. This leads to cumbersome application performance and increasingly complex development requirements.

The MVA Magic

Bungee’s implementation of the MVA applies to both connections to data (web services and databases) as well as the user interface. For the user interface, the Bungee Connect MVA patter enables developers to rapidly create responsive AJAX interfaces without creating and managing triggers, listeners/observers, or other polling and event management facilities. Using MVA, the flow of state is fully automated, from the UI elements presented in the browser to the server-side model objects, and back to the UI -- including the necessary transformations of state and data formats normally required when associating model data in multiple presentation contexts.

MVA follows a similar pattern when interacting with an external service or data source. By automating the create, retrieve, update, delete (CRUD) operations, Bungee Connect obviates the need for developers to write extensive amounts of code just to interact with external data sources. This automation results in the same benefits from the Model to the service or data source as is enjoyed from the Model to the user interface.

One very powerful capability of the MVA pattern, relative to MVC, is the ability of the View to dynamically tailor itself based on the state of the objects in the Model that are encountered at runtime (as communicated to it by the adapter). The presentation that a single View produces can depend on the instance data found in the model on a just-in-time basis as that data is “bound” to the View at run-time. The View can continue to automatically change during the life of user interaction as the underlying Model state changes, either through user directed interaction (that causes state to change) or via Model logic that updates its state appropriately. This means that Views bound to Models that are connected to, and updated by live web services, databases or other time-variant data are automatically updated as the underlying data sources change.

MVA also improves the ability to move an application to new UI form factors and designs while re-using all of the business logic and application functionality. With the separation between Model and View, developers need only layout new forms to move applications to mobile clients. Additionally, entire libraries of business applications can be reused without refactoring the underlying code.

Summary: The Benefits of MVA for Developers

MVA is a new architectural pattern for more highly interactive and rich web applications that present live data from multiple sources as well as multiple services and data sources. Developers write and manage far less code and are required to spend less time architecting the management of state, events and control logic. Through a strict separation between Model and View, developers can re-use controls and application logic throughout their applications, saving a significant amount of time through all the phases of application delivery.