MindTelligent, Inc.

Model View Controller (MVC) Architecture

A Technical White Paper

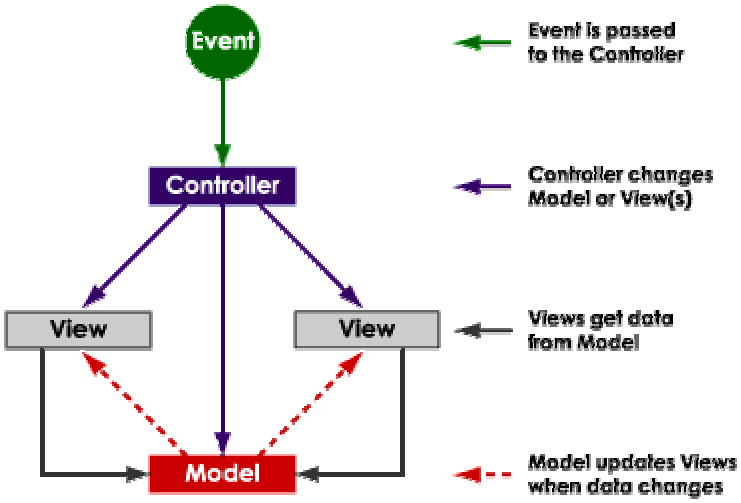
Model View Controller

The Model-View-Controller (MVC) architecture organizes an interactive application design by separating data presentation, data representation, and application behavior.

The Model represents the structure of the data in the application, as well as application-specific operations on those data. A View (of which there may be many) presents data in some form to a user, in the context of some business function. A Controller translates user actions (mouse motions, keystrokes, words spoken, etc.) and user input into business method calls on the Model, and selects the appropriate View based on user preferences and Model state. Essentially, a Model abstracts application state and functionality, a View abstracts application presentation, and a Controller abstracts application behavior in response to user input.

MVC provides many benefits to a design. Separating Model from View (that is, separating data representation from presentation) makes it easy to add multiple data presentations for the same data, and facilitates adding new types of data presentation as technology develops. Model and View components can vary independently (except in their interface), enhancing maintainability, extensibility, and testability. Separating Controller from View (application behavior from presentation) permits run-time selection of appropriate Views based on workflow, user preferences, or Model state. Separating Controller from Model (application behavior from data representation) allows configurable mapping of user actions on the Controller to application functions on the Model. Non-interactive applications can modify the MVC concept, sending input to a controller (that interprets input instead of user gestures), and providing result sets as "views".

MindTelligent has implemented several products based on a multi-tier MVC framework. The Views are JSP pages/HTML Pages/XSL documents, composed with templates and displayed in an HTML browser. The Controller maps user input from the browser to request events, and forwards those events to the Controller in the EJB tier. The Controller performs business methods on the Model Enterprise Beans, and then forwards any resulting updates to a cache of JavaBeans components that mirror Model data in the Web tier. The View JSP pages or Applets may later use these Web-tier Model beans to produce up-to-date dynamic content.



Views (What is a view and how it is implemented.)

The client essentially consists JSP pages/HTML Pages/XSL documents, assembled by the Web tier and displayed by a client-tier Web browser. The View selection and display mechanism is declarative, extensible, and internationalizable.

The Web-tier Controller interprets each HTTP POST or GET request from the browser as a request for an application service. Based on the result of the service request and the user's current state, the Web-tier screen flow manager chooses and assembles the next screen (implemented as a JSP page/HTML Pages/XSL Documents), and serves the resulting HTML to the client.

Selection of the next view is properly the role of the Controller. So the view selection mechanism is Controller functionality, while templating, JSP page assembly, and content delivery are all View functionality. If the JSP page being served is a template, the screen flow manager assembles several pages, one for each section of the template, into a single HTML

response. The JSP pages display current Model data by referring to Web-tier JavaBeans components that mirror data values in EJB tier Model objects.

Screens are defined in an XML file deployed with the application. Screen definitions, which name and describe the contents of each page in the application, are loaded by the screen flow manager on initialization. Screens can be defined or changed by simply changing the contents of the XML screen definitions file and redeploying, with no recompilation necessary. Only the names of the screens are compiled into the application.

The screen definitions file also defines screen flow; that is, selection of the next screen to display. In most cases, the screen flow can be defined declaratively in the screen definitions file. In cases where next screen to display must be determined programmatically, the screen definition may defer the selection of the next screen to a programmer-defined "flow handler" class that selects and returns the identifier of the next screen.

Controller (What is a controller and how is it implemented.)

The Controller is essentially a Servlet that encapsulates and dispatches service requests to the EJB tier. This section describes how the Controller executes calls on the Model in response to user requests, and selects and updates Views.

The JSP pages direct every browser POST or GET operation in the application to the URL of the Front Controller servlet. The operation being requested (create account, create order, etc.) arrives encoded in one of the data items in the request. The Front Controller directs the incoming request to the Request Processor, which translates the HTTP request into an application event corresponding to the requested operation.

The Request Processor sends the application event to the application Controller, a Web-tier proxy for the application Controller (stateful session) enterprise bean. The application Controller proxy sends the application event to the application Controller enterprise bean, and receives as a result a list of enterprise beans that may have been updated. The application controller then uses this update list to notify the affected Model beans, which refresh their contents by pulling data from the

enterprise beans they mirror. As noted above, the JSP pages that form the Views may later use the Model beans to present up-to-date Model data from the Web tier, without incurring a round-trip call to the EJB server.

Model (What is a model and how is it implemented.)

The Model is implemented as a collection of enterprise bean classes. These enterprise beans provide pure business logic, with no reference to how (or if) the information will be displayed.

The core of the Model is the collection of enterprise beans that provide the business logic for the application. When the Application Controller enterprise bean receives a request event from the Web-tier Application Controller, it dispatches the event to an EJB tier "handler" class that fulfills the business request. After the handler executes, the Application Controller enterprise bean returns to the Web tier a list of names of enterprise beans that may have been updated. The Web tier Application Controller notifies the Web-tier beans of the update (as explained above), and the beans refresh their cached data by pulling new values from the enterprise beans they mirror.

Model entity beans persist to the back-end database in the EIS tier using BMP (Bean-Managed Persistence) or CMP (Container Managed Persistence) BMP/CMP in the project should be vendor-independent, because it is written in terms of Data Access Objects, which abstract away the dependencies a resource may have on vendors or versions.