MVC

# Overview

The MVC framework (Model View Controller) sits on the Entity framework (Omniscient.Foundation.Data) to provide a managed view and edit capability for entities. The framework deals with problems such as concurrent edition of the same entity, refreshing views after an entity has changed, and decoupling entities from the UI.

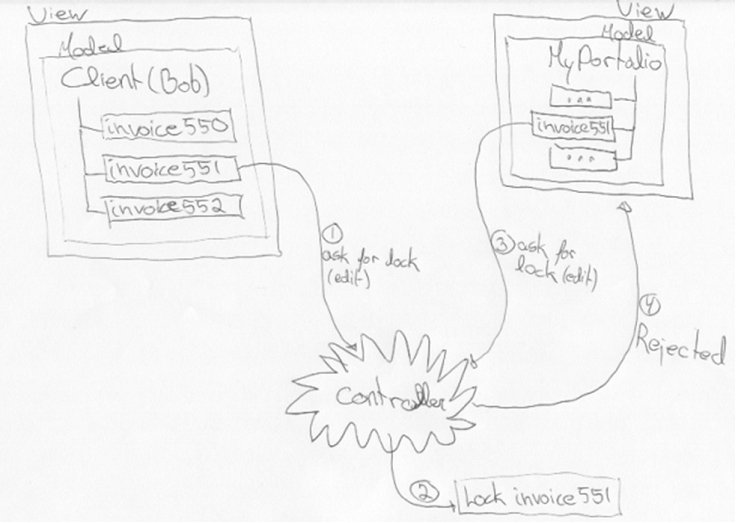
# Model

The model acts as a wrapper around entities. It is built from the metadata, and defines boundaries around the underlying entity. That is, if we have a Client with a list of Invoices, we may be interested solely in the Client properties, not the invoices. The Model would then define its boundaries according to that, and will not even check the invoices when it’s time to save or edit the Client Entity.

The model contains all the metadata needed to secure entities; when the model is displayed in a view, for example, textboxes may get disabled according to security imperatives.

The concept of a Model makes it easy to map it to a view. Displaying a Client may require different views for different scenarios (display the detail of the client, or just the name on top of a report, for example). We wouldn’t be able to make such decision if we were displaying entities.

The Model is responsible for locking entities when the view asks for edit permission. If the entity is already locked by another Model, then the edit permission will be denied. Entity equality is determined by the Id, not the reference, or pointer.



# View

The view is responsible for displaying a Model’s entity(ies). It is declared in the metadata, and loaded when a request is made to display a Model. If it is manually closed by the user, then it must inform the controller so that no change is lost.

The type of Model drives the selection of the View. When a view is selected, it is fed with the Model, and displayed by the ViewController, which manages its lifetime.

The view is responsible of informing the Model that the user wishes to edit the underlying entity through edit controls (text boxes, drop-down selections, etc). The Model then tries to lock the entity; if a lock is already loaned for that entity, then the View must disallow the user to edit the Model. It is good practice for the View to disable any input fields where the underlying entity is unavailable for locking.

# View Controllers

View Controllers are responsible for finding the right View for a given Model. It is also responsible for managing the lifetime of each View. A View Controller is generally responsible for a region of the application where views are all displayed the same way. For example, in a multi-tabbed environment, there could be a View Controller for the main region where Views are displayed in tabs, and there could be a second View Controller for a docking region that displays smaller views. Some Models could open Views only in the tabbed region, other Models could open Views in the docking region only, and other Models could open simultaneously two Views in those two regions. Therefore, it is important for the View Controllers to share locking information, and that brings us to the Application Controller.

# Application Controller

The Application Controller is the entry door for all View Controllers. It is responsible for storing and managing locks on Entities, so that two different Views may not edit the same Entity at the same time. It also controls the lifetime of Models.