MVC is a conceptual model, so it can be hard to understand and apply to your code. Let's explore an example to help you grasp the concept a little better.

I explained each of the MVC layers. I understand if you still have questions—lots of questions—on how to apply it to your code. I'll try to help with that by presenting an example based on a previously explained fictional program. Now let's see how that can be applied to an ILE program.

Reengineering an ILE Program Using the MVC Design Pattern

Take a moment to review Figure 1. It's a simple example of a fictional order-maintenance ILE program.

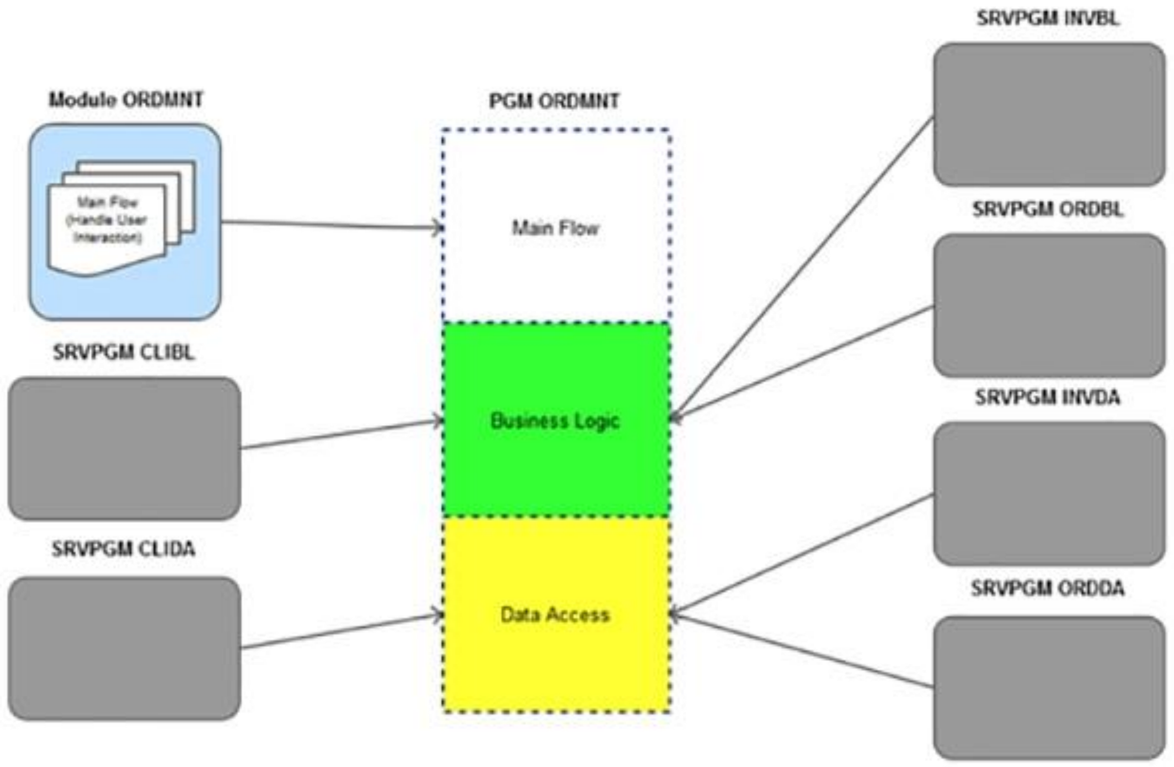

Figure 1: This fictional order-maintenance ILE program.

Just to remind you, let's go over the several pieces depicted here. You'll notice that service programs INVDA, ORDDA, and CLIDA are connected to the data access tier of the program. These service programs would provide data access functions, in the form of RPG or SQL I/O Servers, designed to read and write records to the database.

Service programs INVBL, ORDBL, and CLIBL would supply higher-level procedures and functions like Rtv_Item_Description, Rtv_Order_Delivery_Date, Wrt_Order_Record, Upd_Order_Record, Rtv_Client_Name, Upd_Item_Qty, Clc_Delivery_Fee, Clc_Earliest_Delivery_Date, and Chk_Client_Debt, just to name a few. There are many more procedures and functions involved. I didn't include the utility (to send the order confirmation by email or fax, for instance) and general calculation service programs (such as date operations needed to calculate the actual delivery date assuming it cannot be on a weekend, for example). I also omitted the modules linked to each of the service programs depicted, to try to keep things simple. Only the ORDMNT module was depicted to remind you that the ILE program doesn't contain code; only the respective entry module (ORDMNT, in this case) does. Program ORDMNT's binding directory would contain an entry for each of the service programs used. Otherwise, the compiler wouldn't be able to resolve all of the imports of module ORDMNT.

Let's take the multi-tier architecture implementation example shown in Figure 1 and apply the MVC design pattern to it. Let's see what fits where:

- **The model layer**—The business logic service programs provide the program with its "brains," because they contain the procedures used to get the raw data from the database and transform it into useful information. For instance, the Rtv_Order_Delivery_Date procedure uses the I/O Servers provided by the ORDDA service program to check whether the order number it receives as a parameter is valid and then retrieves the respective delivery date. This means that the data access service programs are also part of the model because they include both the data and the means to retrieve it.
- **The controller layer**—The ORDMNT module represents the controller. It will be reacting to user actions (options and function keys), translating that input into calls to the procedures that are part of the model. Imagine, for instance, that there's an option to show client-related data. When the user chooses that option in the respective display file (omitted for simplicity's sake), the controller (ORDMNT) calls procedure Rtv_Client, which in turn uses an I/O Server to read client raw data and pass it back to the controller.
- **The view layer**—The omitted display file and the part of the ORDMNT that handles the screen constitute the view, because they'll receive data from the controller and display it in a new screen or a subfile, for instance. Continuing with the client-related data example, the view part of ORDMNT would present the retrieved data to the user on a new screen.

This is not a perfect example because of the overlap between the controller and view layers in the ORDMNT module. However, if you're interested in modernizing the UI, you'll want a graphical user interface, not a green-screen. How, might you ask? Well, there are a few options in the market, some more simple than others, but IBM's Rational Open Access, formerly known as RPG Open Access is, in my opinion, the most versatile. In the next TechTip, we'll look at it in more detail, explaining what it is (and what it isn't) and how can you use it (just to give you a preview, there are two ways to do it—the easy and the hard). Until then, leave your comments, questions, and suggestions in the Comments section below or in the LinkedIn forums where my articles usually pop up.