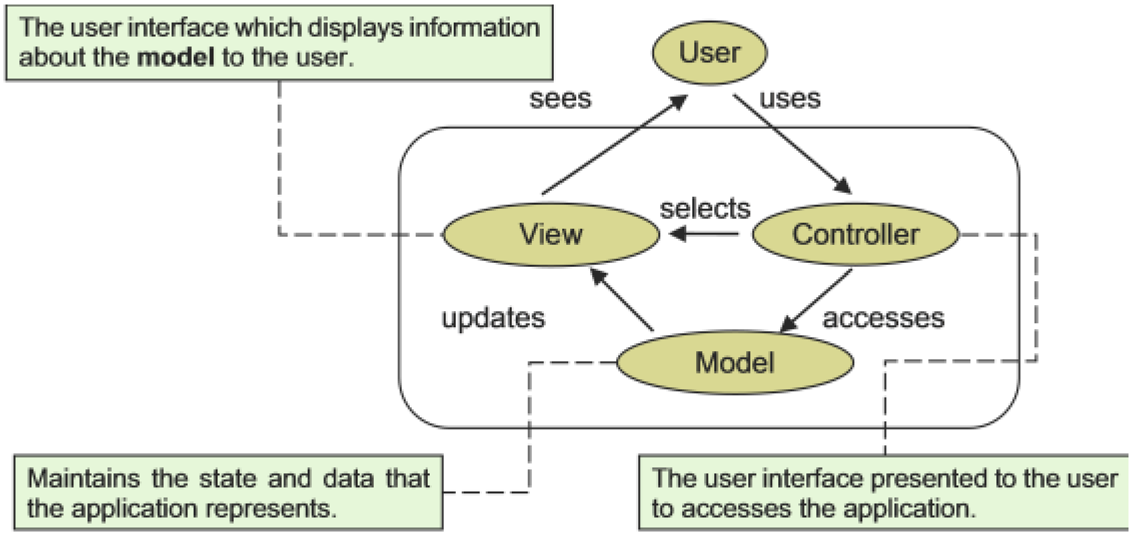
The MVC pattern - useful but not a silver bullet

Design patterns try to suggest the way of the application design, while methodologies try to give suitable models for the application and its whole lifecycle. The main idea behind design patterns is to extract the high level interactions between objects and reuse their behaviour from application to application. Moreover, design patterns help to clarify the way that we can think about a Web application.

The MVC architecture has its roots in Smalltalk, where it was originally applied to map the traditional input, processing, and output tasks to the graphical user interaction model. However, it is straightforward to map these concepts into the domain of multi-tier Web applications. It can improve the application's usability, creating reusable code and helping to understand and clarify the functionality of the program. The MVC pattern is very simple, yet incredible useful. It could support:

- Efficient modularity: allows swapping in and out any of the components as the user or programmer desire. Changing one aspect of the program are not coupled to other aspects
- Reusability: it can support the reuse of previously created code if we act sensibly and design carefully.(Reduces risks of bugs coming from refactoring)
- Ease of growth: controllers and views can grow as the model grow
- Centralized controller: a main module is used to make control more manageable

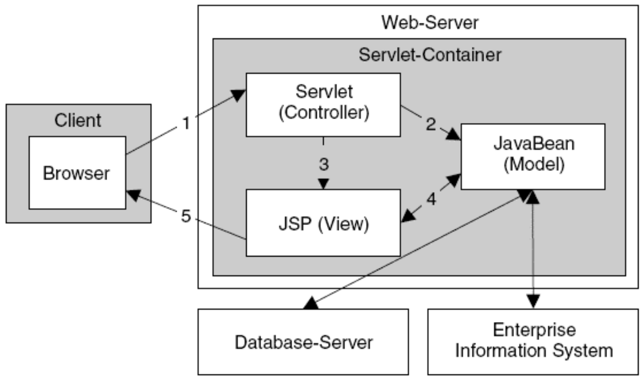

The user interface which displays information about the **model** to the user.
User
sees
uses
selects
View
Controller
updates
accesses
Model
Maintains the state and data that the application represents.
The user interface presented to the user to accesses the application.

The MVC architecture

- **Model**: The model represents enterprise data and the business rules that govern access to and updates of this data.
- **View**: The view renders the contents of a model. It accesses data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes.
- **Controller**: The controller translates interactions with the view into actions to be performed by the model. In a Web application, they appear as GET and POST HTTP requests. The actions performed by the model include activating business processes or changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

Although MVC is undoubtedly a valuable and useful way to design Web applications, but not the only one. The MVC design pattern's importance lies in how could it help to achieve a clear separation of concerns and functional layers. Create a module for the database portion, create another module for the application code, and create a third module for the presentation code. That way, we can swap and change different elements at will, hopefully without affecting the other parts.

Several vendors have applied this design pattern in their solution. As we have seen that the Web application term appeared in the servlet definition from Sun, here we will show the classic version of the MVC pattern's usage in that context (according to Kappel et al. 2006).



The JSP-Model-2 architecture

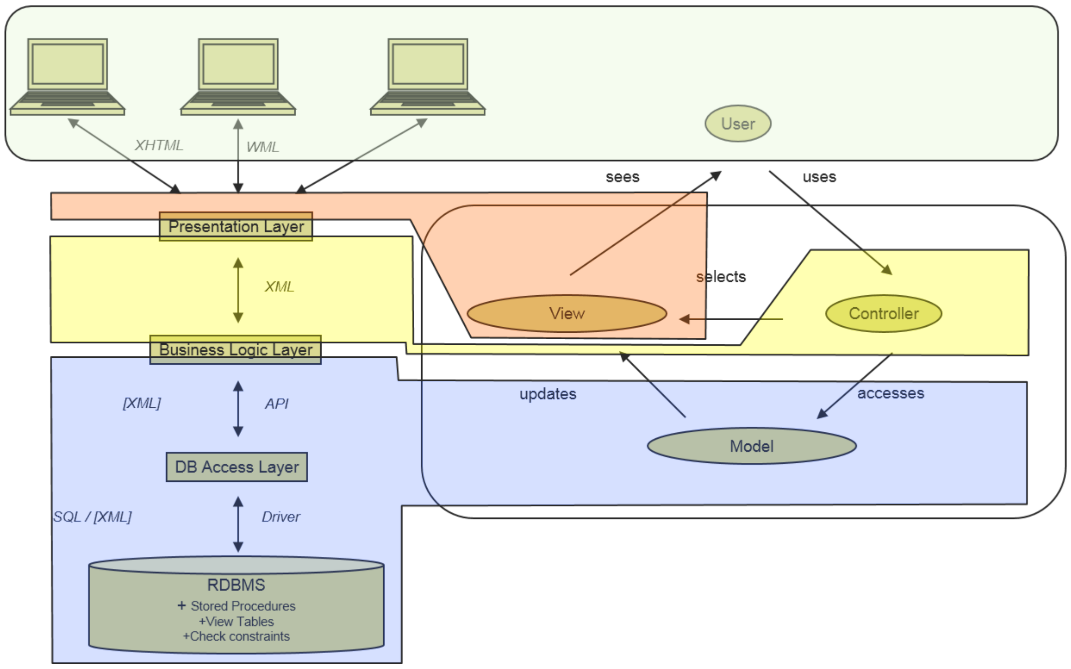
The Layered Architecture and the MVC Design Pattern

In the context of Web applications, by concentrating too closely on applying the MVC design pattern and nothing else, many other important aspects may be overlooked. This could lead to a fragmented and fragile solution, and as a result, is hard to maintain and execute further development. The MVC is a well known and widely used design strategy, but the problem is, how can we adopt this pattern into the layered model of Web applications?

At first sight the answer could say that we have three layers and three modules, the View module is equivalent with the Presentation layer, the Model module is equivalent with the Data layer, and the Controller module is equivalent with the Business Logic.

But if we take a closer look in the functionality of the Model and the Controller – discussed earlier in the paper – we could find that the Model represents business rules, and the Controller translates the interactions (HTTP GET, POST requests).

These suggest that the Controller is only a part of the Business Logic Layer and the Model forms the other part of the Business Logic Layer. We must take under consideration this heterogeneous composition of the Business Logic Layer through the development of a Web-based application.



MVC and the Layers

It should be obvious by now that MVC or any other design pattern is no more a silver bullet by itself than object-oriented programming is. It's just one part of a much bigger system. We know that MVC has several advantages:

- Clean separation of different functional layers
- Reduces maintenance costs
- Reduces risks of bugs coming from refactoring, graphics redesign
- Presence of a central controller raises overall software security level
- Controller can centrally perform tasks like access logging – without central controller such a task would affect the source code if all business logic actions.

but we also know the drawbacks:

- More software design cost (short term)
- More implementation cost
- Programmers cannot use some comfortable system features
- Programmers may feel they are forced to program a complicated way instead of quickly implementing.

Prev Up Next
Chapter 4. Layered Architecture for Web Applications Home Chapter 5. Architectures for Enterprise Level