How is it implemented?

How is it implemented ? How do people actually do it ? Do they use frameworks/libraries which ones do they use for what languages and why e.g. spring

Why do implement the MVC pattern, when should we use it, well unless your application is very simple, all the time. The MVC pattern's reasoning is that of separation of responsibilities, so that when modifying whether it be functionality, data or what the user see's, everything in the application does not have to be changed. It makes our code modular, reusable and just easier to maintain. This just sounds like a way of coding, well it is, but there are many different languages which all behave differently. Is it all hard ? No we've MVC frameworks for each language that help us achieve this. The implementation of these frameworks for some of these languages are listed as such:

Java

* Framework used:
  + Spring
  + Apache Struts
  + jsf (JavaServer Faces)
  + tapestry
  + stripes and wicket
* Popular server-side execution environments :
  + GlassFish
  + Tomcat
  + Jboss
  + Jetty
  + Wildfly

The MVC frameworks for java

Through analysis, the frameworks usually includes a library of mark up tags, the use of these tags vary from framework to framework. Examples:

* Apache Struts uses their own tags to present information to the user as well as suitable functionality e.g. a form and it's validation
* Spring uses tags to bind Java code to pages. The tags are used to map paths to different components of the webapp, the DispatcherServlet uses these tag defined paths to service incoming http-requests by calling the correct component according to the request as defined by the tags. The components used by Spring can be subdivided as follows Handler mapping, Controller, View resolver, view. When the request is received by the Dispatcher servlet, it is passed to the Handler mapping which returns the necessary controller. When the controller recieves the request it calls the appropriate service. The DispatcherServlet passes the request path to the view resolver which then returns the appropriate view after which is returned after the model is updated. Note that the views for spring are written in jsp which utilizes tags.
* JSF provides a standard HTML library

In general there are explicitly different classes for Models, usually POJO's, Views that use some markup language, classes to manage the Model(s) update and return view(s).

PHP

* Framework used:
  + Code Ignitor
  + CakePHP
  + ZEND
  + SYMFONY
* Popular server-side execution environments:
  + Daemon

Through investigation it has been noted that the MVC frameworks for PHP require little to no configurations(e.g. XML/YAML files). They contain a library

Contains a strict folder structure representing MVC as well as a strict naming convention where the framework itself associates files by location and name.

Code ignitor/CakePHP there is a controller class and to use it, we just extend that class. Contollers are initialler loaded from a file that maps pages/uri's to controllers, otherwise controllers can be loaded from controllers. There is a model class and we just extend that class to create our own model. Models are loaded from the controllers from methods defined in the model. Views are written in HTML.

Python

* Framework used:
  + Django
  + Flask
  + Pyramid

Flask/Django uses a python file(routes.py/urls.py) which contains functions to service requests, these functions are assigned a uri, and manipulate/retrieve views. Views are implemented using a 'templating' strategy where you can define general templates and inherit them and modify pieces.

{%extends <template\_name.html> %}

{%Block content%} your modifications go here {%end block%}

These templates allow snippets of python like code which have access to the variables of the function in which it was retrieved, i.e. The function would contain variables passed from the HTTPRequest, and the variables would be used in the inherited template to manipulate the view appropriately.

Javascript

* Framework used:
  + Sails
  + angular.js
  + Backbone
  + Ember
* Server Side execution environments
  + Node.js
  + Rhino.js
  + Helma

Depending on the framework, there may or may not be a prototype defined for a model, but basically a model is represented as a javascript object with properties representing the application's data.

 JavaScript views are about building and maintaining a DOM element.

A view typically observes a model and is notified when the model changes, allowing the view to update itself accordingly. They are implemented through templating (such as Handlebars.js and Mustache) to restrict repetition.

Controllers are an intermediary between models and views which are classically responsible for updating the model when the user manipulates the view. Controllers are defined as objects which may have a prototype defined by the framework with method(s) that manage a view and update the Model, it also contains state from the model for the view.

Model