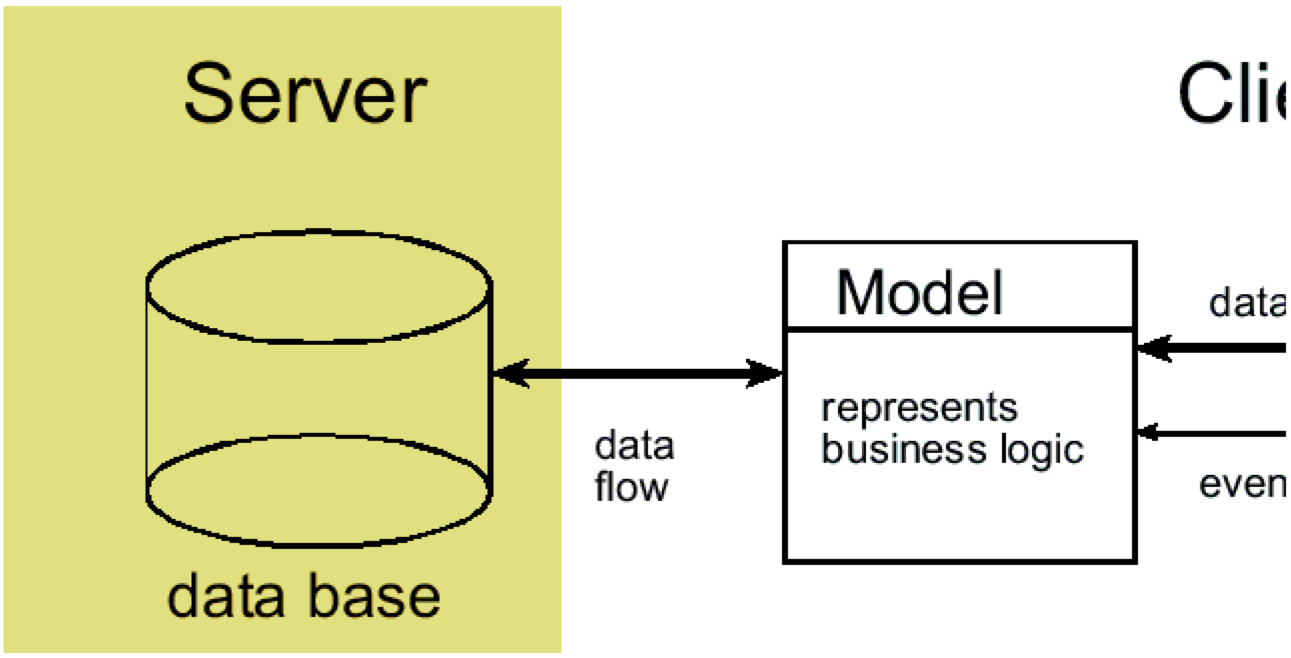
Exercise 8

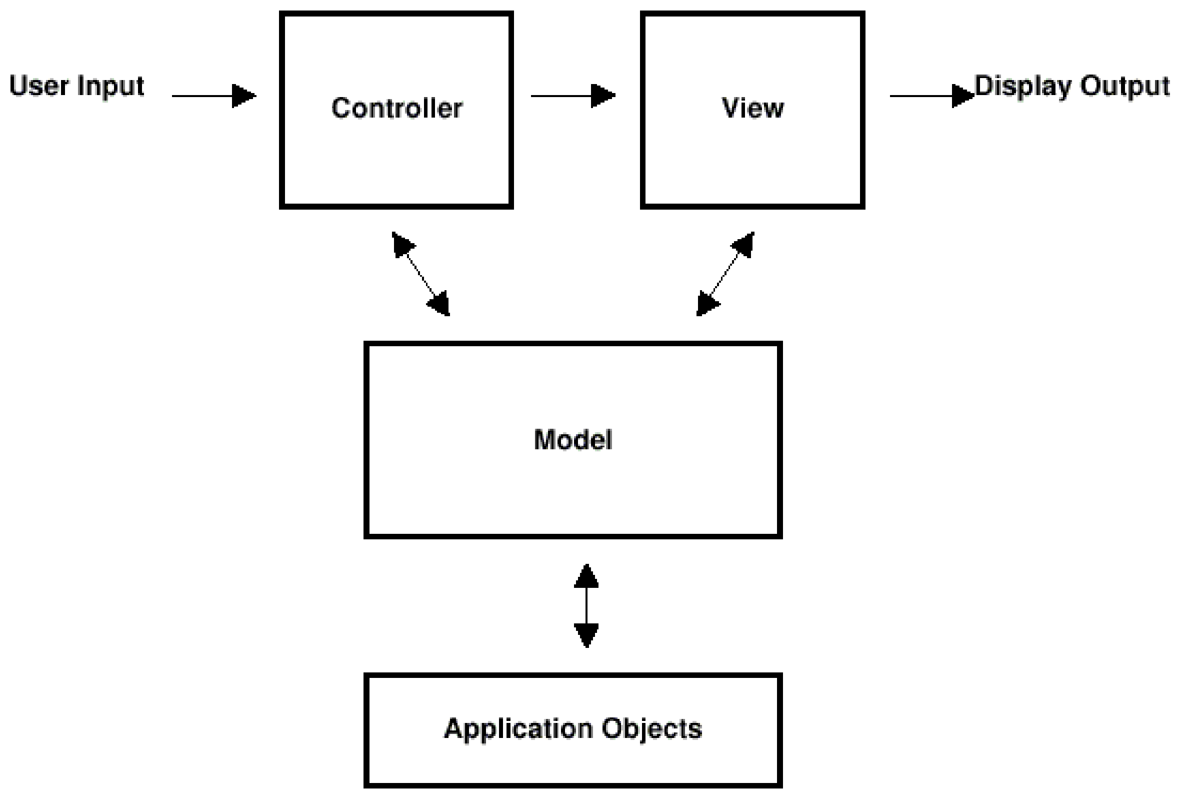
06.12.2002 **Due date: 09.01.2003**

Assignment 8.1.

Many commercial applications have a client/server architecture which follows roughly the schematic representation in the figure below: A client accesses and manipulates data in a data repository called server.The client might be conceptually split into model and view component,where the model corresponds to the particular business logic and the view to the visual representation of the model.

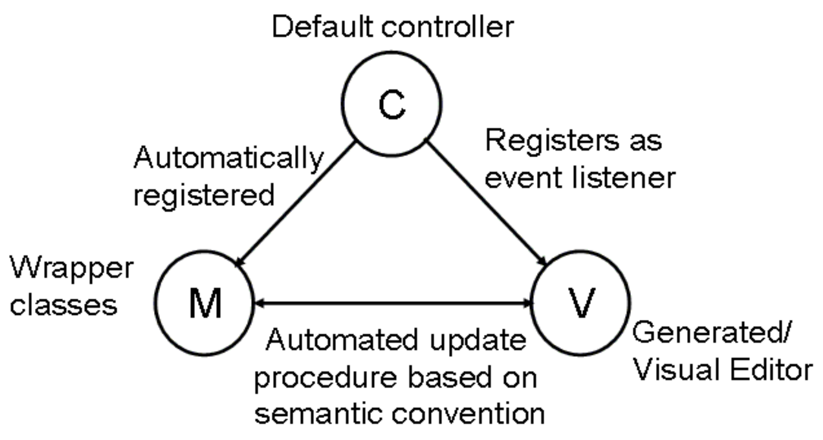


The Model-View-Controller (MVC) paradigm addresses the decomposition of the functionality.. The MVC model has three main components: the model (application), the view (output), and the controller (input). If the model changes, the controller noties the view that the underlying data has changed. It also knows how the user interaction with the view will affect the data in the model. A more general design of the coupling and interdependencies of the model, view, and controller are described by the Observer design pattern (c.f. Design Patterns, Gamma). The MVC model is depicted in the figure below. The view and the controller, since they represent both presentation and interaction aspects are most often tightly coupled and combined in one single component, although the MVC paradigm sees them as separate components.



The objective of the assigment is to clearly separate the view, model, and controller by automatically creating the linking between the view and the model with a **generic default controler** (one that can be used in various application) and naming conventions.

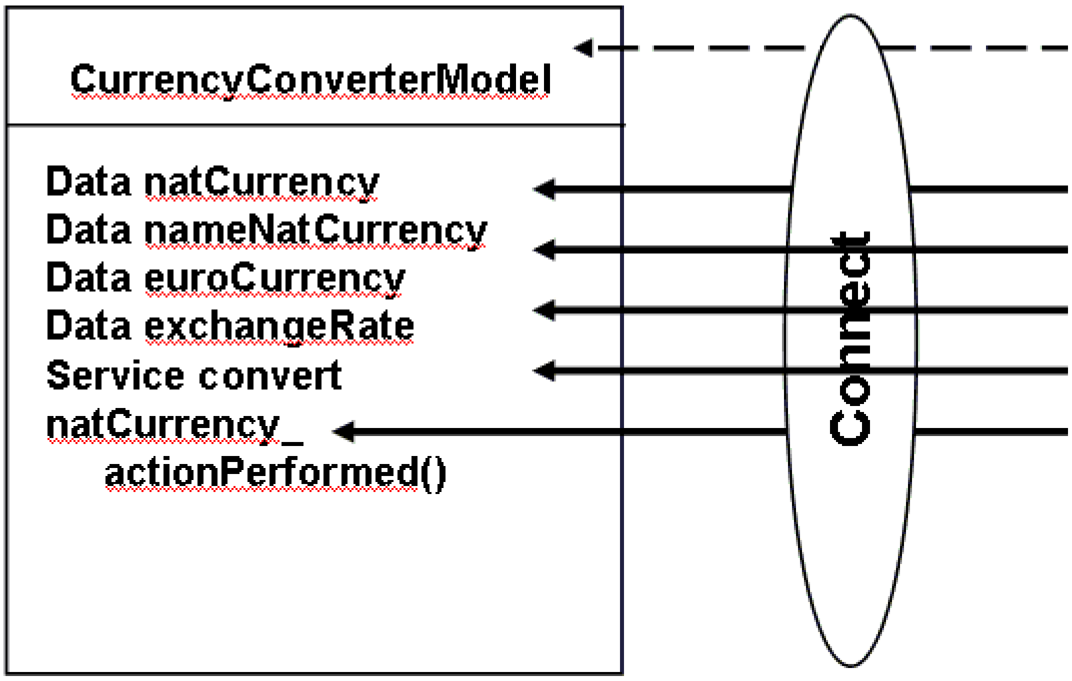
- Model: Wrapper classes for model elements to keep data integrity and to automate view synchronization
- View: Association of model and view elements using a semantic convention to automate the update procedure of the view and to facilitate the automatic generation of the view
- Controller: Default controller to simplify the event handling and to relieve programmer from controller programming



MODEL and VIEW PROGRAMMING

- A developer who defines a model basically defines what we call attributes and services .Attributes store data of a specific type.These types correspond in general to basic types,such as integer,float,double and String, but can also be more complex structures such as a lists or edit fields with special formatting such as date or currency fields.

- Services correspond to GUI elements which trigger action events, such as buttons and menu items.Services and attributes become instance variables of the class that represents the model. An attribute in the model represents a data container for data (taken from the data base,for instance).It usually has a visual representation in the view.One goal of a model/view separation is that model and view should only be semantically linked.
- In the assignment the implementation requires the developer to adhere to a simple naming convention: An element of the model and an element of the view are associated with each other if they have the same name.The programmer has only to ensure that the names of the instance variables are the same and your implementation takes care of the linking.
-



For example,the attribute with the name *lastName* (of type String)corresponds to the GUI field (text field)with the name *lastName* .If there is more than one visual representation for a model item, a *_2 (_3)* is attached to the name of the view elements (*lastName_2,…*) to distinguish them.This mechanism permits an unambiguous connection of model and view elements.

Besides of the functionality of a data container,an attribute further contains a fixed set of GUI status information.This status information somehow introduces a view flavor into the model.For example,lastName contains the data (String)and general information about the text field,whether it is visible,enabled and whether the focus has been set.Thus,the fact that model elements contain view information seems to undermine the strict model/view separation.However, this information belongs to the model,but affects,of course,the user interface.If a service is disabled,neither model nor view (button cannot be clicked)are able to change it directly. To handle the business logic associated with each attribute,a method can be defined which is invoked when the data of the attribute has been changed.The name of the method is composed by the name of the attribute and the String *Changed* as,for instance,lastNameChanged(). The association is based again on a naming convention. A service represents a GUI element that does not contain data but just triggers events.Buttons and menu items would be typical examples.To each service there is a corresponding method implemented in the model,which is invoked every time the GUI element is activated.It has the name exec concatenated with the name of that service,for instance,execSearchCustomer().

For further and more detailed studies refer to Design and Implementation of an MVC-Based Architecture for E-commerce Applications.