Interactive Computer Graphics and Model-View-Controller Architecture

Aaron Hitchcock and Kelvin Sung
Computing and Software Systems, University of Washington Bothell, Bothell, WA, USA

Synonyms

Model-view-controller (MVC); MVC architecture; MVC design pattern

Definition

Interactive graphics applications are a class of application that allows users to interactively update their internal states. These applications provide real-time visualization of their internal states with computer graphics. The model-view-controller (MVC) architecture is effective for presenting, discussing, understanding, and implementing this type of application.

As illustrated in Fig. 1, the **Model** contains the application state, the **View** renders the model graphically, and the **Controller** modifies the model. A **User** interacts with the MVC system by observing the content of the view and manipulating the controller to alter the state of the application.

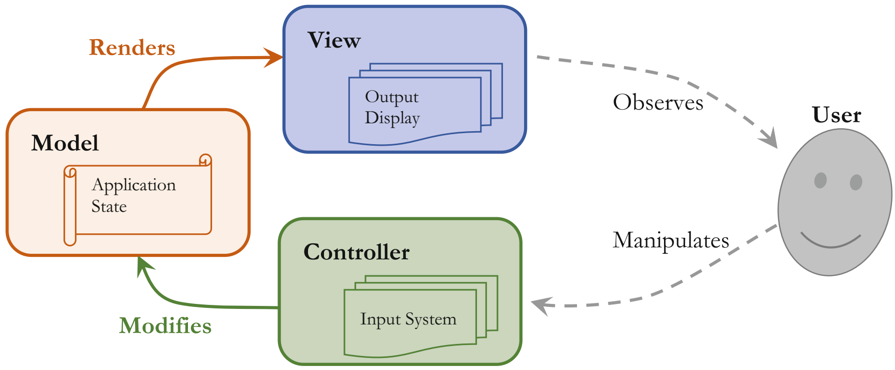
Implementation Considerations

The model defines the persistent application state and implements interface functions which allow it to be modified. The model implementation should be independent from the technologies that build the view and controller components. For example, the model of an image editing application should consist only of data structures and algorithms for defining and maintaining the abstract content of images. In this way, different views and controllers based on distinct libraries can be defined and implemented for the same model. For example, view/controller implementations for a PC-version and a Mac-version are based on the same model.

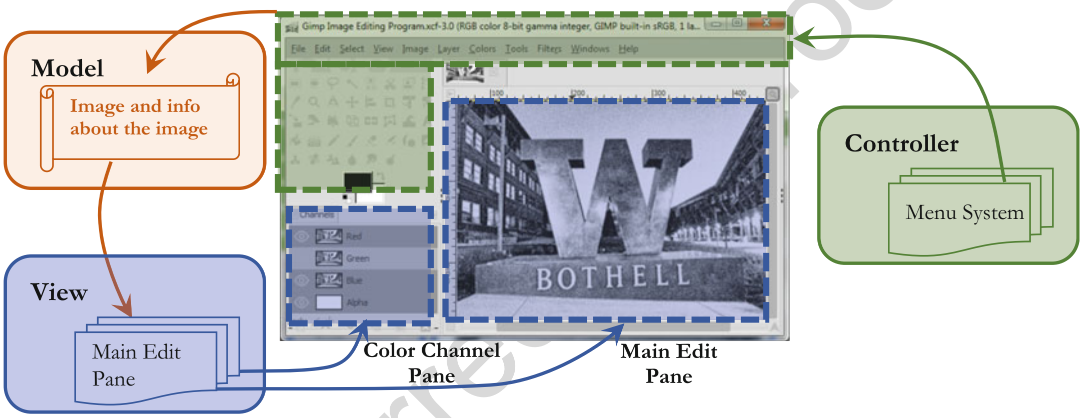
One important benefit of the MVC architecture is the clear enforcement of separation between state modification and visualization. During state modification, the controller receives user input and triggers the model to modify the application state. The MVC architecture ensures that the application state rendering is a completely separate process involving the model triggering the view. During this visualization stage, the application state should be read-only and should not be changed.

Figure 2 illustrates understanding GIMP, an image editor, as an MVC application. In this case, the Model (in orange), or the application state, is simply the image and information about the image. The view (in blue) renders and visualizes the application state as different panes in the application window, and the controller (in green)

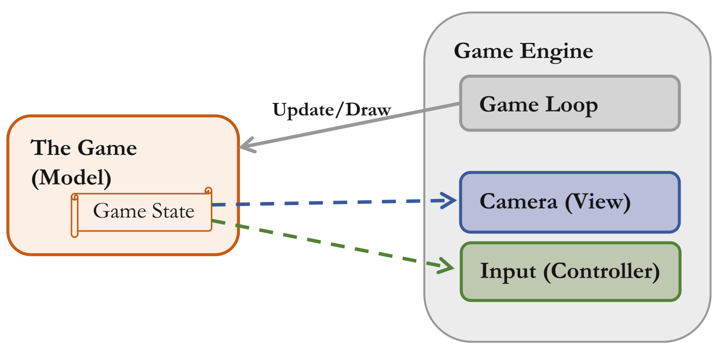
N. Lee (ed.), *Encyclopedia of Computer Graphics and Games*,
https://doi.org/10.1007/978-3-319-08234-9_223-1



Interactive Computer Graphics and Model-View-Controller Architecture, Fig. 1 The model-view-controller architecture



Interactive Computer Graphics and Model-View-Controller Architecture, Fig. 2 GIMP (an image editor) as an example MVC application



Interactive Computer Graphics and Model-View-Controller Architecture, Fig. 3 Modern video games and the MVC architecture provides the interface for the user to manipulate and update the image.

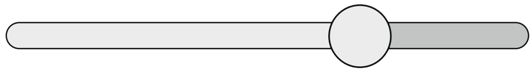
Context of Video Games

Modern video games are examples of interactive graphical applications. Typically, games are built based on specific game engines. As illustrated in Fig. 3, the game loop sub-system in the game engine periodically triggers the game to update and draw its state. In response, the game invokes the game engine functionality: the camera sub-system to render, and input sub-system to receive user commands. In this way, the game is the model responsible for defining and maintaining the game state, and the view and controller functionality are provided by the game engine.

Considering a video game as an MVC application ensures the separation of state update and draw operations. Game state should only be modified during the game engine update call, and only rendered during the game engine draw call. As discussed in the game loop implementation, the update and draw call frequencies are typically independent and can vary with the underlying system performance. Any attempts to draw the game state during update cycles or change the game state during draw cycles can easily result in a chaotic and unmanageable system.

Applying the MVC

It is interesting that the MVC architecture can be applied to interactive graphical systems of any scale. For example, the slider bar shown in Fig. 4 is a fully functional graphical interactive system. In this case, the model is a numeric value (typically a floating-point number), the view presents the numeric value to the user, and the controller allows the user to interactively modify the value. A typical view draws icons (bar and knobs) representing the range and current value in the model, whereas the controller typically supports mouse down and drag events to interactively modify the value in the model component. A slider bar implementation can choose to include an additional view by echoing the numeric value in a separate textbox. The corresponding controller would allow the user to modify the numeric value in the textbox. When the typing functionality is disabled, the view exists without a corresponding controller. AU2



Interactive Computer Graphics and Model-View-Controller Architecture, Fig. 4 A Unity3D slider bar

Cross-References

- ▶ Character Animation Scripting Environment
- ▶ Decoupling Game Tool GUIs from Core Editing Operations
- ▶ Game Engine
- ▶ Game Loop
- ▶ Physical, Virtual, and Game World Persistence

References AU3

Model-view-controller. Available https://en.wikipedia.org/wiki/Model-view-controller. Accessed 28 June 2018

Model-view-viewmodel. Available https://en.wikipedia.org/wiki/Model-view-viewmodel. Accessed 28 June 2018

Sung, K., Shirley, P., Baer, S.: Essentials of Interactive Computer Graphics: Concepts and Implementation. Taylor & Francis Group (2009). ISBN: 978-1-5688-1257-1 AU4