Model–view–controller

Model–view–controller (usually known as MVC) is a software design pattern[1] commonly used for developing user interfaces which divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user.[2][3] This kind of pattern is used for designing the layout of the page.

Traditionally used for desktop graphical user interfaces (GUIs), this pattern has become popular for designing web applications.[4] Popular programming languages like JavaScript, Python, Ruby, PHP, Java, C#, and Swift have MVC frameworks that are used for web or mobile application development straight out of the box.

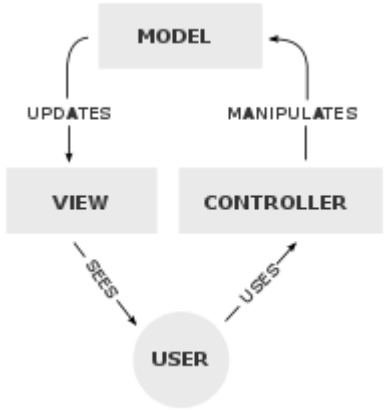

Diagram of interactions within the MVC pattern.

Contents

Components

Model
: The central component of the pattern. It is the application's dynamic data structure, independent of the user interface.[5] It directly manages the data, logic and rules of the application.

View
: Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

Controller
: Accepts input and converts it to commands for the model or view.[6]

In addition to dividing the application into these components, the model–view–controller design defines the interactions between them.[7]

- The model is responsible for managing the data of the application. It receives user input from the controller.
- The view means presentation of the model in a particular format.
- The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

As with other software patterns, MVC expresses the "core of the solution" to a problem while allowing it to be adapted for each system.[8] Particular MVC designs can vary significantly from the traditional description here.[9]

History

One of the seminal insights in the early development of graphical user interfaces, MVC became one of the first approaches to describe and implement software constructs in terms of their responsibilities.[10]

Trygve Reenskaug introduced MVC into Smalltalk-79 while visiting the Xerox Palo Alto Research Center (PARC)[11][12] in the 1970s. In the 1980s, Jim Althoff and others implemented a version of MVC for the Smalltalk-80 class library. Only later did a 1988 article in *The Journal of Object Technology* (JOT) express MVC as a general concept.[13]

The MVC pattern has subsequently evolved,[14] giving rise to variants such as hierarchical model–view–controller (HMVC), model–view–adapter (MVA), model–view–presenter (MVP), model–view–viewmodel (MVVM), and others that adapted MVC to different contexts.

The use of the MVC pattern in web applications exploded in popularity after the introduction of NeXT's WebObjects in 1996, which was originally written in Objective-C (that borrowed heavily from Smalltalk) and helped enforce MVC principles. Later, the MVC pattern became popular with Java developers when WebObjects was ported to Java. Later frameworks for Java, such as Spring (released in October 2002), continued the strong bond between Java and MVC. The introduction of the frameworks Django (July 2005, for Python)

and Rails (December 2005, for Ruby), both of which had a strong emphasis on rapid deployment, increased MVC's popularity outside the traditional enterprise environment in which it has long been popular. MVC web frameworks now hold large market-shares relative to non-MVC web toolkits.

Use in web applications

Although originally developed for desktop computing, MVC has been widely adopted as a design for World Wide Web applications in major programming languages. Several web frameworks have been created that enforce the pattern. These software frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server.[15]

Some web MVC frameworks take a thin client approach that places almost the entire model, view and controller logic on the server. This is reflected in frameworks such as Django, Rails and ASP.NET MVC. In this approach, the client sends either hyperlink requests or form submissions to the controller and then receives a complete and updated web page (or other document) from the view; the model exists entirely on the server.[15] Other frameworks such as AngularJS, EmberJS, JavaScriptMVC and Backbone allow the MVC components to execute partly on the client (also see Ajax).

Goals of MVC

Simultaneous development

Because MVC decouples the various components of an application, developers are able to work in parallel on different components without affecting or blocking one another. For example, a team might divide their developers between the front-end and the back-end. The back-end developers can design the structure of the data and how the user interacts with it without requiring the user interface to be completed. Conversely, the front-end developers are able to design and test the layout of the application prior to the data structure being available.

Code reuse

The same (or similar) view for one application can be refactored for another application with different data because the view is simply handling how the data is being displayed to the user. Unfortunately this does not work when that code is also useful for handling user input. For example, DOM code (including the application's custom abstractions to it) is useful for both graphics display and user input. (Note that, despite the name *Document Object Model*, the DOM is actually not an MVC model, because it is the application's interface to the user).

To address these problems, MVC (and patterns like it) are often combined with a *component architecture* that provides a set of UI elements. Each UI element is a single higher-level *component* that combines the 3 required MVC components into a single package. By creating these higher-level components that are independent of each other, developers are able to reuse components quickly and easily in other applications.

Advantages & disadvantages

Advantages

- *Simultaneous development* – Multiple developers can work simultaneously on the model, controller and views.
- *High cohesion* – MVC enables logical grouping of related actions on a controller together. The views for a specific model are also grouped together.
- *Loose coupling* – The very nature of the MVC framework is such that there is low coupling among models, views or controllers
- *Ease of modification* – Because of the separation of responsibilities, future development or modification is easier
- *Multiple views for a model* – Models can have multiple views

Disadvantages

The disadvantages of MVC can be generally categorized as overhead for incorrectly factored software.

- *Code navigability* – The framework navigation can be complex because it introduces new layers of indirection and requires users to adapt to the decomposition criteria of MVC.
- *Multi-artifact consistency* – Decomposing a feature into three artifacts causes scattering. Thus, requiring developers to maintain the consistency of multiple representations at once.
- *Undermined by inevitable clustering* – Applications tend to have heavy interaction between what the user sees and what the user uses. Therefore each feature's computation and state tends to get clustered into one of the 3 program parts, erasing the purported advantages of MVC.
- *Excessive boilerplate* – Due to the application computation and state being typically clustered into one of the 3 parts, the other parts degenerate into either boilerplate shims or *code-behind*[16] that exists only to satisfy the MVC pattern.
- *Pronounced learning curve* – Knowledge on multiple technologies becomes the norm. Developers using MVC need to be skilled in multiple technologies.
- *Lack of incremental benefit* – UI applications are already factored into components, and achieving code reuse and independence via the component architecture, leaving no incremental benefit to MVC.

See also

- Multitier architecture
- Hierarchical model–view–controller
- Model–view–adapter
- Model–view–presenter
- Model–view–viewmodel
- Entity-Control-Boundary pattern
- Presentation–abstraction–control
- Action–domain–responder
- Observer pattern
- Separation of concerns
- Strategy pattern
- Naked objects

References

1. https://www.youtube.com/watch?v=o_TH-Y78tt4&t=1667
2. Reenskaug, Trygve; Coplien, James O. (20 March 2009). "The DCI Architecture: A New Vision of Object-Oriented Programming" (https://web.archive.org/web/20090323032904/https://www.artima.com/articles/dci_vision.html). *Artima Developer*. Archived from the original (https://www.artima.com/articles/dci_vision.html) (html) on 23 March 2009. Retrieved 3 August 2019. "More deeply, the framework exists to separate the representation of information from user interaction."
3. Burbeck (1992): "... the user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of object."
4. Davis, Ian. "What Are The Benefits of MVC?" (http://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/). *Internet Alchemy*. Retrieved 2016-11-29.
5. Burbeck, Steve (1992) Applications Programming in Smalltalk-80:How to use Model–View–Controller (MVC) (https://web.archive.org/web/20120729161926/http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html)
6. Simple Example of MVC (Model–View–Controller) Architectural Pattern for Abstraction (http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design)
7. Buschmann, Frank (1996) *Pattern-Oriented Software Architecture*.
8. Gamma, Erich et al. (1994) *Design Patterns*
9. Moore, Dana et al. (2007) *Professional Rich Internet Applications: Ajax and Beyond*: "Since the origin of MVC, there have been many interpretations of the pattern. The concept has been adapted and applied in very different ways to a wide variety of systems and architectures."

10. Model–View–Controller History (http://c2.com/cgi/wiki?ModelViewControllerHistory). C2.com (2012-05-11). Retrieved on 2013-12-09.
11. Notes and Historical documents (http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html) from Trygve Reenskaug, inventor of MVC.
12. "A note on DynaBook requirements", Trygve Reenskaug, 22 March 1979, SysReq.pdf (http://folk.uio.no/trygver/1979/sysreq/SysReq.pdf).
13. Krasner, Glenn E.; Pope, Stephen T. (Aug–Sep 1988). "A cookbook for using the model–view controller user interface paradigm in Smalltalk-80" (http://dl.acm.org/citation.cfm?id=50757.50759). *The Journal of Object Technology*. SIGS Publications. **1** (3): 26–49. Also published as "A Description of the Model–View–Controller User Interface Paradigm in the Smalltalk-80 System (https://web.archive.org/web/20100921030808/http://www.itu.dk/courses/VOP/E2005/VOP2005E/8_mvc_krasner_and_pope.pdf)" (Report), ParcPlace Systems; Retrieved 2012-06-05.
14. The evolution of MVC and other UI architectures (http://martinfowler.com/eaaDev/uiArchs.html) from Martin Fowler.
15. Leff, Avraham; Rayfield, James T. (September 2001). *Web-Application Development Using the Model/View/Controller Design Pattern*. IEEE Enterprise Distributed Object Computing Conference. pp. 118–127.
16. "Is Model-View-Controller dead on the front end?" (https://medium.freecodecamp.org/is-mvc-dead-for-the-frontend-35b4d1fe39ec). 2016-10-09.

Bibliography