UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Model-View-Controller Design Pattern

Este patrón (diseño reusable) de diseño fue desarrollado como una solución orientada a objetos, al problema de crear buenas interfaces gráficas de usuario. Involucra tres objetos y sus asociaciones: model, view y controller.

Los diseñadores de los componentes de las Java Foundation Classes (JFC) de la interface de usuario Swing diseñaron cada componente alrededor del pattern MVC. El resultado es un toolkit de interface de usuario de flexibilidad casi incomparable.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Model-View-Controler Design Pattern

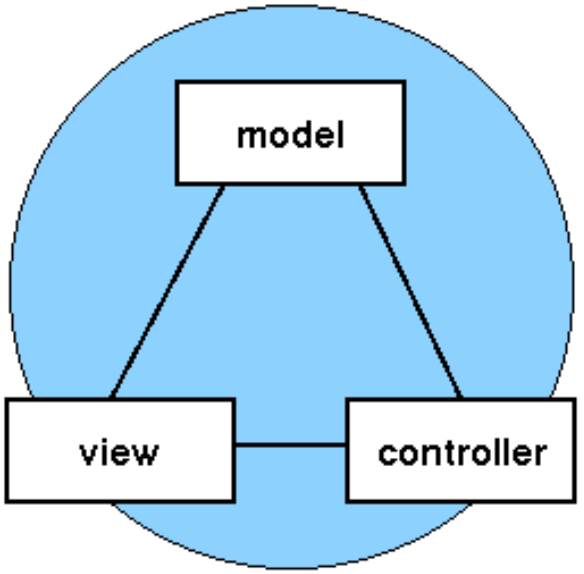
Los componentes GUI son fuentes de eventos y listeners que se implementan en clases que reciben eventos.

- Muchos de los componentes swing están basados en el pattern Model-View-Controler o MVC.
- La mayoría de los mecanismos para escuchar los eventos swing están basados en MVC.

Mientras que el pattern MVC se usa típicamente para construir interfaces de usuario completas, los diseñadores del JFC lo usaron como la base para cada componente individual de Swing. Cada componente de interface de usuario (button, table, scrollbar) tiene un modelo, una vista, y un controller. Además, ellos pueden cambiar, incluso mientras el componente está en uso. El resultado es un toolkit de interface de usuario de gran flexibilidad.

Model-View-Controler Design Pattern

Dijimos que MVC divide un componente de software en tres piezas distintas:



El *modelo* (model) es la pieza que representa el estado y el comportamiento a bajo-nivel del componente. Maneja el estado y conduce todas las transformaciones en ese estado. El modelo no tiene ningún conocimiento específico de sus controllers o sus views. El propio sistema mantiene links entre modelo y vistas y notifica a las vistas cuando el modelo cambia su estado.

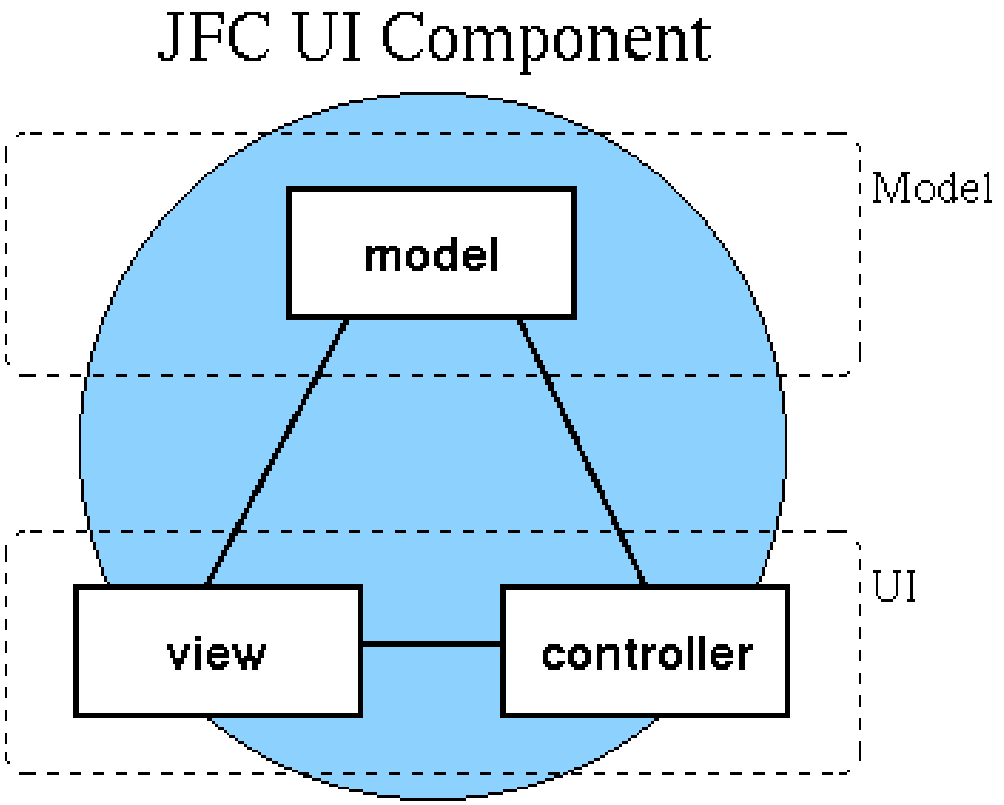
Model-View-Controler Design Pattern

La vista es la pieza que maneja el display visual del estado representado por el modelo.

El controller es la pieza que maneja la interacción del usuario con el modelo. Proporciona el mecanismo por el que se hacen cambios al estado del modelo.

Model-View-Controler Design Pattern

La figura siguiente ilustra cómo partir un componente de interface de usuario en un model, view y controller. Nota que view y controller se combinan en una pieza, una adaptación común del modelo de MVC básico. Ellos forman la interface del usuario para el componente.



Model-View-Controler Design Pattern

Veamos cada componente del pattern en detalle:

Model

- Hay un modelo en el pattern MVC.
- Este objeto tiene la responsabilidad de mantener el estado de los datos y responder a los requerimientos de esos datos.
- Debe proveer métodos de interfaz que permitan a la vista (view) requerir información sobre la colección de datos y otros que permitan al controller requerir cambios en el estado de los datos.
- Si los datos en el modelo pueden cambiar independientemente de las acciones de los objetos MVC, entonces el modelo debe ser capaz de iniciar actualizaciones de la Vista.

Model-View-Controler Design Pattern

View

- La vista tiene la responsabilidad de mostrar los datos del modelo.
- Para cada model, puede haber múltiples vistas. Por ej., una lista numérica puede mostrarse en un Table view y un Graph View.
- El modelo es un objeto único, de modo que los datos presentados son consistentes a través de todas las vistas.
- Dado que el modelo es pobremente acoplado a la vista, pueden introducirse nuevas vistas sin cambiar el modelo. El controller en cambio, es fuertemente acoplado a la vista.

Model-View-Controler Design Pattern

Controller

- Este objeto tiene la responsabilidad de interpretar las acciones del usuario sobre el teclado o con el mouse y comunicar esa información a la Vista o al modelo.
- Cada vista necesita tener su propio controller.
- La respuesta del controller al requerimiento del usuario comunicado vía mouse o teclado puede ser envío de mensajes al model solicitando que cambie su estado o envío de mensajes a la Vista requiriendo que actualice los datos que está mostrando.

Model-View-Controler Design Pattern

Cómo Java basa en MVC la implementación de los componentes Swing

Para entender cómo se relaciona MVC con los componentes Swing, veamos el ej. de un botón:

El modelo (Model)

El comportamiento del modelo en el botón es capturado por la interface ButtonModel. Una instancia de button model encapsula el estado interno de un botón simple y define cómo se comporta. Sus métodos pueden agruparse en las siguientes 4 categorías:

- Interrogar estado interno (si está habilitado, si ha sido presionado)
- Manipular estado interno
- Agregar y remover listeners de eventos
- Disparar eventos.

Notar que nada en el modelo está relacionado con la apariencia del botón o la forma en que responde al pressing o click (eventos).

Model-View-Controler Design Pattern

La vista (view) y el controller

El comportamiento de view y controller para el botón son capturados por la interface ButtonUI. Las clases que implementan esta interface son responsables de crear la representación visual del botón y manejar los inputs del usuario provistos vía mouse y teclado. Sus métodos pueden agruparse en las siguientes categorías:

- Paint
- Retornar información geométrica
- Manejar los eventos AWT.

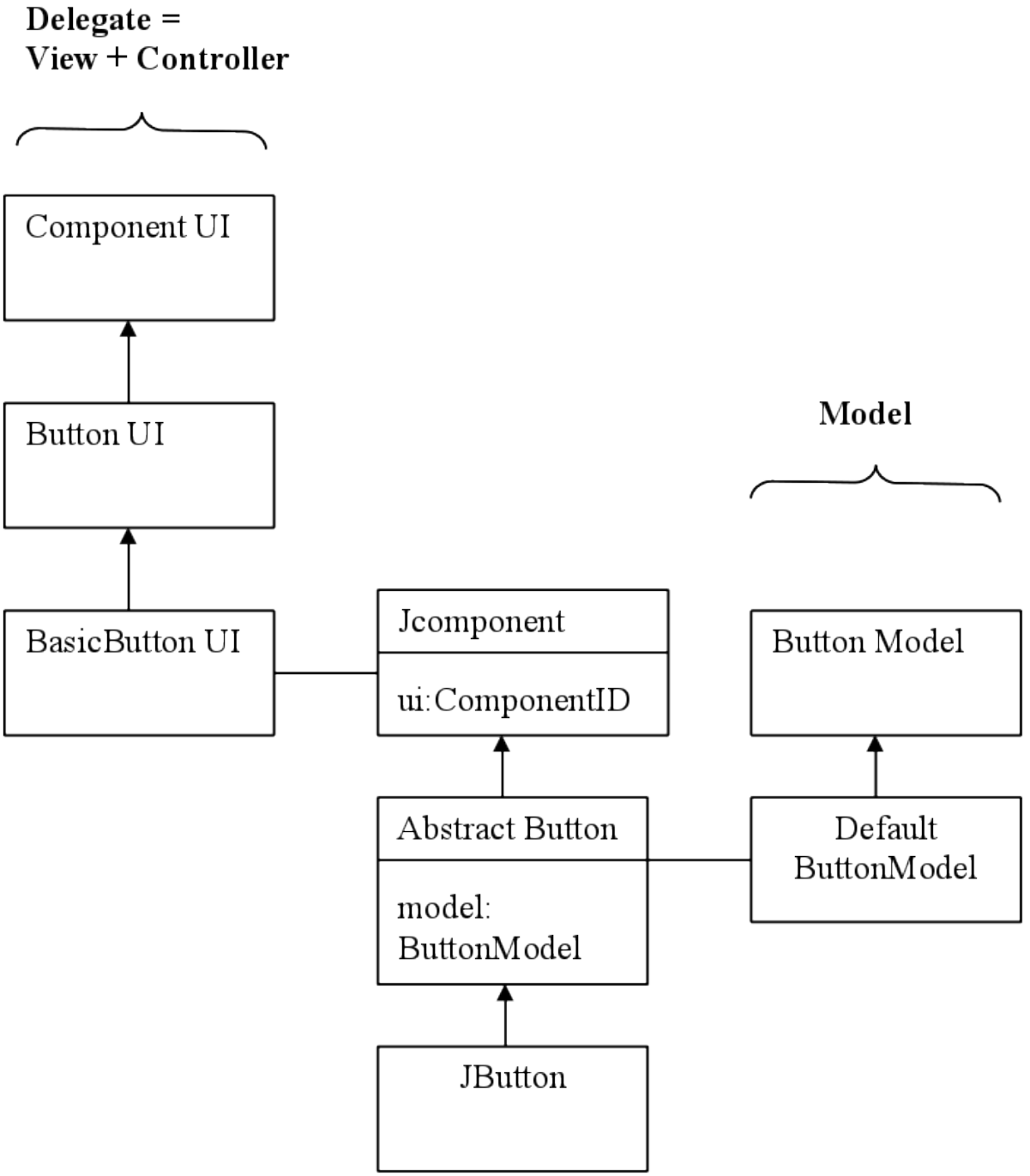
En principio, view y controllers son entidades separadas. Sin embargo están fuertemente relacionadas.
Este acoplamiento entre ambos, hace que sean considerados como una entidad común: un delegado. En la figura se muestra la relación entre JButton y su Delegate.

Model-View-Controler Design Pattern

El “Andamio”

Los programadores típicamente no trabajan directamente con las clases model y view/controller. Su presencia se esconde en una clase componente (subclase de java.awt.Component). Esta clase actúa como un “pegamento” que mantiene unido el MVC.

Model-View-Controler Design Pattern



UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Model-View-Controler Design Pattern

Resumiendo:
El concepto de MVC es el siguiente:
• Almacenar el estado interno en un conjunto de clases llamado *model.*
• Mostrar los datos del *model* en un *view*.
• Modificar los datos en el *model* en un *controller*.

Así como se utilizó el concepto MVC para las componentes swing, se utiliza para las aplicaciones.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

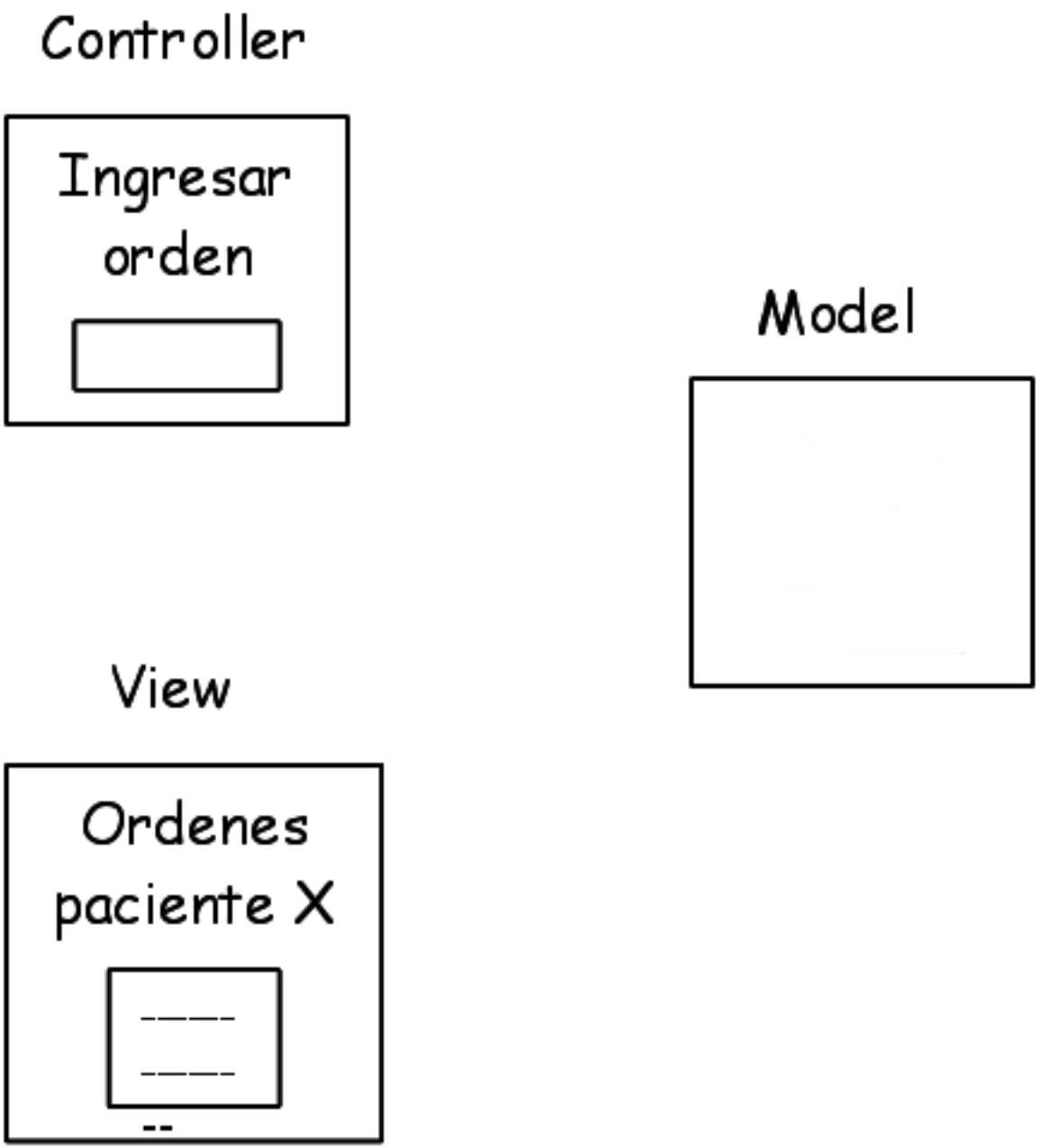
Model-View-Controler Design Pattern

Ejemplo:
Médicos que están continuamente ingresando y suspendiendo en un sistema, nuevas órdenes de medicación.

- El modelo (model) contiene una lista de todas las órdenes actuales para cada paciente.
- El controller es implementado en la interface que usan los médicos para prescribir órdenes.
- El sistema puede soportar muchas vistas: una lista de las órdenes actuales, una lista de posibles reacciones a determinada droga, etc.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Model-View-Controler Design Pattern



UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Model-View-Controler Design Pattern

Una aplicación “user-centered” manejada por eventos puede ser dividida en tres tipos de clases:

- un modelo, implementando la funcionalidad básica y los datos a ser preservados,
- una o más vistas, implementando el "look and feel" de la aplicación, y
- para cada vista, un controller para dar al usuario humano el control de la aplicación.

La vista incorpora los elementos visuales de la interface del usuario y el controller representa los mecanismos de control para la aplicación. (MVC era originalmente un paradigma de diseño básico en Smalltalk).

Para cada vista en la aplicación hay varios botones y campos de entrada de texto para darle comandos al modelo. Éstos representan los controllers, aunque al usuario ellos le parecen ser parte de las vistas.

Model-View-Controler Design Pattern

Los controllers en java son listeners de eventos como ActionListener y MouseListener.

La manera con la cual el sistema trabaja es la siguiente:

- El programador crea un modelo y por lo menos una vista.
- A cada vista se le da un conjunto de controllers y a cada control se da uno o más listeners de evento.
- Cada vista se "registra" con el modelo enviándole el mensaje addObserver. El modelo guarda una lista de todas las vistas registradas. Igualmente, cada listener se registra con el componente de Java para el que manejará eventos. Esto se hace con métodos como addActionListener.

Cuando un model cambia, tanto debido a cómputos internos, o porque algún agente externo (un controller) ha llamado a uno de sus métodos setter, ejecuta un método que envía un mensaje de actualización a todas las vistas registradas con una referencia al modelo cambiado, y un parámetro adicional de referencia que puede ser un objeto de cualquier tipo.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Model-View-Controler Design Pattern

Es relativamente fácil agregar nuevas vistas a una aplicación, dado que el modelo subyacente generalmente no necesita ser modificado para hacerlo. También es más fácil actualizar el modelo para nueva funcionalidad, dado que la interface y la funcionalidad se mantienen separadas.

Notar que la vista y sus controllers pueden mantenerse herméticamente acoplados. El mecanismo usual es hacer a las clases controller (listener), clases inners a los elementos de vista a los que ellos corresponden.

Por sobre todo, el paradigma model-view provee una forma de pensar sobre el software user-centered, que nos permite separar claramente los elementos de la interface de los elementos de funcionalidad.