Model View Controller

26 Marzo 2019

Scaglione San Pietro

RESPONSABILI

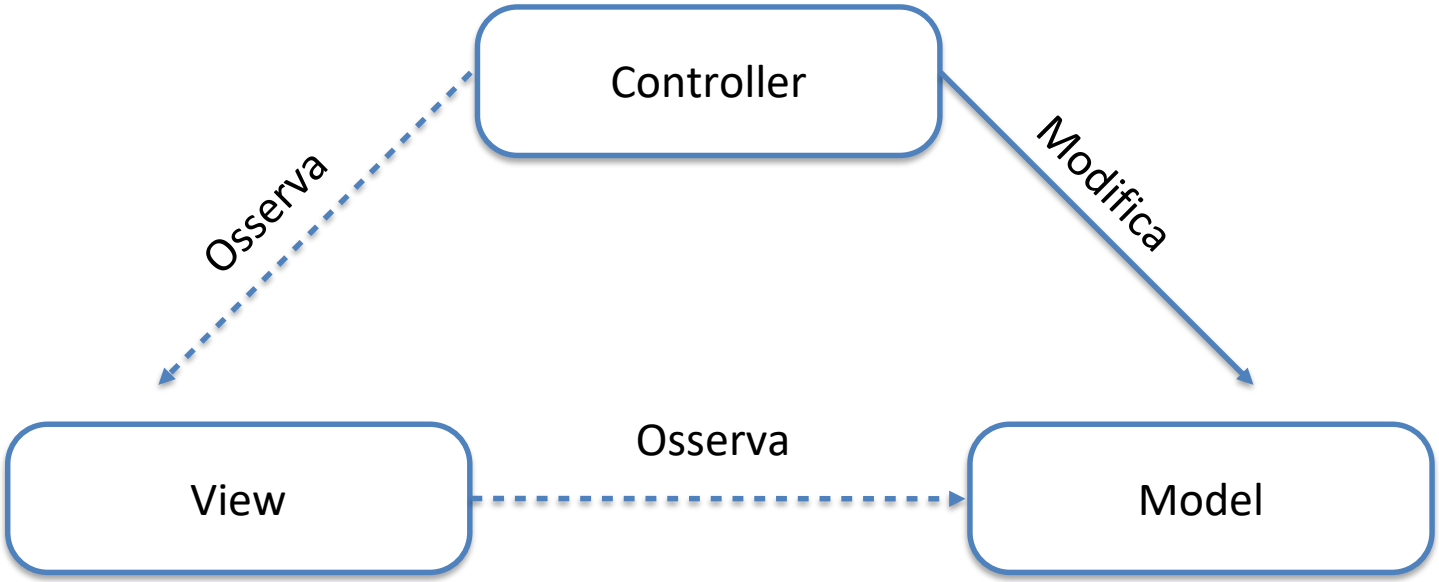
Giovanni Meroni
Amarildo Likmeta

TUTOR

Marco Bacis
Valentina Deda

Model-View-Controller

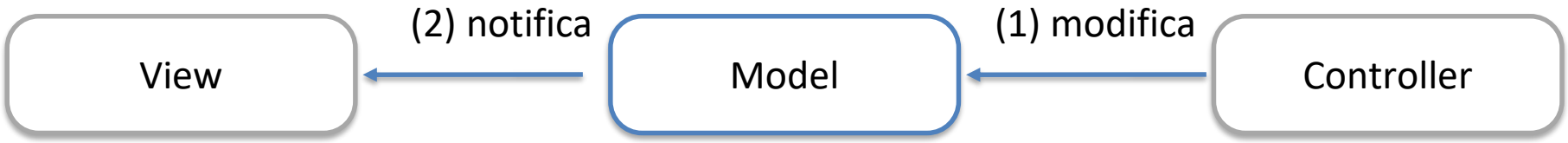
- ***Pattern architetturale*** che separa il modello dei dati di una applicazione dalla rappresentazione grafica (view) e dalla logica di controllo (controller)



Model

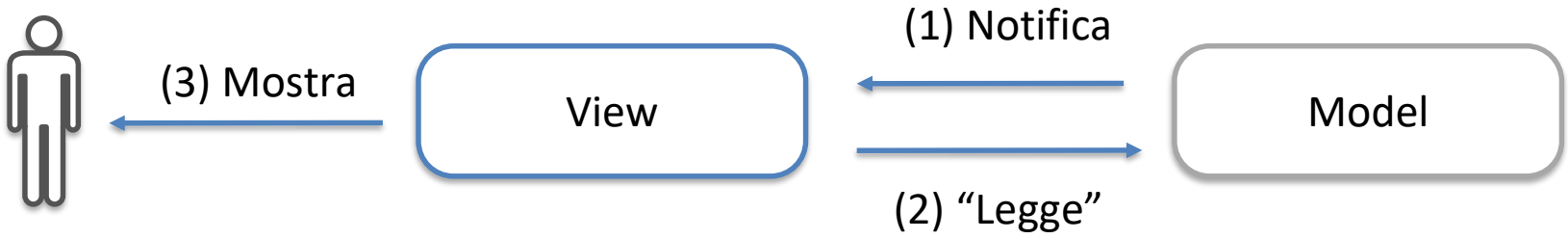
- ***Incapsula lo <u>stato</u>*** *di una applicazione*
  - *contiene le classi che descrivono i dati dell'applicazione e le operazioni per manipolarli*

- *Fornisce i metodi per* ***accedere*** ai dati
- ***<u>Notifica</u>*** i cambiamenti di stato (alla view)

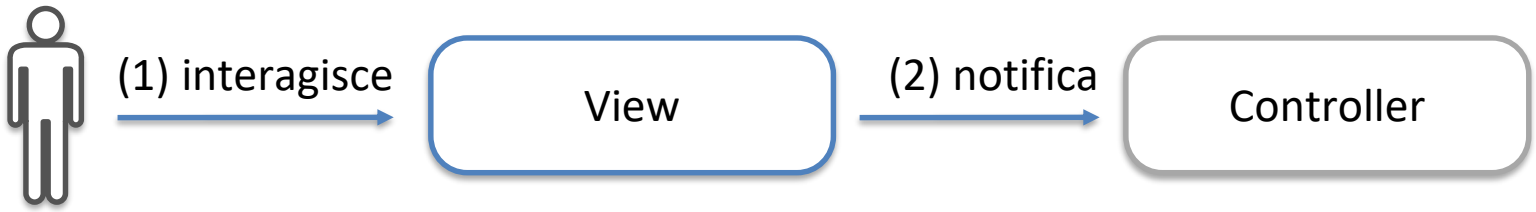


View

- ***Mostra*** *lo stato dell'applicazione (modello)*



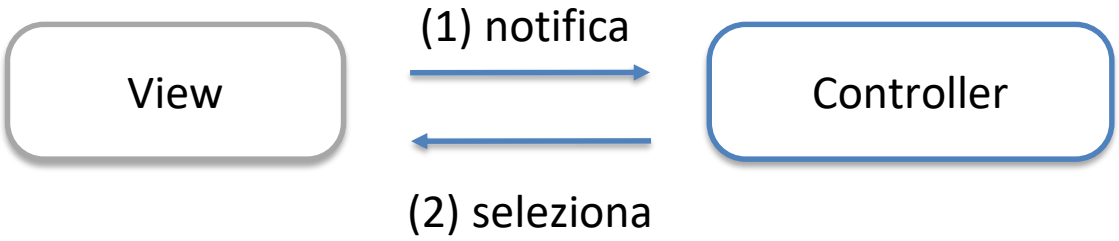
- *Gestisce* ***l'interazione*** *con l'utente*



Controller

- *Rappresenta la **logica applicativa***
- *Collega le **azioni** dell'utente con **modifiche allo stato***

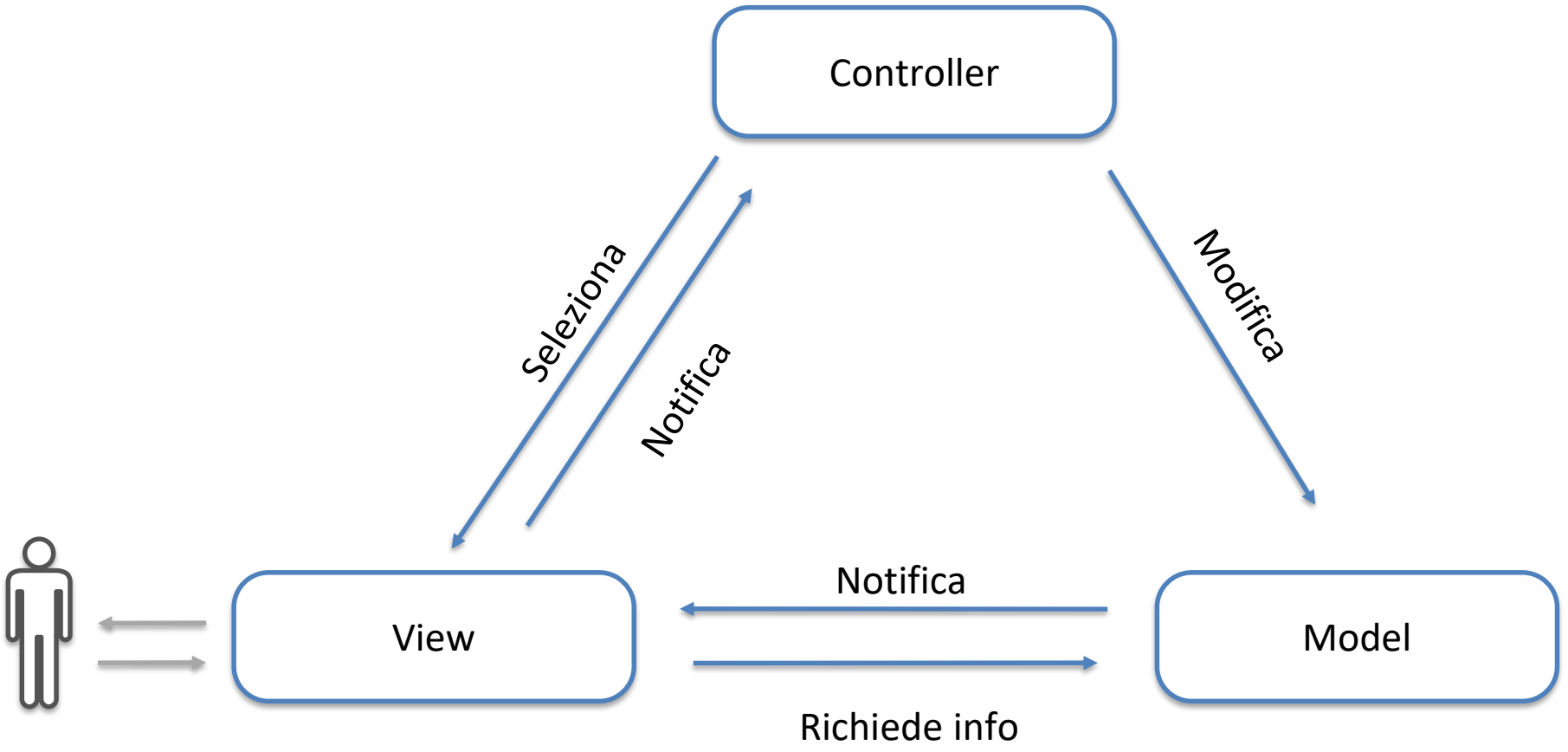
- *Sceglie cosa deve essere mostrato*



View - Controller

- *La view deve rendere accessibile al controller metodi generici / comportamentali (es. showMessage, resetBoard, showFinalScores …). Il controller non deve mai intervenire direttamente sugli elementi grafici.*

Model-View-Controller



Come si realizza?

- *Utilizzo della **programmazione ad eventi***
- *Un oggetto sorgente genera eventi*
- *Un ascoltatore registrato viene notificato degli eventi*

Programmazione ad eventi

EVENTI

- *Azioni dell'utente o notifiche dal model al controller*
- *Sono classi che contengono informazioni dettagliate sull'evento*

ASCOLTATORI (Listeners)

- *Si mettono in ascolto di un evento*
- *Devono avere dei metodi per poter reagire agli eventi*
- *Possono esserci più ascoltatori per un evento*

SORGENTI DI EVENTI

- *Notificano gli eventi agli interessati*
- *La notifica avviene invocando i metodi sugli ascoltatori*
- *Devono avere un metodo per permettere la registrazione degli ascoltatori*

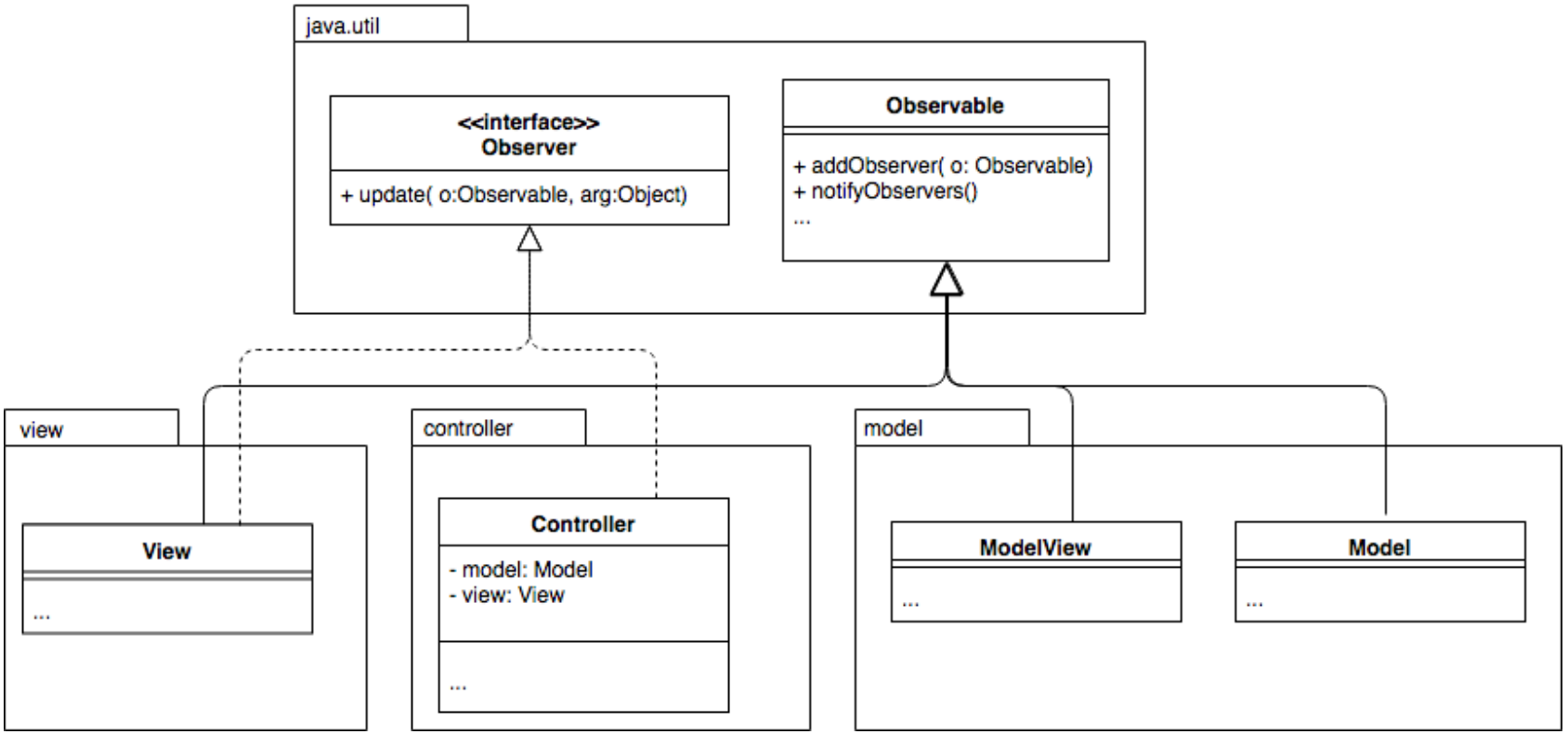
Passaggio Oggetti

- ***Sempre*** *oggetti che rappresentano eventi o creati appositamente*
- *Oggetti modificabili del modello* ***non devono*** *arrivare all'interfaccia*

Soluzioni:

- ***Interfacce limitate*** *(solo metodi per la visualizzazione delle informazioni selezionate)*
- ***Oggetti immutabili***
- *Oggetti creati appositamente per la visualizzazione*

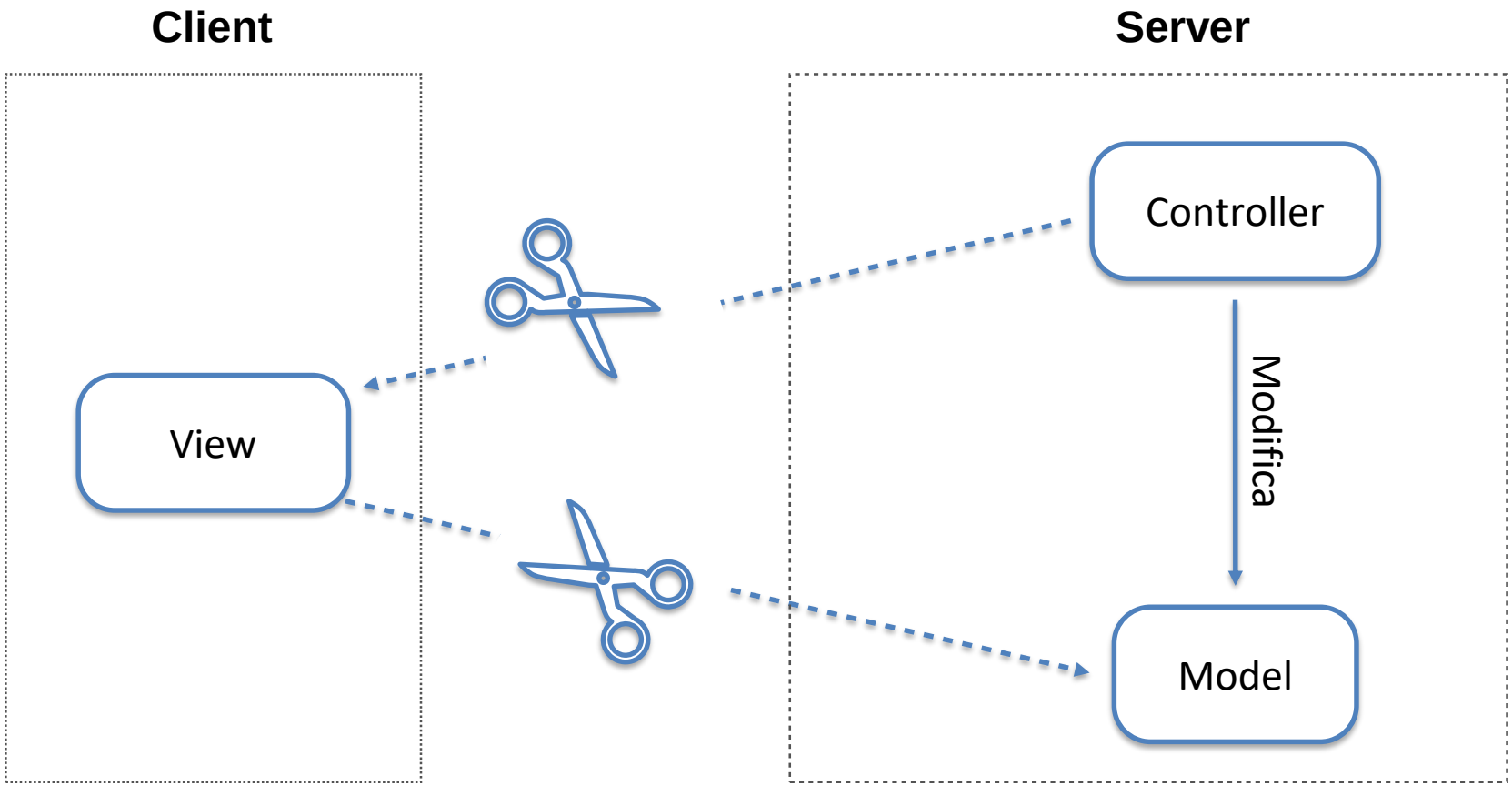
Observer e Observable



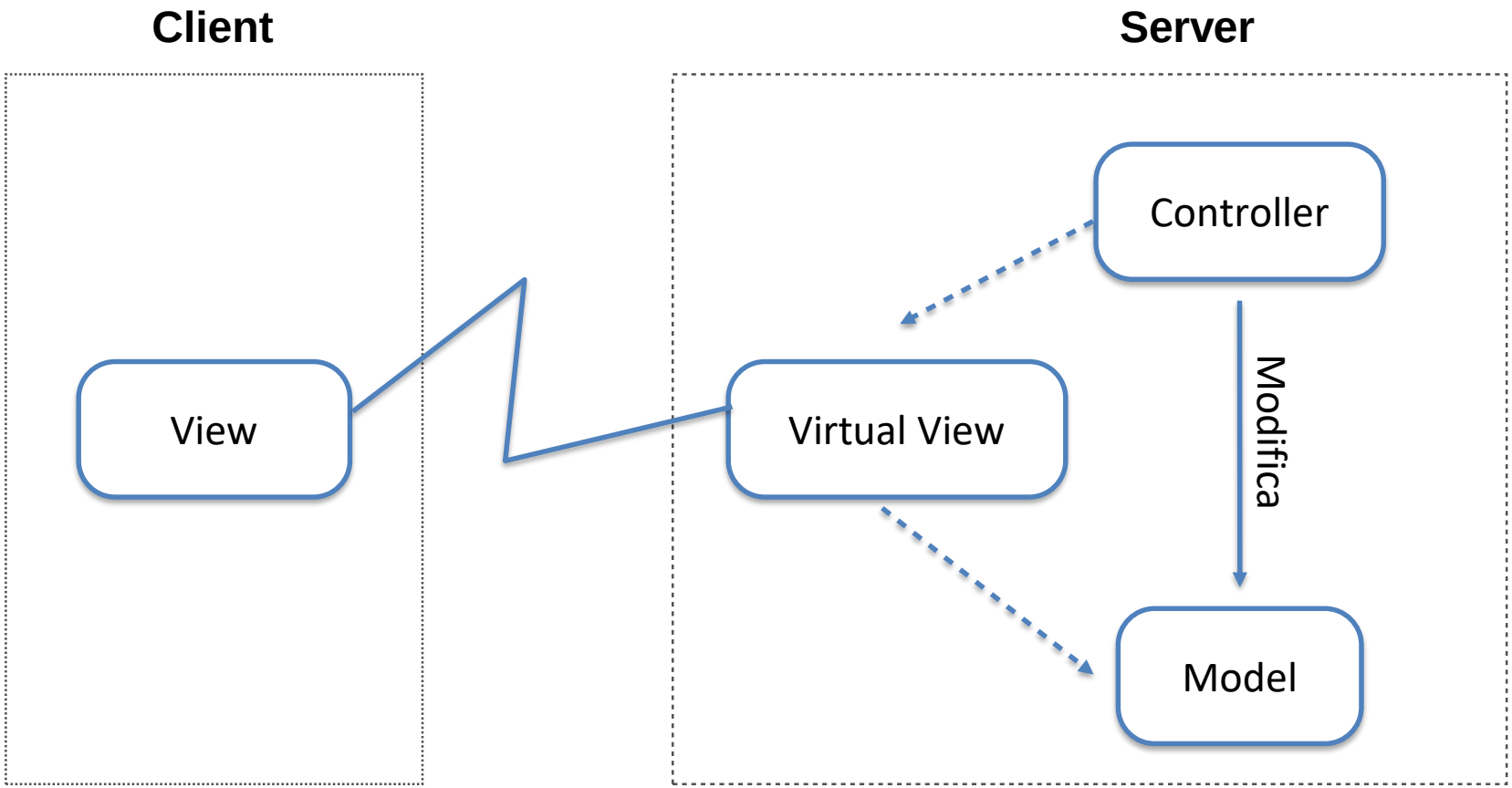
MVC + Rete

- *Non far confusione tra MVC e network (separazione dei task)*
- *Non hardcodare le tecnologie di rete (estensibilità del codice)*
- *Nascondi la gestione della rete alla classi MVC (encapsulation)*

MVC + Rete



MVC + Rete



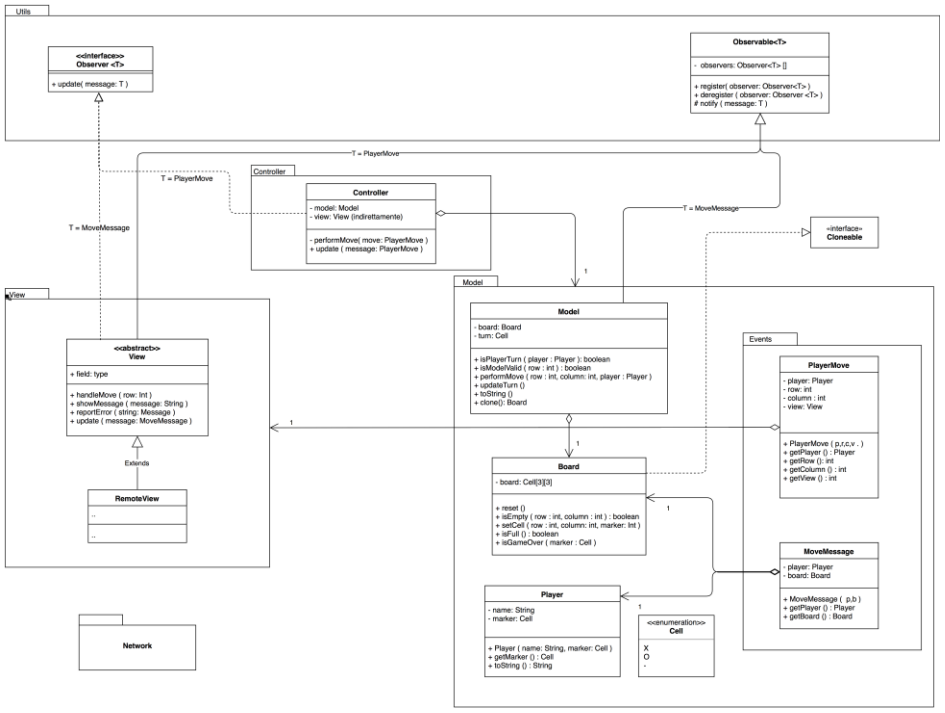
MVC + Rete

Virtual View

- *Come una normale View è **Observer** del model*
- *Come una normale View è **Observable** del controller*
- ***Agisce sulla rete** "all'insaputa" di model e controller*
  - *Inoltra gli eventi ricevuti dal modello attraverso la rete (alla view del client)*
  - *Riceve eventi dalla view del client attraverso la rete e li inoltra al controller*

Esempio: Tris

- *Esempio visto durante le esercitazioni*
- *UML nel PDF*
- *Rete omessa*



Scelte progettuali dell'esempio

- *Vengono implementati **Observer** od **Observable** custom facenti uso dei Java generics per poter distinguere tra i vari eventi*
- *La View notifica il controller della mossa effettuata da un giocatore tramite l'evento **PlayerMove***
- *Il modello notifica i cambiamenti alla View tramite l'evento **MoveMessage***
- *E' accettabile il passaggio di classi del modello alla view (tramite eventi) in quanto Board è **clonabile** e Player è **immutabile***
- *Il controller non ha un riferimento diretto alla view. Il riferimento alla view è tenuto in PlayerMove.*
- *La view rende accessibili al controller metodi generici / comportamentali; il controller non interviene sugli elementi grafici*
- *La view usa semplici getter e setter sugli immutabili / cloni provenienti dal model*