ISAM34
Prova finale di Ingegneria del Software

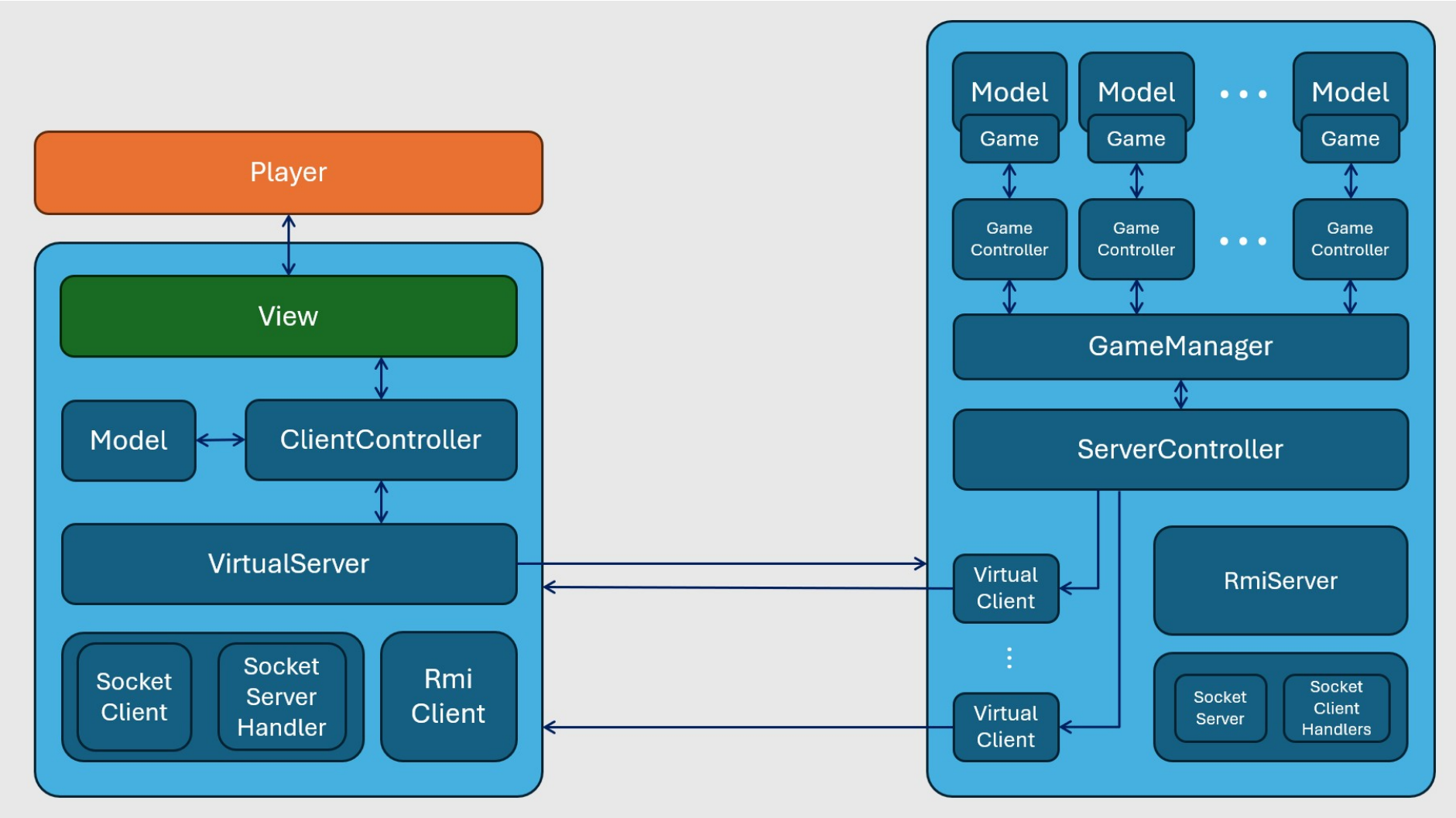
Antonio Augello, Lorenzo Baggi, Stefano Bernardotto, Andrea Brugnera

Politecnico di Milano

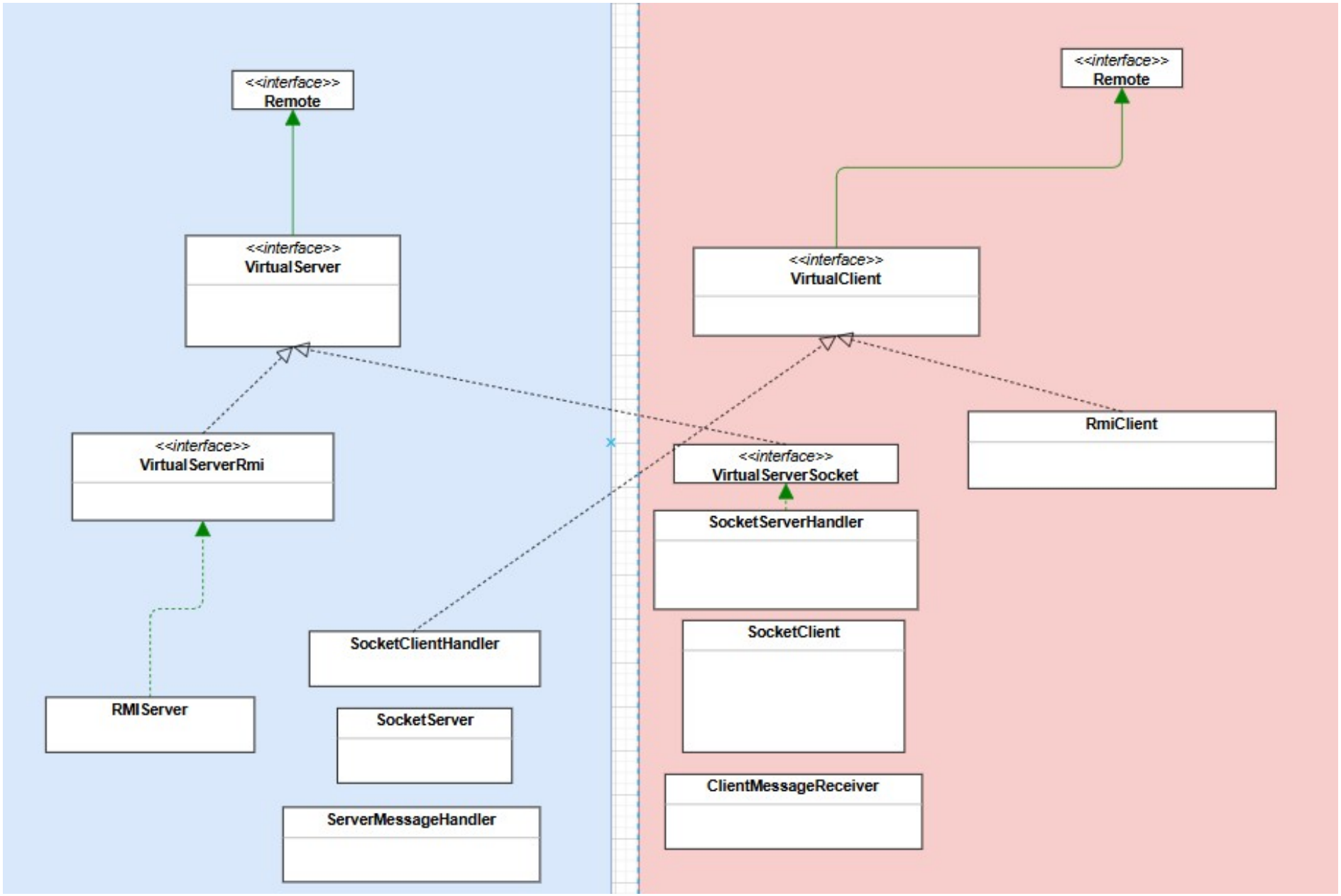
1 Luglio 2025

Indice

Struttura di rete



Architettura Client Server



SocketClient

Client → Server

- SocketServerHandler che gestisce ObjectOutputStream e viene chiamato direttamente da ClientController.

Server → Client

- ServerMessageThread : sta in ascolto su ObjectInputStream e aggiunge i messaggi alla coda
- ServerReceiverThread : legge dalla coda e chiama i metodi sul ClientController attraverso Executors.newSingleThreadExecutor.

RMIClient

Server → Client

- usa Executors.newFixedThreadPool(4) per chiamare i metodi sul ClientController.

Client → Server

- il ClientController chiama RMIClient solo quando fa la prima connect(), successivamente ottiene direttamente il riferimento al server e chiama direttamente il Server.

RMIServer

Client → Server

- utilizziamo un pool di Executors, vengono creati su richiesta : ThreadPoolExecutor(0 , Integer.MAX_VALUE, 0L, TimeUnit.MILLISECONDS, new SynchronousQueue<>()).

Server → Client

- Thread all'interno di GameController che gestisce clientCommunication per aggiornare gli stati al Client.

SocketServer

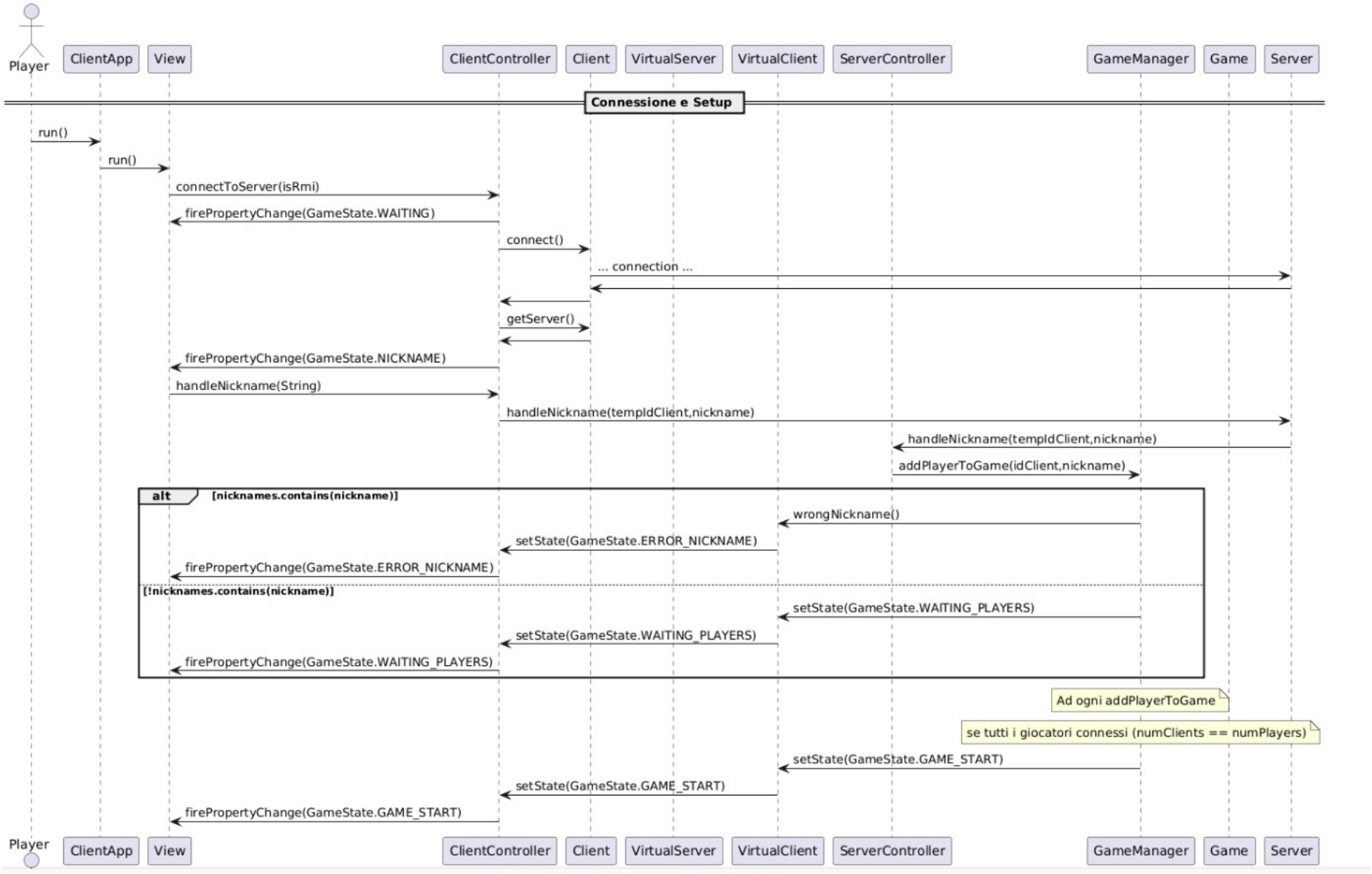
Client → Server

- Processo principale resta in un loop per accettare nuove connessioni.
- Quando accetta connessioni crea un SocketClientHandler e avvia due thread : uno per mettere i messaggi in cosa l'altro per eseguire la coda dei messaggi.

Server → Client

- Thread all'interno di GameController che gestisce clientCommunication per aggiornare gli stati al Client.

Sequence diagram connessione



Design Pattern Utilizzati

- **Iterator:** svariati utilizzi
- **Observer (firePropertyChange):**
  - Reattività GUI
  - Messaggi broadcast ai client
- **Factory Pattern:**
  - Diverse modalità di gioco: EASY & NORMAL
  - Usato per `FlyBoard` e `ShipBoard`
  - Modalità di connessione
- **Sealed Classes con Pattern Matching:**
  - Messaggi
  - Eventi
  - Carte avventura

Modalità di aggiornamento

Gestione aggiornamenti:
Il sistema utilizza una **coda di stati di gioco** che raccoglie i cambiamenti e aggiorna la vista in base al contesto.

- **TUI:** il circuito è mostrato solo su richiesta o al termine di una carta; nessun aggiornamento attivo necessario.
- **GUI:** il circuito è sempre visibile, quindi è gestito da una **coda dedicata** svuotata da un thread separato.
- **Pattern Observer:** aggiornamenti eseguiti con `PropertyChangeEvent`; l'interfaccia `View` estende `PropertyChangeListener` e riceve notifiche dal `ClientController` tramite `firePropertyChange`.