# Esercitazione: Architetture

Vincenzo Gervasi, Laura Semini
Ingegneria del Software
Dipartimento di Informatica
Università di Pisa

# Ex 1

Si assuma che sia stata definita un'architettura con i seguenti componenti:

- app_cliente (che implementa l'app usata dal cliente)
- sistema_prenotazione (che implementa la business logic del sistema)
- db_anagrafica (che raccoglie dati anagrafici su autisti e clienti)
- db_corse (che raccoglie dati su prenotazioni, corse assegnate, corse effettuate)
- app_autista (che implementa l'app usata dall'autista)

Si disegni il diagramma C&C, specificando gli stereotipi corrispondenti al tipo di interazione che si immagina sui connettori.

# Ex2

Sotto le stesse assunzioni dell'esercizio 1, e assumendo un artefatto (omonimo) per componente, si disegni un diagramma di dislocazione (deployment) per il sistema.
Si assuma, che il db_anagrafica venga aggiornato poco frequentemente (solo quando si iscrivono nuovi autisti e nuovi clienti), e che venga interrogato con query semplici, mentre il db_corse sia soggetto ad aggiornamenti più frequenti, e debba eseguire query complesse (per esempio, per verificare quali autisti saranno liberi a un dato orario occorre incrociare le ore di inizio e fine servizio di tutte le corse attualmente prenotate e delle altre richieste pendenti).

# Estensione Aeroporti

- Il servizio di prenotazione di REBU si arricchisce di una nuova feature: la possibilità di associare un numero di volo, in caso di corse da/per un aeroporto.
- Nel caso di spostamenti verso un aeroporto, al momento della richiesta, il cliente specifica il numero del volo e con quanto anticipo, in minuti, desidera arrivare in aeroporto rispetto alla partenza del volo. Il sistema si interfaccia con i servizi dell'aeroporto e ottiene l'orario di partenza. Si interfaccia quindi con un servizio di calcolo di percorso stradale (tipo google maps) per ottenere i tempi di percorrenza tra la posizione di ritrovo e l'aeroporto, e stabilire un orario di partenza per la corsa. A questo punto procede come con una normale prenotazione.
- 
- Nel caso di spostamenti da un aeroporto, al momento della richiesta, il cliente specifica il numero del volo e se avrà del bagaglio imbarcato da attendere dopo l'atterraggio. Il sistema si interfaccia con i servizi dell'aeroporto, ottiene l'orario di atterraggio del volo, stabilisce un orario di partenza per la corsa, e procede come con una normale prenotazione.
- 
- Nelle ultima due ore prima dell'inizio della corsa, REBU monitora eventuali ritardi dei voli. In caso di ritardo superiore ai 15 minuti, ricalcola l'orario di partenza della corsa, cerca un nuovo autista per la corsa ritardata (allo stesso prezzo della corsa già pattuita con l'utente), e cancella la precedente. Quindi informa l'utente del nuovo orario.

# Estensione Aeroporti: Ex

La progettazione architetturale ha portato all'individuazione di una componente GestioneRitardi che si occupa di monitorare eventuali ritardi dei voli e, se necessario, modificare la corsa. I servizi offerti dai vari aeroporti non sono uniformi: alcuni offrono la possibilità di abbonarsi alle informazioni su un volo e informano REBU di eventuali ritardi (modalità PUSH), altri offrono solo un servizio di informazioni domanda/risposta (modalità PULL). Descrivere, con un diagramma di sequenza, le interazioni tra GestioneRitardi e l'aeroporto, nei due casi.

# Estensione: condivisione auto

- *Al fine di estendere il servizio, REBU attiva un programma di condivisione delle auto. Un autista durante le ore di riposo può decidere di lasciare la sua auto a disposizione di autisti che possono lavorare ma non hanno un'auto di proprietà con gli standard richiesti da REBU. A tal fine deve indicare la posizione dell'auto e l'intervallo orario in cui non la userà.*
- *Un autista che voglia usare un'auto condivisa cerca tra quelle disponibili, la prenota per il tempo necessario a raggiungerla (le prenotazioni durano max 20 minuti), ne prende possesso, lo segnala. Quando termina il turno la parcheggia in un raggio di 500 metri rispetto a dove l'ha trovata e segnala la nuova posizione. Il sistema provvede ad addebitare/accreditare il noleggio sull'account degli autisti coinvolti.*
- *Il meccanismo di apertura e messa in moto di auto condivise si basa sull'uso di un codice monouso inviato in risposta a una prenotazione.*

# Estensione: condivisione auto, Ex

La progettazione architetturale ha individuato le seguenti componenti:

- **DB auto**, che mantiene il parco auto, con il loro stato corrente,
- **DB autisti**, che mantiene informazioni sugli autisti, comprensive del loro conto economico; **Gestione Prestiti** che realizza la business logic.

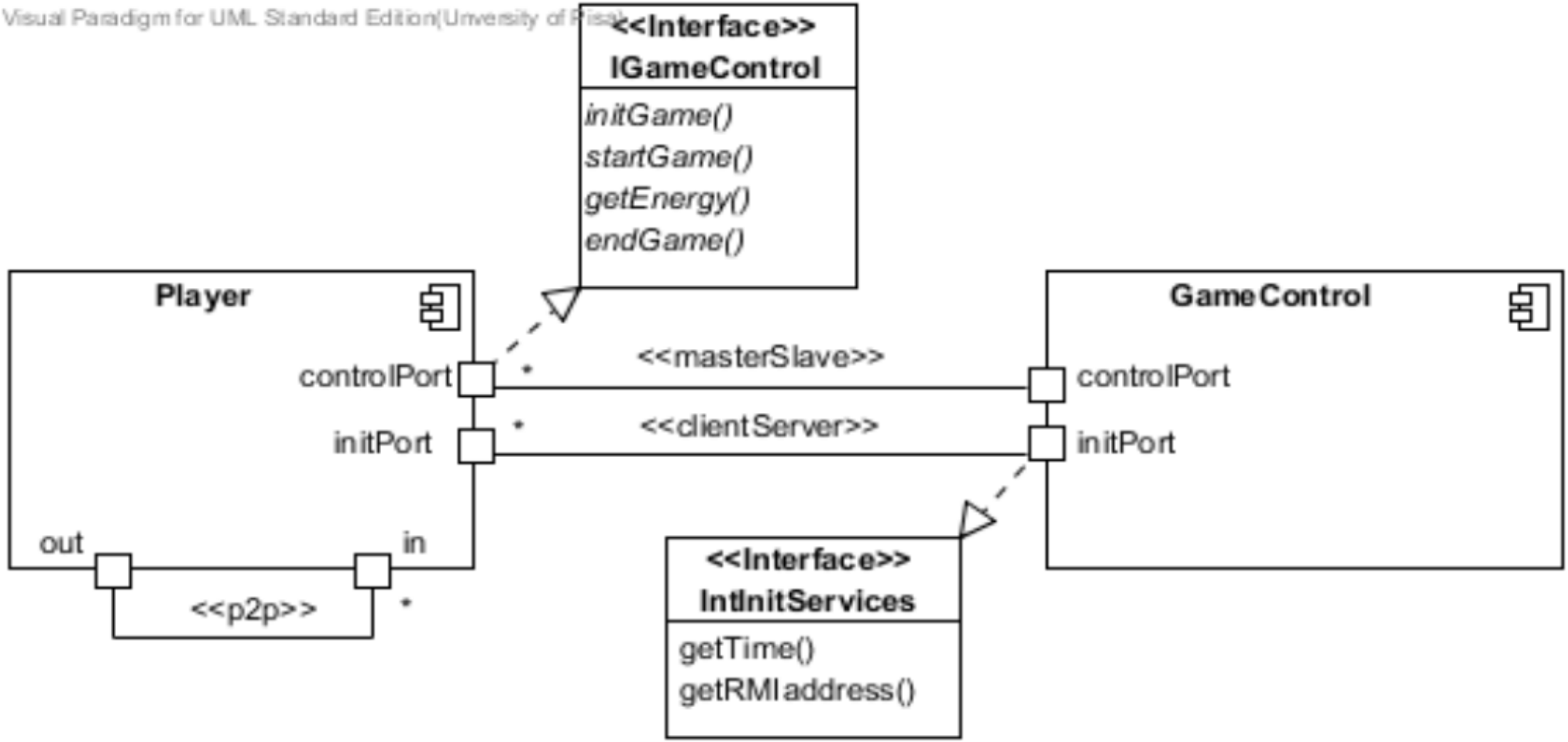
Dare

- La vista C&C
- un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso Presa in Prestito, dal momento in cui un autista inizia una ricerca fra le auto condivise, fino a quando segnala la presa di possesso di un'auto. Il caso d'uso prevede una notifica al proprietario nel momento in cui l'auto è prenotata, e un'altra quando viene effettivamente presa in uso.

# Cops & Robbers

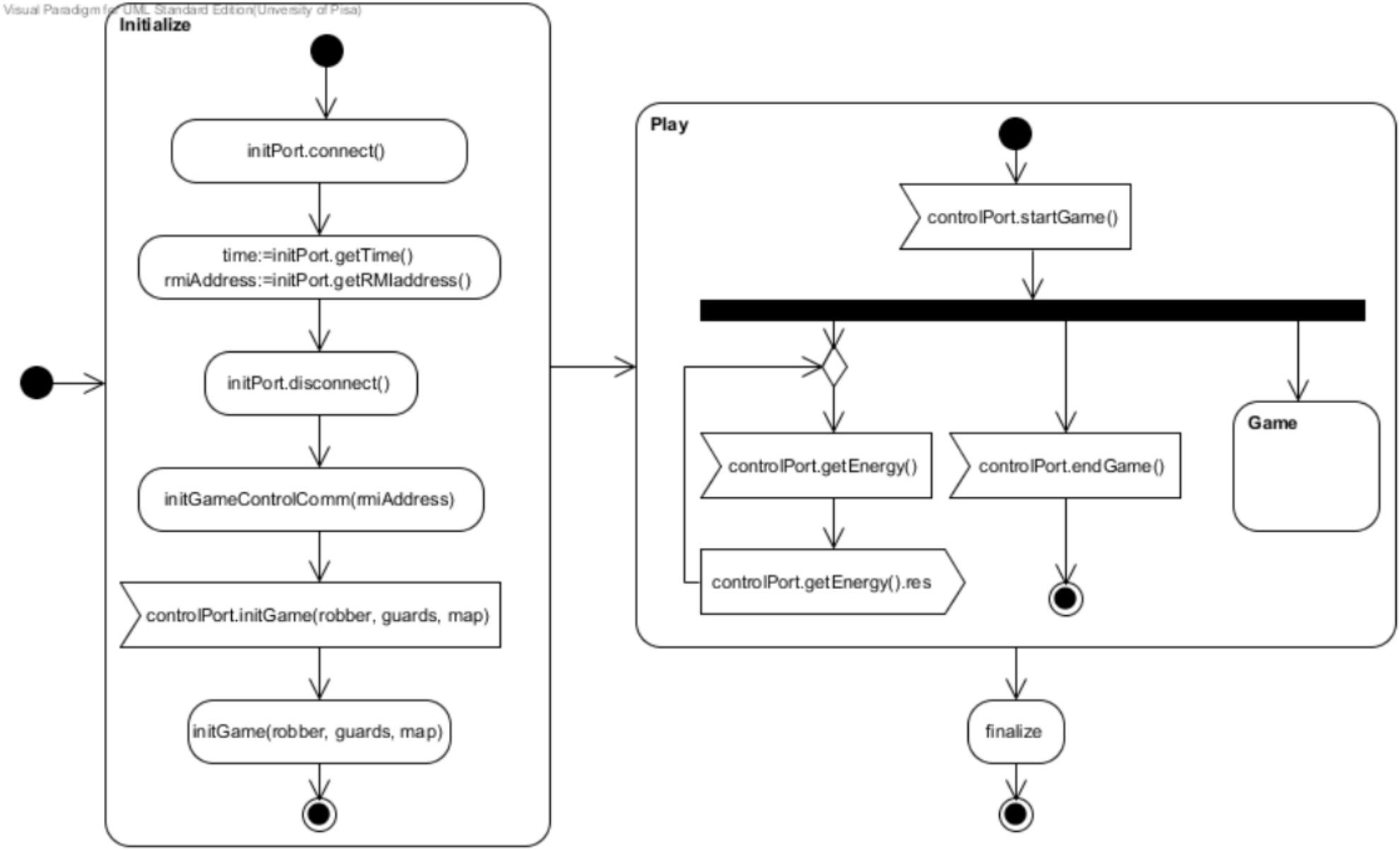
- Progetto del corso di Laboratorio di Reti di alcuni anni fa.
- Documenti alla pagina didawiki del corso:
  - Dodumento consegnato dalla Prof. Ricci
  - Documento dei requisiti
  - Architettura
- Scopo del progetto e quello di realizzare Cops And Robbers (Guardie eLadri) un semplice gioco multiplayer.
- In Cops And Robbers il mondo virtuale è popolato da un certo numero Ladri e da un insieme di Guardie.

# Vista C&C

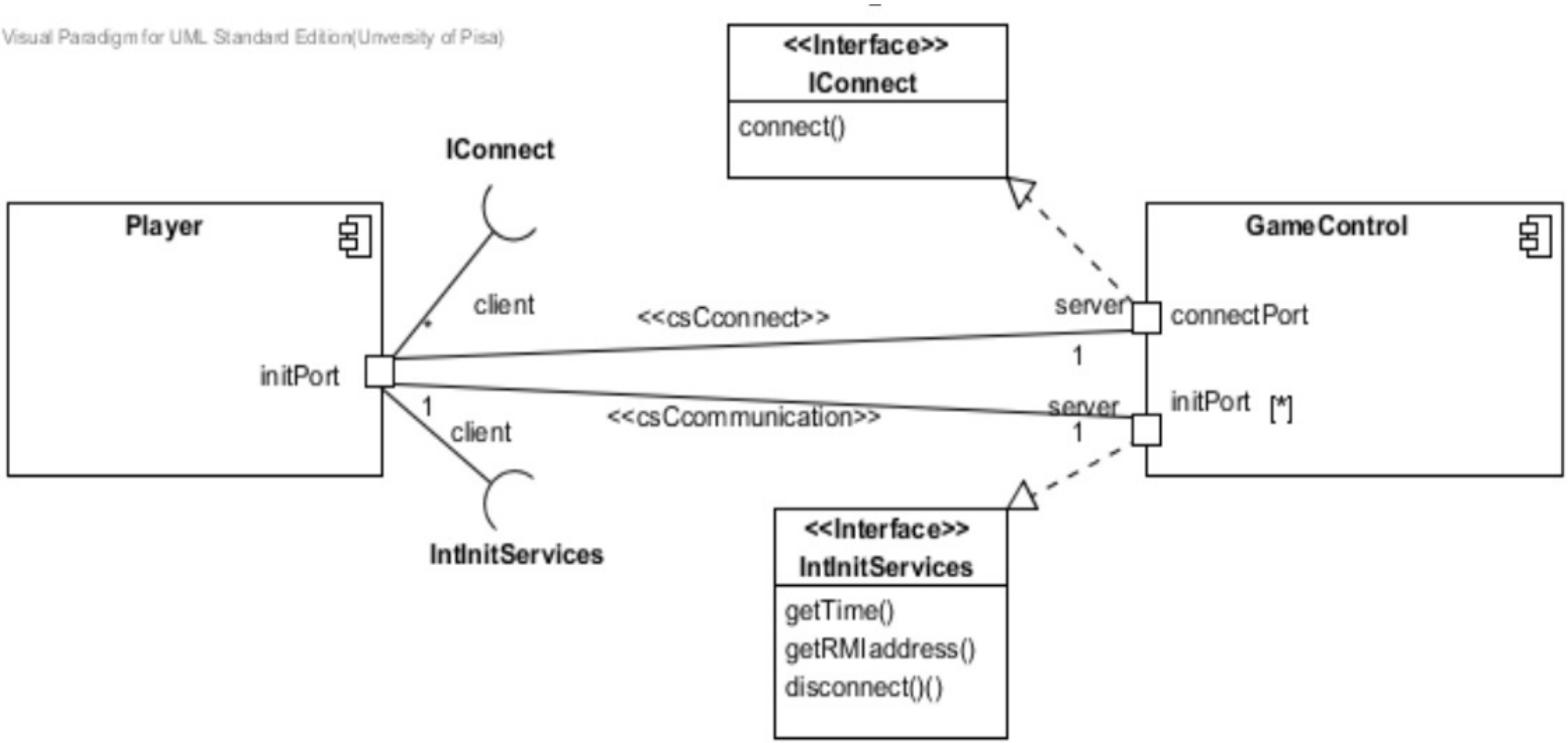



Questa vista mostra le due diverse componenti di CopsAndRobbers e i diversi connettori tra di esse.
Il connettore <<clientserver>> è usato nella fase iniziale da ogni Player, per ottenere i dati iniziali per sincronizzare gli orologi e inizializzare il connettore <<masterSlave>>, con cui GameControl gestisce il gioco.

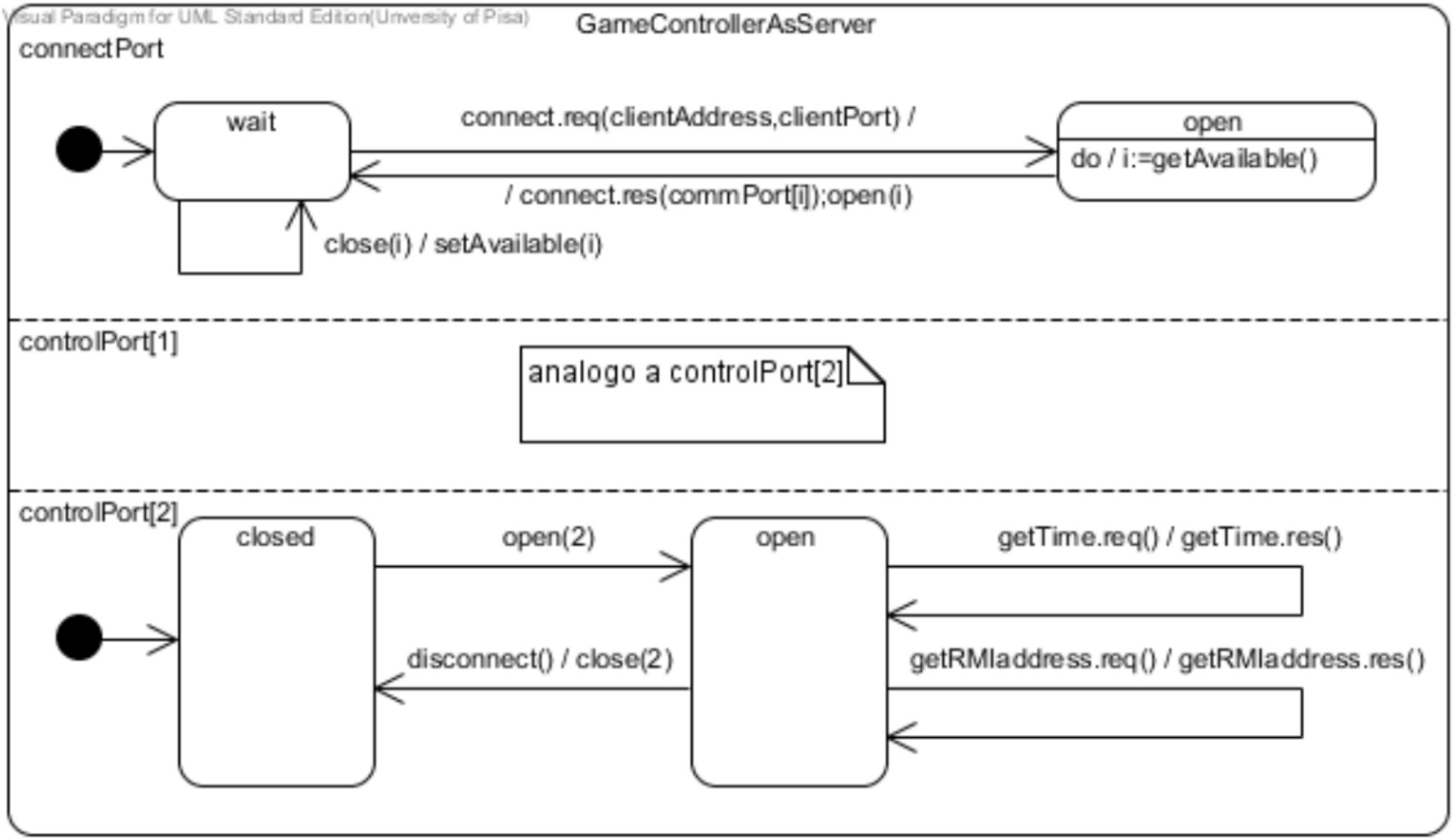
# PlayerActivity



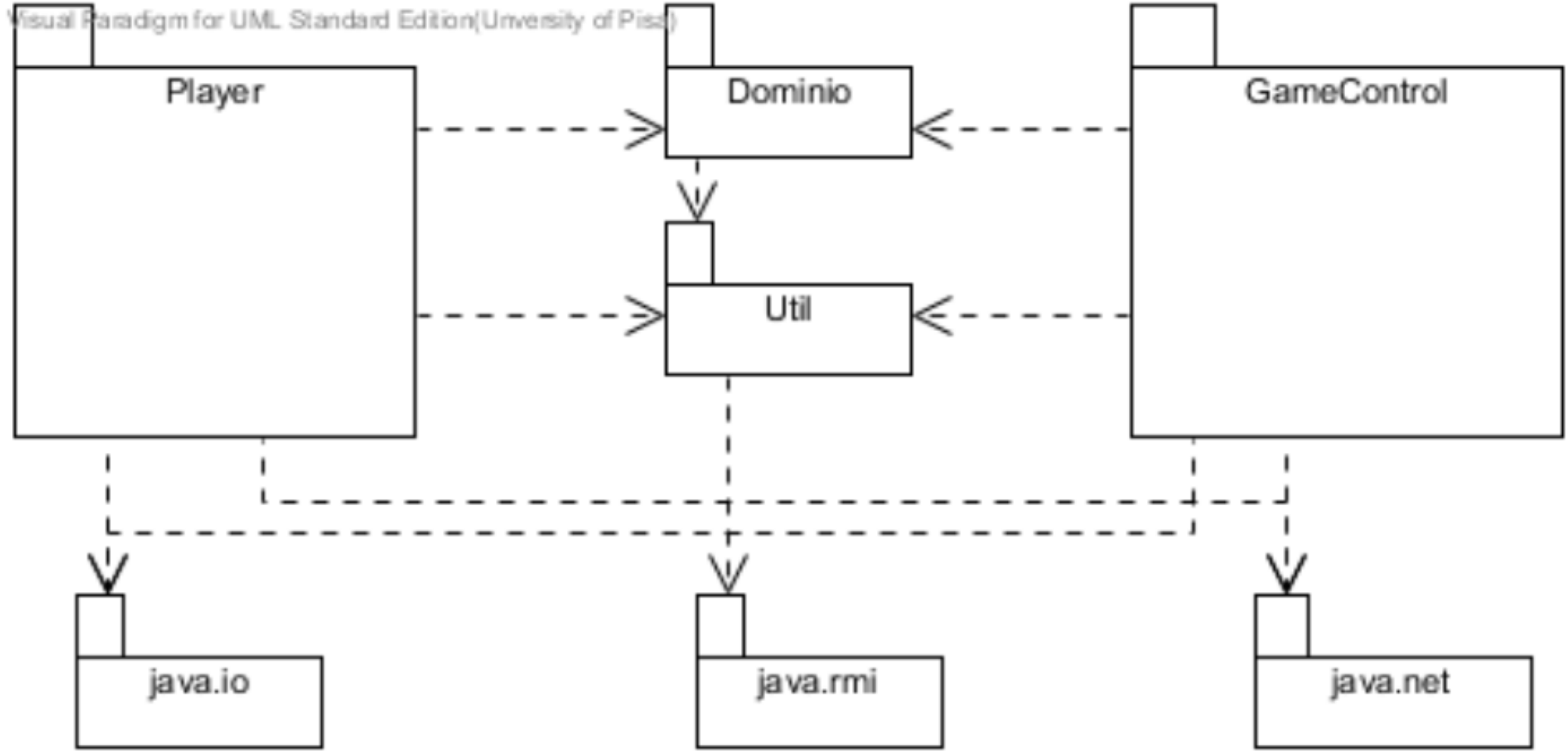
# VistaC&C 2


Visual Paradigm for UML Standard Edition(University of Pisa)
<<Interface>>
IConnect
connect()
IConnect
Player
client
<<csCconnect>>
server
connectPort
1
initPort
1
client
<<csCcommunication>>
server
1
initPort [*]
GameControl
IntInitServices
<<Interface>>
IntInitServices
getTime()
getRMIaddress()
disconnect()()

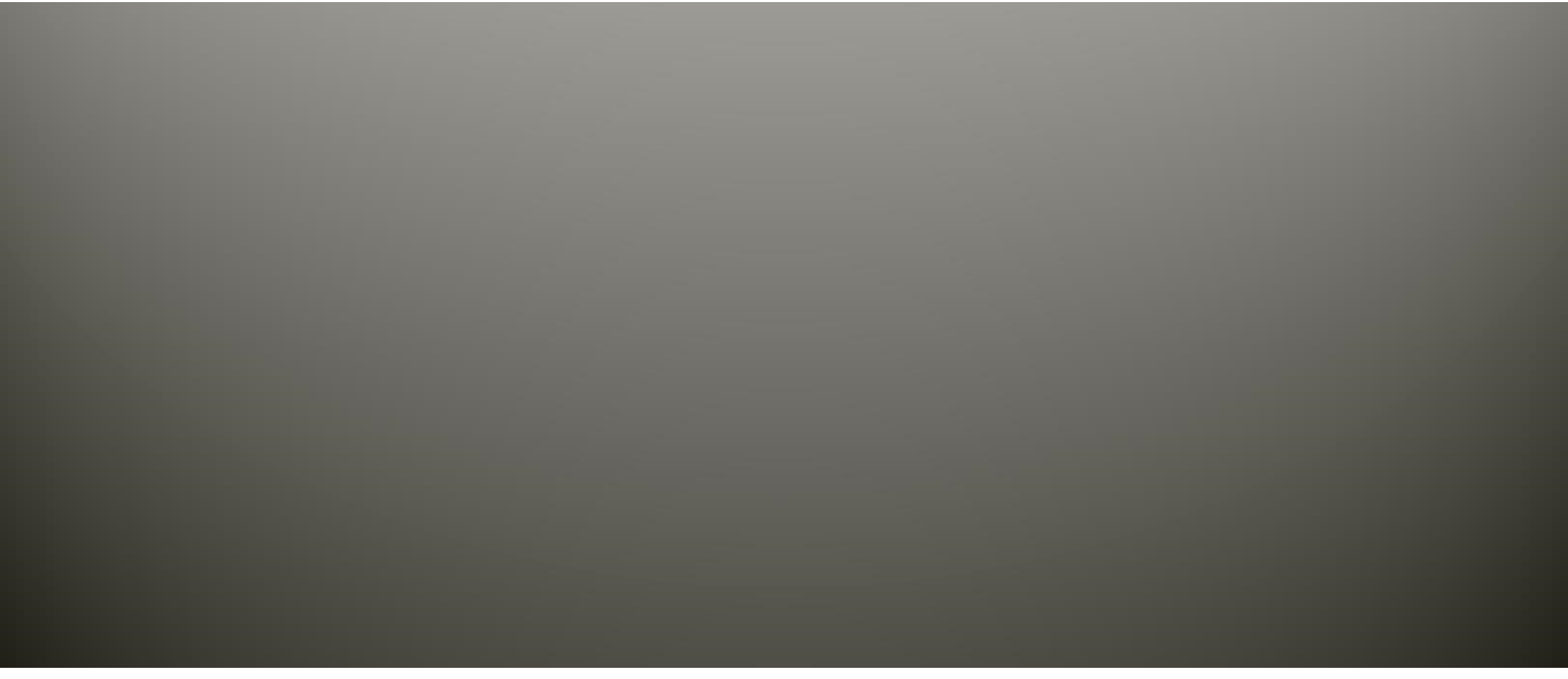
# GameControl: stati



# VistaStrutturale



# Grande distribuzione

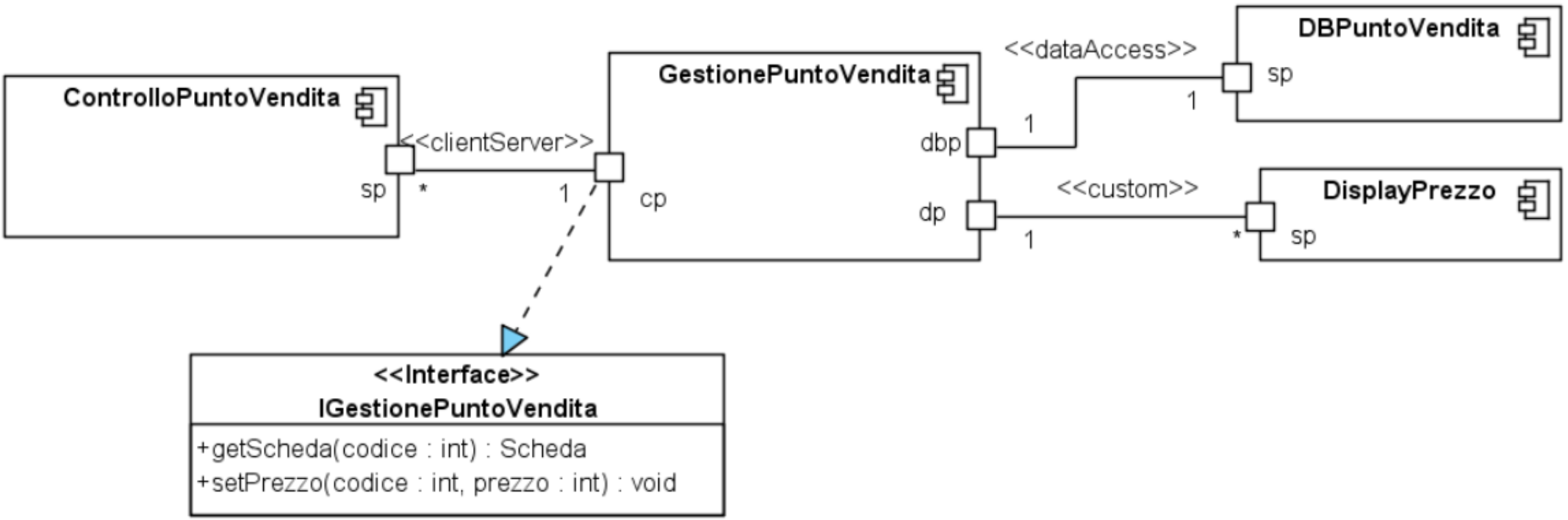


# Aggiornamento dei prezzi (C&C)

Fornire la vista C&C considerando la seguente descrizione per le componenti, e mostrare l'interfaccia che GestionePuntoVendita deve offrire per realizzare il caso d'uso.

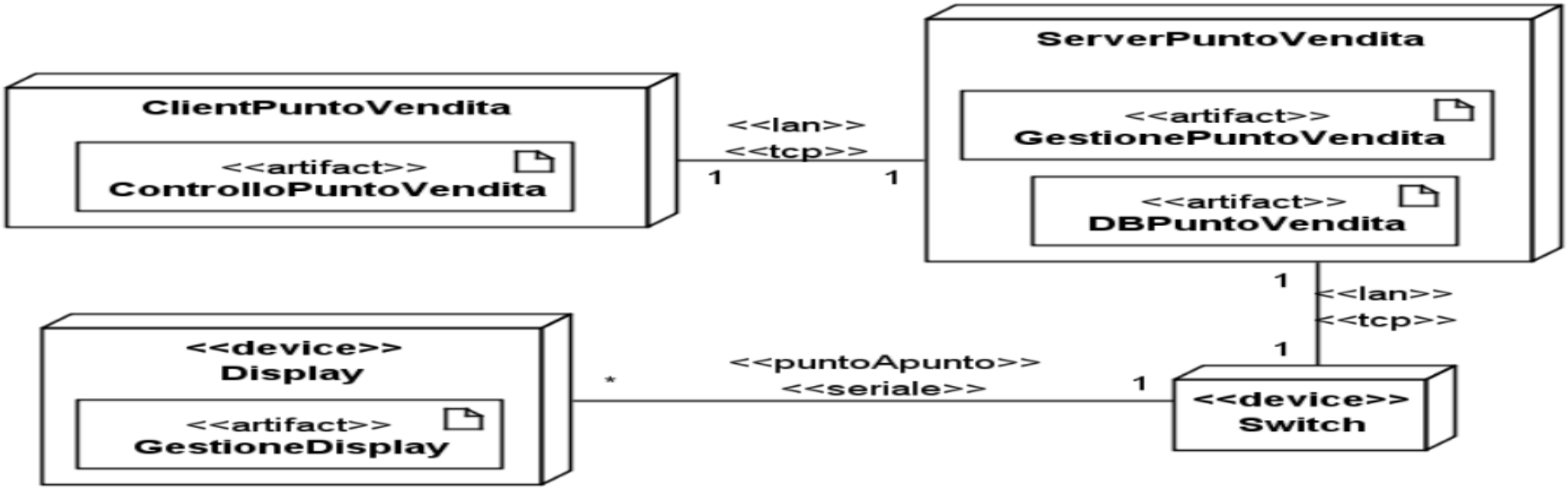
| **Componente** | **Responsabilità** |
|---|---|
| DBPuntoVendita | Memorizza i dati sui prodotti in vendita. |
| GestionePuntoVendita | Gestisce l'aggiornamento dei prezzi del punto vendita. |
| ControlloPuntoVendita | Permette al ResponsabilePuntoVendita di accedere al sistema per aggiornare i prezzi del punto vendita. |
| DisplayPrezzo | Gestisce la visualizzazione dei prezzi sugli scaffali. |

# Aggiornamento dei prezzi (C&C)



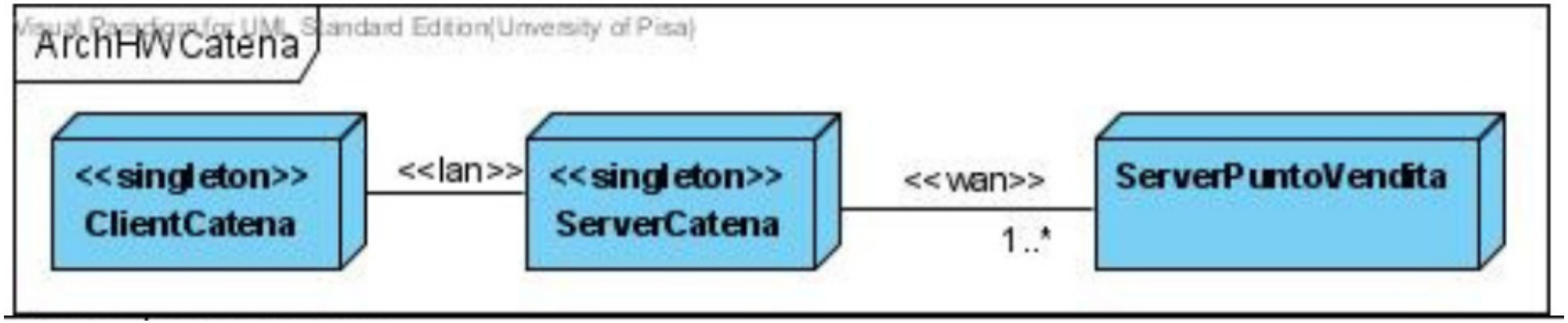
# Aggiornamento dei prezzi (Dislocazione)

Fornire la vista di dislocazione. Si tenga conto che, oltre ai nodi hardware descritti nel testo, è previsto uno switch per inoltrare i messaggi ai Display, connesso via rete locale al Server e via cavo ai singoli Display. Il software dello switch è reso disponibile dal produttore, e non interessa in questa vista.



# Vista fisica (1/2)

Dare un diagramma di deployment parziale, che consideri solo la struttura hardware di TradingSystem (vista fisica). Si assuma l'esistenza di un router che funge da gateway per il sotto-sistema barriera.



# Vista fisica (2/2)