Esercizi su architetture

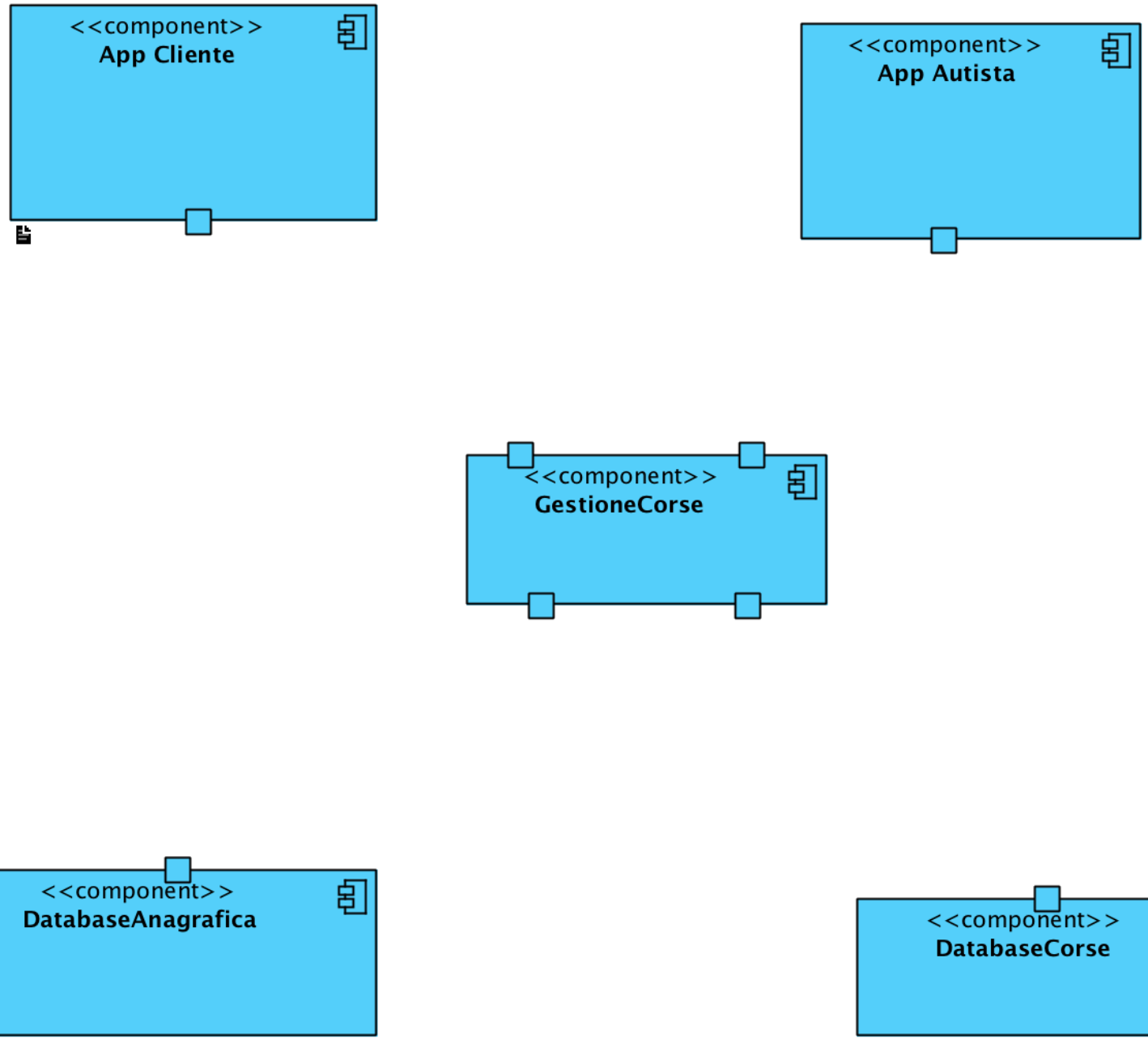
Roberta Gori, Laura Semini
Ingegneria del Software
Dipartimento di Informatica
Università di Pisa

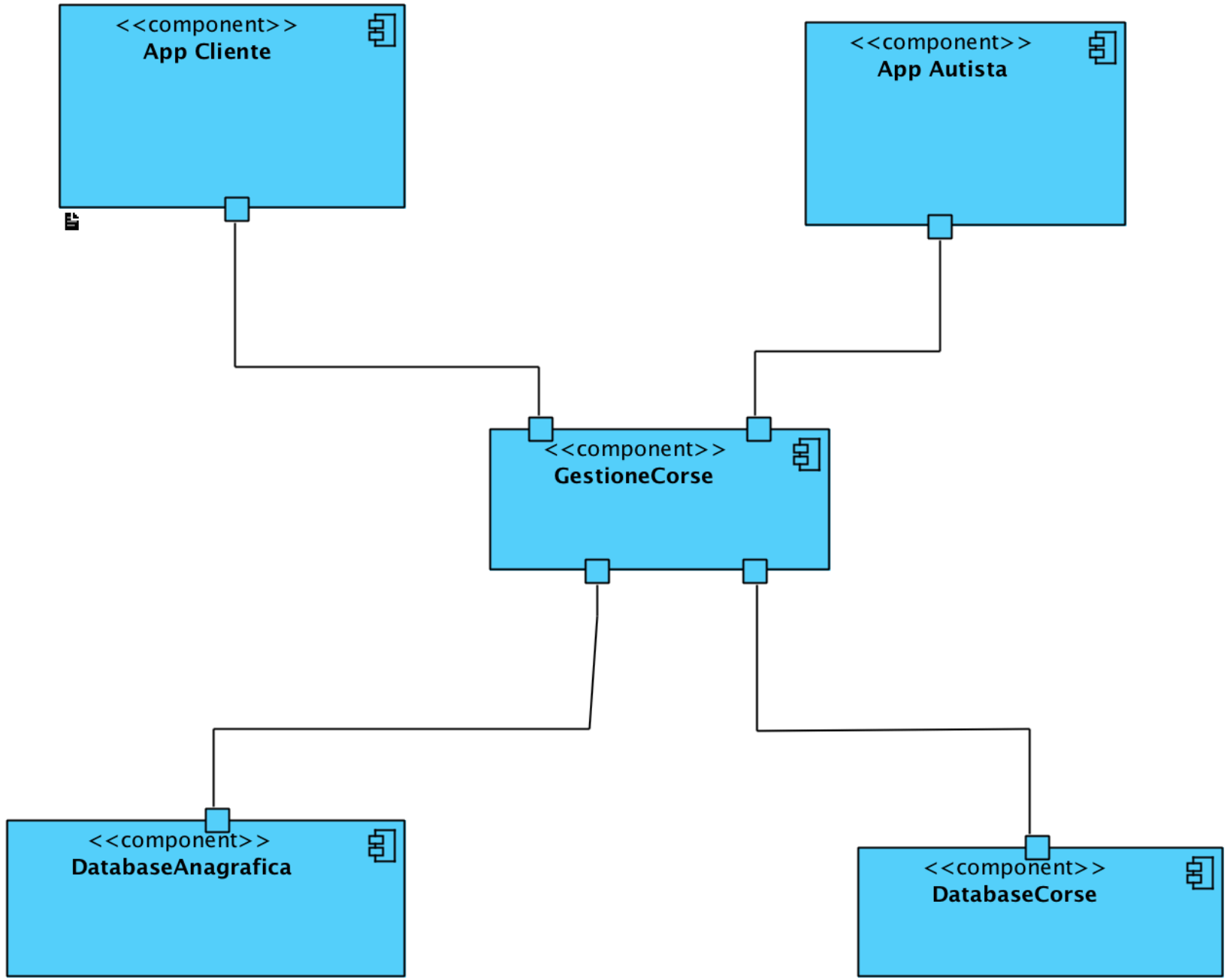
Ex 1: REBU

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

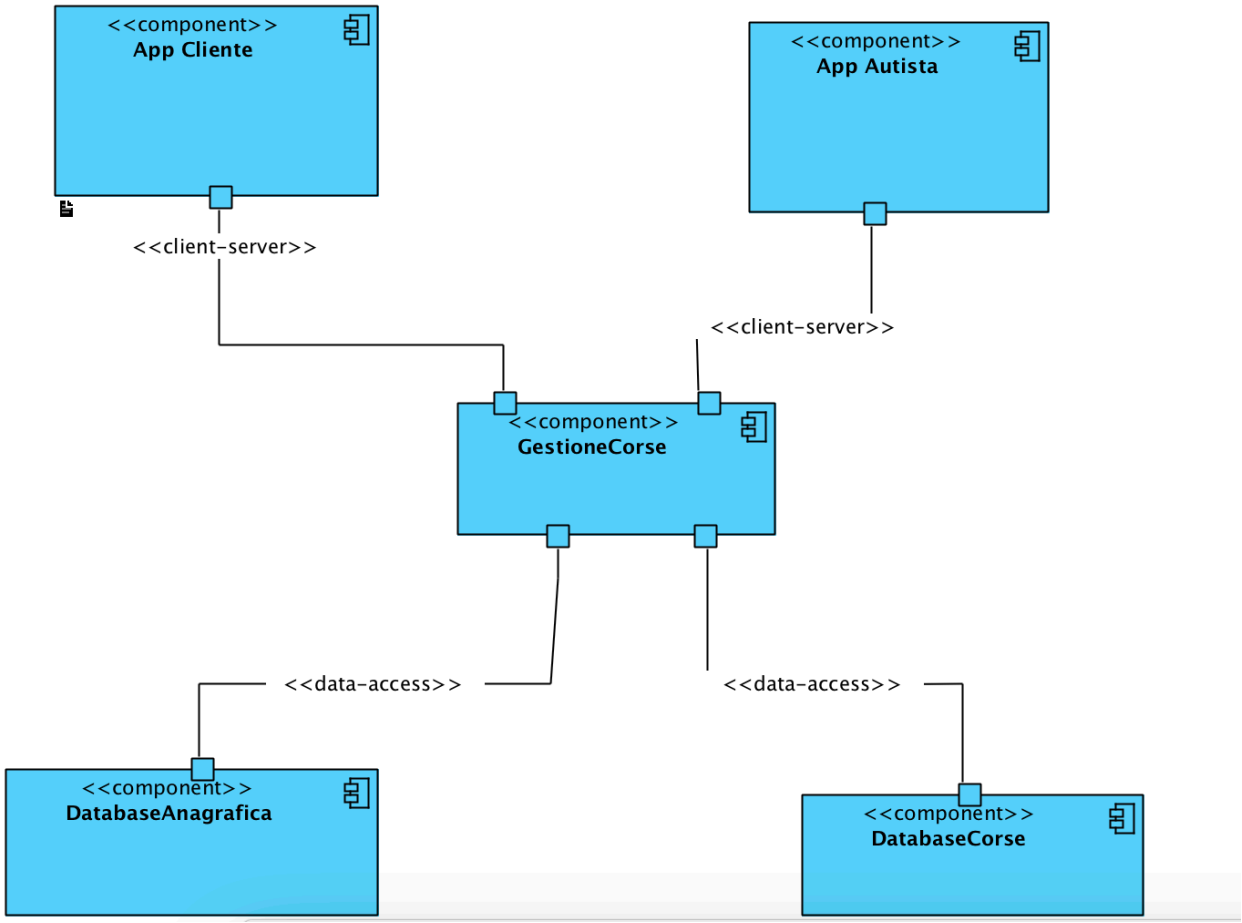
- app_cliente (che implementa l'app usata dal cliente)
- sistema_prenotazione (che implementa la business logic del sistema)
- db_anagrafica (che raccoglie dati anagrafici su autisti e clienti)
- db_corse (che raccoglie dati su prenotazioni, corse assegnate, corse effettuate)
- app_autista (che implementa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

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

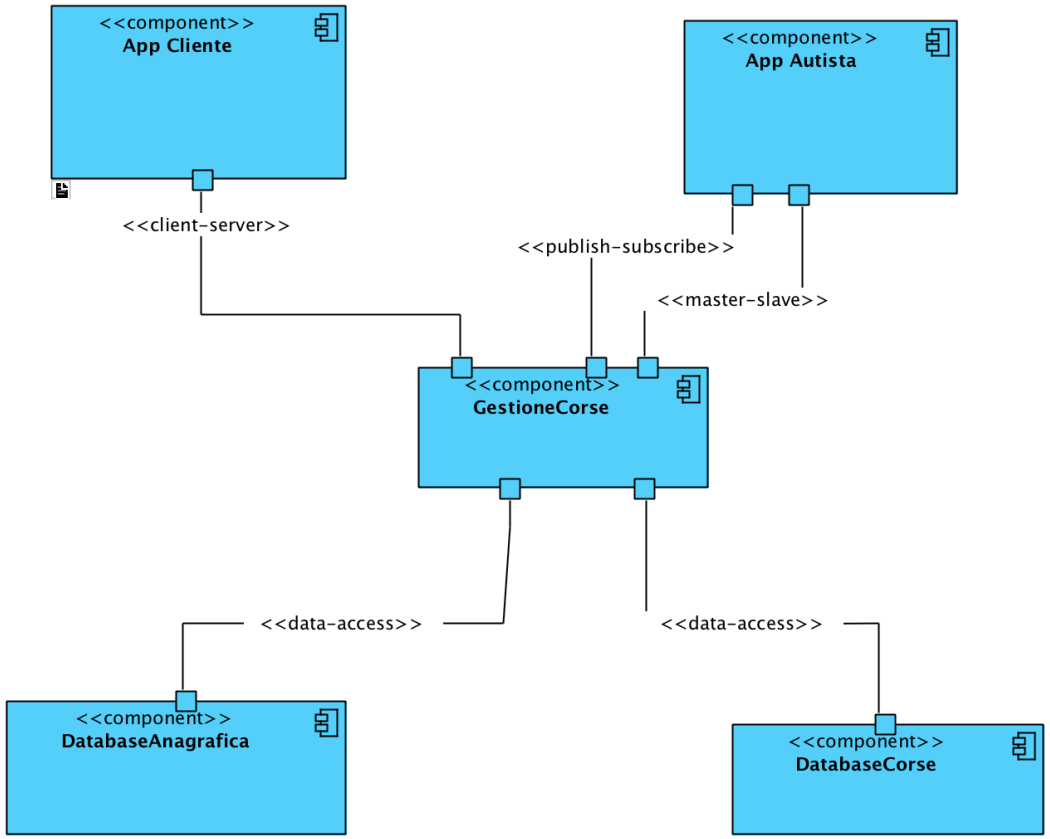

<<component>>
App Cliente
<<component>>
App Autista
<<component>>
GestioneCorse
<<component>>
DatabaseAnagrafica
<<component>>
DatabaseCorse


<<component>>
App Cliente
<<component>>
App Autista
<<component>>
GestioneCorse
<<component>>
DatabaseAnagrafica
<<component>>
DatabaseCorse

Prima soluzione



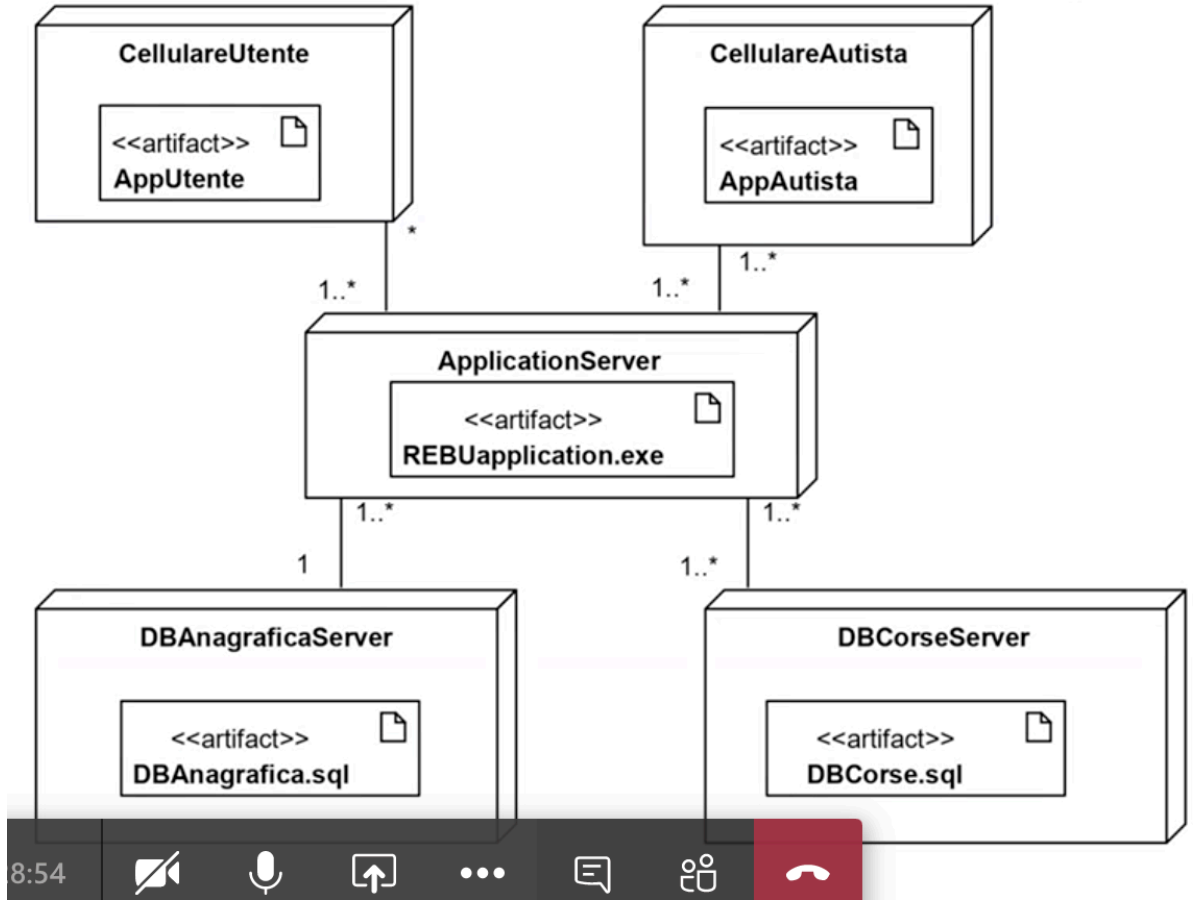
Altra soluzione



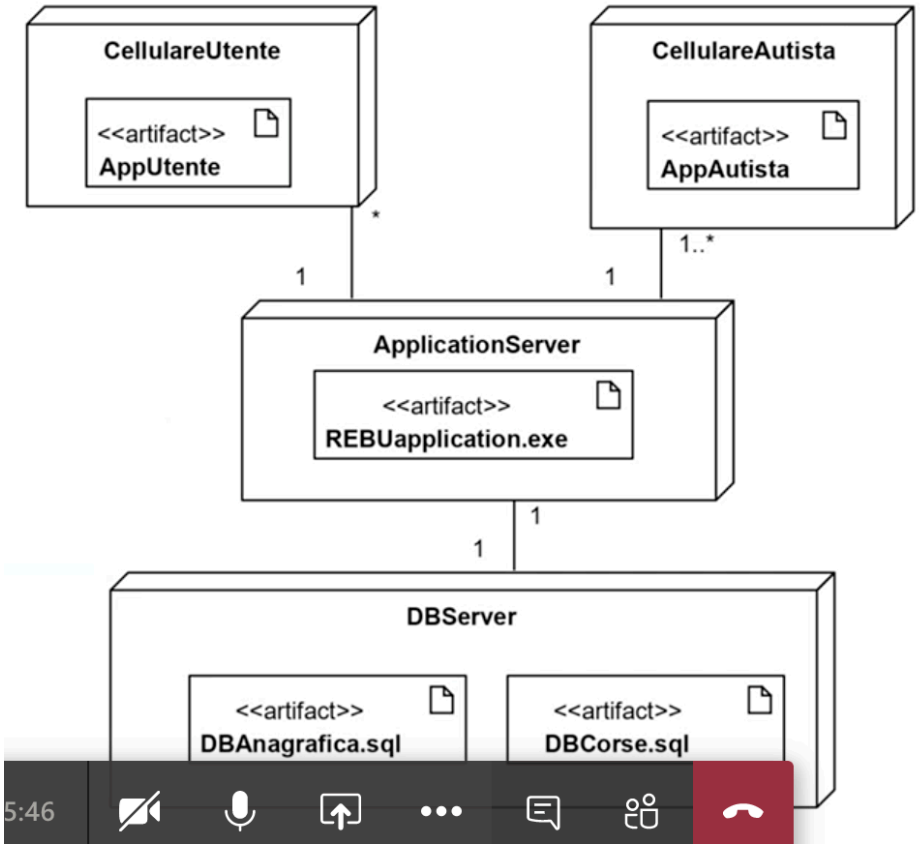
Ex2: REBU

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).

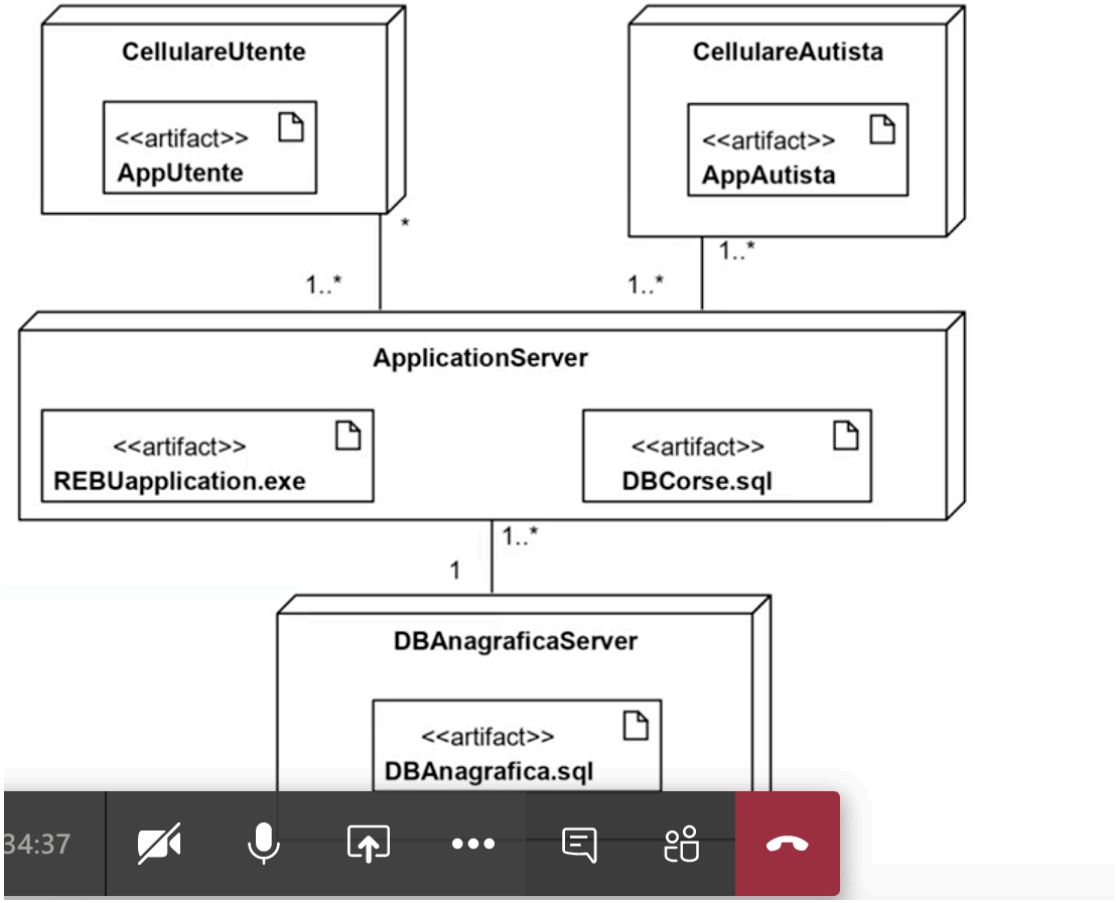
Prima soluzione



Seconda soluzione



Terza soluzione



REBU: 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.

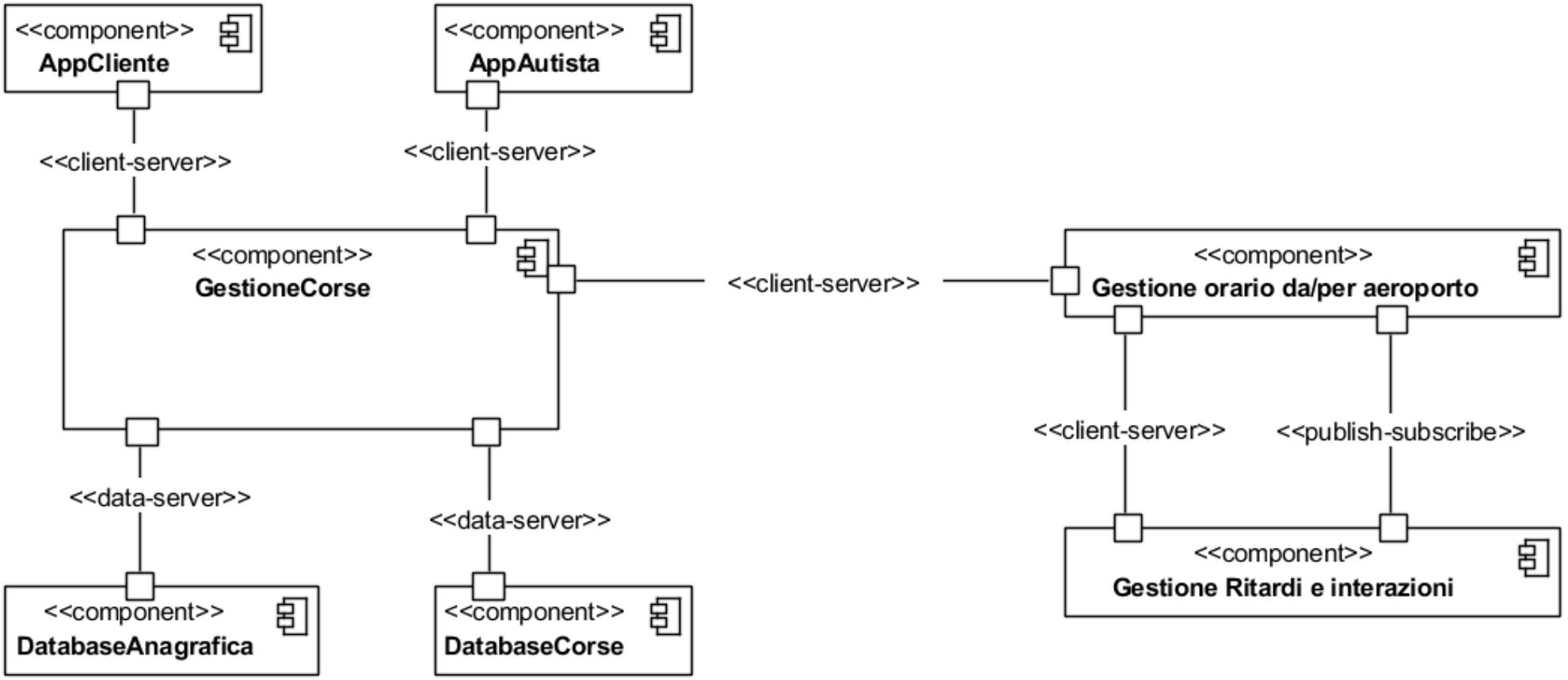
Nelle ultima due ore prima dell'inizio della corsa, REBU monitora eventuali ritardi dei voli. In caso di ritardo superiore ai 15 minuti, ricalcola l'orario di partenza della corsa, cerca un nuovo autista per la corsa ritardata (allo stesso prezzo della corsa già pattuita con l'utente), e cancella la precedente. Quindi informa l'utente del nuovo orario.

Ex3: REBU (Estensione Aeroporti)

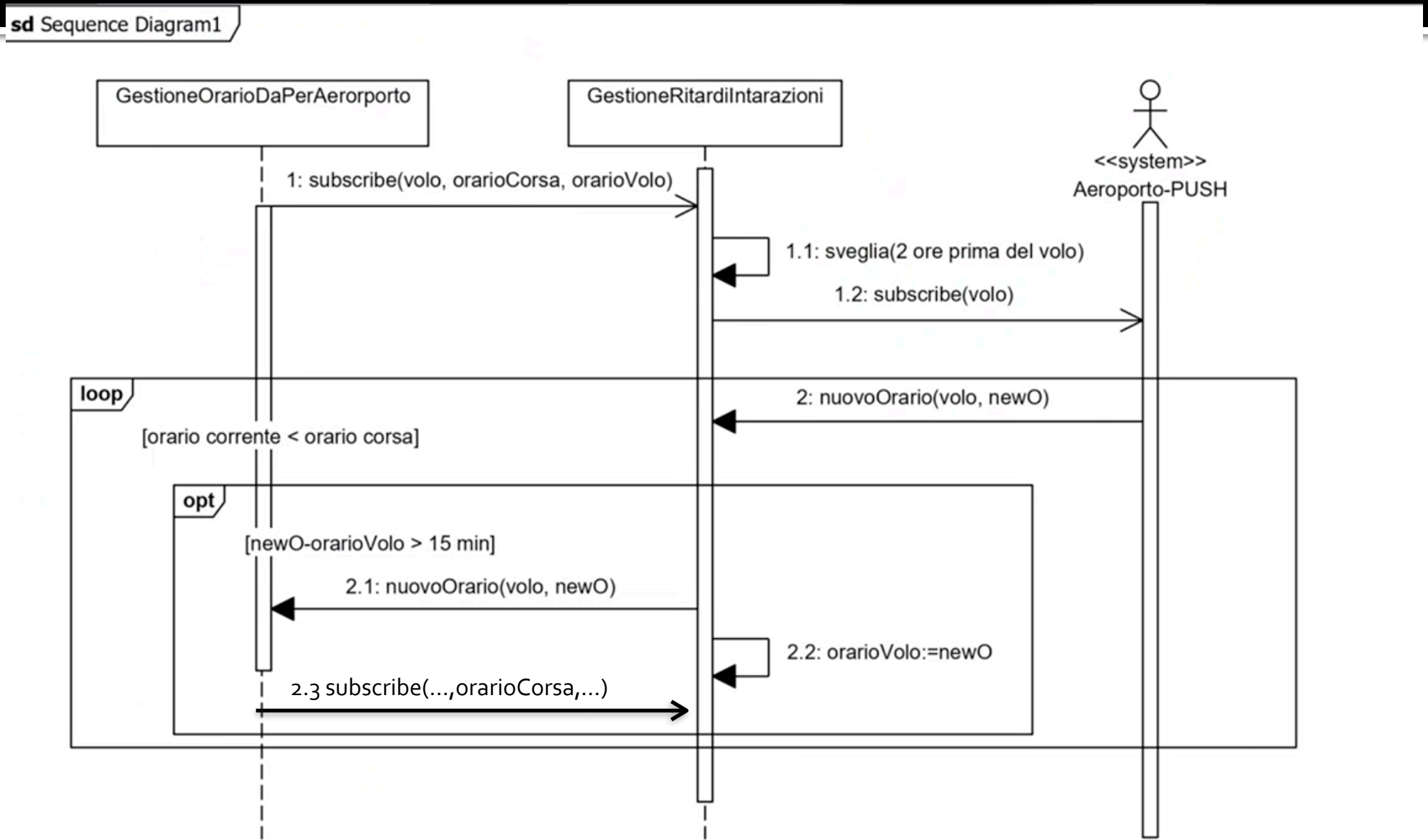
La progettazione architetturale ha portato all'individuazione di una componente GestioneRitardi che si occupa di monitorare eventuali ritardi dei voli e, se necessario, modificare la corsa. I servizi offerti dai vari aeroporti non sono uniformi: alcuni offrono la possibilità di abbonarsi alle informazioni su un volo e informano REBU di eventuali ritardi (modalità PUSH), altri offrono solo un servizio di informazioni domanda/risposta (modalità PULL). Nella modalita' PULL si ipotizzi che il sistema REBU si aggiorni sui voli ogni 20 minuti.

Descrivere, con un diagramma di sequenza, le interazioni tra GestioneRitardi e l'aeroporto, nei due casi.

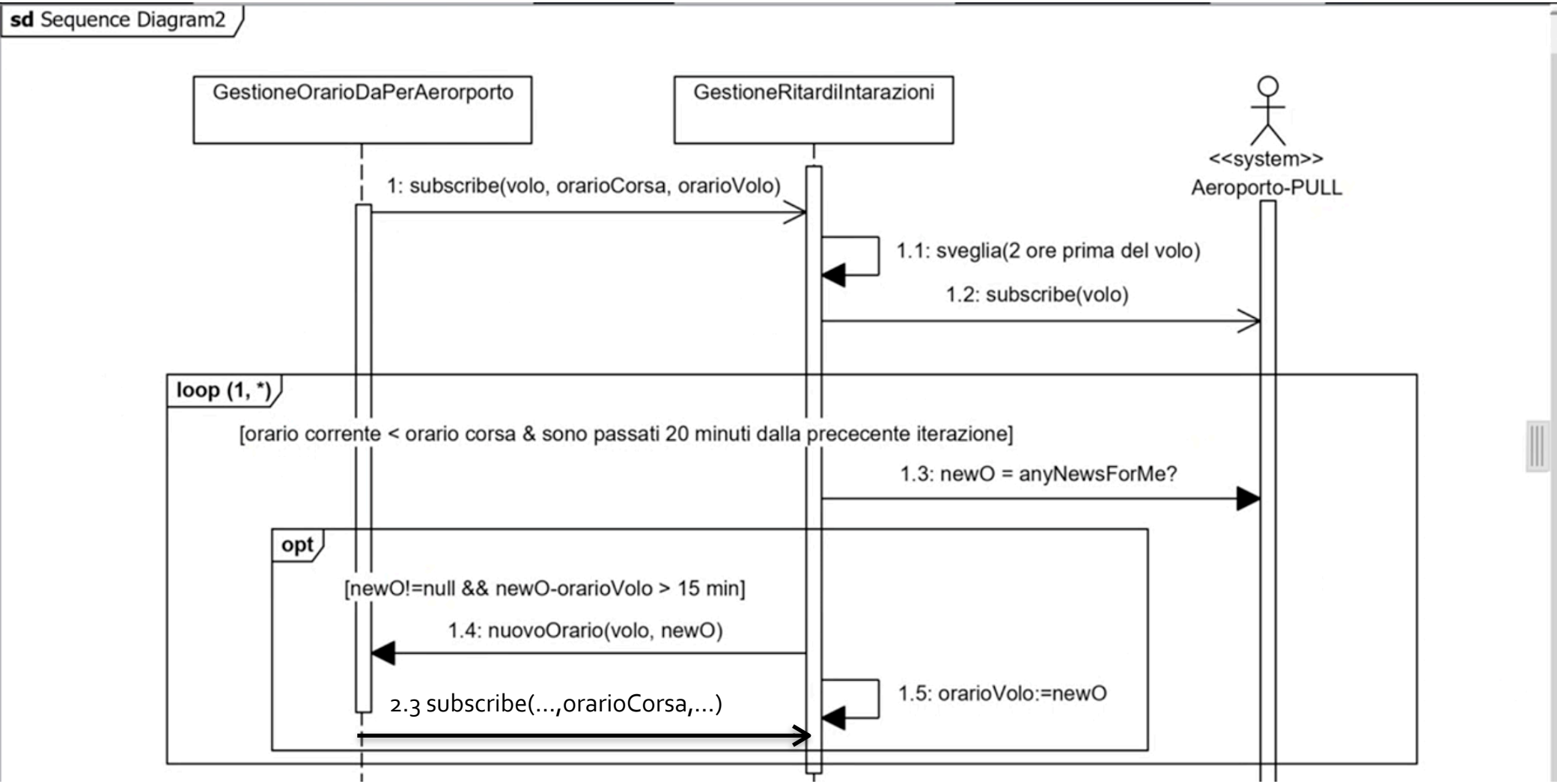
Es 3


<<component>>
AppCliente
<<component>>
AppAutista
<<client-server>>
<<client-server>>
<<component>>
GestioneCorse
<<client-server>>
<<component>>
Gestione orario da/per aeroporto
<<client-server>>
<<publish-subscribe>>
<<data-server>>
<<data-server>>
<<component>>
Gestione Ritardi e interazioni
<<component>>
DatabaseAnagrafica
<<component>>
DatabaseCorse

Es 3 Push


sd Sequence Diagram1
GestioneOrarioDaPerAerorporto
GestioneRitardiIntarazioni
<<system>>
Aeroporto-PUSH
1: subscribe(volo, orarioCorsa, orarioVolo)
1.1: sveglia(2 ore prima del volo)
1.2: subscribe(volo)
loop
2: nuovoOrario(volo, newO)
[orario corrente < orario corsa]
opt
[newO-orarioVolo > 15 min]
2.1: nuovoOrario(volo, newO)
2.2: orarioVolo:=newO
2.3 subscribe(...,orarioCorsa,...)

Es 3 Pull


sd Sequence Diagram2
GestioneOrarioDaPerAerorporto
GestioneRitardiIntarazioni
<<system>>
Aeroporto-PULL
1: subscribe(volo, orarioCorsa, orarioVolo)
1.1: sveglia(2 ore prima del volo)
1.2: subscribe(volo)
loop (1, *)
[orario corrente < orario corsa & sono passati 20 minuti dalla prececente iterazione]
1.3: newO = anyNewsForMe?
opt
[newO!=null && newO-orarioVolo > 15 min]
1.4: nuovoOrario(volo, newO)
1.5: orarioVolo:=newO
2.3 subscribe(...,orarioCorsa,...)

Ex 4 e 5

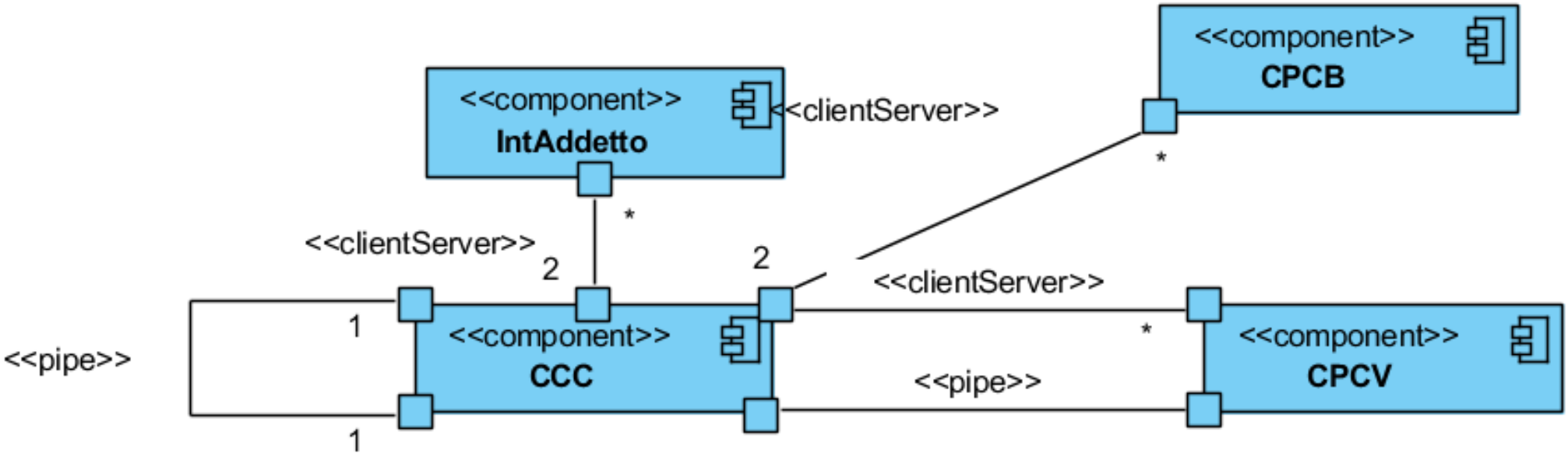
Si consideri il sottosistema che controlla il movimento e l'apertura-chiusura delle porte del PisaMover, che abbreviamo in SSC. Per garantirne la fault-tolerance, è stata adottata una politica di ridondanza: i server fisici sono duplicati e così le componenti software. In caso di fallimenti di un server, la componente sul server replica prende il controllo.
In particolare, sono state individuate le seguenti componenti e le seguenti risorse hardware su cui dislocare gli artefatti che manifestano le componenti date.

Componente	Specifica	HW
IntAddetto	Interfaccia utente, per impostare la modalità operativa e inviare altri comandi alla CCC	PC, presso la sala controllo
Componente centrale di controllo (CCC)	Decide apertura/chiusura porte e movimento/arresto. Comunica le decisioni alle componenti periferiche CPCB e CPCV	Server, presso la sala controllo
Componente periferica di controllo porte banchina (CPCB)	Riceve ordini dalla CCC, invia agli attuatori il segnale apertura/chiusura porte	Microcontrollore embedded, presso ogni stazione
Componente periferica di controllo vagone (CPCV)	Riceve ordini dalla CCC, invia agli attuatori il segnale apertura/chiusura porte, trasmette alla CCC le richieste di passare in modalità Emergenza 2	Microcontrollore embedded, su ogni vagone

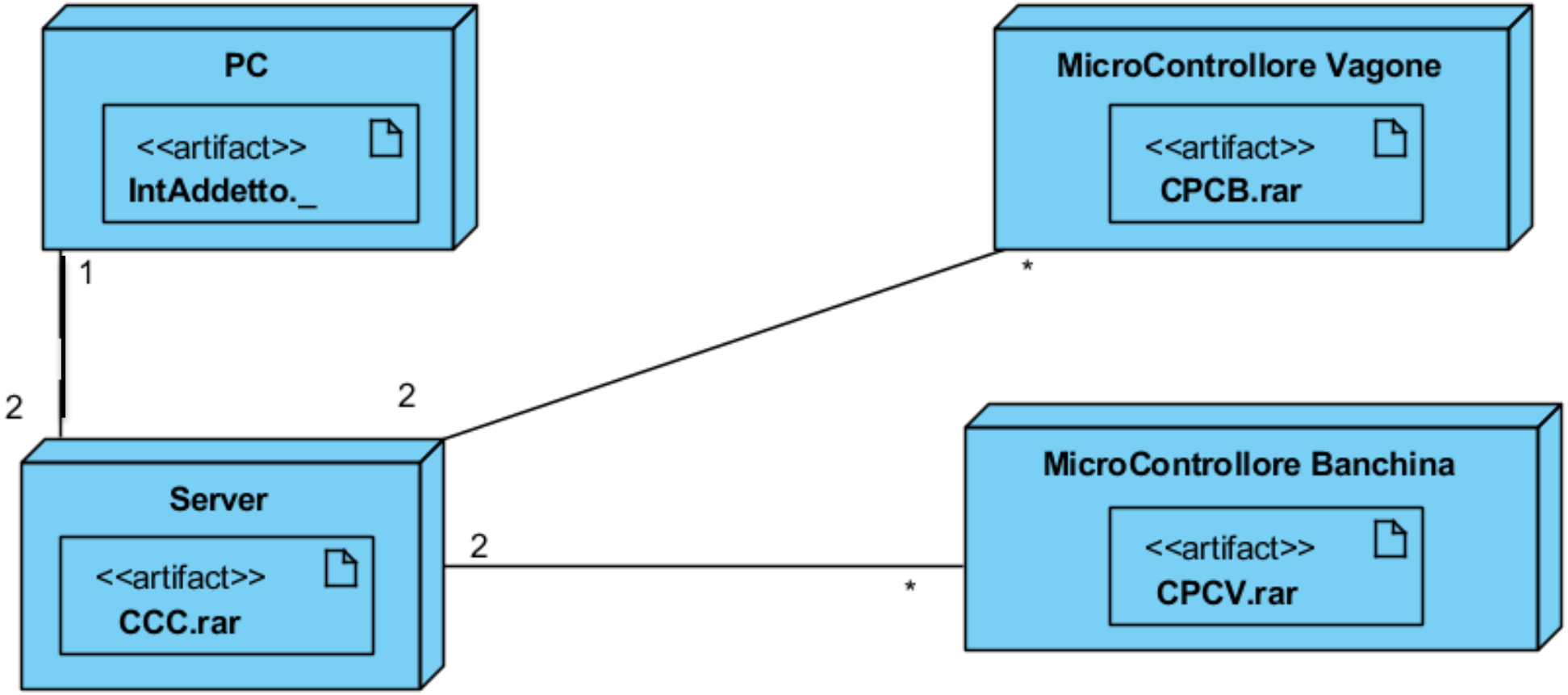
Ex5: Dare la vista Componenti & Connettori del SSC.

Ex6: Dare la vista di dislocazione del SSC.

Ex 4


<<component>>
IntAddetto
<<clientServer>>
<<component>>
CPCB
*
<<clientServer>>
2
*
2
<<clientServer>>
1
<<component>>
CCC
*
<<component>>
CPCV
<<pipe>>
<<pipe>>
1

Ex 5


PC
<<artifact>>
IntAddetto._
MicroControllore Vagone
<<artifact>>
CPCB.rar
1
*
2
2
Server
<<artifact>>
CCC.rar
2
*
MicroControllore Banchina
<<artifact>>
CPCV.rar

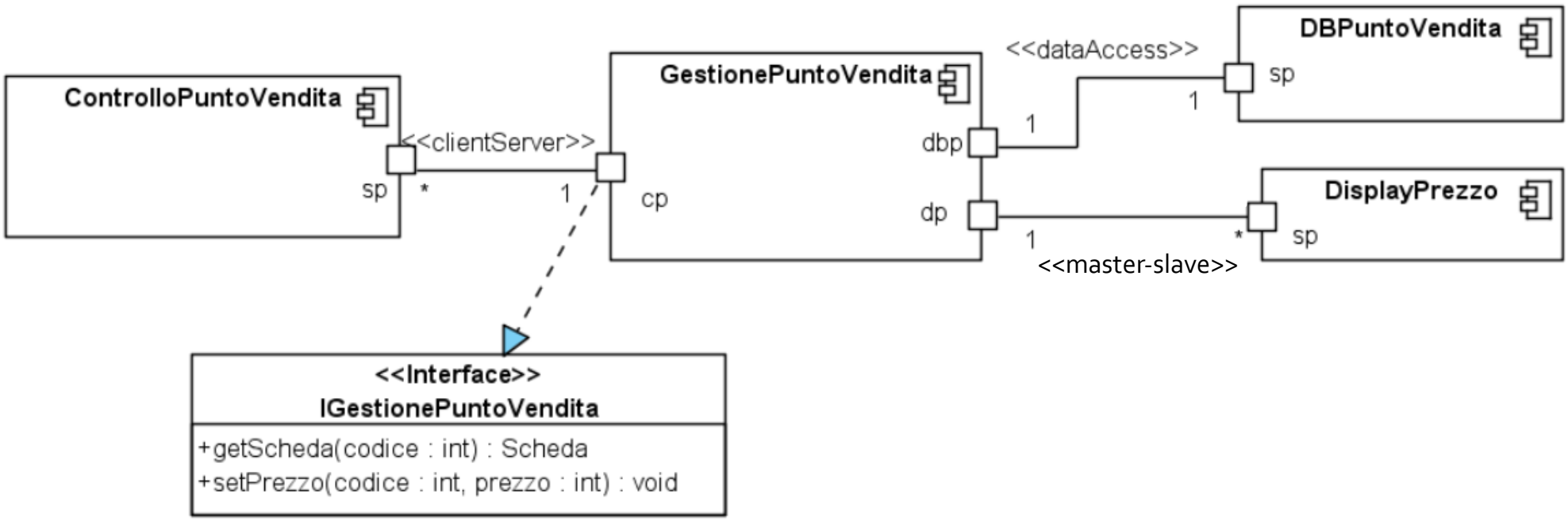
Grande distribuzione

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)



Cops & Robbers

- 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.

Per l'amministrazione del gioco, CopsAndRobbersAdmin:
2.1.deve ricevere le richieste di connessione al gioco da parte dei giocatori;
2.2.quando ha ricevuto esattamente k richieste, deve
    2.2.1.inviare ai giocatori la posizione degli oggetti all'interno della mappa, e
    2.2.2.notificare a ogni giocatore che può iniziare il gioco;
2.3.deve inviare ai giocatori che ne fanno richiesta la lista dei giocatori che partecipano al gioco, con la loro energia;
2.4.deve notificare la vincita/perdita ai giocatori.

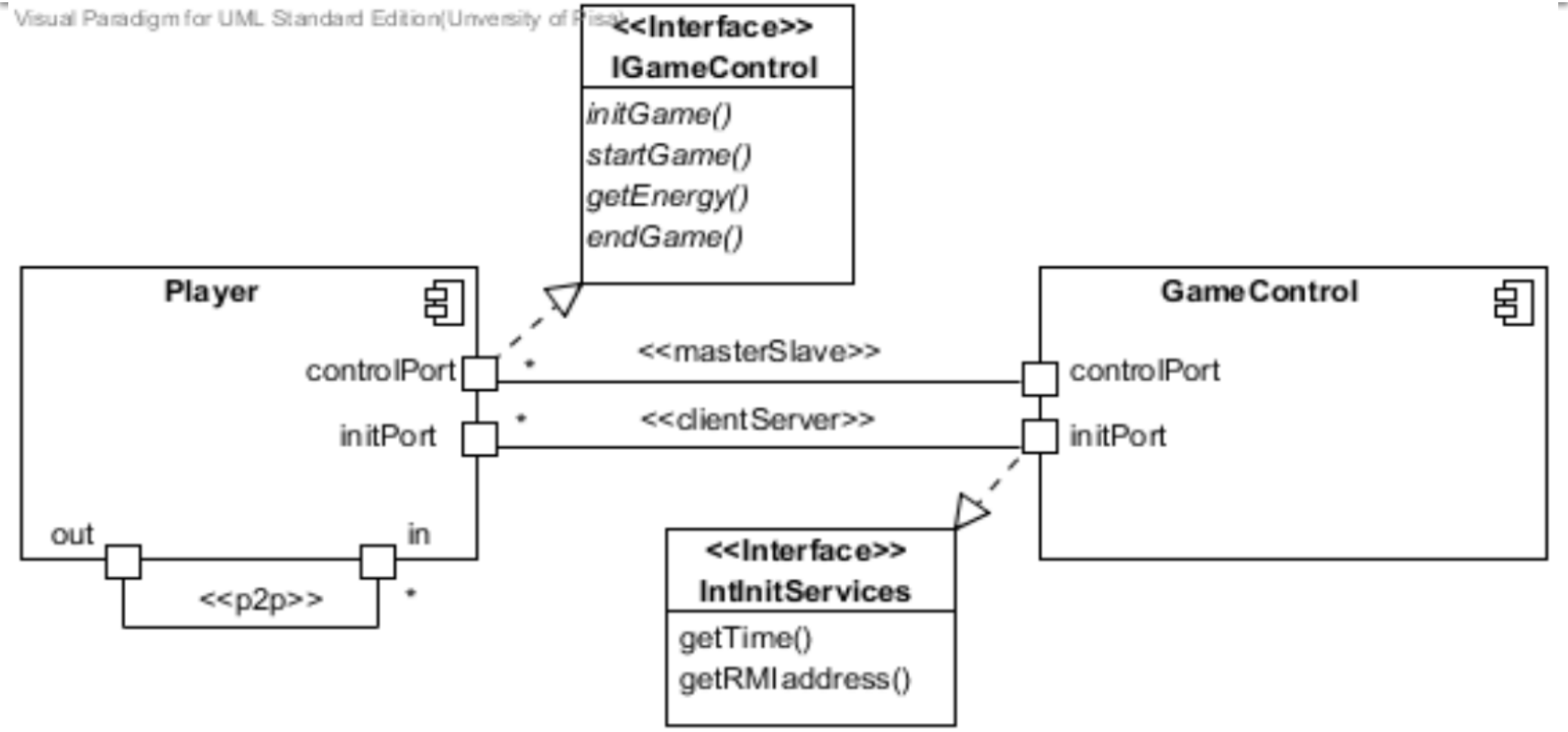
Per permettere lo svolgimento del gioco, PeerCopsAndRobbers:
3.1.deve dar corso alle richieste del giocatore, e notificargliene l'esito;
3.2.deve muovere le guardie assegnategli.

Esercizio

Fornire:

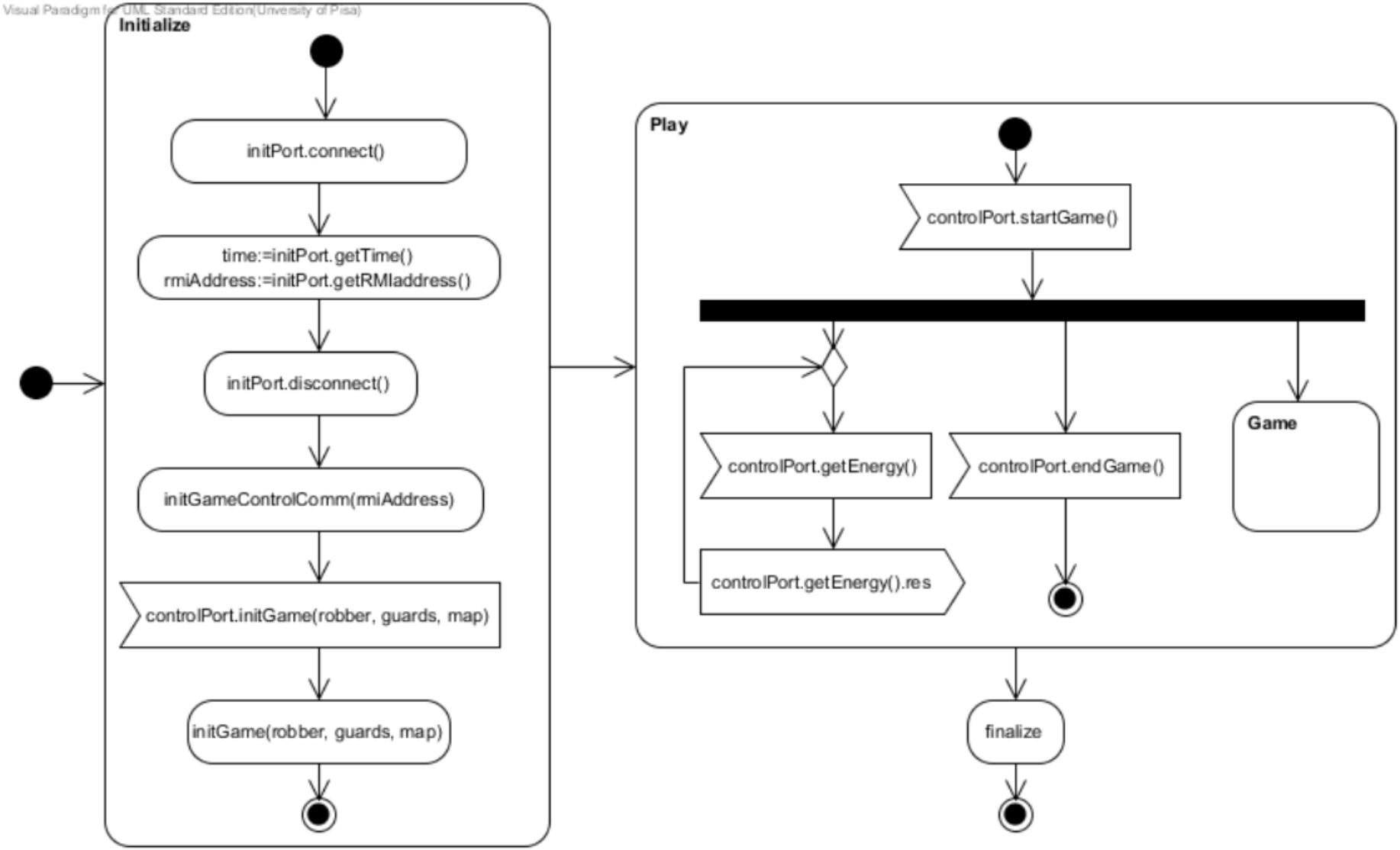
1. una vista C&C
2. un diagramma delle attivita' del giocatore
3. un diagramma degli stati del controllore
4. una vista strutturale dell'architettura

Vista C&C

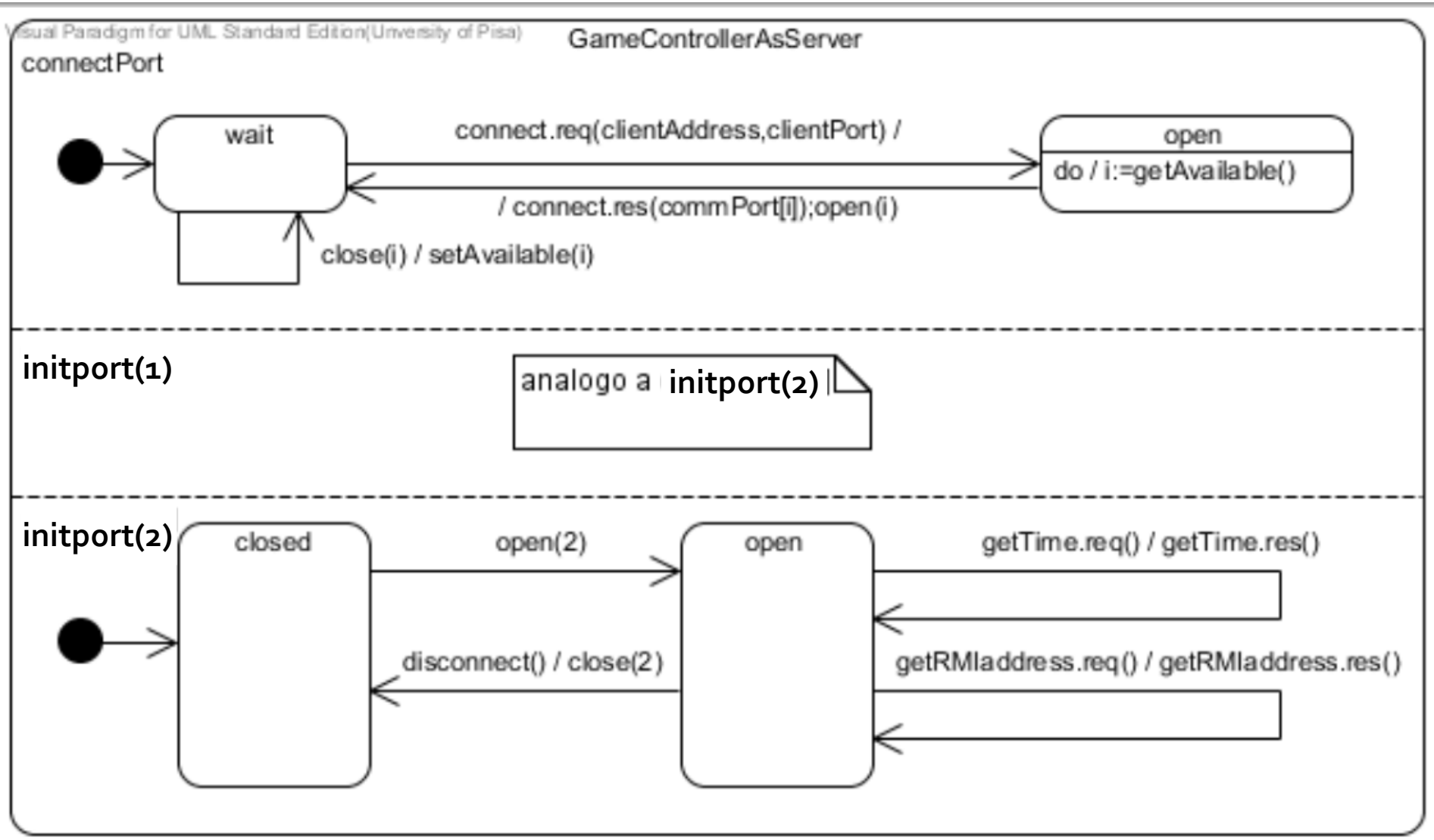


Questa vista mostra le due diverse componenti di CopsAndRobbers e i diversi connettori tra di esse.
Il connettore <<clientserver>> è usato nella fase iniziale da ogni Player, per ottenere i dati iniziali per sincronizzare gli orologi e inizializzare il connettore <<masterSlave>>, con cui GameControl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

PlayerActivity



Game Control: initport



VistaStrutturale