UNIVERSITÀ DI PISA

Dipartimento di Informatica
Corso di Laurea in Informatica

Un giorno al Museo

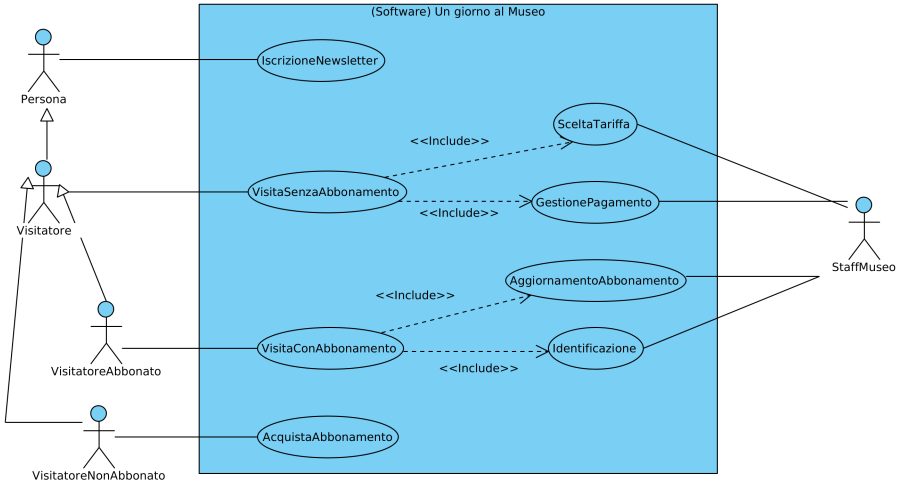
Ingegneria del Software a.a. 2020/2021

Prof.ssa Roberta Gori

S C
-
D I
-
Giacomo Trapani
-

Esercizio 1

Si assume che un utente possa possedere al massimo **uno e un solo abbonamento** alla volta.



Narrative

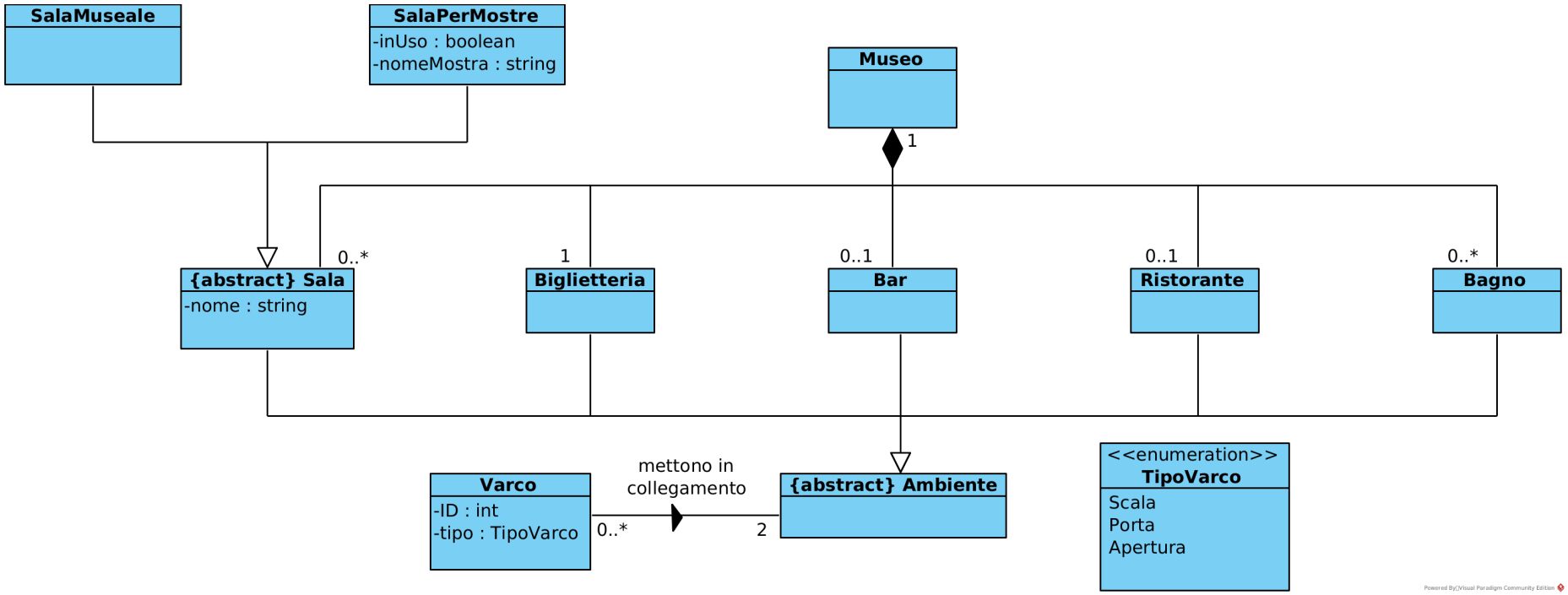
Nome	VisitaConAbbonamento
Descrizione	Un visitatore abbonato effettua una visita al museo.
Attore primario	VisitatoreAbbonato.
Attore secondario	StaffMuseo.
Precondizioni	Il visitatore dispone di un abbonamento valido.
Sequenza principale degli eventi	1. Ingresso del visitatore.
	2. Include Identificazione.
	3. Assegnazione dispositivo.
	4. Visita del museo.
	5. Consegna dispositivo.
	6. Include AggiornamentoAbbonamento
	7. Uscita del visitatore dal museo.
Postcondizioni	Il tempo e il numero di sale rimanenti nell'
	abbonamento sono stati aggiornati.
Sequenze alternative degli eventi	Identificazione fallisce; il visitatore sceglie se
	effettuare una visita senza abbonamento o andare via.

Nome	Identificazione
Descrizione	Identifica il visitatore come possessore di un abbonamento valido.
Attore primario	VisitatoreAbbonato.
Attore secondario	StaffMuseo.
Precondizioni	Il visitatore ha con sé un abbonamento.
Sequenza principale degli eventi	1. Il sistema verifica la validità dell'abbonamento.
	2. SE (abbonamento non valido)
	2.1 Identificazione termina con esito negativo.
	3. Lo staff del museo si accerta che l'identità del visitatore coincida
	con quella del titolare dell'abbonamento.
	4. SE (identità non coincidenti)
	4.1 Identificazione termina con esito negativo.
	5. Identificazione termina con esito positivo.
Postcondizioni	Identificazione termina con un esito,
Sequenze alternative degli eventi	Nessuna.

Nome	AggiornamentoAbbonamento
Descrizione	Aggiorna il numero di minuti e di sale rimanenti allo
	scadere dell'abbonamento.
Attore primario	StaffMuseo.
Attore secondario	VisitatoreAbbonatore.
Precondizioni	Il visitatore ha con sé un abbonamento valido e ha
	concluso una visita al museo.
Sequenza principale degli eventi	1. Il sistema ricava il numero di sale visitate
	e la durata complessiva della visita al museo.
	2. SE (sale visitate > sale rimanenti OR
	tempo trascorso > tempo rimanente)
	2.1 Segna l'abbonamento come scaduto
	2.2 Il visitatore paga la differenza scegliendo una tariffa
	o registrando un nuovo abbonamento da cui
	verrà sottratta la parte eccedente.
	2.3 AggiornamentoAbbonamento termina.
	3. All'abbonamento viene sottratto il numero di sale
	visitate e la durata totale della visita.
Postcondizioni	L'abbonamento del visitatore viene aggiornato.
Sequenza alternativa degli eventi	Nessuna.

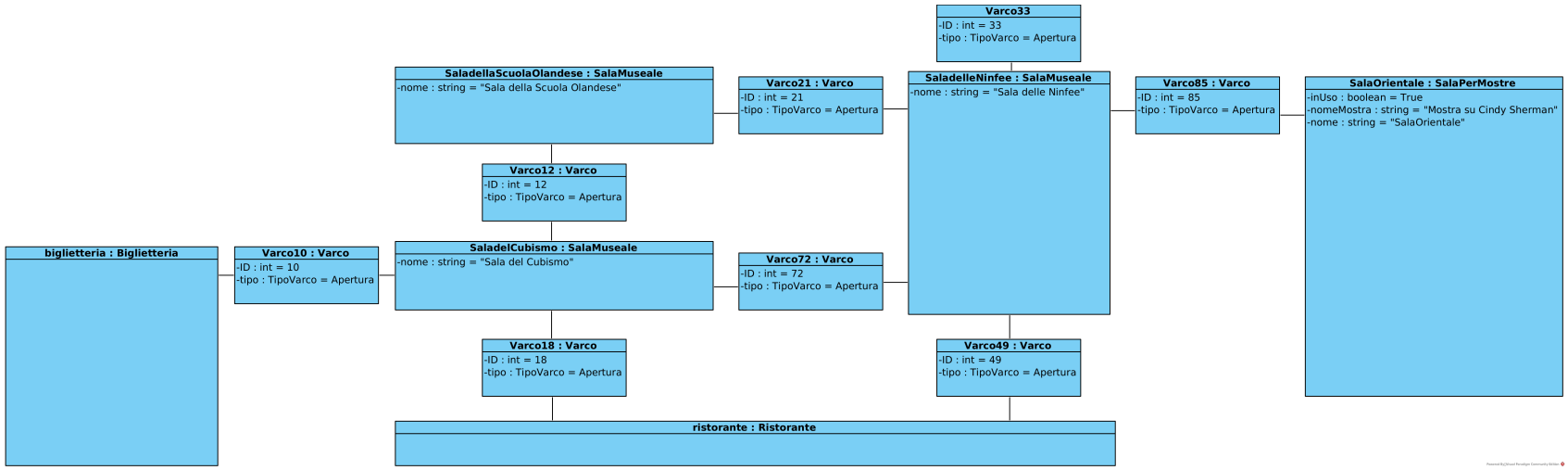
Esercizio2

Come si evince dal testo, all'interno di un museo esiste **una e una sola Biglietteria**, risultano **opzionali** le presenze di **Bar**, **Ristorante** (presenti al più **una e una sola volta** ciascuno), **Sale** e **Bagni** (presenti invece in quantità non meglio precisate).



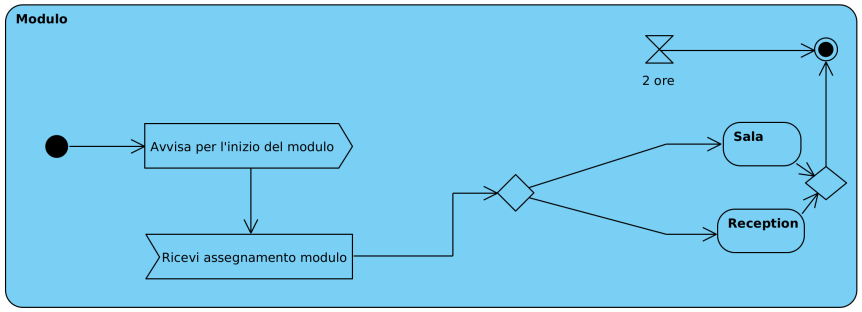
Esercizio 3

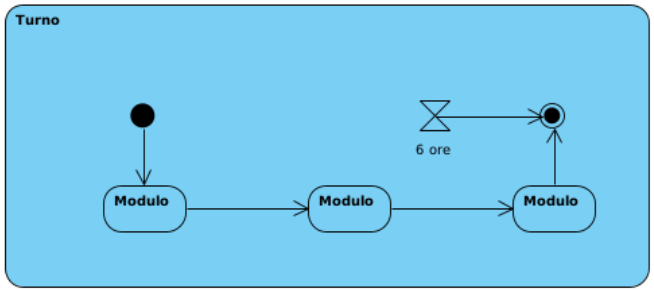
Si ricorda che il frammento di piantina del museo da rappresentare **non** costituisce una istanza completa o corretta di un museo: nello specifico, manca un **Ambiente** connesso al **Varco 33**.

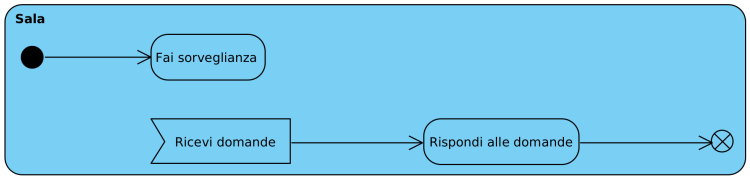


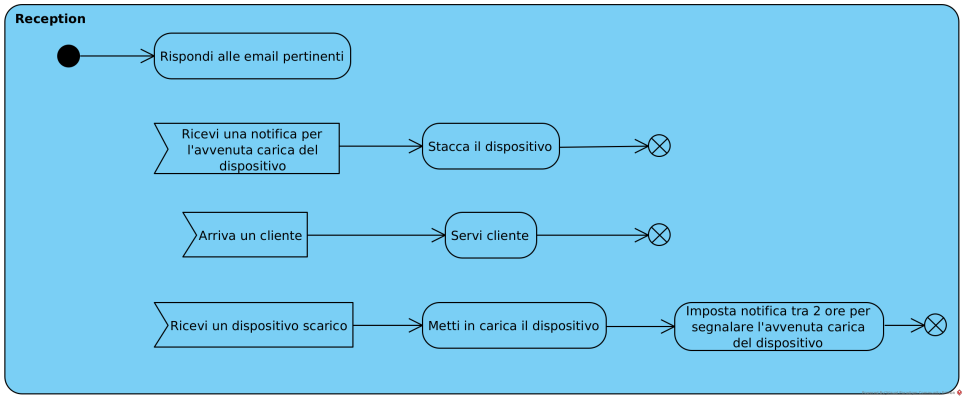
Esercizio 4

Scegliendo di modellizzare la scelta del modulo a cui un dipendente del museo viene assegnato come un segnale ricevuto dall'esterno ed essendo presente un vincolo sulla **durata** dei singoli moduli, si sceglie di eliminare la logica (della durata) temporale dal diagramma delle attività di **Sala** e di **Reception** (che vengono dunque modellizzati come un ciclo infinito) e di forzare la terminazione di questi al livello superiore (ossia al livello di **Modulo**.) In questo modo l'attività **Turno** non è altro che una sequenza di 3 attività modulo.



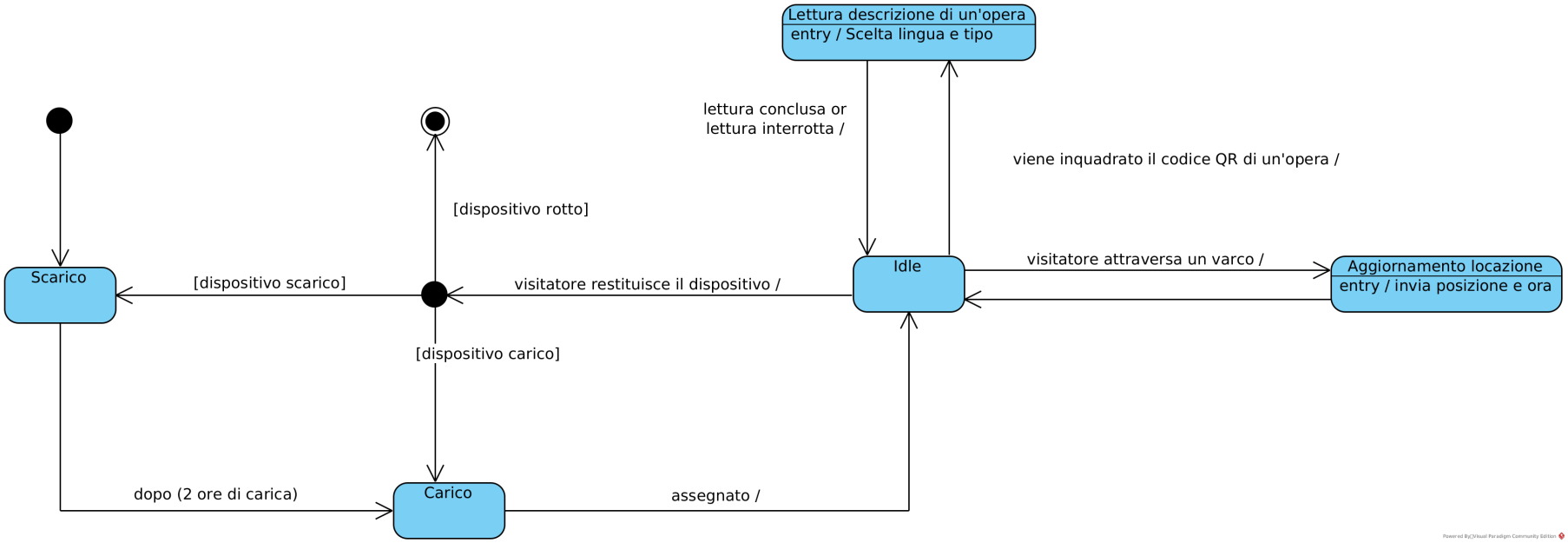






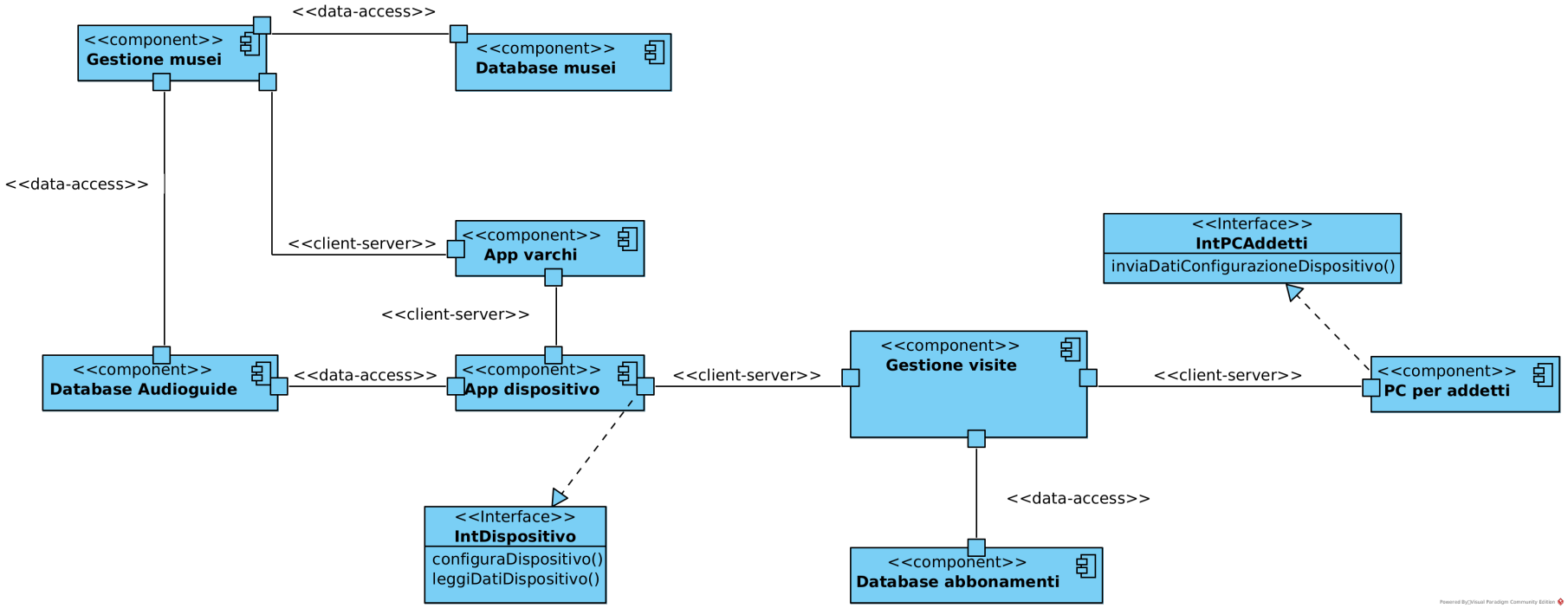
Esercizio 5

Si assume che la vita di un dispositivo non termini né alla chiusura del museo né quando questo si scarichi (questa infatti termina se e solo se il dispositivo risulta **rotto**) e che questi partano scarichi in modo tale da uniformarli.



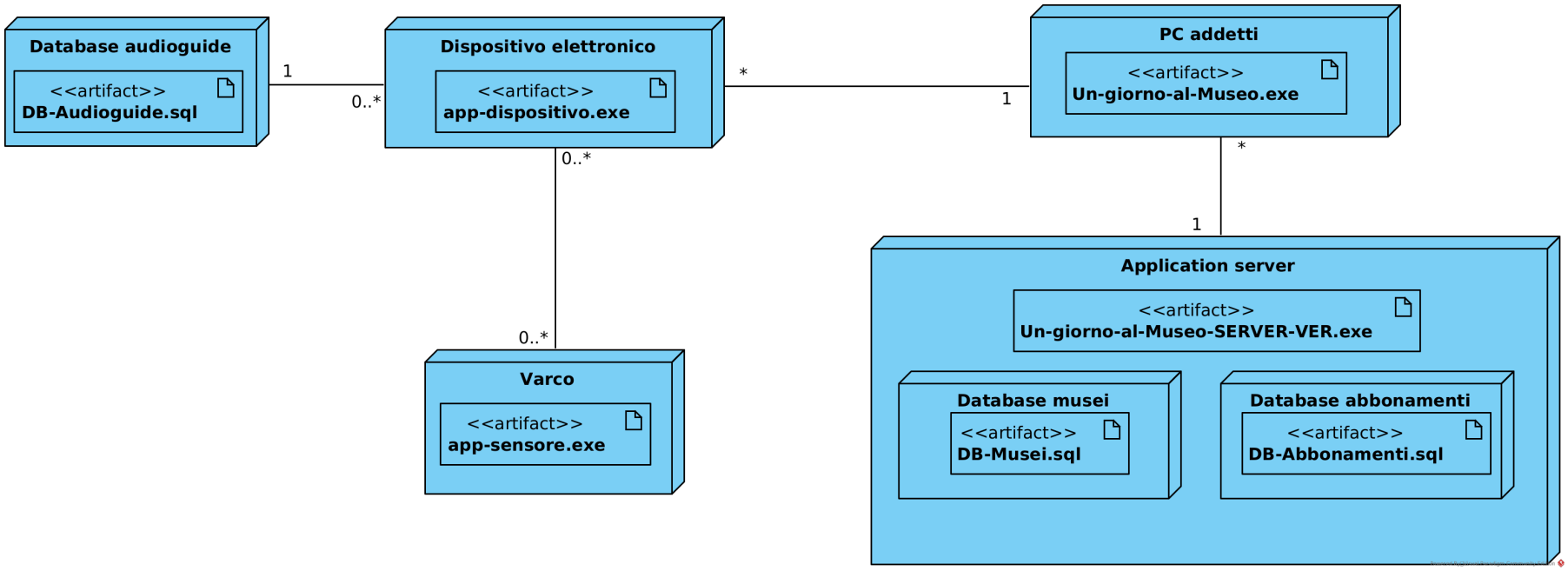
Esercizio 6

Si sceglie di dividere il sistema software Un giorno al Museo in due grosse componenti: **Gestione musei** e **Gestione Visite**. Si assume che la prima gestisca il database delle audioguide a cui accede ogni dispositivo elettronico nel momento in cui un visitatore inquadra un codice QR per avere a disposizione - appunto - l'audioguida dell'opera scelta. Dal testo si deduce inoltre che i varchi comunicano direttamente col dispositivo elettronico scrivendo in questo i dati riguardo il transito in una determinata sala e che questi vengano elaborati dalla seconda componente menzionata (poiché a disposizione di quest'ultima viene messa una operazione di lettura dei dati dai dispositivi elettronici).



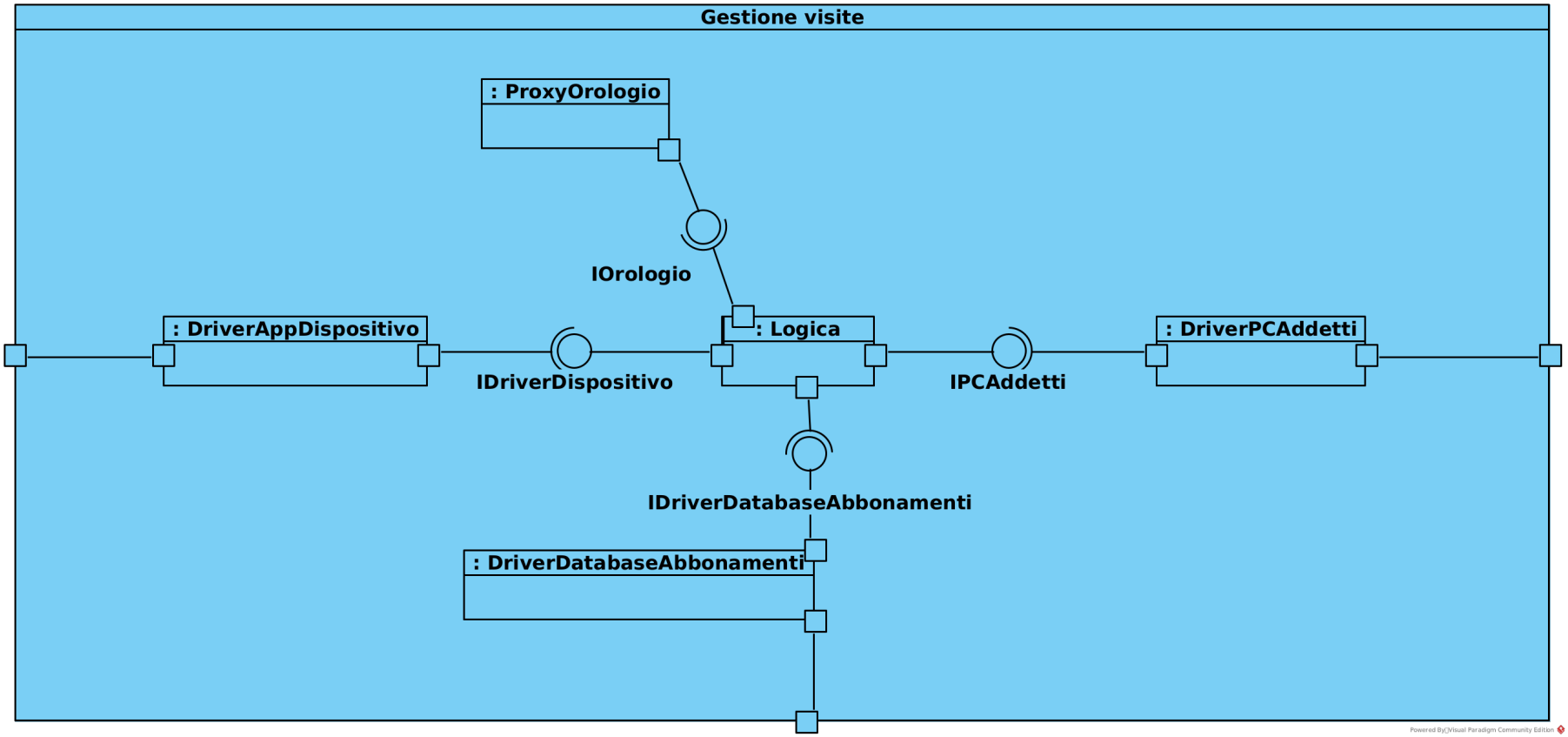
Esercizio 7

Si assume che il software Un giorno al Museo fornisca a ogni museo appartenente al sistema museale un eseguibile che comunica col **server centrale** sul quale sono salvati sia gli abbonamenti sia i dati dei singoli musei. Si assume inoltre che la gestione del database delle audioguide non sia affidata al sistema centrale, ma sia responsabilità dei singoli musei e che questo comunichi coi dispositivi elettronici.



Esercizio 8

Si assume che per il calcolo delle tariffe e per il passaggio da tempo assoluto (che si assume essere il formato adottato dai dati inviati dai varchi) a **tempo relativo** (ossia il formato che verrà utilizzato nei punti successivi) sia necessario un **ProxyOrologio**.



Esercizio 9

1 Esercizio 9.a

Assumiamo nello svolgimento dell'esercizio che:

- l'identificatore di ogni sala sia un intero (tipicamente una chiave primaria nel database delle sale);
- i metodi *bool salaPermanente(int ID)* e *bool salaTemporanea(int ID)* internamente usino un oggetto di tipo che contiene gli attributi corrispondenti presenti nella sala e che usino uno stub che simuli la connessione al database per recuperare le informazioni necessarie.

Un possibile stub è il seguente:

```
bool salaPermanente(int ID)
{
    return ID % 3 == 0;
}

bool salaTemporanea(int ID)
{
    return ID % 3 == 1;
}
```

che rispetta le specifiche del sistema in quanto non permette di avere sale contemporaneamente temporanee e permanenti. Si definiscono inoltre le due classi di equivalenza:

$$errore = \{[Passaggio]\ lp \mid (lp = NULL)\ \vee\ (\exists\ int\ i\ \in [0; lp.length()] \mid lp[i] = NULL))\}.$$

$$(\forall (h,k) \mid k + h\ \in [0; lp.length() - 1]\ \wedge h * k \geq 0.\ \mathbf{X}_{h,k} = \{[Passaggio]\ lp \mid (\exists\ I = \{i_1, ..., i_k\} \subseteq [1, ..., lp.length() - 1] \mid$$
$$(\forall i \in I.\ lp[i].orario - lp[i-1].orario \geq 30\ \wedge\ salaPermanente(lp[i-1].sala)) \wedge$$
$$(\exists J = \{j_1, ..., j_h\} \subseteq [1, ..., lp - length() - 1] \mid (\forall j \in J. lp[j].orario - lp[j-1].orario \geq 30\ \wedge salaTemporanea(lp[j-1].sala))\ \wedge\ I \cap J = \emptyset \wedge$$
$$(\forall h \in I \cup J)^C.\ (lp[h].orario - lp[h-1].orario \leq 30 \vee (!salaPermanente(lp[h-1].sala) \wedge !salaTemporanea(lp[h-1].sala)))))\}).$$

Ovvero la classe di equivalenza $\mathbf{X}_{h,k}$ contiene tutti e soli i vettori che corrispondono a visite in cui bisogna pagare per k sale permanenti e h sale temporanee indipendentemente dall'ordine in cui queste sono state visitate ponendo dunque che il valore restituito da *calcolaTariffa* è del tipo 3k + 5h.

Si osserva inoltre che senza perdita di generalità si può supporre di considerare $lp.length() \leq 3$ poiché il comportamento atteso non dipende dalla lunghezza del vettore in ingresso (oltre una dimensione che permetta almeno un pagamento) ai fini del testing del metodo.

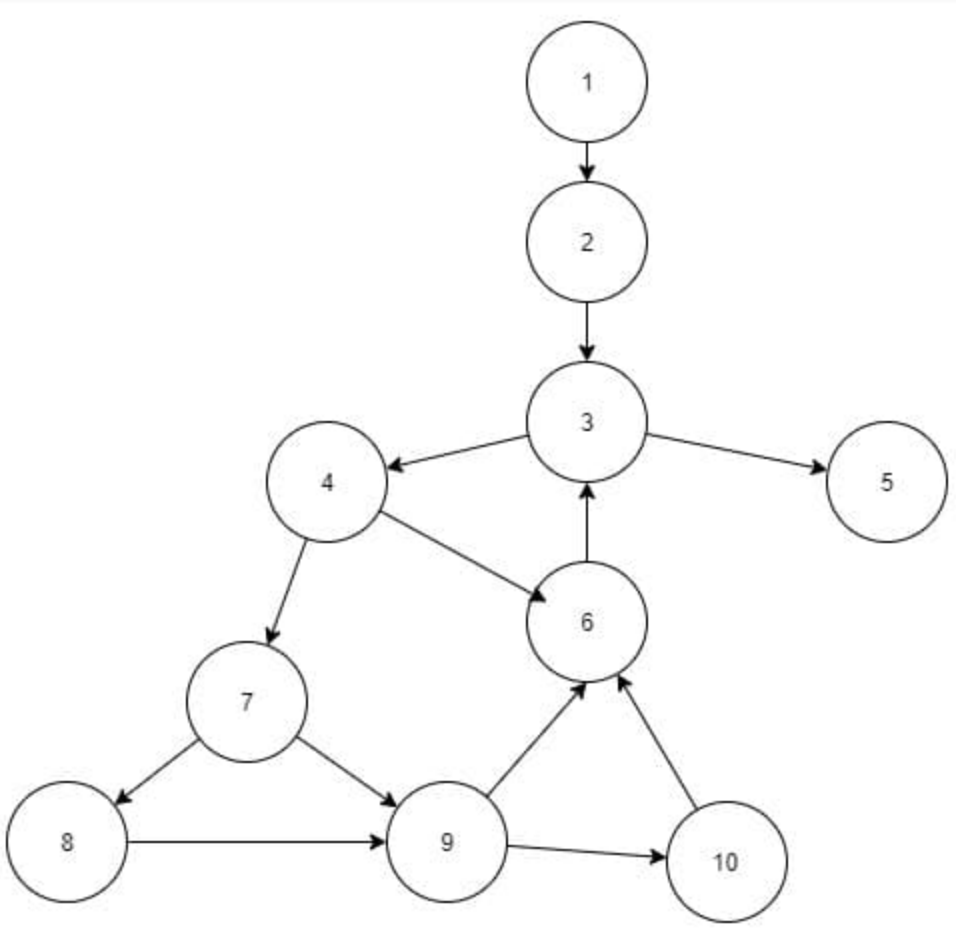
Si fissa dunque $n = 3$ e si suppone di identificare con p un oggetto di tipo Passaggio con i parametri (p.orario, p.sala), una batteria di test composta da un test di errore e un test di frontiera per ogni classe di equivalenza individuata può essere definita come:

- {NULL; errore; A};
- k = 0, h = 0 : {{[0, 0], [10, 1], [20, 0]}, 0, A};
- k = 0, h = 1 : {{[0, 0], [20, 1], [50, 2]}, 5, A};
- k = 1, h = 0 : {{[0, 0], [30, 1], [50, 0]}, 3, A};
- k = 1, h = 1 : {{[0, 0], [30, 1], [60, 0]}, 8, A};
- k = 0, h = 2 : {{[0, 1], [30, 4], [60, 0]}, 10, A};
- k = 2, h = 0 : {{[0, 0], [30, 3], [60, 0]}, 6, A};

2 Esercizio 9.b

Il corpo del metodo viene partizionato in questo modo:

1. int tariffa = 0;
2. for (i = 1;
3. i < lp.length;
4. if (lp[i].orario - lp[i-1].orario >= 30)
5. return tariffa;
6. i++)
7. if (salaPermanente(lp[i-1].sala))
8. tariffa += 3;
9. if (salaTemporanea(lp[i-1].sala))
10. tariffa += 5;



Grafo di flusso

3 Esercizio 9.c

La batteria di test definita al punto (a) con l'aggiunta dell'ulteriore test $\{\{[0,0]\},0,A\}$ è sufficiente per garantire una copertura del 100% del grafo di flusso precedente. Infatti per garantire tale copertura sono sufficienti i seguenti testcase:

- un testcase in cui il corpo del ciclo *for* non viene mai raggiunto (quello appena aggiunto);
- un testcase in cui il corpo del ciclo *for* viene eseguito almeno una volta (succede in ogni caso in cui lp $\neq$ NULL);
- un testcase in cui il corpo del primo ramo *if* non viene mai raggiunto (k = h = 0);
- un testcase in cui il corpo del primo ramo *if* viene eseguito almeno una volta (k = 1, h = 0);
- un testcase in cui il corpo del secondo ramo *if* viene eseguito almeno una volta, il terzo invece no (k = h = 1, iterazione n. 1);
- un testcase in cui il corpo del terzo ramo *if* viene eseguito almeno una volta, il secondo invece no (k = h = 1, iterazione n. 2).

4 Esercizio 9.d

La specifica pone il vincolo $\neg(\exists int\ i \mid salaPermanente(i) = salaTemporanea(i) = True)$.

Si propongono due soluzioni all'esercizio affrontando il problema da due punti di vista differenti.

4.1 Prima soluzione

Suppongo le due implementazioni di *salaPermanente* e di *salaTemporanea* come due funzioni del tipo $f : R \to \{T, F\}$. Ponendo ad esempio ID $= \infty \vee$ ID $= -\infty$ le operazioni usate all'interno delle due funzioni potrebbero non essere definite in quanto $\infty \mod n$ ha un risultato che dipende dall'implementazione del linguaggio in cui si sviluppa (e.g. supponendo che i metodi siano implementati in modo simile a quelli dello stub fornito al punto (a), in Java il problema sussiterebbe poiché il modulo di infinito risulta essere *NaN* e - supponendo anche che quella fosse una Sala - entrambi i metodi restituirebbero un valore *False* che permetterebbe dunque al visitatore di non pagare per quella Sala; la coppia di risposte dello stub sarebbe dunque {False, False}).

4.2 Seconda soluzione

Si suppone che le definizioni di salaPermanente e di salaTemporanea siano mal poste, ad esempio:

```
1    bool salaPermanente(int ID)
2    {
3        return ID % 3 == 0;
4    }
```

```
5
6       bool salaTemporanea(int ID)
7       {
8           return ID % 3 == 0;
9       }
```

in questo modo una sala risulta contemporaneamente permanente e temporanea (si restituisce dunque la coppia {True, True}) - in contrasto con la specifica. Assumendo che tutti i valori per cui vale $ID \mod 3 == 0$ siano sale permanenti e supponendo che un visitatore non attraversi solo queste, nel momento in cui (il visitatore) richiederà la possibilità di procedere al pagamento (n.d.r. con **tariffa bianca**) l'ammontare effettivo non sarebbe coerente con quanto il visitatore si aspetterebbe.