PISICINAMIMU

PiscinaMiMu è un software sviluppato da Paolo Milone in collaborazione con Salvatore Roberto Musumeci, il lavoro sul software è stato svolto in modo iterativo ed è visualizzabile al seguente link: https://github.com/giocottildi/PisicinaMiMu

Il repository è organizzato in delle cartelle: una di ideazione in cui si trovano i documenti aggiornati fino alla fase finale, e tre di iterazione in cui si trovano i documenti relativi all'iterazione in esame, e tutto il codice sviluppato nella realizzazione del software.

Nelle pagine successive verrà inserito il contenuto già presente nei pdf di ogni iterazione, ai fini di poter fornire un documento che racchiuda tutto ciò che è stato fatto dal nostro gruppo nelle relative iterazioni.

ITERAZIONE 1

1.0 Introduzione

La prima iterazione si basa sull'analisi e progettazione dei primi due casi d'uso, ritenuti fondamentali ai fini del funzionamento dell'applicativo e dei casi d'uso 8 e 10.

Scopo di questa, così come delle iterazioni che verranno dopo, è quello di andare ad Implementare in maniera iterativa l'applicativo software raffinando costantemente la sua Visione, identificando e implementando la maggior parte dei requisiti richiesti e test per garantire il corretto funzionamento del codice dell'applicazione.

In particolare, in questa prima iterazione andremo ad implementare:

- UC1: NUOVO CORSO
- UC2: MODIFICA CORSO
- UC8: ASSEGNA ISTRUTTORE
- UC10: ASSUMI ISTRUTTORE

1.1 Aggiornamento dei casi d'uso

Durante questa fase sono stati specificati e rivisti dei dettagli dei casi d'uso implementati. Per visionare le modifiche apportate ai suddetti, si consulti il documento "Modello dei casi d'uso" riportato nella cartella in versione aggiornata per questa iterazione.

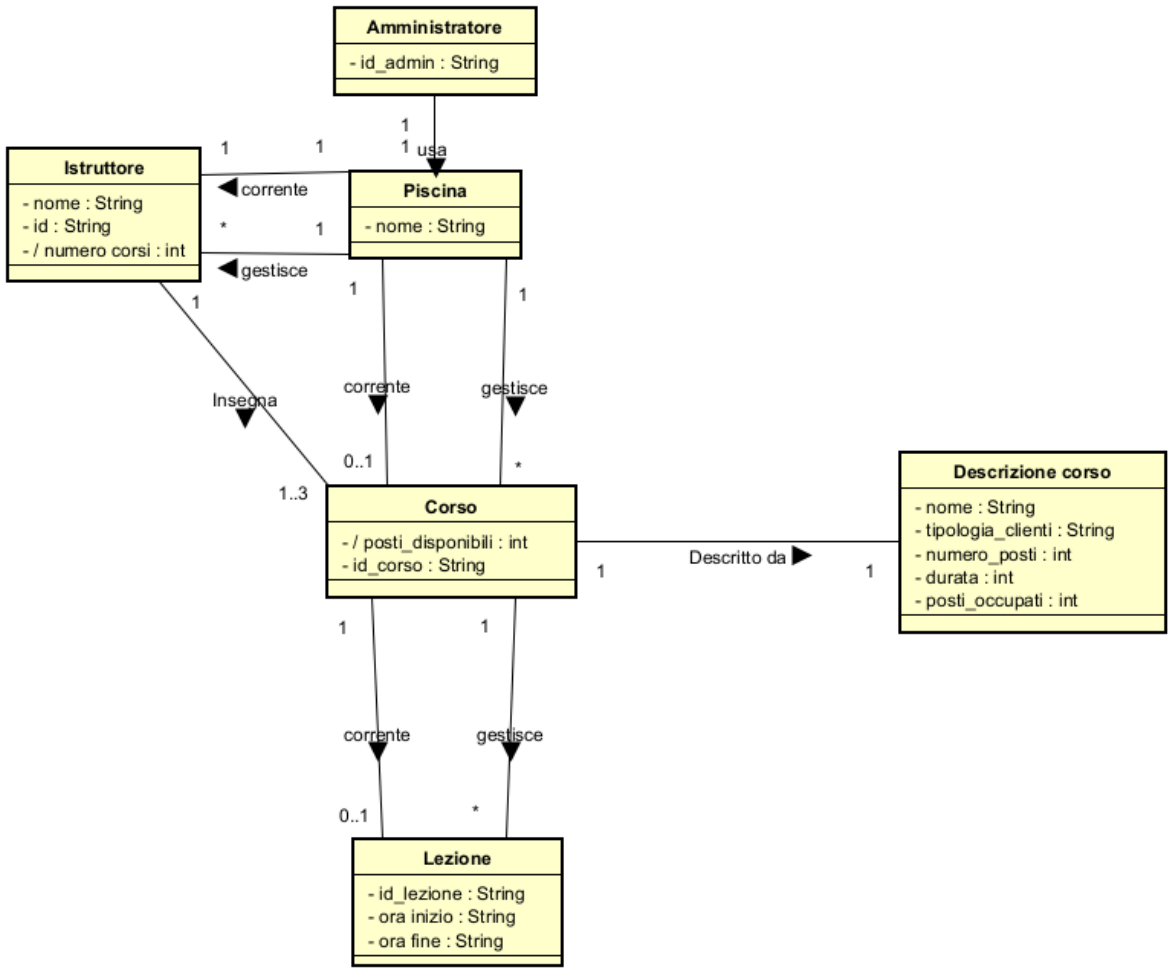
Eventuali ulteriori modifiche ai documenti saranno visionabili tramite la cronologia.

Fase di Analisi

Le classi concettuali identificate durante questa prima iterazione sono:

- Amministratore: Attore primario che interagisce con il sistema.
- Piscina: Rappresenta il sistema PiscinaMiMu
- Istruttore
- Corso
- Descrizione Corso
- Lezione

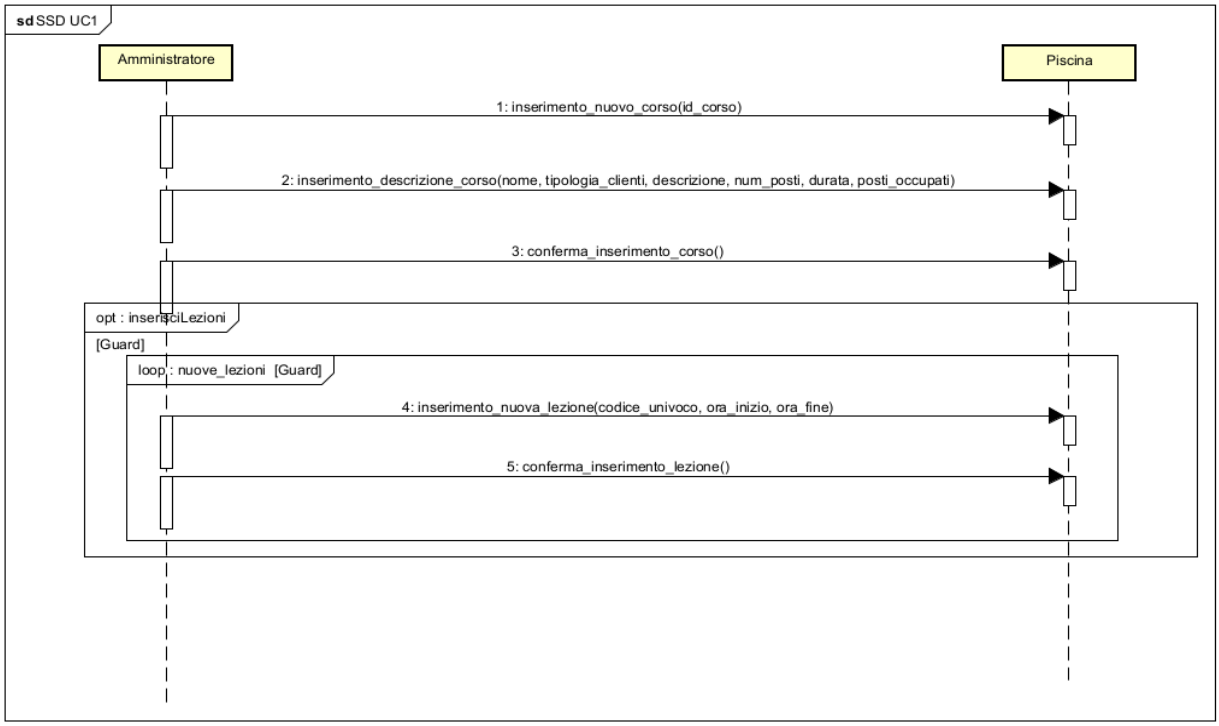
Da cui tenendo conto di associazioni e attributi, ottenuti analizzando i primi tre casi d'uso, è stato ricavato il seguente Modello di Dominio:



2.1 Diagrammi di sequenza di sistema UC1

- UC1: INSERISCI NUOVO CORSO

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'amministratore e il sistema Piscina.



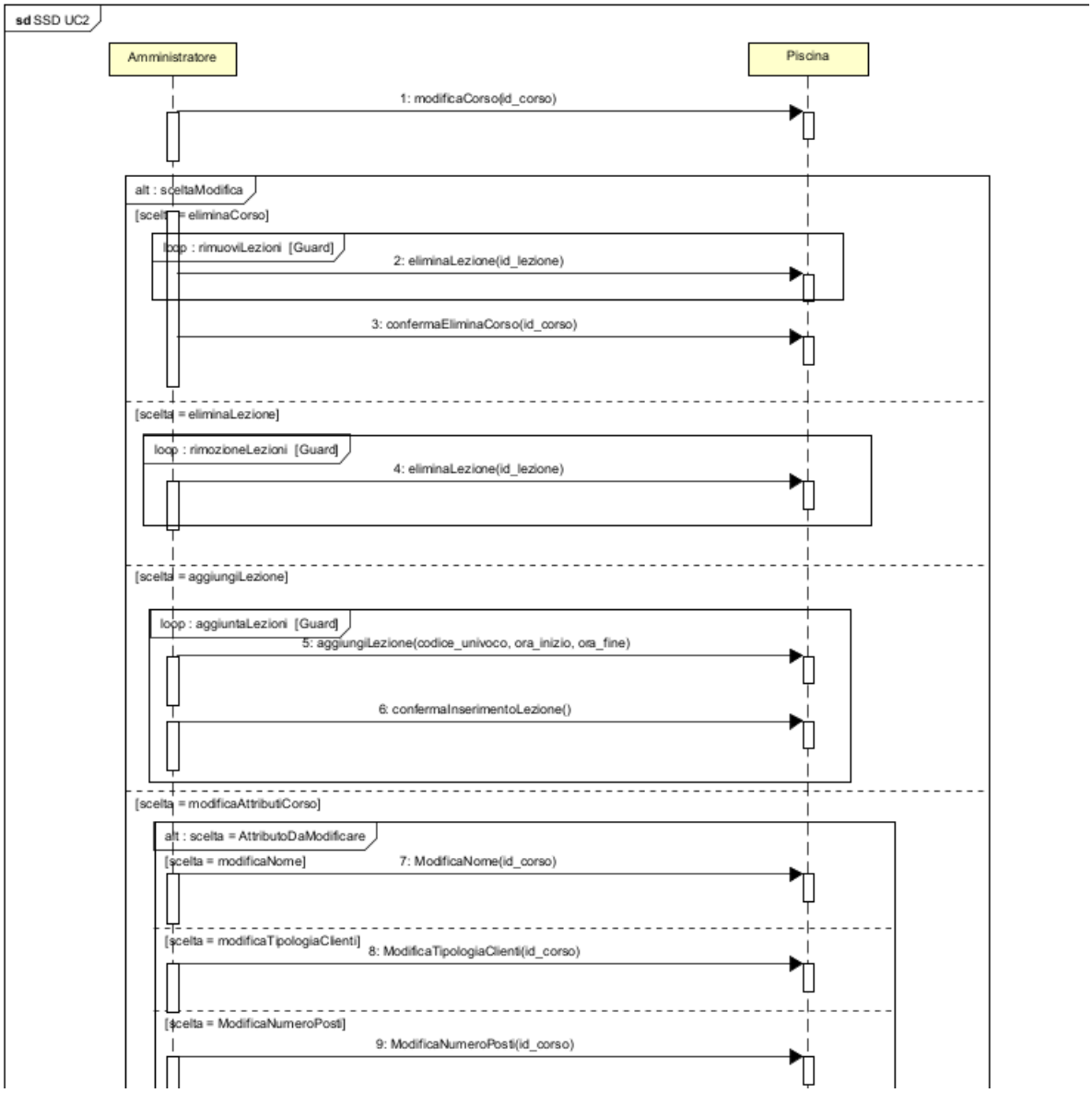
2.1.1 Contratti delle operazioni

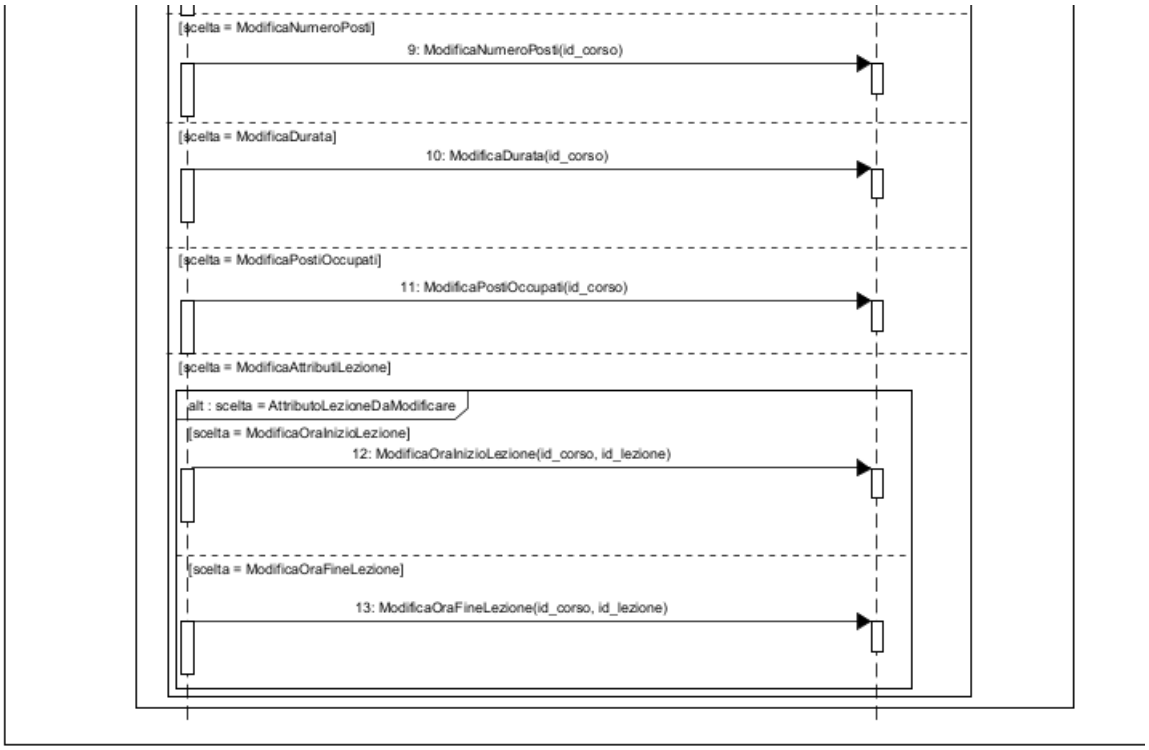
Non sono stati scritti contratti per questo caso d'uso.

2.2 Diagrammi di sequenza di sistema UC2

- UC2: MODIFICA CORSO ESISTENTE

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'amministratore e il sistema Piscina.


sd SSD UC2
Amministratore
Piscina
1: modificaCorso(id_corso)
alt : sceltaModifica
[scelta = eliminaCorso]
loop : rimuoviLezioni [Guard]
2: eliminaLezione(id_lezione)
3: confermaEliminaCorso(id_corso)
[scelta = eliminaLezione]
loop : rimozioneLezioni [Guard]
4: eliminaLezione(id_lezione)
[scelta = aggiungiLezione]
loop : aggiuntaLezioni [Guard]
5: aggiungiLezione(codice_univoco, ora_inizio, ora_fine)
6: confermaInserimentoLezione()
[scelta = modificaAttributiCorso]
alt : scelta = AttributoDaModificare
[scelta = modificaNome]
7: ModificaNome(id_corso)
[scelta = modificaTipologiaClienti]
8: ModificaTipologiaClienti(id_corso)
[scelta = ModificaNumeroPosti]
9: ModificaNumeroPosti(id_corso)



2.2.1 Contratti delle operazioni

I seguenti Contratti descrivono le principali operazioni eseguite dal sistema sulla base degli eventi individuati nell'SSD.

Riferimento:	UC2.1
Operazione:	modificaCorso(id_corso)
Pre-condizioni:	• Deve esistere un'istanza di Corso c
Post-condizioni:	• Viene creata un'associazione tra GestoreCorsi e un'istanza c di Corso

Riferimento:	UC2.2
Operazione:	eliminaLezione(id_lezione)
Pre-condizioni:	• Esiste un'istanza di corso c associata a GestoreCorsi • Esiste un'istanza l di lezione associata all'istanza c di Corso

Post-condizioni:	• Viene rimossa un'associazione tra l'istanza c di Corso ed un'istanza l di corso

Riferimento:	UC2.3
Operazione:	confermaEliminaCorso(id_corso)
Pre-condizioni:	• Deve esistere l'istanza Corso c associata a GestoreCorsi
Post-condizioni:	• Viene eliminata l'istanza c di Corso

Riferimento:	UC2.5
Operazione:	inserimentoNuovaLezione(id_lezione, ora_inizio, ora_fine)
Pre-condizioni:	• Esiste un'istanza c di Corso
Post-condizioni:	• Viene creata un'istanza l di Lezione

Riferimento:	UC2.6
Operazione:	confermaInserimentoLezione()
Pre-condizioni:	• Esiste un'istanza c di Corso • Esiste un'istanza l di Lezione
Post-condizioni:	• Viene creata un'associazione tra Corso c e Lezione l

Riferimento:	UC2.7
Operazione:	ModificaNome(id_corso)
Pre-condizioni:	• Esiste un'istanza c di Corso
Post-condizioni:	• L'attributo nome della DescrizioneCorso cd di Corso c viene modificato

Riferimento:	UC2.8
Operazione:	ModificaTipologiaClienti(id_corso)
Pre-condizioni:	• Esiste un'istanza c di Corso
Post-condizioni:	• L'attributo TipologiaClienti della DescrizioneCorso cd di Corso c viene modificato

Riferimento:	UC2.9
Operazione:	ModificaDurata(id_corso)
Pre-condizioni:	• Esiste un'istanza c di Corso
Post-condizioni:	• L'attributo Durata della DescrizioneCorso cd di Corso c viene modificato

Riferimento:	UC2.10
Operazione:	ModificaNumeroPosti(id_corso)
Pre-condizioni:	• Esiste un'istanza c di Corso
Post-condizioni:	• L'attributo numero_posti della DescrizioneCorso cd di Corso c viene modificato

Riferimento:	UC2.11
Operazione:	ModificaPostiOccupati(id_corso)
Pre-condizioni:	• Esiste un'istanza c di Corso

Post-condizioni:	• L’attributo posti_occupati della DescrizioneCorso cd di Corso c viene modificato

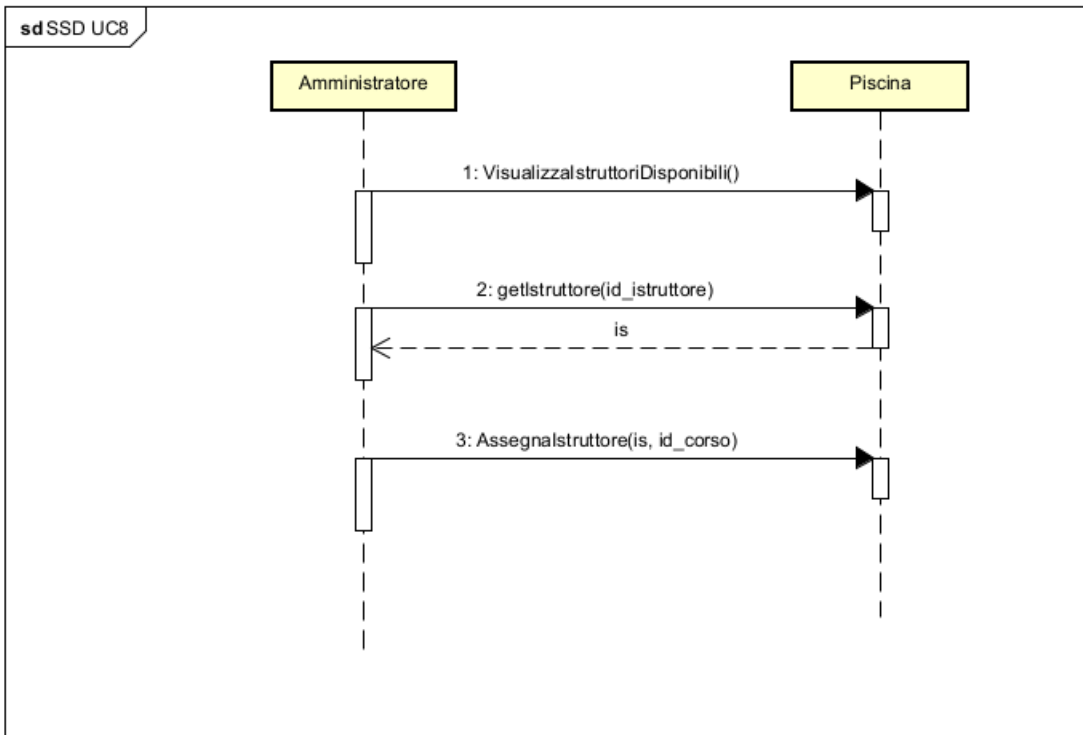
Riferimento:	UC2.12
Operazione:	ModificaOraInizioLezione(id_corso, id_lezione)
Pre-condizioni:	• Esiste un’istanza c di Corso • Esiste un’istanza l di Lezione associata all’istanza c di Corso
Post-condizioni:	• L’attributo ora_inizio di Lezione l viene modificato

Riferimento:	UC2.12
Operazione:	ModificaOraFineLezione(id_corso, id_lezione)
Pre-condizioni:	• Esiste un’istanza c di Corso • Esiste un’istanza l di Lezione associata all’istanza c di Corso
Post-condizioni:	• L’attributo ora_fine di Lezione l viene modificato

2.3Diagrammi di sequenza di sistema UC8

- UC8 ASSEGNA ISTRUTTORE AL CORSO

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l’amministratore e il sistema Piscina.



2.3.1 Contratti delle operazioni

Riferimento:	UC8
Operazione:	VisualizzaIstruttoriDisponibili()
Pre-condizioni:	• Nessuna
Post-condizioni:	• Nessuna

Riferimento:	UC8
Operazione:	getIstruttore(id_istruttore)
Pre-condizioni:	• Deve esistere un'istanza is di Istruttore
Post-condizioni:	• Nessuna

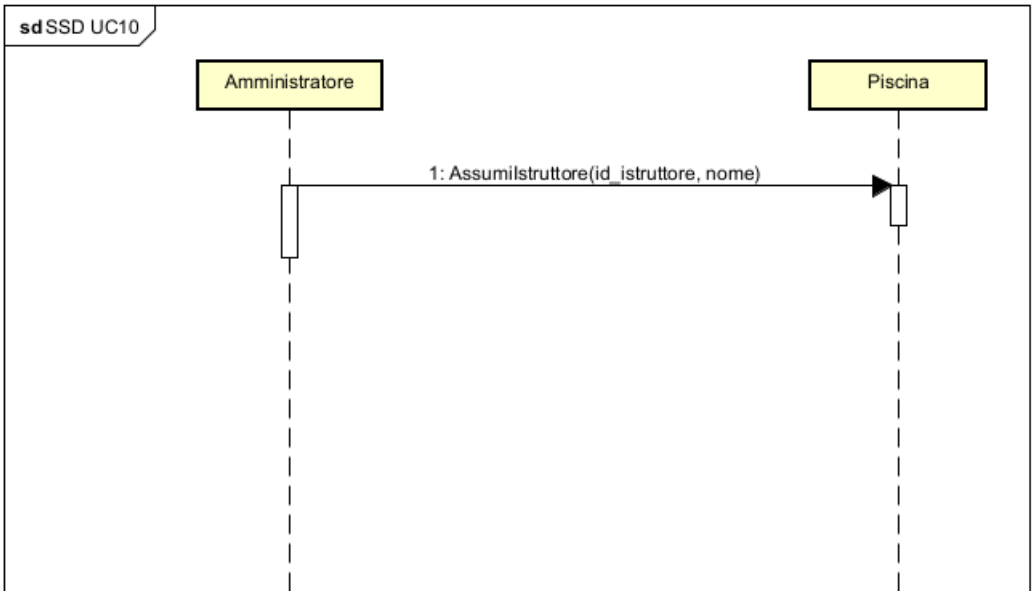
Riferimento:	UC8
Operazione:	AssegnaIstruttore(is, id_corso)
Pre-condizioni:	• Deve esistere un'istanza is di Istruttore • Deve esistere un'istanza c di Corso
Post-condizioni:	• È stata creata l'associazione tra l'istanza di corso c e l'istanza is di istruttore

	• È stata creata l'associazione tra l'istanza is di istruttore e l'istanza c di corso

2.4 Diagrammi di sequenza di sistema UC10

- UC10: ASSUMI ISTRUTTORE

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'amministratore e il sistema Piscina.



2.4.1 Contratti delle operazioni

I seguenti Contratti descrivono le principali operazioni eseguite dal sistema sulla base degli eventi individuati nell'SSD.

Riferimento:	UC10
Operazione:	AssumiIstruttore(id_istruttore, nome)
Pre-condizioni:	• Nesuna
Post-condizioni:	• Viene creata un'istanza di Istruttore • Vengono inizializzati I campi id_istruttore, nome e num_corsi di istruttore

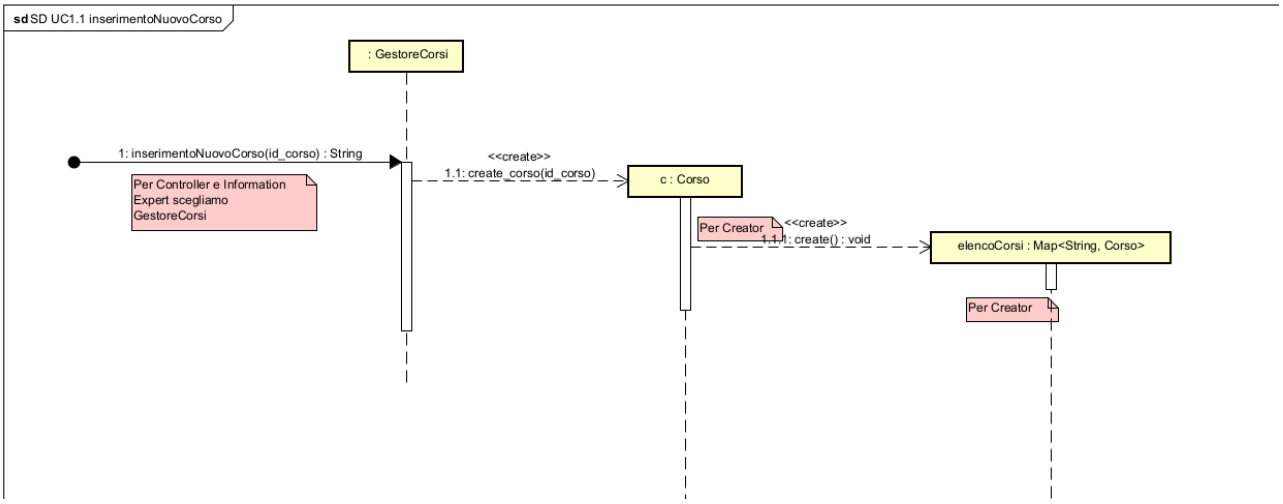
3.0 Fase di Progettazione

Gli elaborati principali della fase di progettazione sono i diagrammi di interazione, articolati in diagrammi di sequenza e diagrammi di interazione, con lo scopo di descrivere il comportamento del sistema da un punto di vista dinamico durante i casi d'uso presi in considerazione per questa prima iterazione. Insieme a questi, il diagramma delle classi rappresenta il sistema da un punto di vista statico. Di seguito vengono riportati:

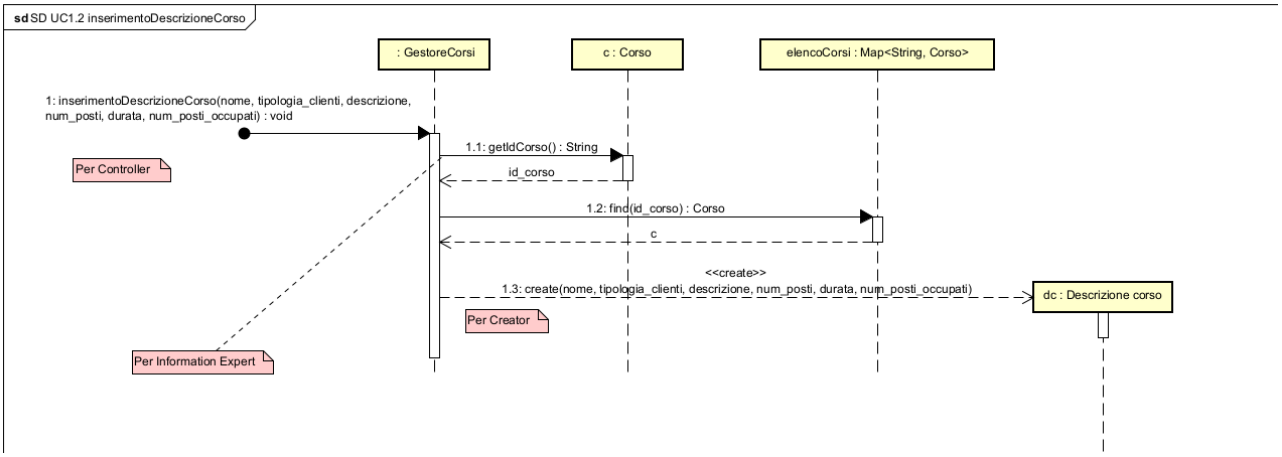
3.1 Diagrammi di Sequenza UC1

Si è utilizzato un Controller GestoreCorsi per lavorare come interfaccia del sistema. Avrà il compito di creare le istanze di Corso, Descrizione Corso e Lezione.

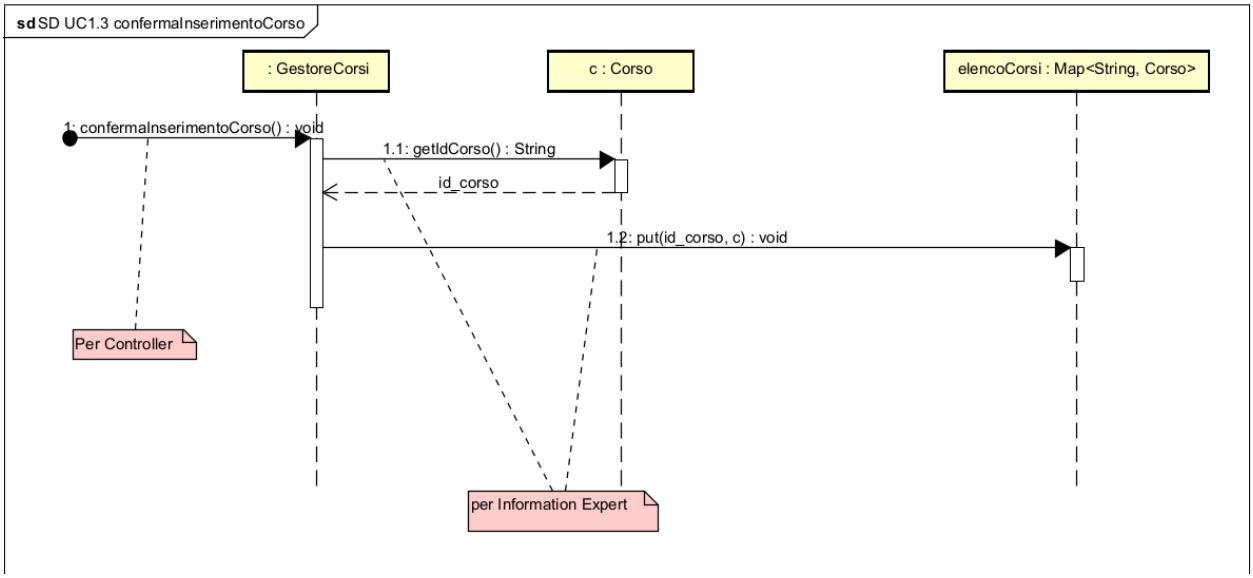
inserimentoNuovoCorso



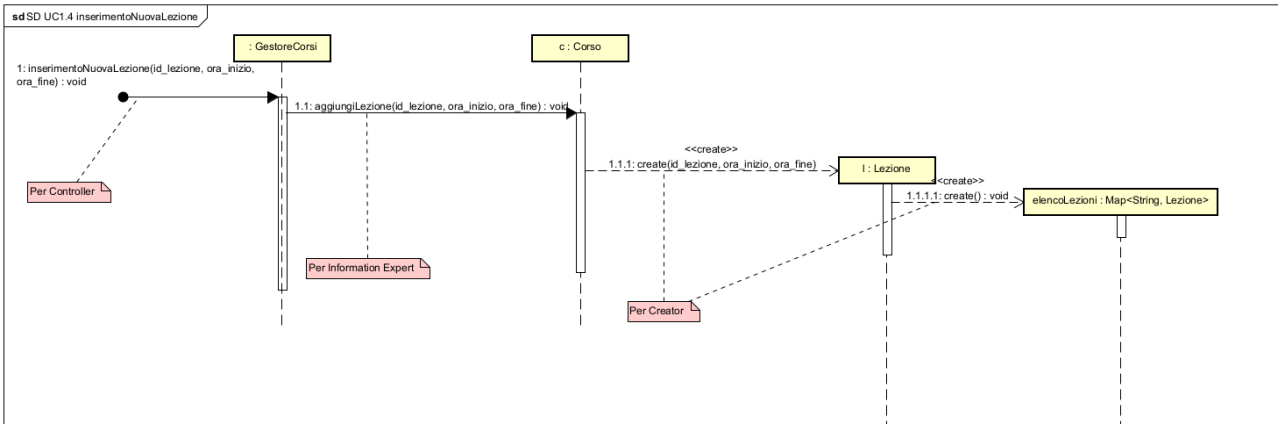
inserimentoDescrizioneCorso



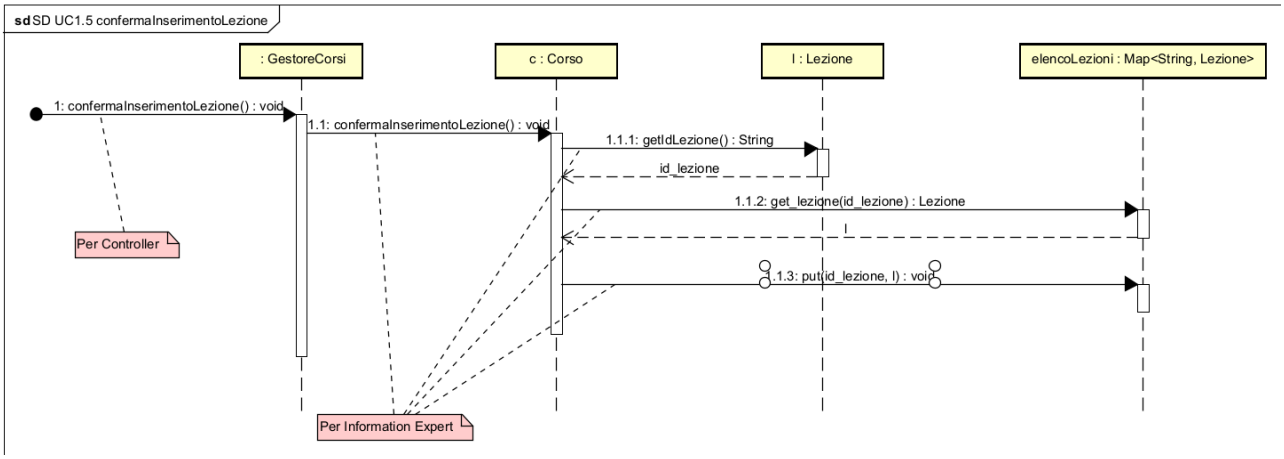
confermaInserimentoCorso



inserimentoNuovaLezione



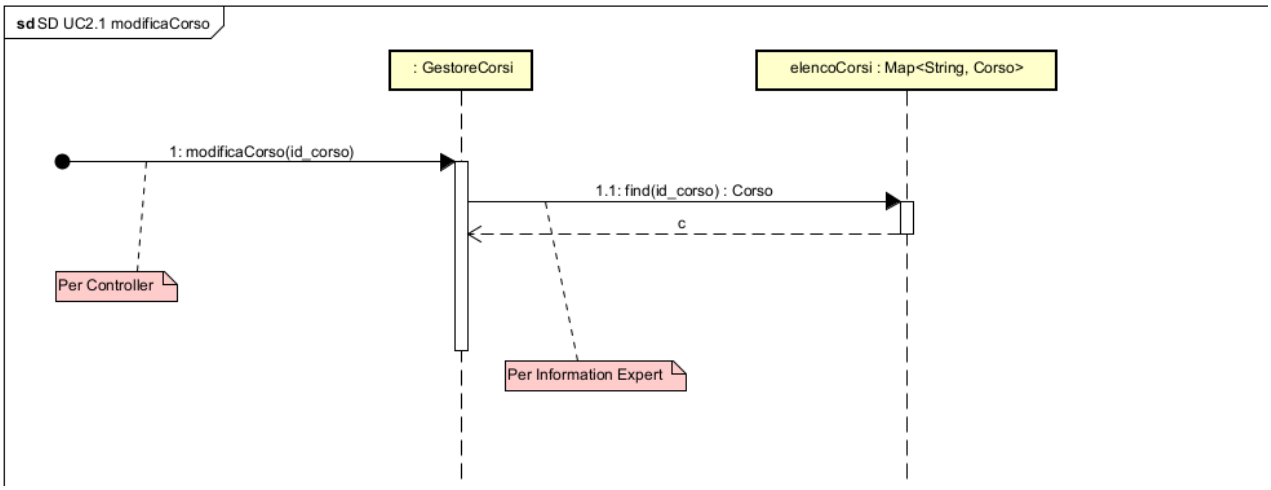
confermaInserimentoLezione



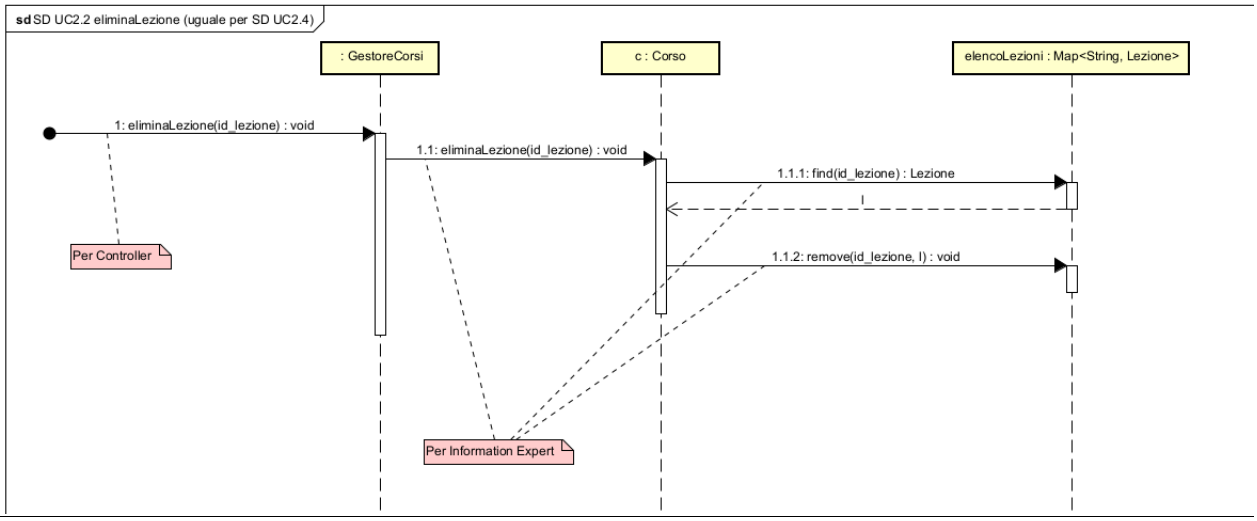
3.2 Diagrammi di Sequenza UC2

Si è utilizzato un Controller GestoreCorsi per lavorare come interfaccia del sistema. Avrà il compito di modificare le istanze di Corso, Descrizione Corso e Lezione.

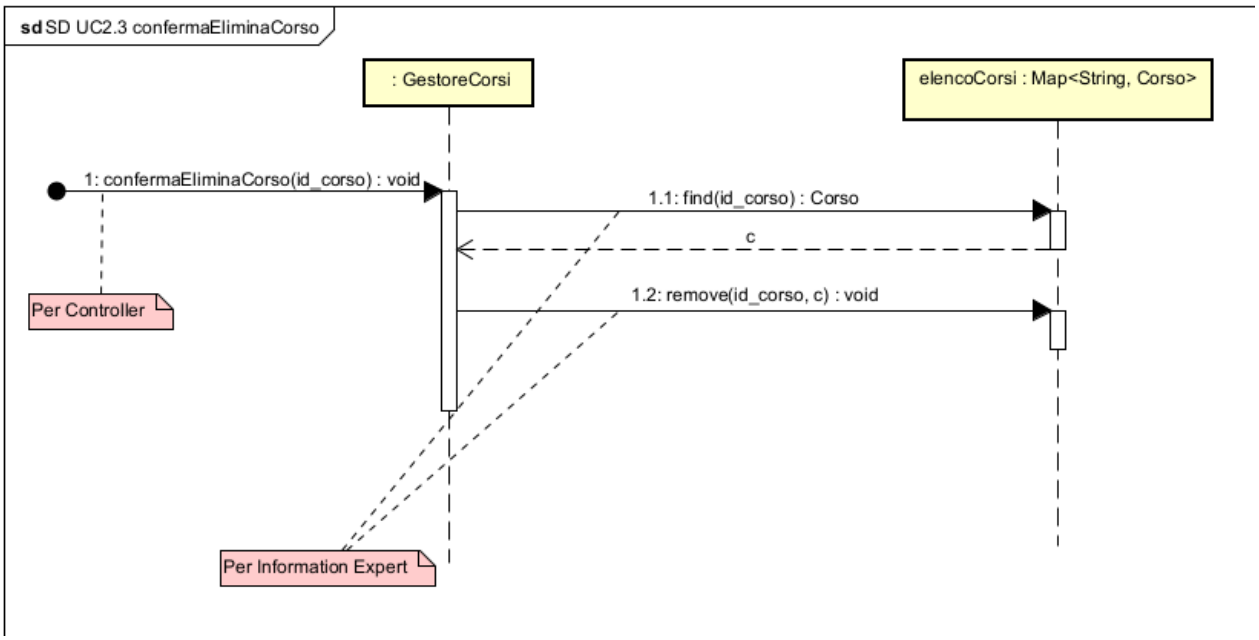
modificaCorso



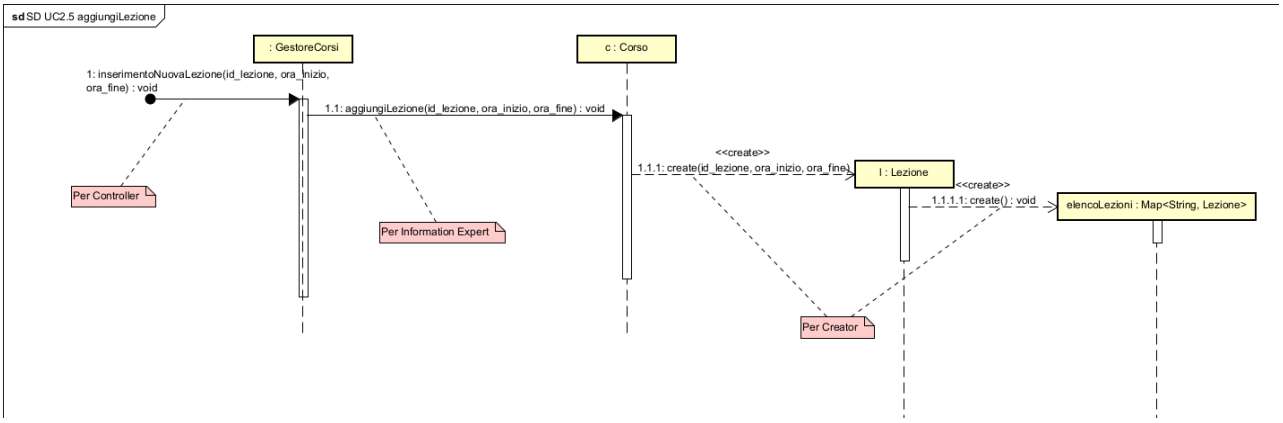
eliminaLezione



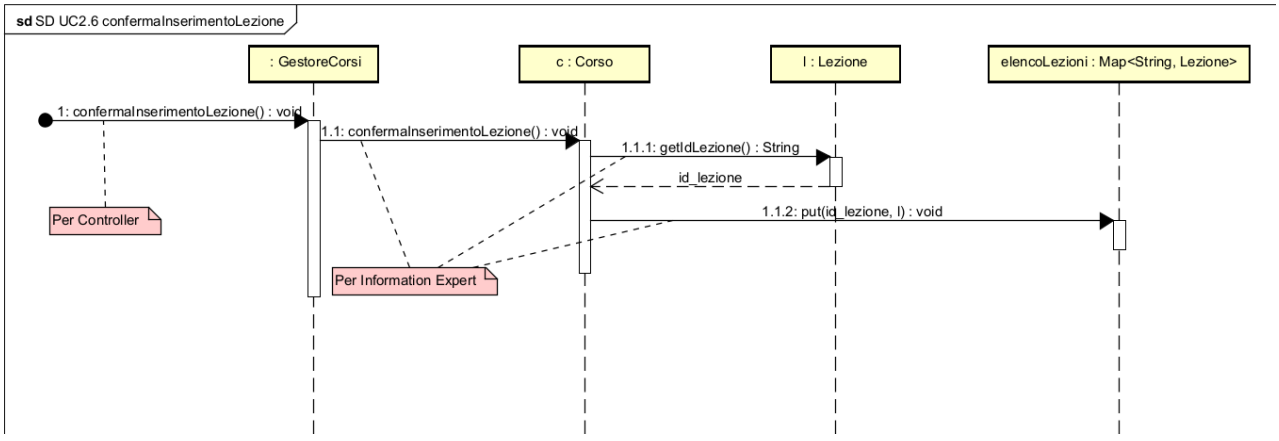
confermaEliminaCorso



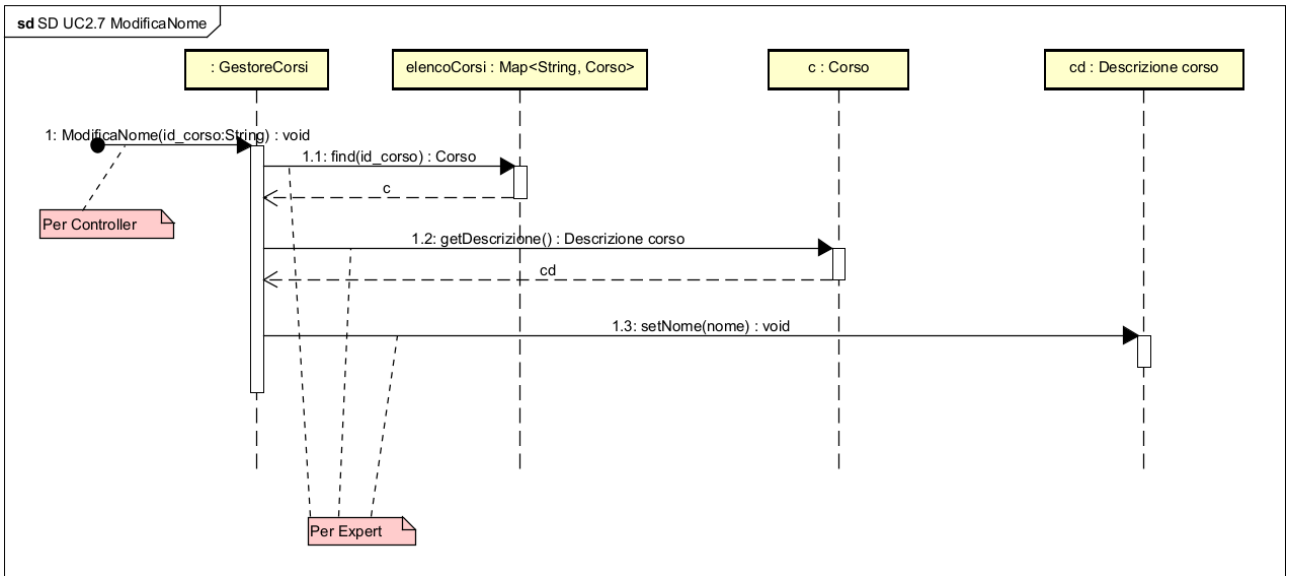
aggiungiLezione



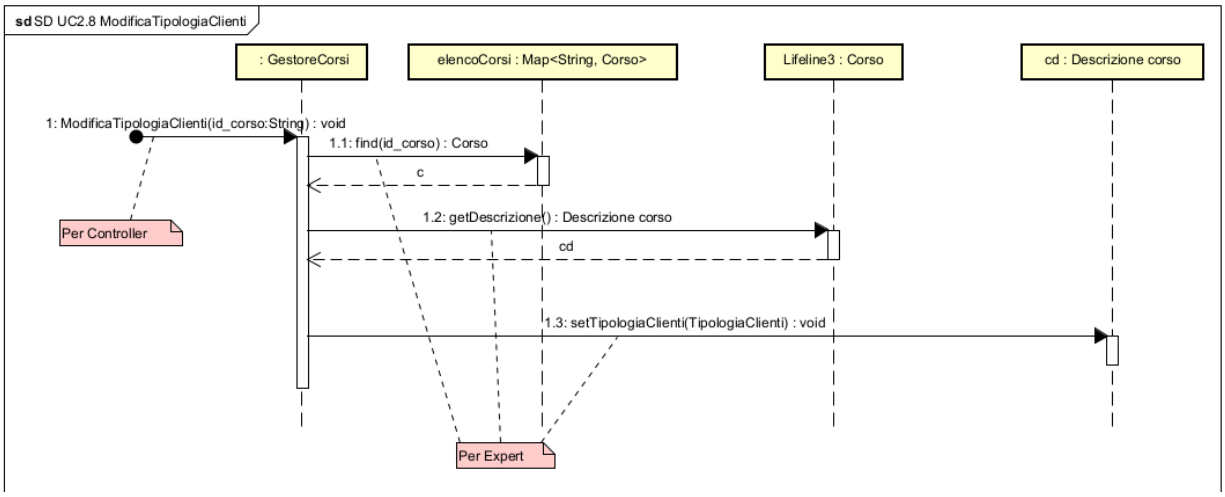
confermaInserimentoLezione



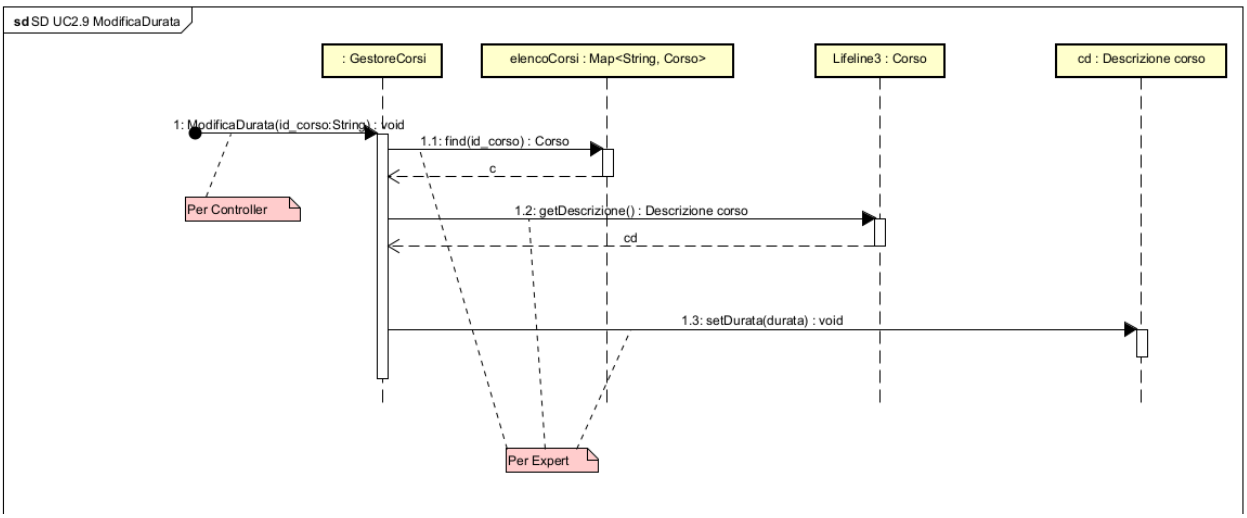
modificaNome



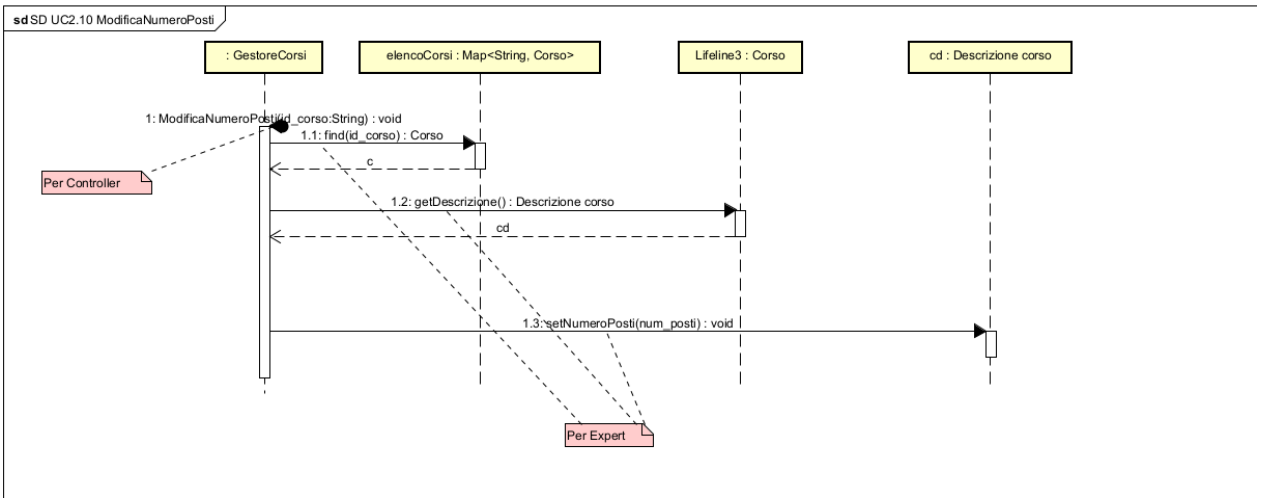
modificaTipologiaClienti



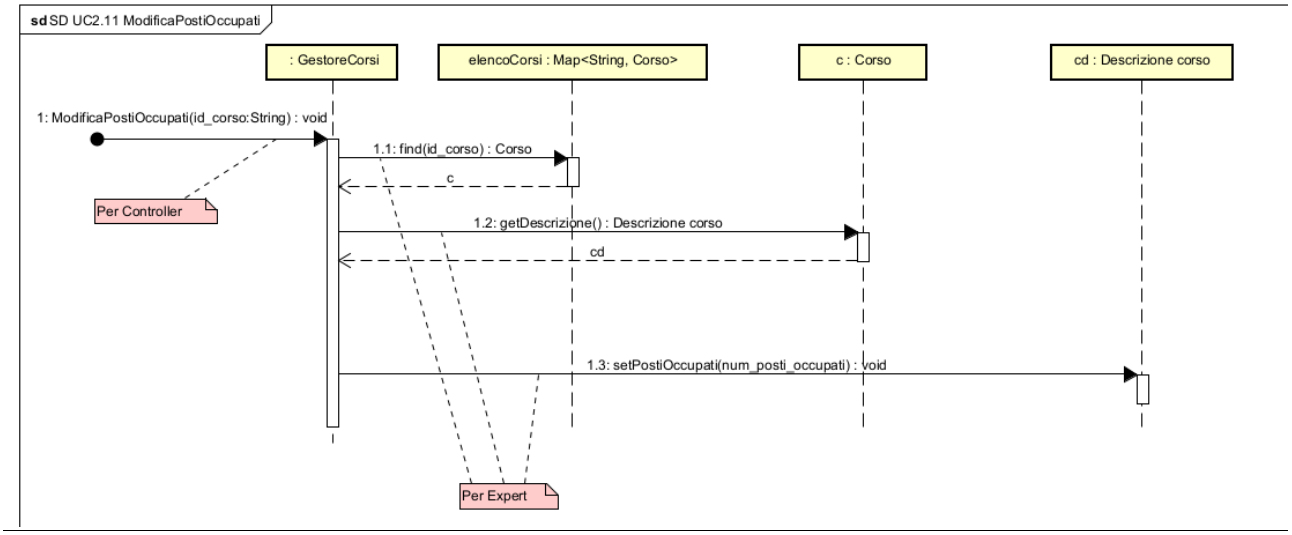
modificaDurata



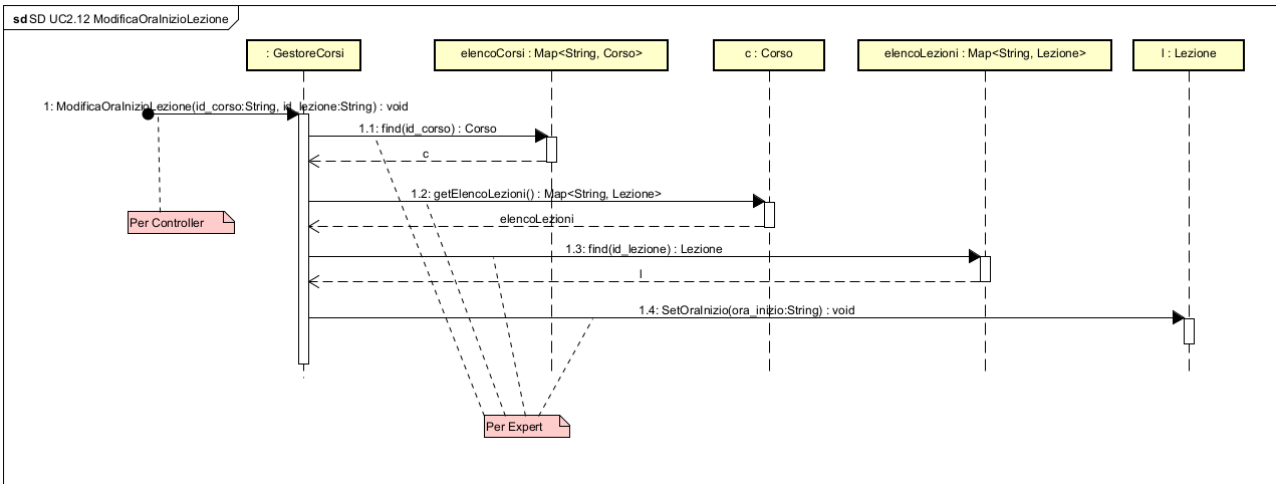
modificaNumeroPosti



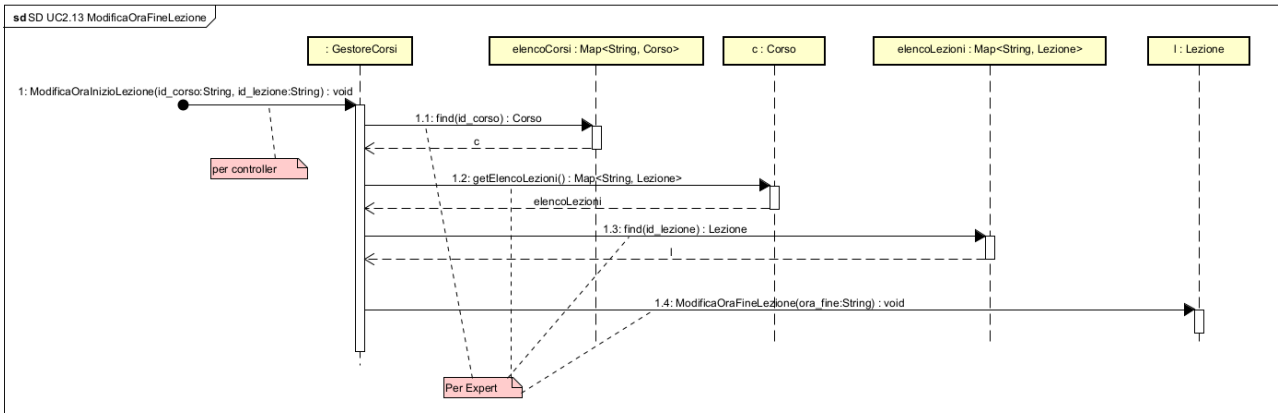
modificaPostiOccupati



modificaOraInizioLezione



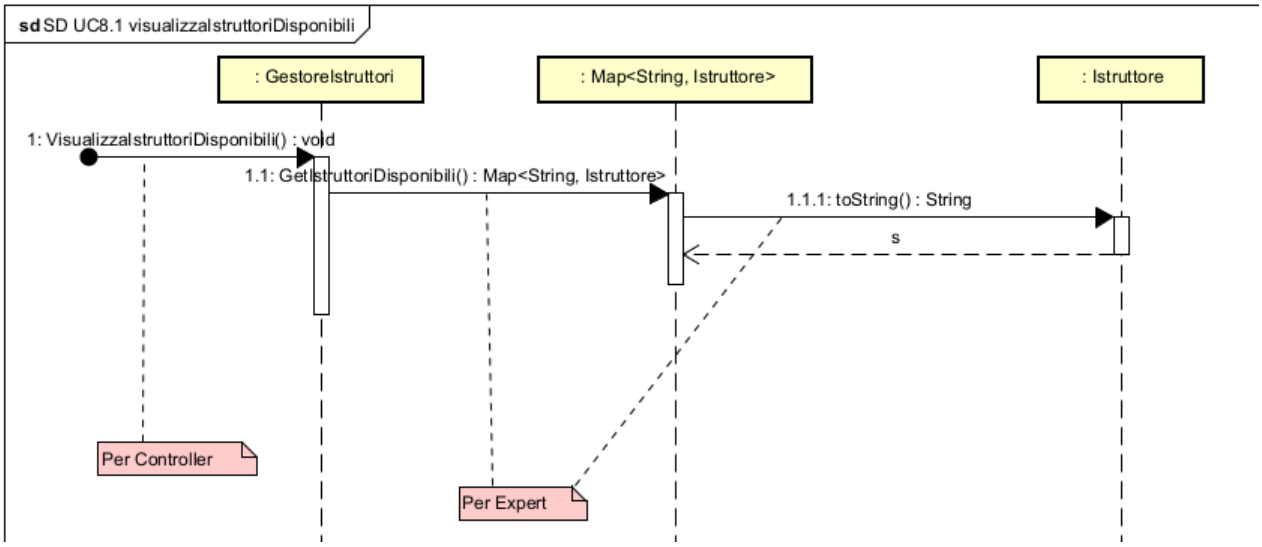
modificaOraFineLezione



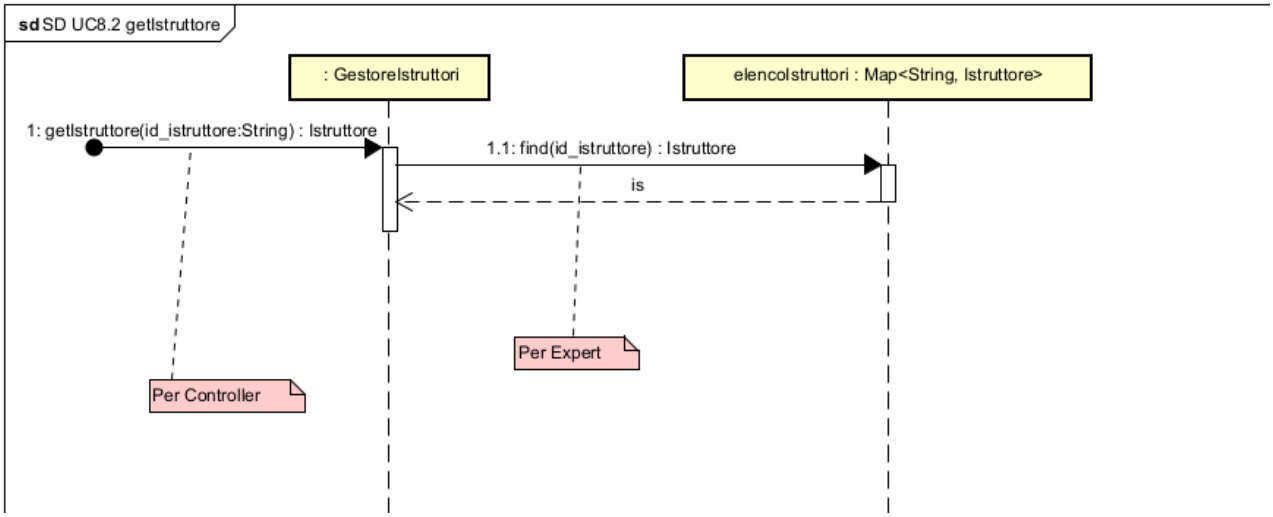
3.3 Diagrammi di Sequenza UC8

Si è fatto uso di due controller: un Controller GestoreIstruttori e un Controller GestoreCorsi per lavorare come interfaccia del sistema. GestoreIstruttori conosce la lista degli istruttori disponibili e quindi si occupa di prelevare la giusta istanza, GestoreCorsi invece si occupa di prelevare la giusta istanza di corso e di creare l'associazione tra le due.

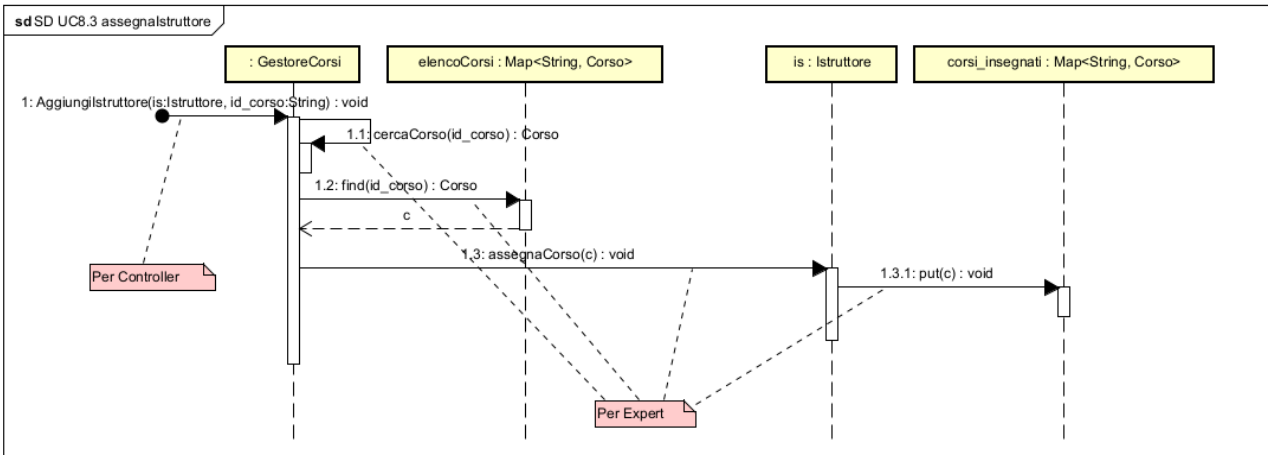
visualizzaIstruttoriDisponibili



getIstruttore



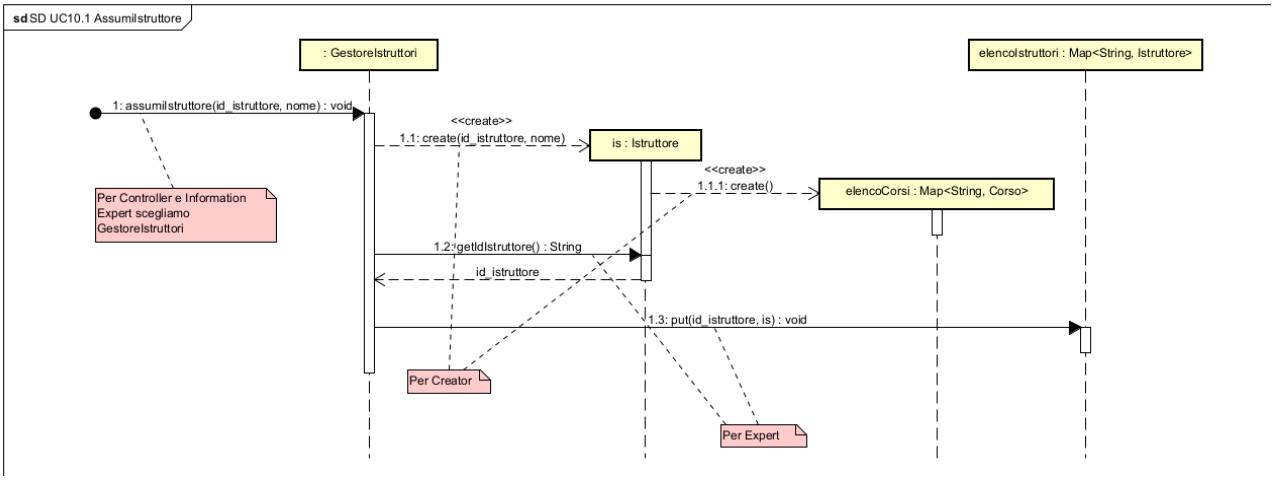
assegnaIstruttore



3.4 Diagrammi di Sequenza UC10

Si è utilizzato un Controller GestoreIstruttori per lavorare come interfaccia del sistema. Avrà il compito di creare le istanze di Istruttore.

assumiIstruttore

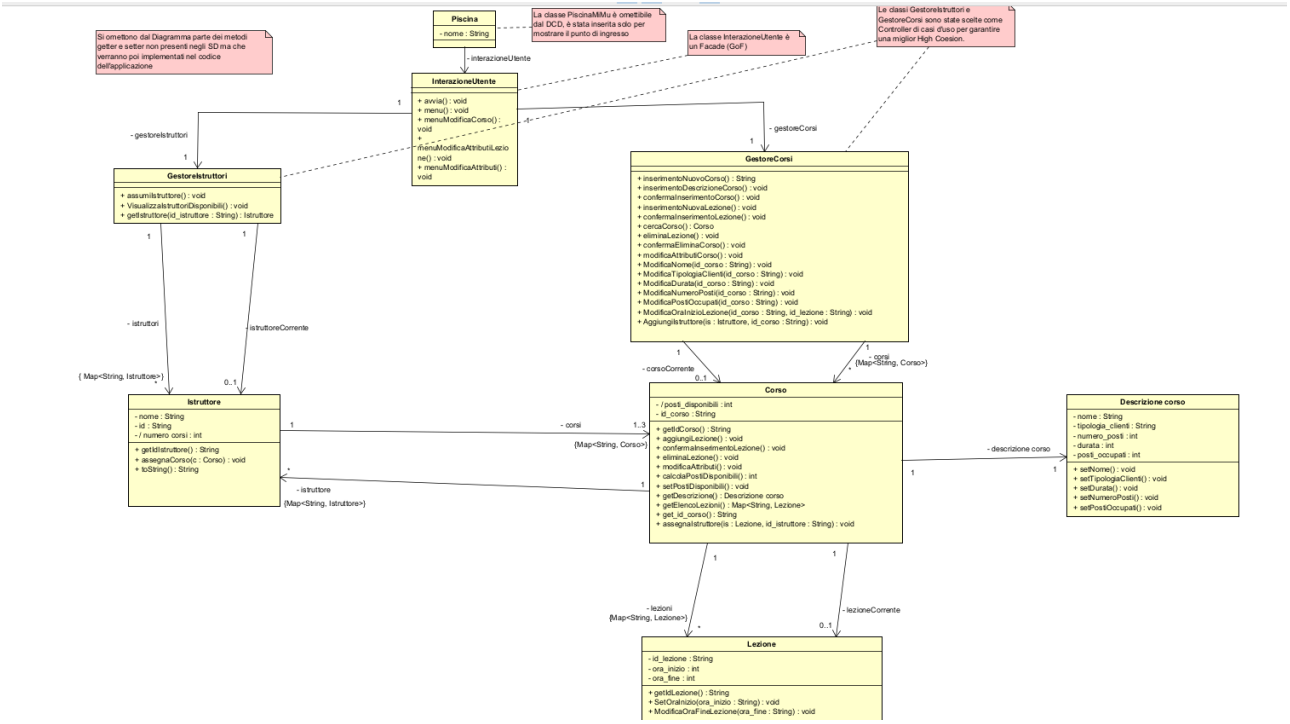


3.4 Diagramma delle Classi

La classe **InterazioneUtente** funge da **Facade (Pattern GoF)** del sistema, fornendo un punto di accesso unificato alle funzionalità principali. Le classi **GestoreCorsi** e **GestoreIstruttori** sono state modellate come **Controller**, ciascuna responsabile dei rispettivi casi d'uso, contribuendo a garantire una maggiore **High Cohesion** all'interno del sistema.

La classe **Piscina** è inclusa nel DCD principalmente per indicare il punto di ingresso del sistema, ma la sua presenza non è strettamente necessaria.

Di seguito il DCD:



5.0 Testing

5.1 Individuazione dei casi di test

In una fase preliminare al testing, si è svolta un'analisi del codice per individuare le classi e i metodi da testare. La strategia è stato quello di dare priorità ai metodi cruciali per il funzionamento del sistema relativamente ai casi d'uso implementati.

Di seguito sono elencati i principali casi di test individuati per le varie classi:

CorsoTest

1. **aggiungiLezione_ok**
   Verifica che sia possibile aggiungere correttamente una lezione a un corso e che questa sia effettivamente presente.
2. **aggiungiLezione_giaPresente**
   Controlla che provare ad aggiungere una lezione già presente lanci l'eccezione LezioneGiaPresenteException.
3. **aggiungiLezione_programmazionePiena**
   Verifica che non sia possibile aggiungere più lezioni del numero massimo consentito dal corso, generando ProgrammazionePienaException.
4. **eliminaLezione_ok**
   Controlla che una lezione possa essere rimossa correttamente e che non sia più presente nel corso.
5. **eliminaLezione_nonPresente**
   Verifica che tentare di eliminare una lezione inesistente lanci LezioneNonPresenteException.
6. **cercaLezione_nonPresente**
   Controlla che cercare una lezione inesistente generi LezioneNonPresenteException.
7. **toString_test**
   Verifica che il metodo toString() restituisca una rappresentazione corretta dello stato del corso, sia quando non ci sono lezioni sia quando sono presenti.

GestoreCorsiTest

1. **aggiungiCorso_ok**
   Verifica che un corso possa essere aggiunto correttamente e recuperato tramite cercaCorso.
2. **aggiungiCorso_giaPresente**
   Controlla che aggiungere un corso con lo stesso ID di uno esistente lanci CorsoGiaPresenteException.
3. **cercaCorso_nonPresente**
   Verifica che cercare un corso inesistente lanci CorsoNonPresenteException.
4. **eliminaCorso_ok**
   Controlla che un corso possa essere eliminato correttamente e che dopo l'eliminazione non sia più recuperabile.
5. **eliminaCorso_nonPresente**
   Verifica che tentare di eliminare un corso inesistente lanci CorsoNonPresenteException.
6. **aggiungiLezione_alCorsoEsistente**
   Controlla che una lezione possa essere aggiunta a un corso già presente e verificata tramite cercaLezione.

7. **aggiungiLezione_giaPresente**
   Verifica che aggiungere una lezione già esistente in un corso lanci LezioneGiaPresenteException.
8. **aggiungiLezione_programmazionePiena**
   Controlla che non si possano aggiungere più lezioni del numero massimo consentito dal corso, lanciando ProgrammazionePienaException.
9. **eliminaLezione_nonPresente**
   Verifica che tentare di eliminare una lezione inesistente in un corso lanci LezioneNonPresenteException.
10. **ModificaNome_ok**
    Controlla che il nome di un corso possa essere modificato correttamente.
11. **ModificaNome_not_ok**
    Verifica che tentare di modificare il nome di un corso inesistente lanci CorsoNonPresenteException.
12. **ModificaOraInizioLezione_ok**
    Controlla che l'ora di inizio di una lezione possa essere modificata correttamente.
13. **ModificaOraInizioLezione_corso_not_ok**
    Verifica che modificare l'ora di inizio di una lezione di un corso inesistente lanci CorsoNonPresenteException.
14. **ModificaOraInizioLezione_lezione_not_ok**
    Verifica che modificare l'ora di inizio di una lezione inesistente lanci LezioneNonPresenteException.
15. **ModificaOraInizioLezione_lezione_orari_not_ok**
    Controlla che modificare l'ora di inizio in un modo che crei orari non validi lanci LezioniConOrariNonValidiException.
16. **ModificaOraFineLezione_ok**
    Controlla che l'ora di fine di una lezione possa essere modificata correttamente.
17. **ModificaOraFineLezione_lezione_orari_not_ok**
    Verifica che modificare l'ora di fine creando orari non validi lanci LezioniConOrariNonValidiException.
18. **AssegnaCorso_ok**
    Verifica che un istruttore disponibile possa essere assegnato correttamente a un corso e che il corso venga inserito tra quelli insegnati.
19. **AssegnaCorso_nonDisponibile**
    Controlla che tentare di assegnare un corso a un istruttore che ha già raggiunto il numero massimo di corsi consentiti lanci IstruttoreNonDisponibile.

GestoreIstruttoriTest

1. **AssumiIstruttore_ok**
   Verifica che un istruttore possa essere assunto correttamente e aggiunto al gestore, controllando che sia recuperabile tramite getIstruttore.
2. **AssumiIstruttore_not_ok**
   Controlla che tentare di assumere un istruttore con lo stesso ID di uno già presente lanci IstruttoreGiaAssuntoException.
3. **stampaTutto_nonLanciaEccezioni**
   Verifica che il metodo StampaTutto() del gestore degli istruttori non generi eccezioni quando viene invocato.

4. **getIstruttore_not_ok**
   Verifica che tentare di recuperare un istruttore inesistente tramite getIstruttore lanci IstruttoreNonDisponibile.

5. **visualizzaIstruttoriDisponibili_ok**
   Controlla che VisualizzaIstruttoriDisponibili restituisca solo gli istruttori effettivamente disponibili e che quelli con tre o più corsi non vengano inclusi nella mappa.

IstruttoreTest

1. **isDisponibile_ok**
   Verifica che un istruttore senza corsi assegnati risulti disponibile.
2. **isDisponibile_not_ok**
   Controlla che un istruttore che ha già raggiunto il numero massimo di corsi assegnati non sia disponibile, lanciando IstruttoreNonDisponibile.
3. **toString_test**
   Verifica che il metodo toString() dell'istruttore restituisca una rappresentazione corretta contenente il nome e l'ID dell'istruttore.
4. **assegnaCorso_ok**
   Verifica che un corso venga assegnato correttamente a un istruttore disponibile e che possa essere recuperato tramite getCorsoInsegnato.

ITERAZIONE 2

1.0 Introduzione

La seconda iterazione si concentrerà sui seguenti aspetti:

- Implementazione di nuovi casi d'uso
- Utilizzo di Pattern GoF per raffinare il modello di dominio
- Refactoring del codice e del testing.

In particolare, in questa seconda iterazione andremo ad implementare:

- UC3: NUOVA VASCA
- UC4: ISCRIZIONE PISCINA
- UC6: ASSEGNAZIONE CORSIE AD UNA LEZIONE
- UC7: REPORT PISCINA
- UC11: MODIFICA VASCA

La scelta di questi casi d'uso è stata dettata dalla necessità di avere l'ultimo pilastro fondante del sistema, seppur possano sembrare molti si consideri che parte del report è stata implementata in codice nell'iterazione 1 come utility.

In questa iterazione ci si è resi conto che uc3 risultava essere troppo ampio per cui, in combinazione con il vecchio uc6, si è arrivati alla conclusione di trasformare questi due uc in tre uc: i nuoci uc3 e uc6 a cui si aggiunge uc11.

Il refactoring del codice ha portato ad avere delle modifiche a nomi dei metodi e aggiunta di alcuni attributi per alcune classi ai fini di essere più chiare e coerenti. Si rimanda al diagramma delle classi nella sua versione aggiornata.

1.1 Aggiornamento dei casi d'uso

Come detto durante l'introduzione, in questa fase sono stati specificati e rivisti dei dettagli dei casi d'uso implementati. Per visionare le modifiche apportate ai suddetti, si consulti il documento "Modello dei casi d'uso" riportato nella cartella in versione aggiornata per questa iterazione.

Eventuali ulteriori modifiche ai documenti saranno visionabili tramite la cronologia.

Fase di Analisi

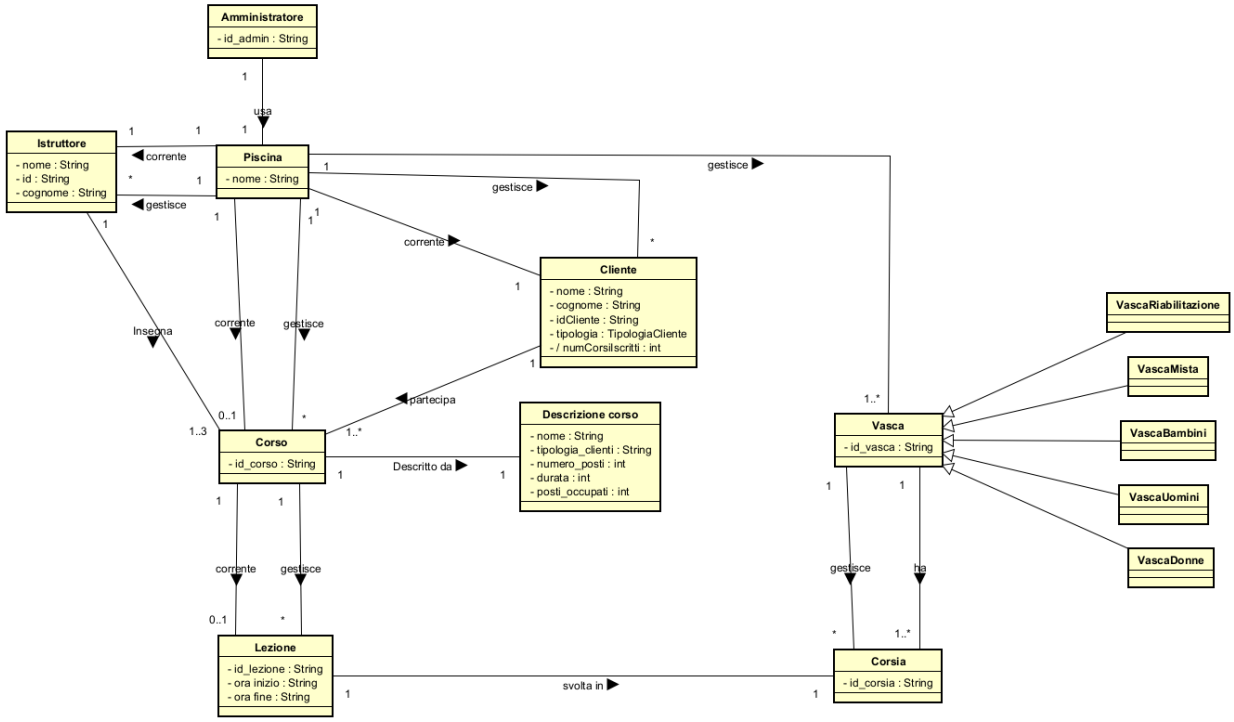
Le classi concettuali identificate durante questa seconda iterazione sono:

- Cliente
- Vasca
- Corsia

Inoltre, sono stati aggiornati alcuni parametri di classi concettuali già definite in precedenza:

- Aggiunto l'attributo cognome su Istruttore

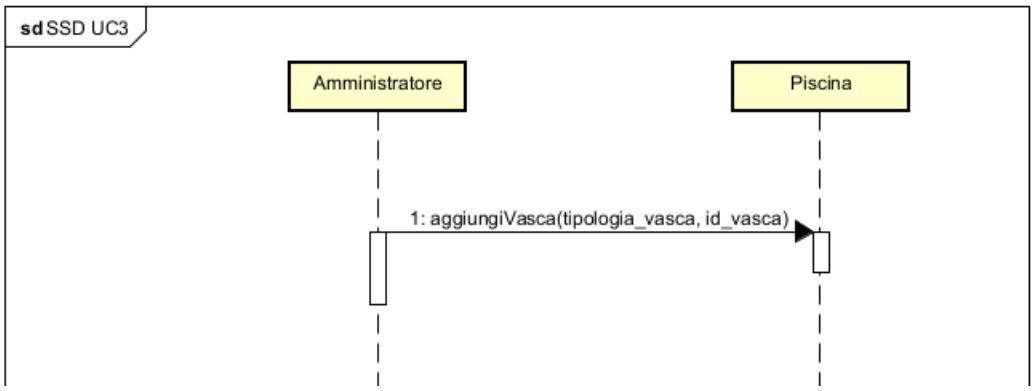
Si è arrivati al seguente Modello di Dominio:



2.1 Diagrammi di sequenza di sistema UC3

- UC3: INSERISCI NUOVA VASCA

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'amministratore e il sistema Piscina.



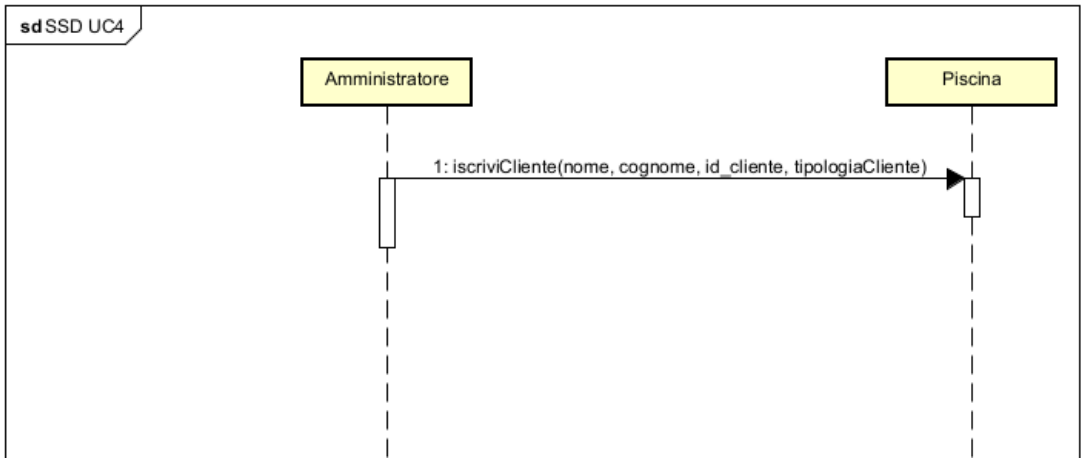
2.1.1 Contratti delle operazioni

Riferimento:	UC3
Operazione:	aggiungiVasca(tipologiaVasca, id_vasca)
Pre-condizioni:	• Nessuna
Post-condizioni:	• Viene creata un'istanza di vasca, in base alla tipologia di vasca passata come parametro

2.2 Diagrammi di sequenza di sistema UC4

- UC4: ISCRIZIONE PISCINA

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'amministratore e il sistema Piscina.



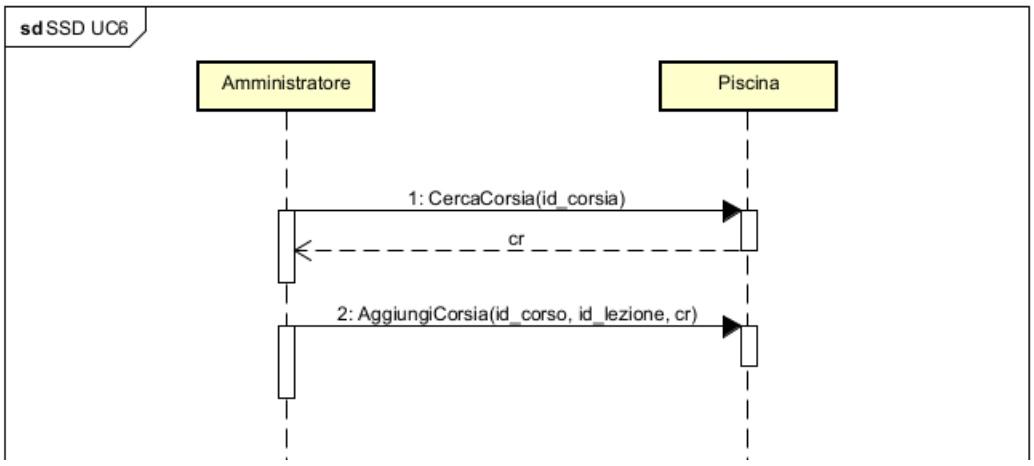
2.2.1 Contratti delle operazioni

Non sono stati scritti contratti per questo UC.

2.3Diagrammi di sequenza di sistema UC6

- UC6 ASSEGNAZIONE CORSIE AD UNA LEZIONE

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'amministratore e il sistema Piscina.



2.3.1 Contratti delle operazioni

Riferimento:	UC6.1
Operazione:	CercaCorsia(id_corsia)

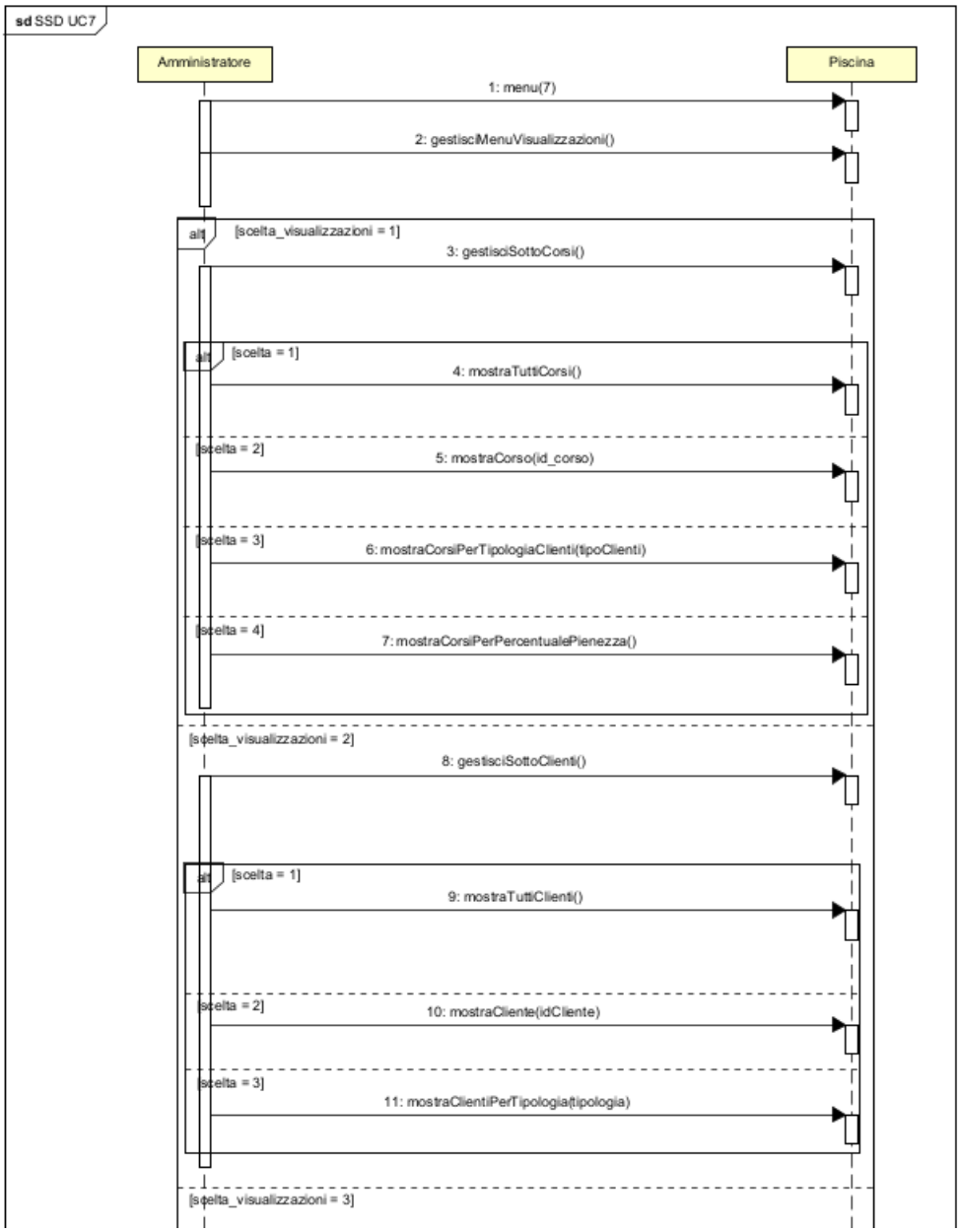
Pre-condizioni:	• Nessuna
Post-condizioni:	• Se esiste viene recuperata l'istanza cr di Corsia

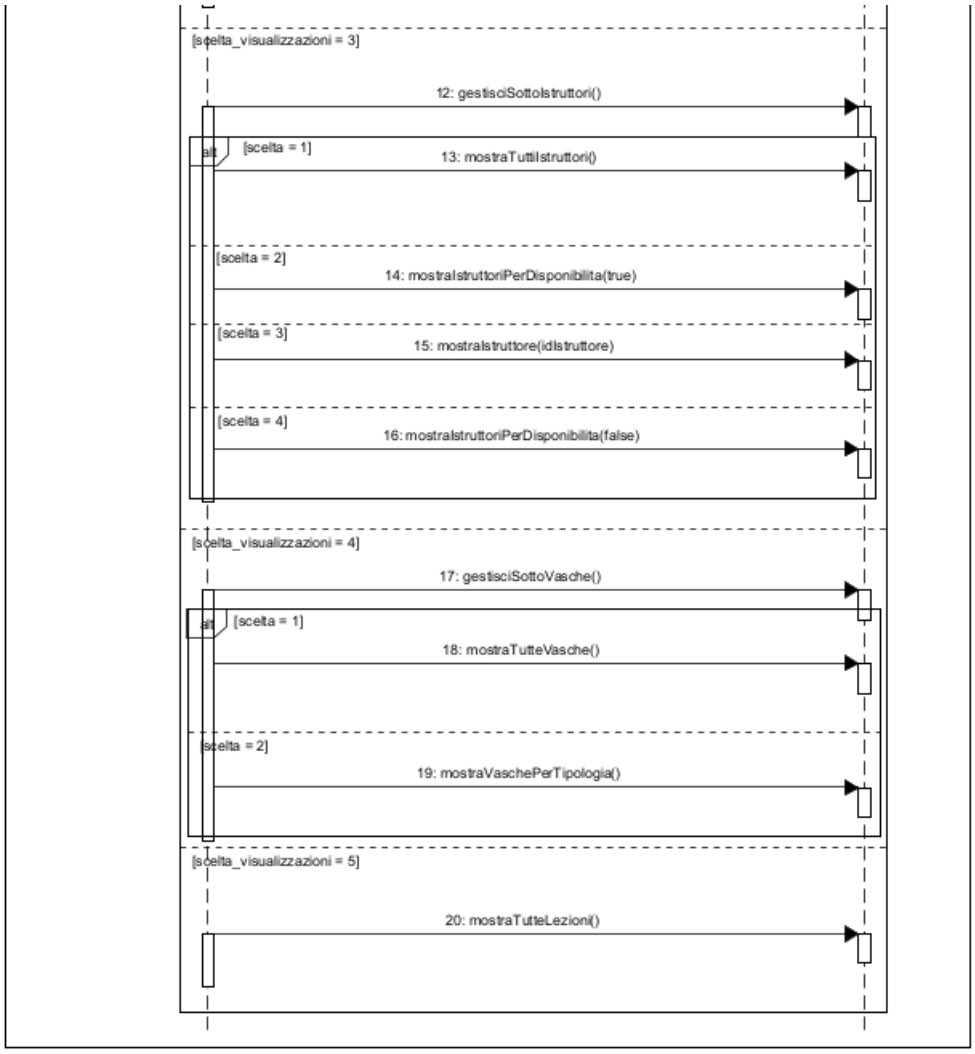
Riferimento:	UC6.2
Operazione:	AggiungiCorsia(id_corso, id_lezione, cr)
Pre-condizioni:	• Deve esistere un'istanza c di Corso • Deve esistere un'istanza l di Lezione associata all'istanza c di Corso • Deve esistere l'istanza cr di Corsia
Post-condizioni:	• Viene creata l'associazione tra l'istanza cr di Corsia e l'istanza l di Lezione

2.4 Diagrammi di sequenza di sistema UC7

- UC7: REPORT PISCINA

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'amministratore e il sistema Piscina.


sd SSD UC7
Amministratore
Piscina
1: menu(7)
2: gestisciMenuVisualizzazioni()
alt
[scelta_visualizzazioni = 1]
3: gestisciSottoCorsi()
alt
[scelta = 1]
4: mostraTuttiCorsi()
[scelta = 2]
5: mostraCorso(id_corso)
[scelta = 3]
6: mostraCorsiPerTipologiaClienti(tipoClienti)
[scelta = 4]
7: mostraCorsiPerPercentualePienezza()
[scelta_visualizzazioni = 2]
8: gestisciSottoClienti()
alt
[scelta = 1]
9: mostraTuttiClienti()
[scelta = 2]
10: mostraCliente(idCliente)
[scelta = 3]
11: mostraClientiPerTipologia(tipologia)
[scelta_visualizzazioni = 3]



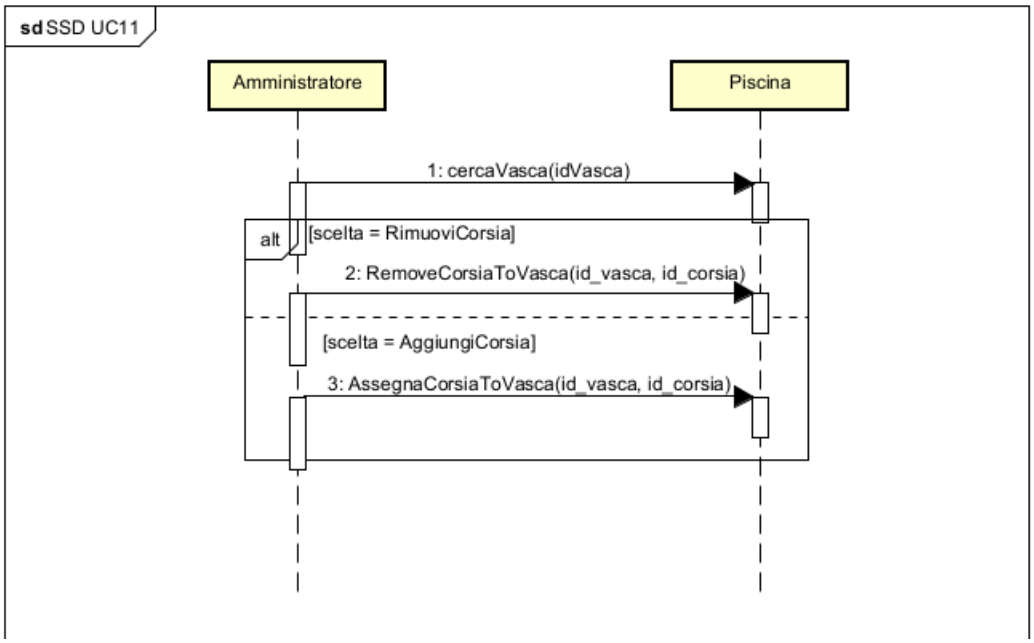
2.4.1 Contratti delle operazioni

Non sono stati scritti contratti per questo caso d'uso.

2.5 Diagrammi di sequenza di sistema UC11

- UC11: MODIFICA VASCA

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'amministratore e il sistema Piscina.



2.4.1 Contratti delle operazioni

Riferimento:	UC11.1
Operazione:	RemoveCorsiaToVasca(id_vasca, id_corsia)
Pre-condizioni:	• Deve esistere un'istanza v di Vasca • Deve esistere un'istanza c di Corsia
Post-condizioni:	• Vengono elminate tutte le associazioni tra l'istanza c di Corsia e le relative lezioni • Viene eliminata l'associazione tra l'istanza c di Corsia e v di Vasca • Viene eliminata l'istanza c di Corsia

Riferimento:	UC11.2
Operazione:	AssegnaCorsiaToVasca(id_vasca, cr)
Pre-condizioni:	• Deve esistere un'istanza v di Vasca
Post-condizioni:	• Viene creata l'istanza c di Corsia • Viene creata l'associazione tra l'istanza v di Vasca e l'istanza c di Corsia

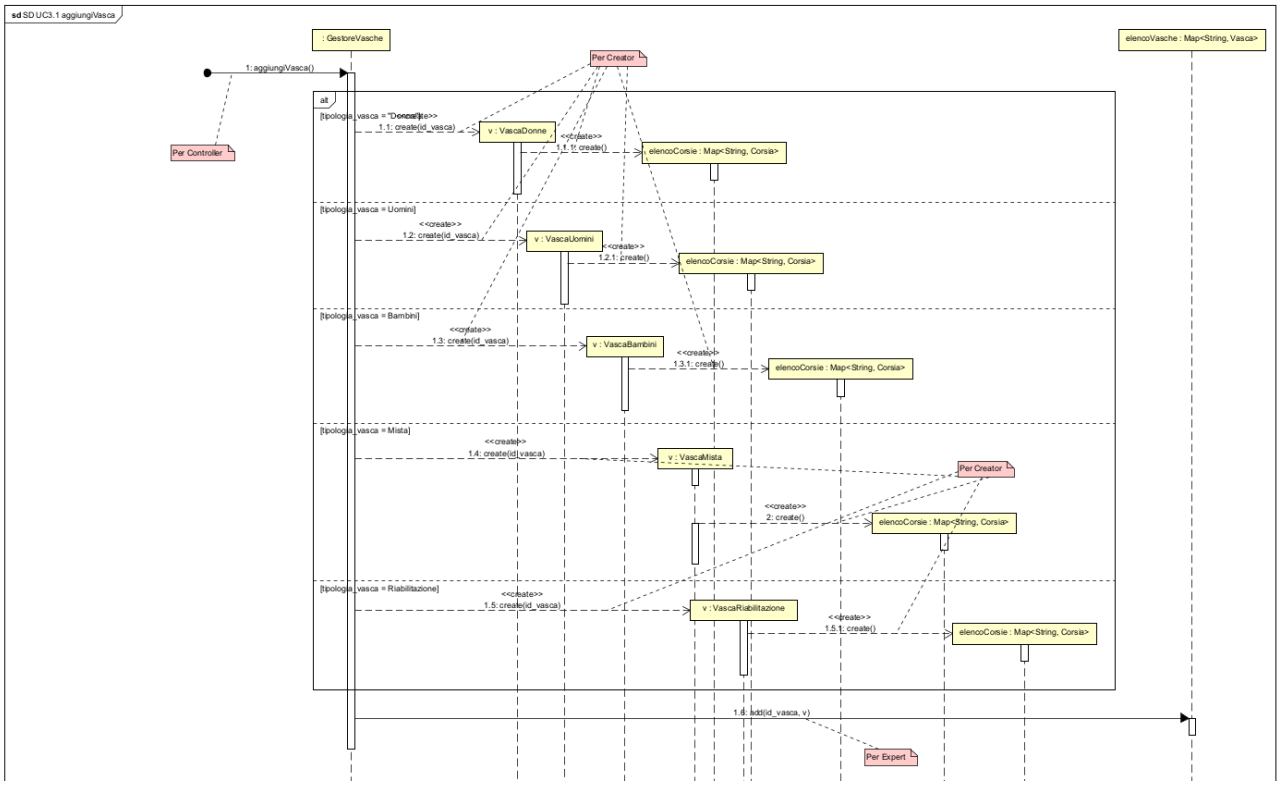
3.0 Fase di Progettazione

Gli elaborati principali della fase di progettazione sono i diagrammi di interazione, articolati in diagrammi di sequenza e diagrammi di interazione, con lo scopo di descrivere il comportamento del sistema da un punto di vista dinamico durante i casi d'uso presi in considerazione per questa seconda iterazione. Insieme a questi, il diagramma delle classi rappresenta il sistema da un punto di vista statico. Di seguito vengono riportati:

3.1 Diagrammi di Sequenza UC3

Si è utilizzato un Controller GestoreVasche per lavorare come interfaccia del sistema. Avrà il compito di creare le istanze di Vasca, le sue generalizzazioni e Corsia.

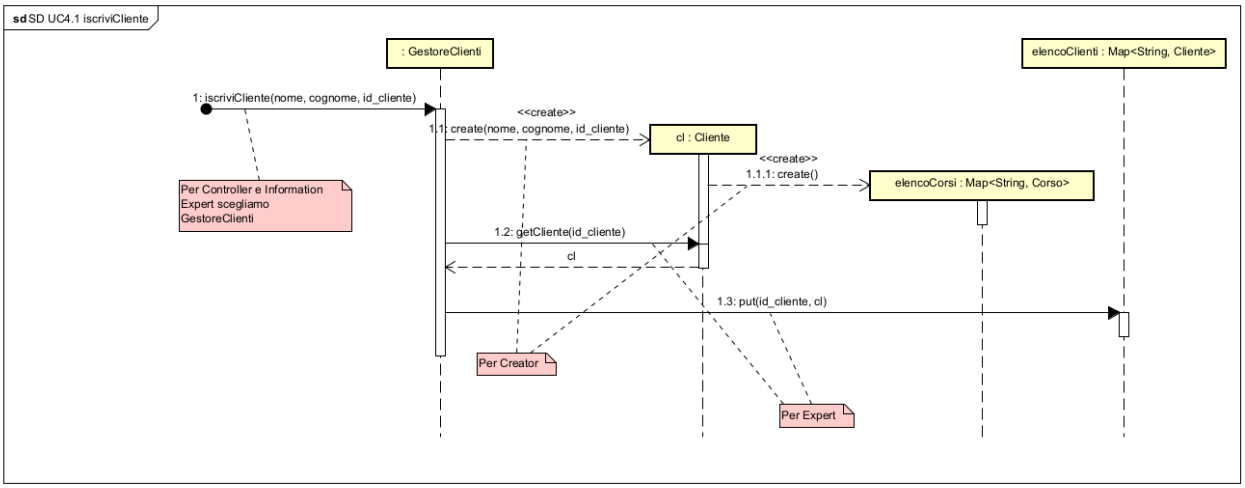
aggiungiVasca



3.2 Diagrammi di Sequenza UC4

Si è utilizzato un Controller GestoreClienti per lavorare come interfaccia del sistema. Avrà il compito di creare le istanze di Cliente.

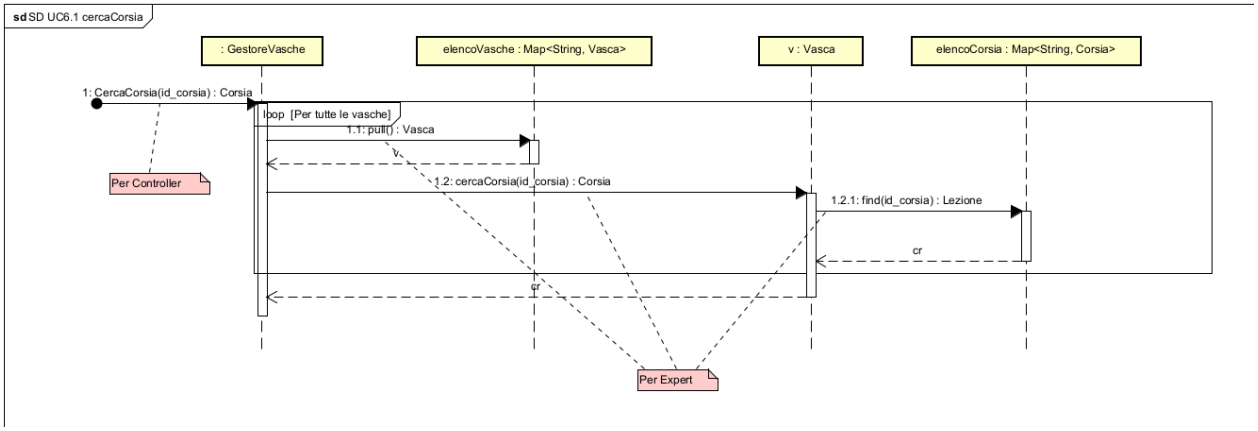
iscriviCliente



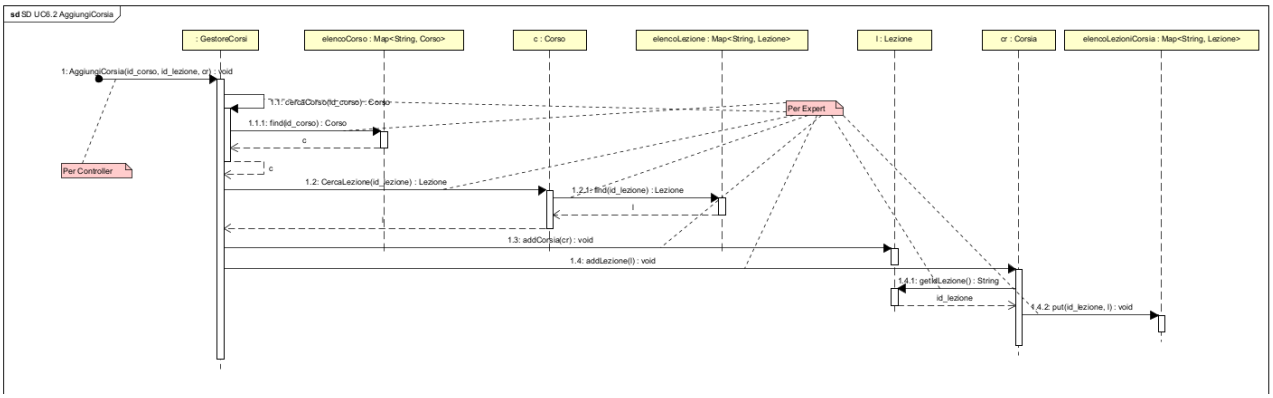
3.3 Diagrammi di Sequenza UC6

Si è fatto uso di un controller GestioneVasche per lavorare come interfaccia del sistema. Avrà il compito di creare o modificare l'associazione fra l'istanza di Corsia all'istanza di vasca.

AssegnaCorsiaToVasca



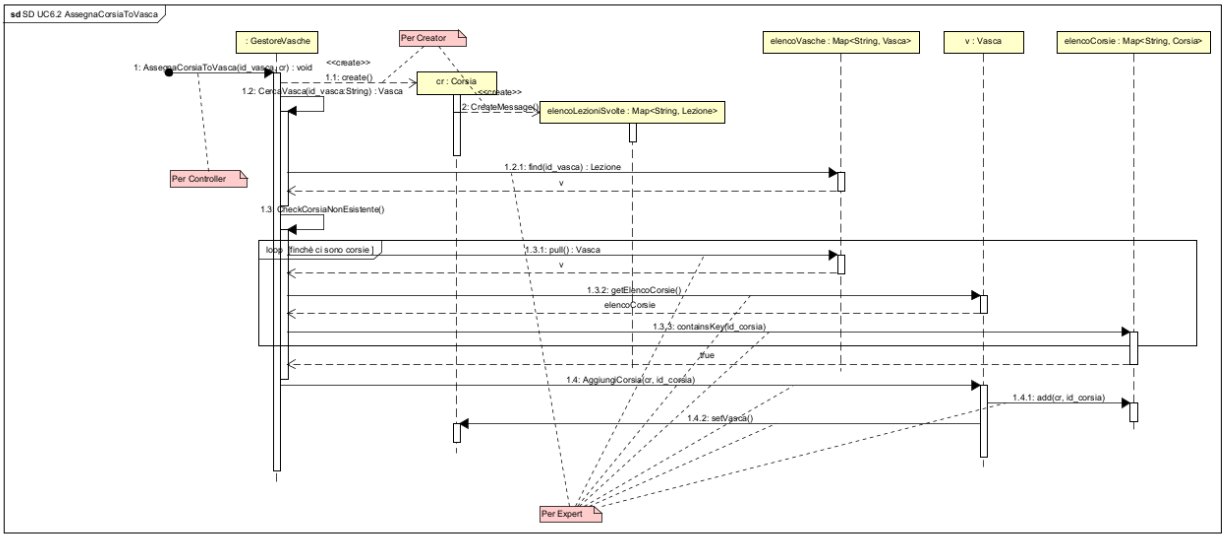
RemoveCorsiaToVasca



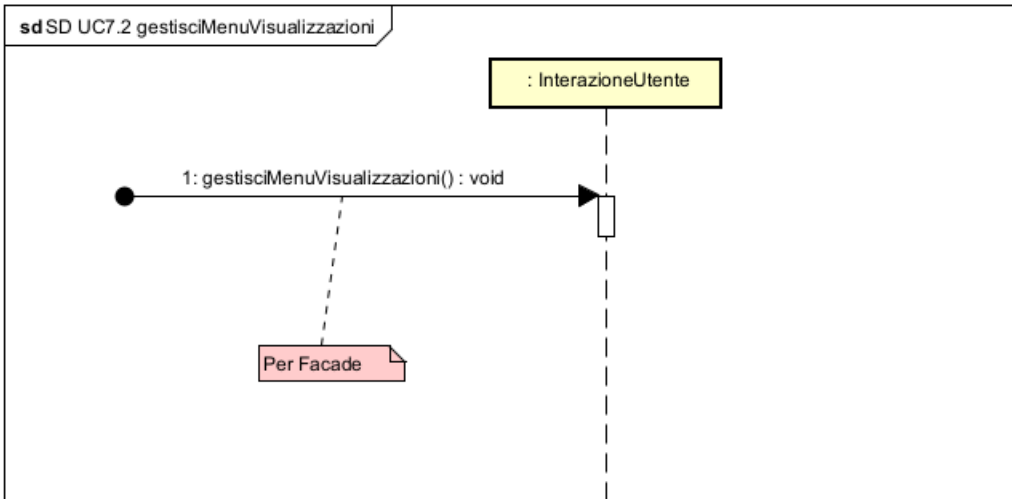
3.4 Diagrammi di Sequenza UC7

Si è utilizzato un il Facade InterazioneUtente insieme ai 4 controller: GestioneVasca, GestioneCorso, GestioneClienti, GestioneIstruttori per lavorare come interfaccia del sistema. Il Facade in combinazione con uno dei Controller fornirà la visualizzazione del report richiesto.

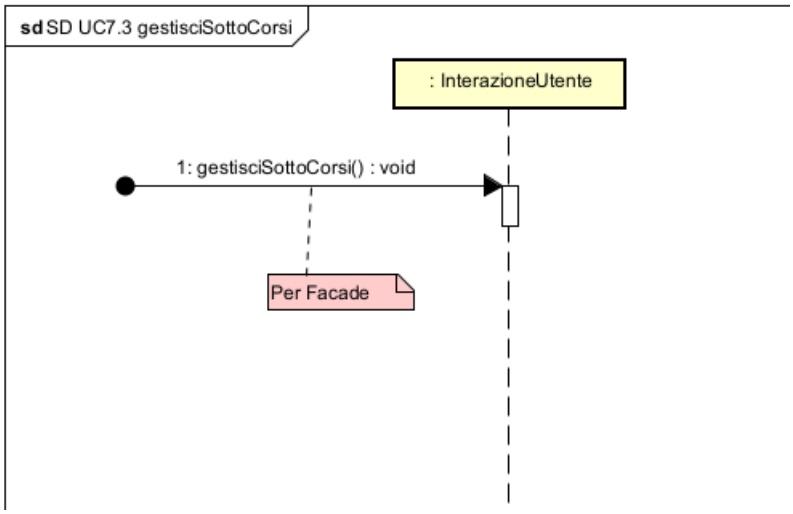
menu



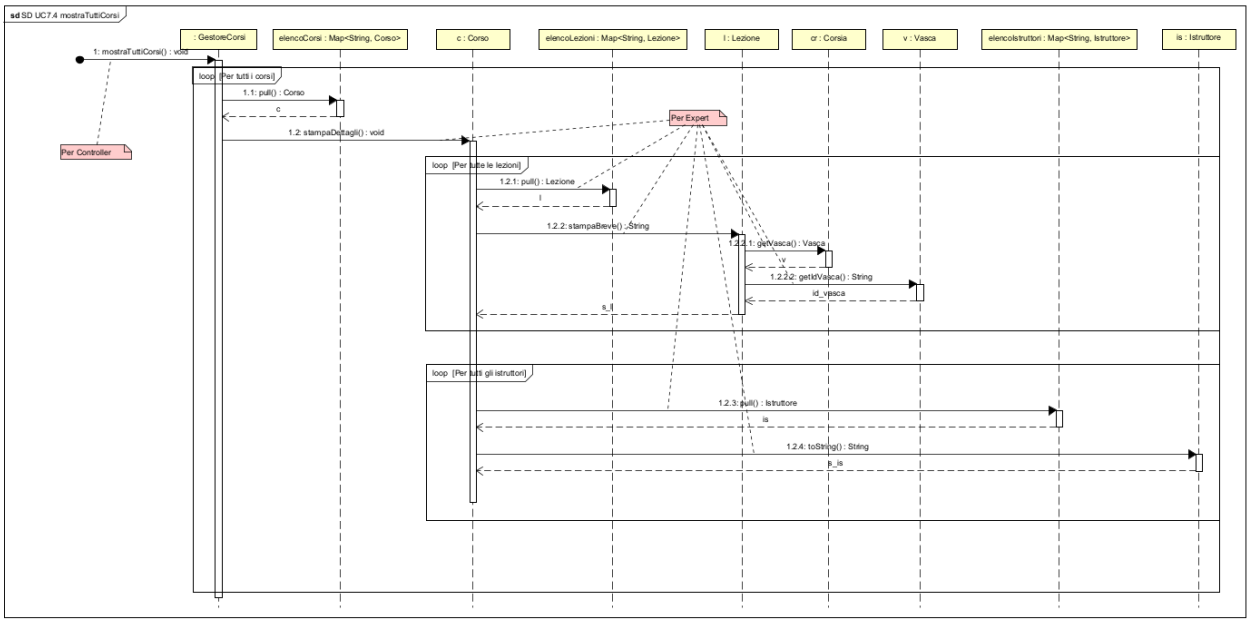
gestisciMenuVisualizzazioni



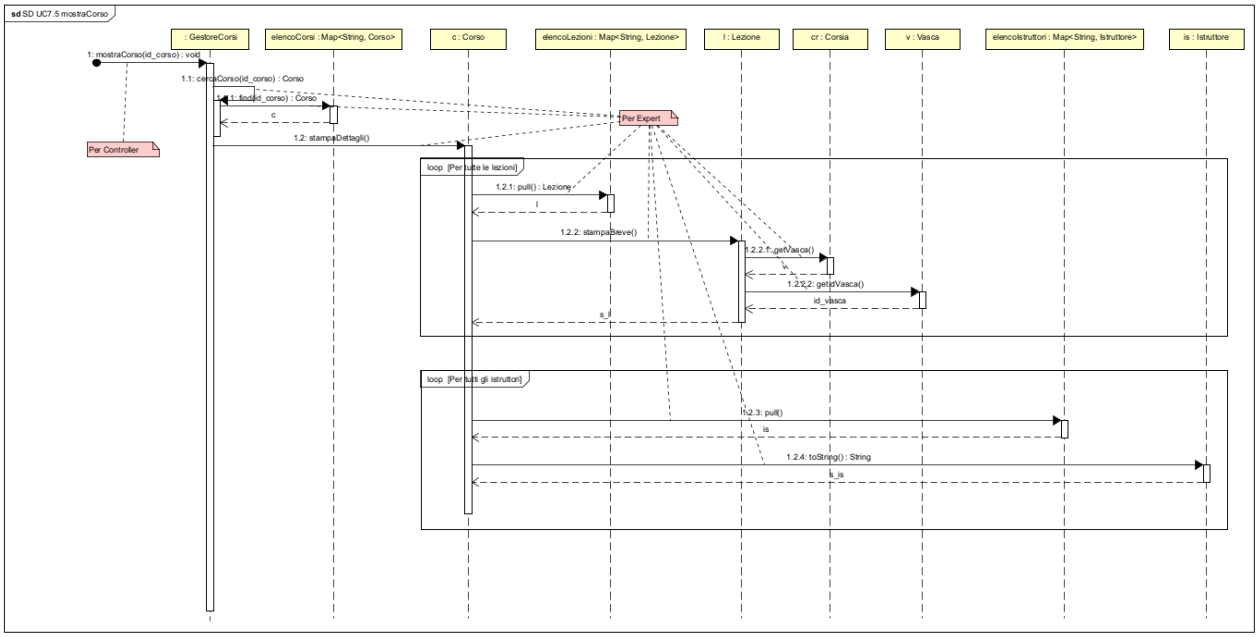
gestisciSottoCorsi



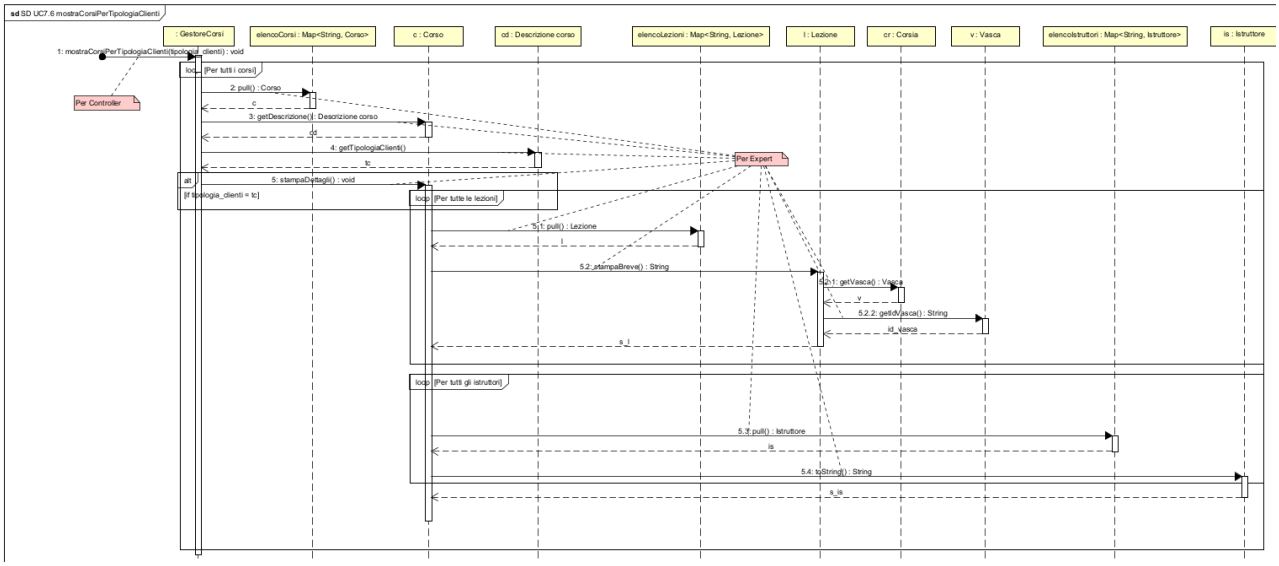
MostaTuttiCorsi



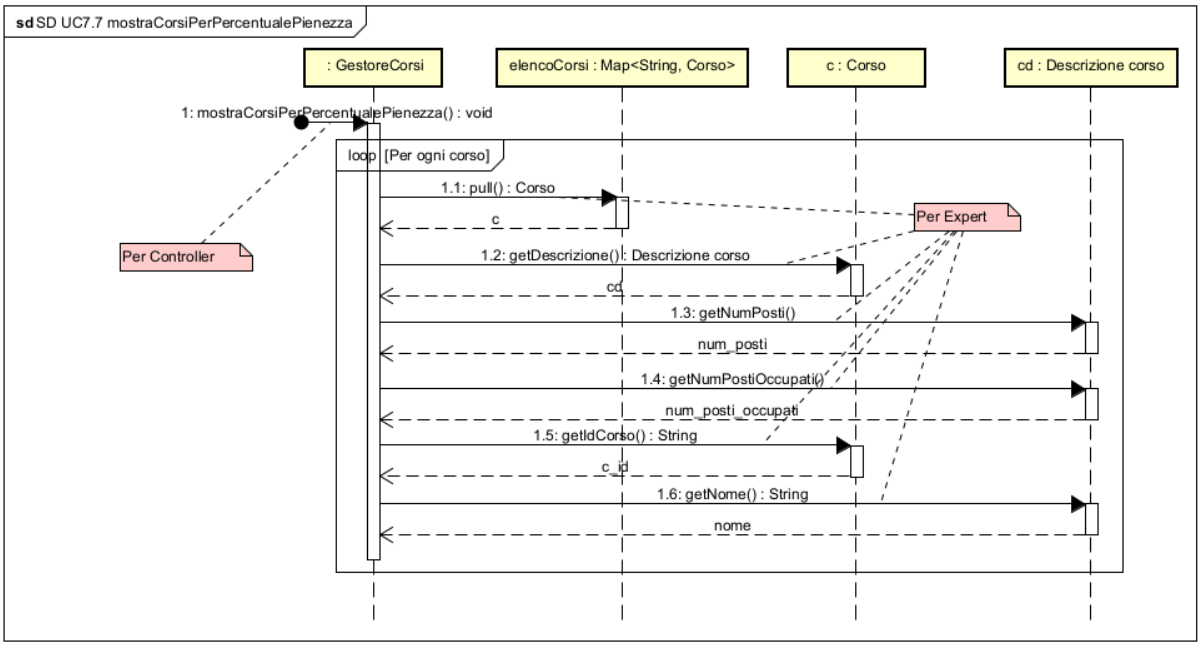
MostraCorso



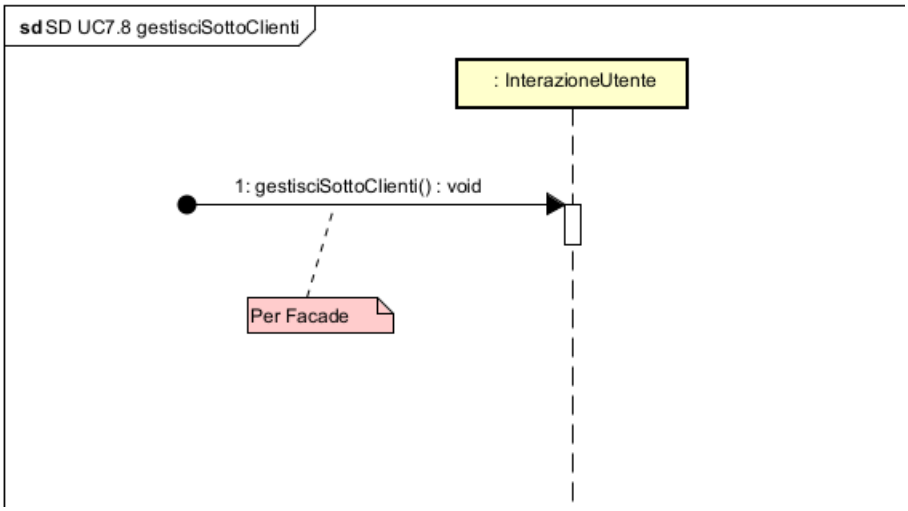
mostraCorsiPerTipologia



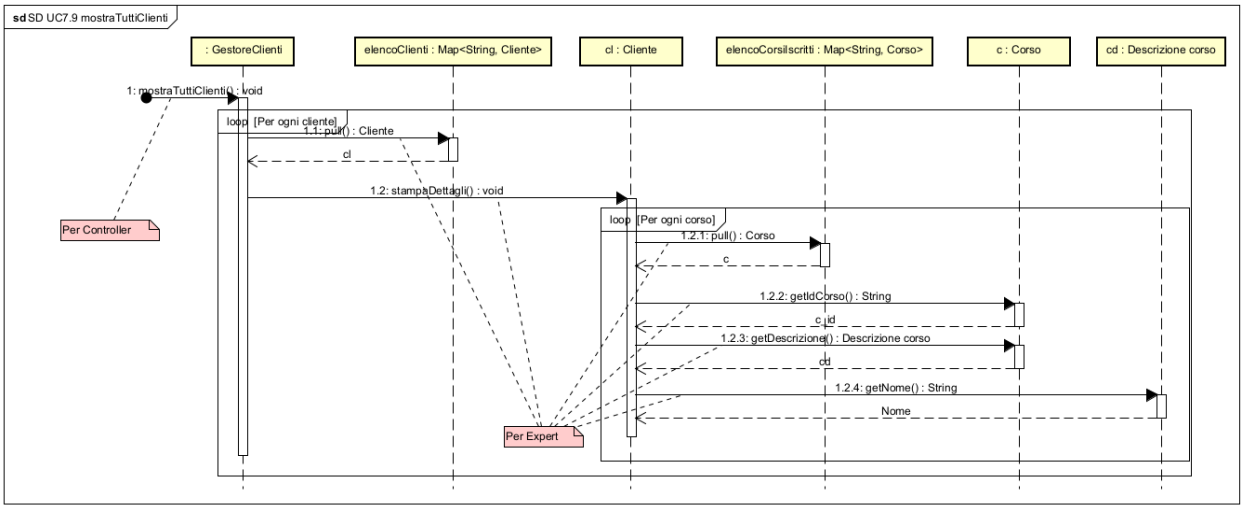
mostraCorsiPercentualePienezza



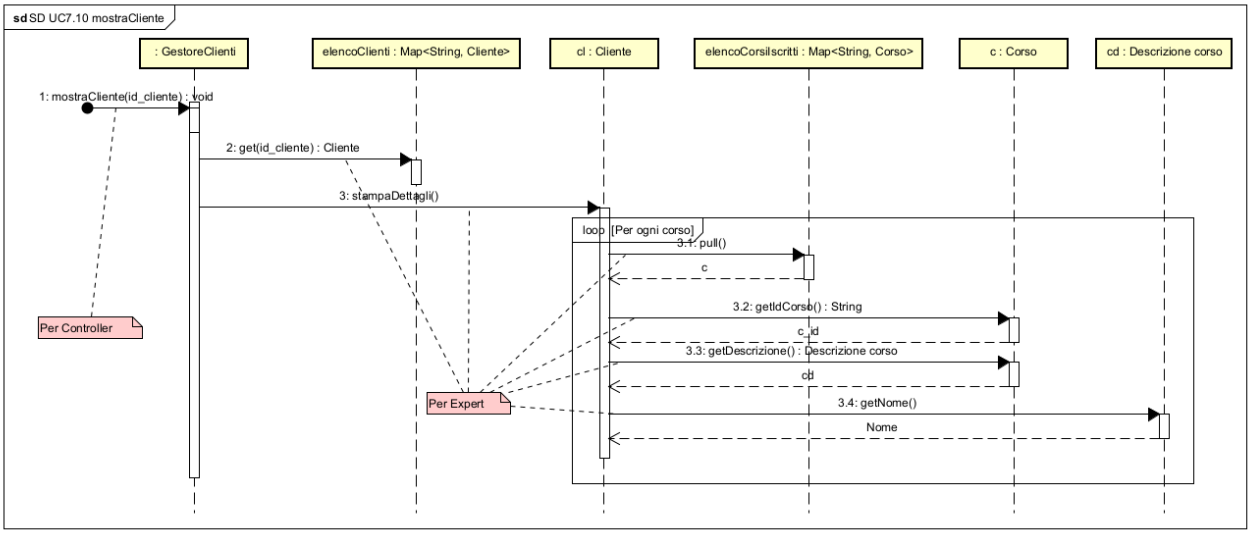
gestisciSottoClienti



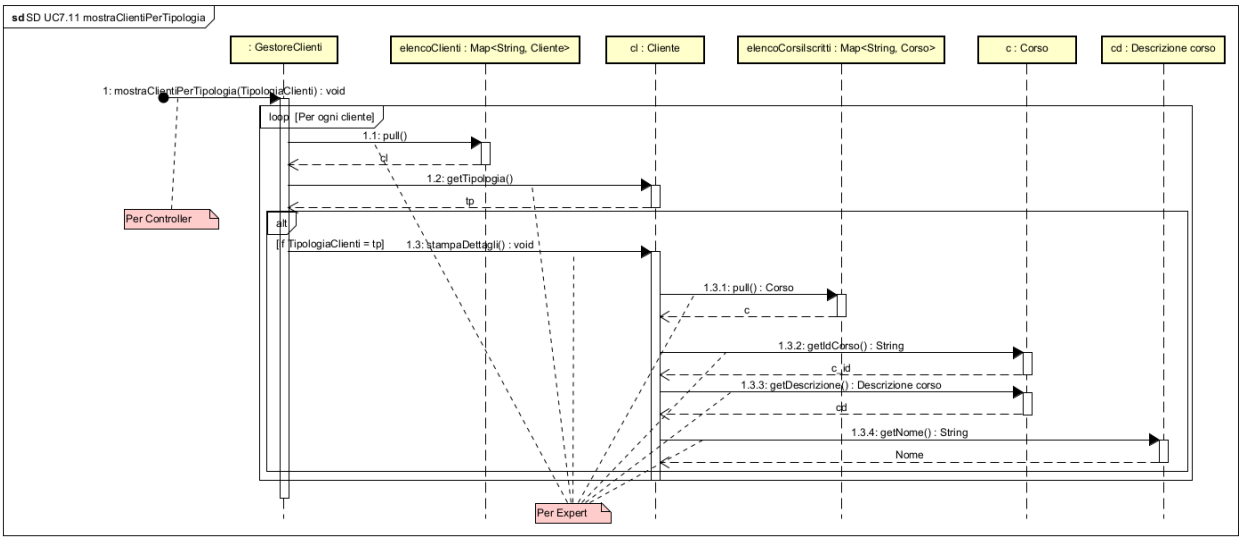
mostraTuttiClienti



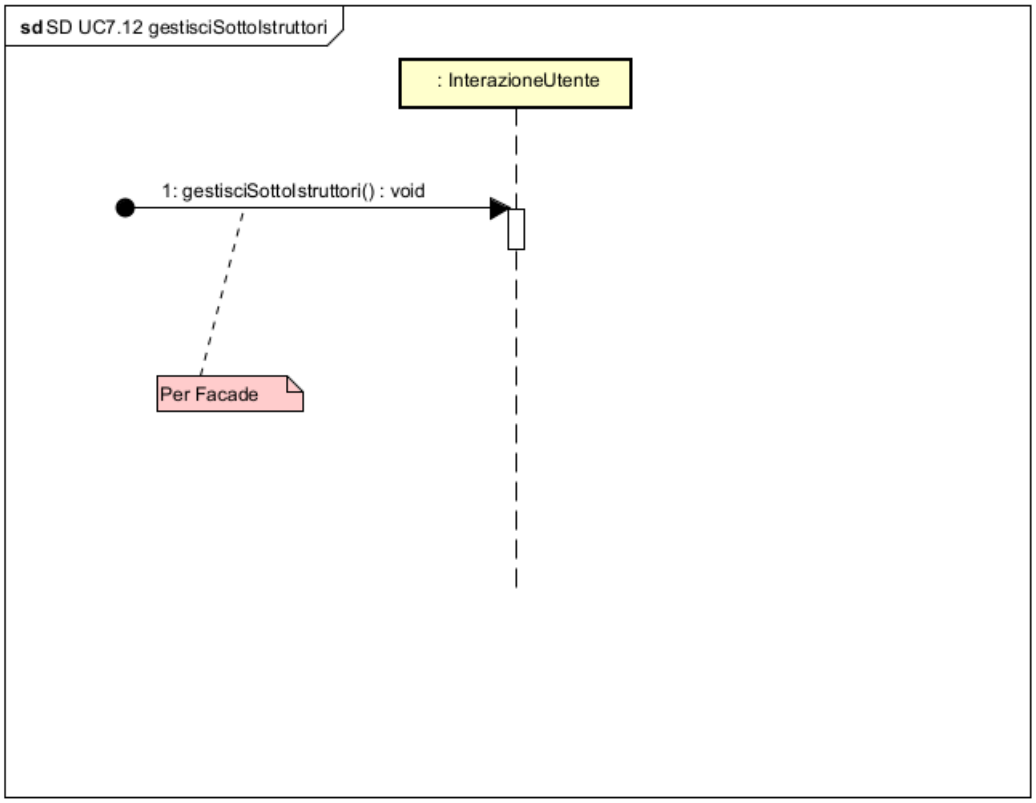
mostraCliente



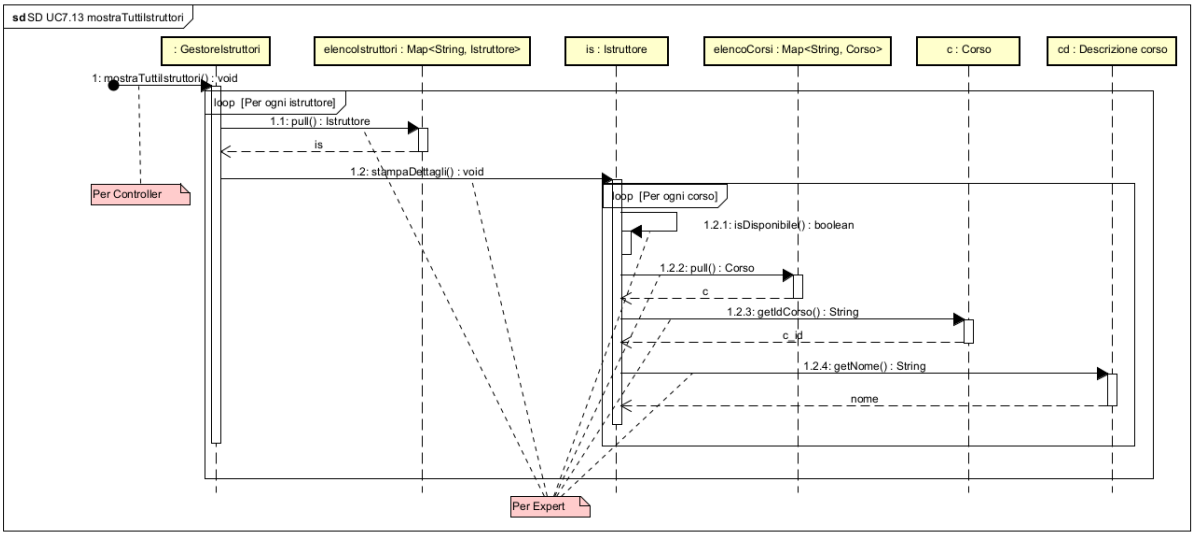
mostraClientiPerTipologia



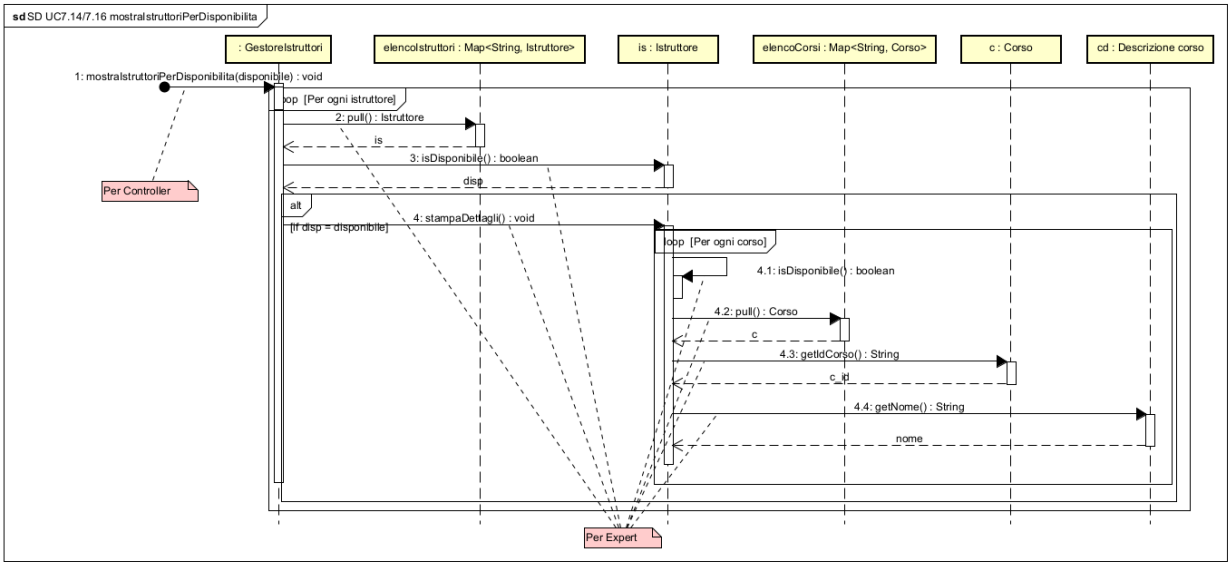
gestisciSottoIstruttori



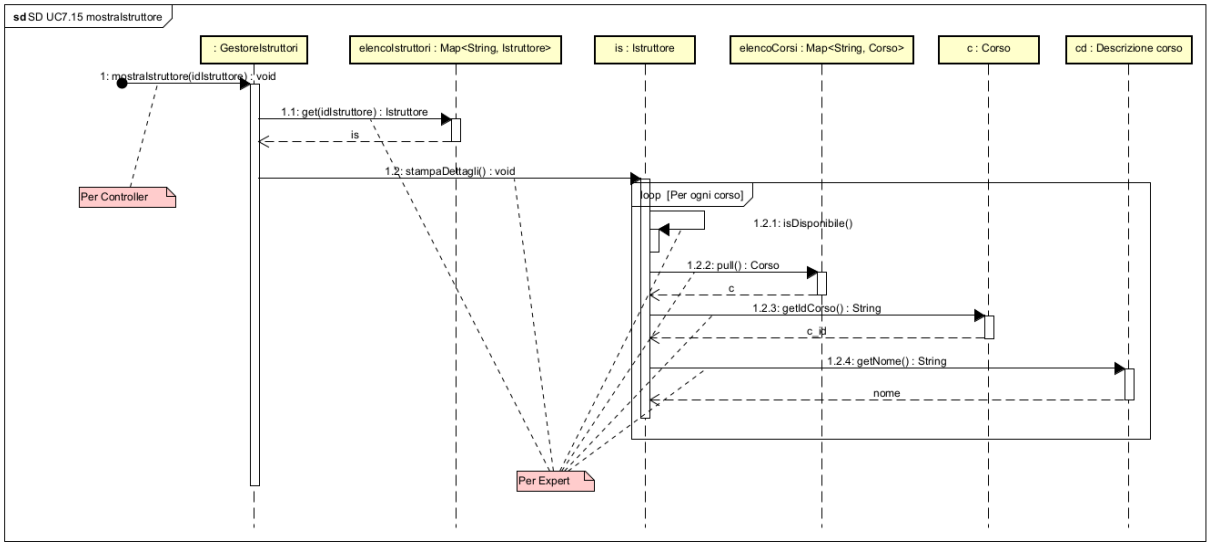
mostraTuttiIstruttori



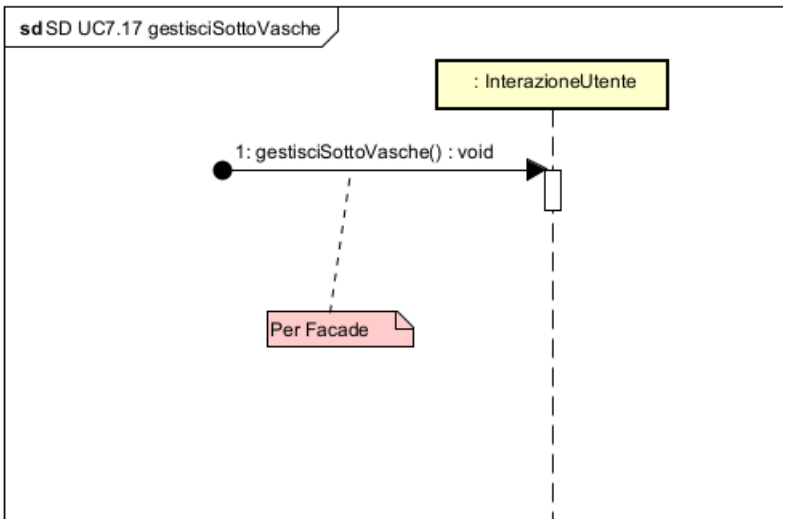
mostraIstruttoriPerDisponibilità



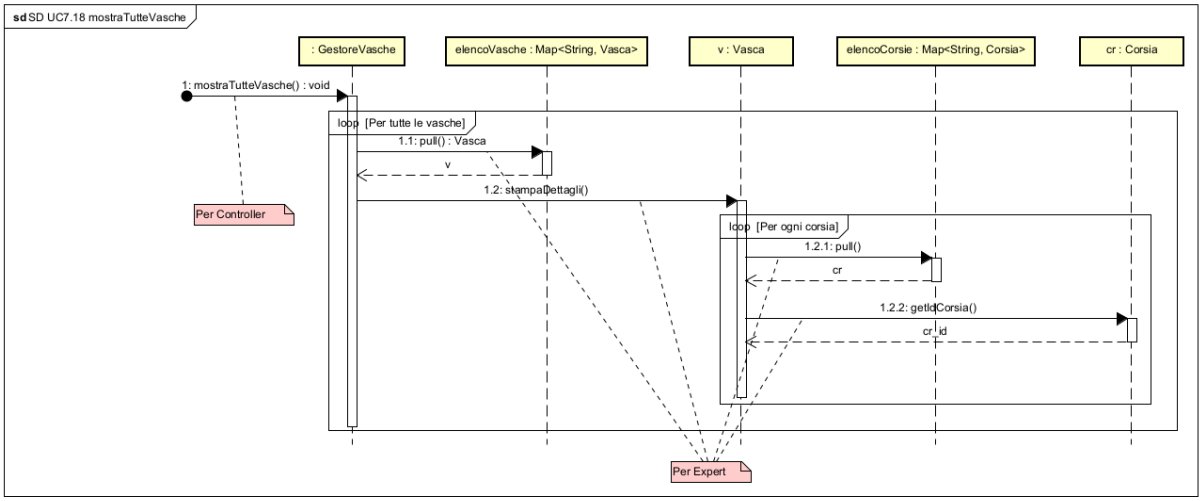
mostraIstruttore



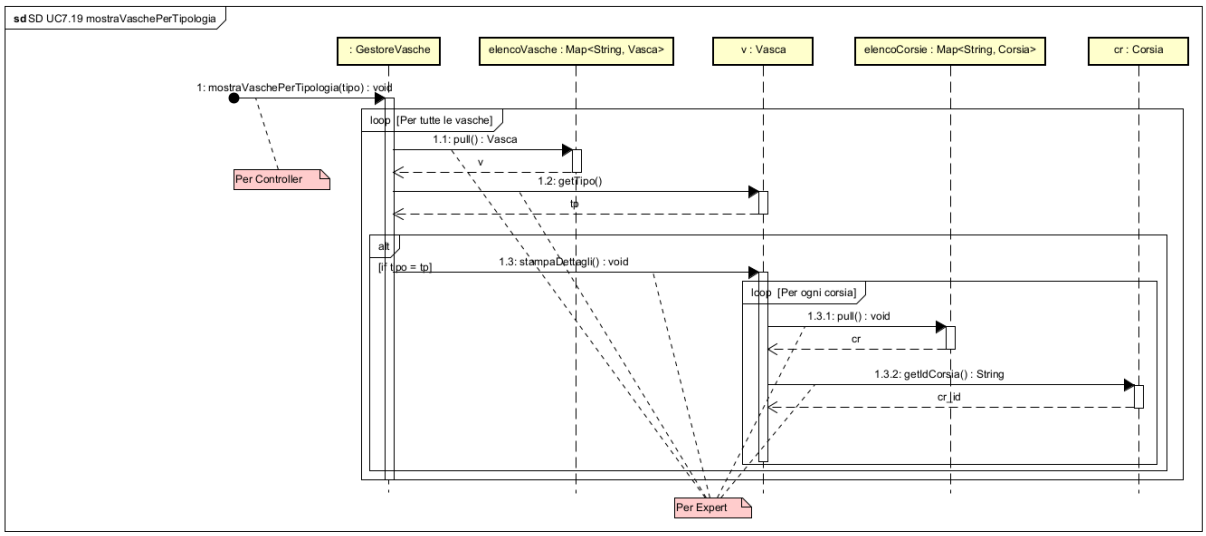
gestisciSottoVasche



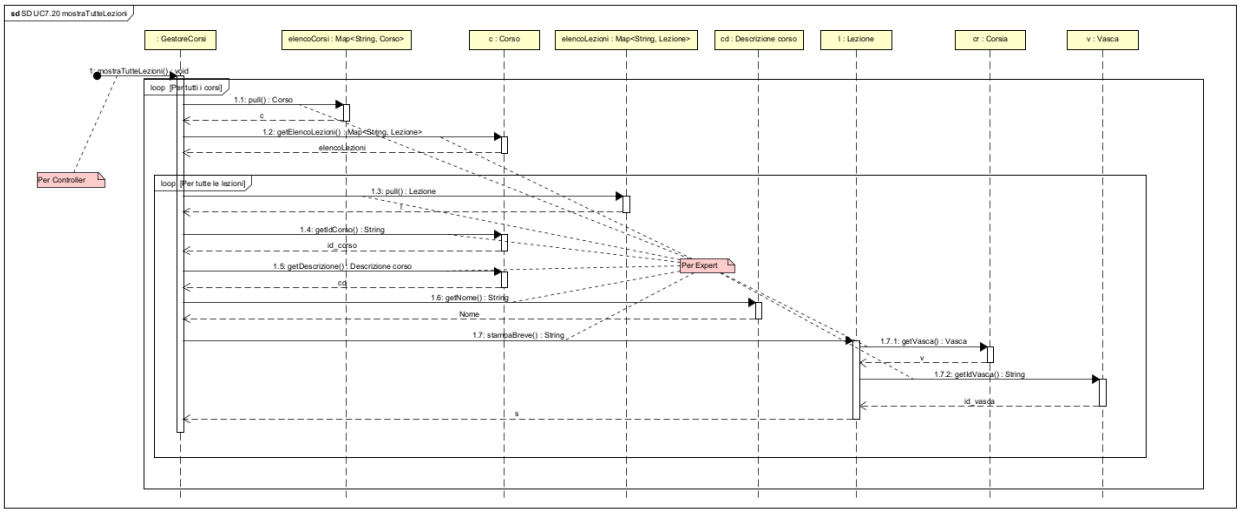
mostraTutteVasche



mostraVaschePerTipologia

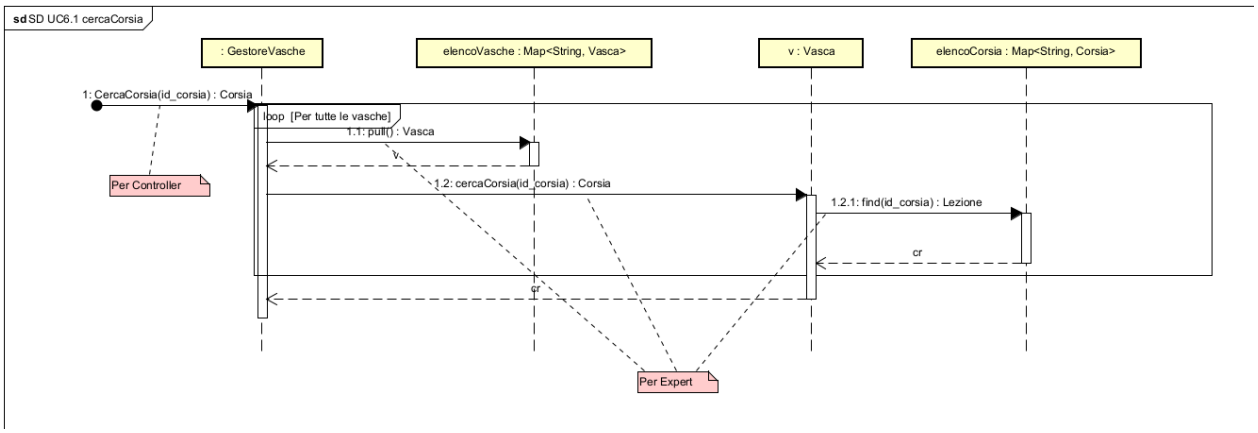


mostraTutteLezioni



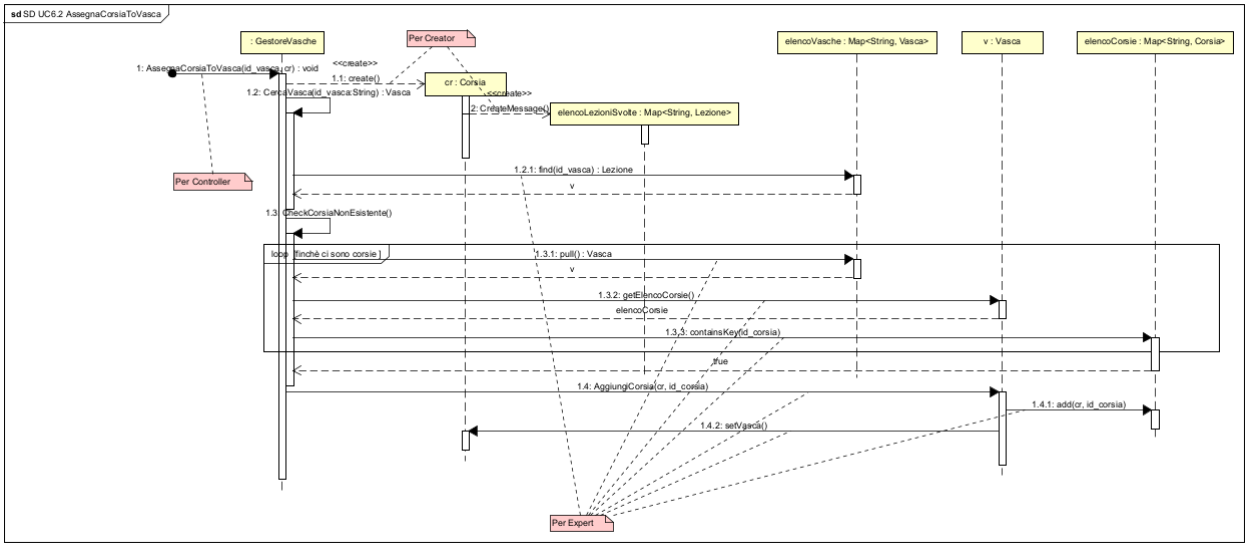
3.5 Diagrammi di Sequenza UC11

Si è fatto uso di un controller GestioneVasche per lavorare come interfaccia del sistema. Questo avrà il compito di creare istanze di Corsia da associare all'opportuna istanza di

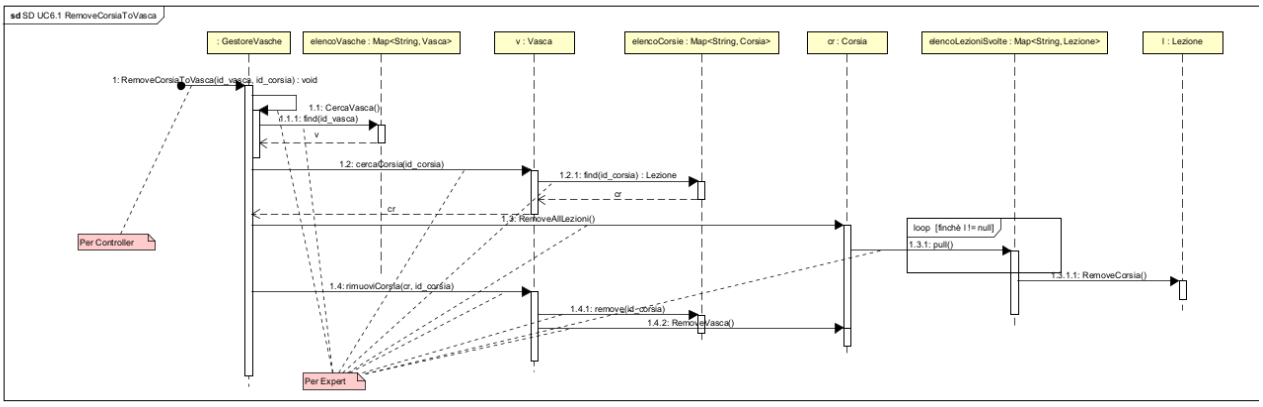


Vasca.

AssegnaCorsiaToVasca

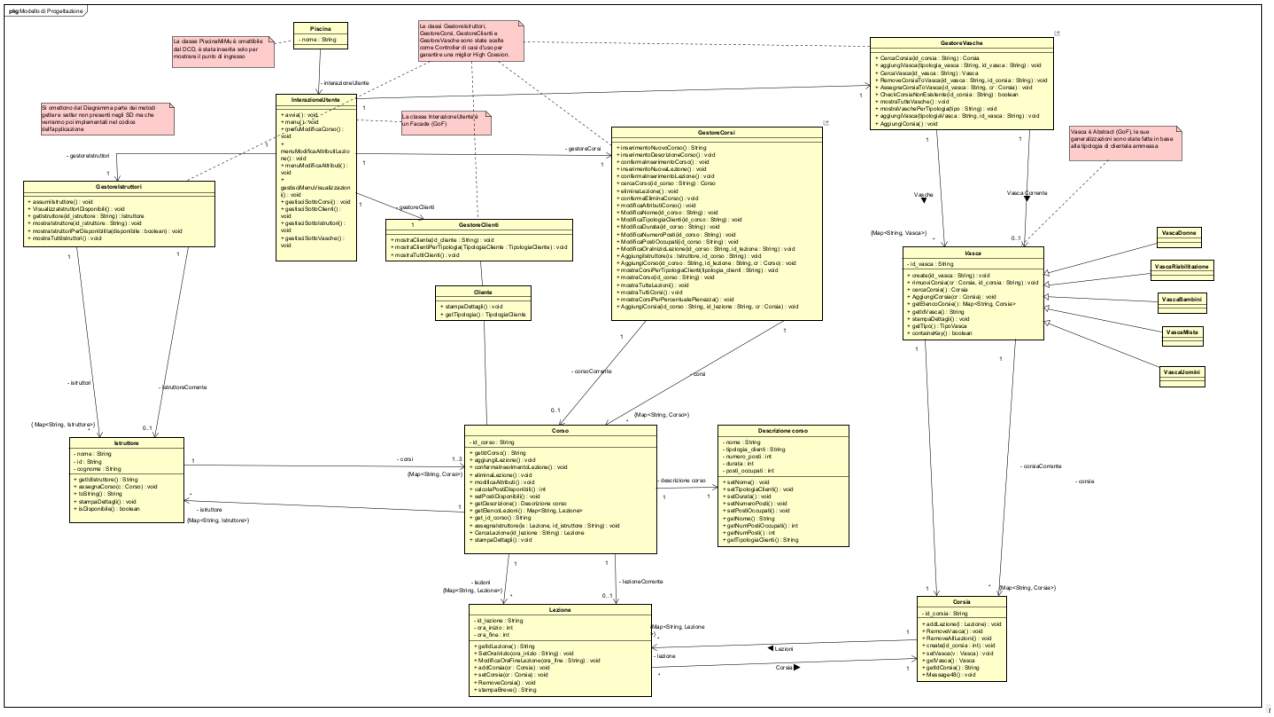


RemoveCorsiaToVasca



3.4 Diagramma delle Classi

Oltre all'evoluzione delle già presenti classi, in questa iterazione sono state aggiunte le classi di progettazione **GestoreClienti** e **GestoreVasche** sono state modellate come **Controller responsabili dei casi d'uso** e risultano fondamentali per garantire una miglior **High Cohesion** all'interno del sistema. Inoltre, è stato applicato il pattern GoF Abstract alla classe Vasca che ora presenta delle generalizzazioni basate sulla tipologia di clientela ammessa. Di seguito il DCD:



5.0 Testing

5.1 Individuazione dei casi di test

In una fase preliminare al testing, si è svolta un'analisi del codice per individuare le classi e i metodi da testare. La strategia è stato quello di dare priorità ai metodi cruciali per il funzionamento del sistema relativamente ai casi d'uso implementati e al fixing di test precedenti per il già citato refactoring.

Di seguito sono elencati i nuovi metodi di test:

ClienteTest

1. **testCostruttoreEGetters**
   Verifica che il costruttore inizializzi correttamente nome, cognome, ID e lista dei corsi, controllando che sia vuota.
2. **testNumCorsiIniziale**
   Controlla che il numero iniziale di corsi di un cliente appena creato sia zero.
3. **testAggiuntaCorso**
   Testa l'aggiunta di un corso alla mappa dei corsi iscritti, verificando presenza e conteggio corretto.
4. **testToStringContieneInfoCliente**
   Verifica che toString del cliente includa tutte le informazioni principali prima e dopo l'aggiunta di un corso.

CorsiaTest

1. **testCostruttoreEGetter**
   Verifica che il costruttore di Corsia inizializzi correttamente l'ID, la vasca sia nulla e l'elenco lezioni sia vuoto.

2. **testSetVascaERemoveVasca**
   Controlla l'assegnazione e la rimozione della vasca, assicurandosi che getVasca ritorni correttamente l'oggetto o null.
3. **testAddLezioneECercaLezione**
   Testa l'aggiunta di lezioni e la ricerca tramite ID, verificando anche il lancio dell'eccezione per lezioni inesistenti.
4. **testRemoveLezione**
   Verifica la rimozione di una lezione e che le lezioni rimanenti restino accessibili, controllando le eccezioni per lezioni mancanti.
5. **testRemoveAllLezioni**
   Controlla che tutte le lezioni vengano rimosse correttamente dall'elenco delle lezioni.
6. **testToString**
   Verifica che il metodo toString riporti correttamente lo stato della vasca sia quando non è assegnata sia quando è assegnata.

GestoreClientiTest

1. **testAggiungiClienteNuovo**
   Verifica che un nuovo cliente venga aggiunto correttamente e sia recuperabile tramite il suo ID.
2. **testAggiungiClienteDuplicato**
   Controlla che l'aggiunta di un cliente con ID già presente lanci l'eccezione ClienteGiaPresenteException.
3. **testGetClienteNonEsistente**
   Verifica che la ricerca di un cliente con ID inesistente ritorni null.
4. **testGetElencoClienti**
   Controlla che getElencoClienti restituisca tutti i clienti aggiunti e con la dimensione corretta.
5. **testMostraTuttiClienti**
   Testa che il metodo mostraTuttiClienti venga eseguito senza errori (stampa dell'elenco clienti).
6. **testMostraClientiPerTipologiaConClienti**
   Verifica che il metodo mostraClientiPerTipologia stampi correttamente i clienti della tipologia specificata quando esistono.
7. **testMostraClientiPerTipologiaSenzaClienti**
   Controlla che mostraClientiPerTipologia gestisca correttamente il caso in cui non ci siano clienti della tipologia richiesta.

GestoreCorsiTest

1. **testAggiungiCorsia_OK**
   Verifica l'aggiunta corretta di una Corsia a una lezione esistente di un corso, controllando l'assegnazione reciproca.
2. **testAggiungiCorsia_CorsoNonPresente**
   Controlla che venga lanciata l'eccezione CorsoNonPresenteException quando si prova ad aggiungere una corsia a un corso inesistente.
3. **testAggiungiCorsia_LezioneNonPresente**
   Verifica che l'aggiunta di una corsia a una lezione inesistente lanci LezioneNonPresenteException.
4. **testMostraTuttiCorsi**
   Assicura che il metodo mostraTuttiCorsi venga eseguito senza generare errori.
5. **testMostraCorso**
   Controlla che la visualizzazione di un corso esistente tramite mostraCorso non generi eccezioni.

6. **testMostraCorsiPerTipologiaClienti**
   Verifica che mostraCorsiPerTipologiaClienti funzioni senza errori per una tipologia di clienti specifica.
7. **testMostraTutteLezioni**
   Assicura che mostraTutteLezioni stampi correttamente tutte le lezioni dei corsi senza lanciare eccezioni.

GestoreIstruttoriTest

1. **mostraIstruttoriPerDisponibilita_conDisponibili**
   Verifica che la visualizzazione degli istruttori disponibili funzioni correttamente quando ci sono istruttori con disponibilità.
2. **mostraIstruttoriPerDisponibilita_senzaDisponibili**
   Controlla che il metodo gestisca correttamente il caso in cui non ci siano istruttori disponibili senza lanciare eccezioni.
3. **mostraIstruttore_esistente**
   Assicura che la visualizzazione di un istruttore esistente tramite ID funzioni senza errori.
4. **mostraIstruttore_nonEsistente**
   Verifica che la ricerca di un istruttore inesistente non generi eccezioni.

LezioneTest

1. **testCostruttoreEGetter**
   Verifica che il costruttore di Lezione inizializzi correttamente ID, orari e che la corsia sia inizialmente nulla.
2. **testAddCorsiaOK**
   Controlla che l'aggiunta di una corsia alla lezione funzioni correttamente e venga associata.
3. **testAddCorsiaDuplicata**
   Verifica che l'aggiunta della stessa corsia più volte lanci l'eccezione CorsiaGiaPresenteException.
4. **testCheckValidTimeStatic**
   Controlla il metodo statico checkValidTimeStatic per validare correttamente orari di inizio e fine lezione.
5. **testStampaBreve**
   Verifica che stampaBreve riporti correttamente informazioni sulla corsia e vasca, sia quando assegnate sia quando non assegnate.

VascaTest

1. **testCostruttoreEGetter**
   Verifica che il costruttore di Vasca inizializzi correttamente l'ID e che l'elenco delle corsie sia vuoto.
2. **testAggiungiCorsiaOK**
   Controlla che l'aggiunta di una corsia alla vasca funzioni correttamente e stabilisca il collegamento bidirezionale.
3. **testAggiungiCorsiaDuplicata**
   Verifica che l'aggiunta della stessa corsia due volte lanci l'eccezione CorsiaGiaPresenteException.
4. **testCercaCorsiaOK**
   Assicura che la ricerca di una corsia esistente nella vasca restituisca correttamente l'oggetto.

5. **testCercaCorsiaNonPresente**
   Verifica che la ricerca di una corsia inesistente lanci l'eccezione CorsiaNonPresenteNellaVascaException.
6. **testRimuoviCorsiaOK**
   Controlla che la rimozione di una corsia aggiorni sia l'elenco della vasca sia la referenza della corsia.
7. **testRimuoviCorsiaNonPresente**
   Verifica che la rimozione di una corsia non presente nella vasca lanci l'eccezione CorsiaNonPresenteNellaVascaException.
8. **testToString**
   Assicura che il metodo toString della vasca riporti correttamente l'ID e informazioni sulle corsie presenti.

GestioneVascheTest

1. **testAggiungiVascaECerca**
   Verifica che una vasca possa essere aggiunta e ricercata correttamente, controllando anche la gestione di vasche inesistenti.
2. **testAggiungiVascaDuplicata**
   Controlla che l'aggiunta di una vasca con ID già presente lanci VascaGiaPresenteException.
3. **testAssegnaCorsiaToVasca**
   Testa l'assegnazione di una corsia a una vasca e la corretta associazione bidirezionale tra corsia e vasca.
4. **testCheckCorsiaNonEsistente**
   Verifica che il metodo CheckCorsiaNonEsistente lanci l'eccezione CorsiaGiaEsistenteException quando la corsia è già presente.
5. **testRemoveCorsiaToVasca**
   Controlla che la rimozione di una corsia dalla vasca aggiorni sia l'elenco della vasca sia la referenza della corsia.
6. **testCercaCorsia**
   Assicura che la ricerca di una corsia nella gestione delle vasche restituisca correttamente l'oggetto o lanci CorsiaNonEsistenteException.
7. **testMostraVaschePerTipologia**
   Verifica che la visualizzazione delle vasche per tipologia funzioni correttamente, gestendo anche tipologie non esistenti senza errori.

ITERAZIONE 3

1.0 Introduzione

La terza iterazione si concentrerà sui seguenti aspetti:

- Implementazione di nuovi casi d'uso
- Refactoring del codice e del testing.

In particolare, in questa seconda iterazione andremo ad implementare:

- UC5: ISCRIZIONE CLIENTE A CORSO
- UC9: VALUTAZIONE PIENEZZA DI UN CORSO

Si tratta degli ultimi due casi d'uso del sistema.

Il refactoring del codice ha portato ad avere delle modifiche in alcuni metodi e in particolare al tipo dell'attributo tipologiaCliente in Descrizione Corso. Si rimanda al diagramma delle classi nella sua versione aggiornata.

1.1 Aggiornamento dei casi d'uso

Per visionare le modifiche apportate ai suddetti, si consulti il documento "Modello dei casi d'uso" riportato nella cartella in versione aggiornata per questa iterazione.

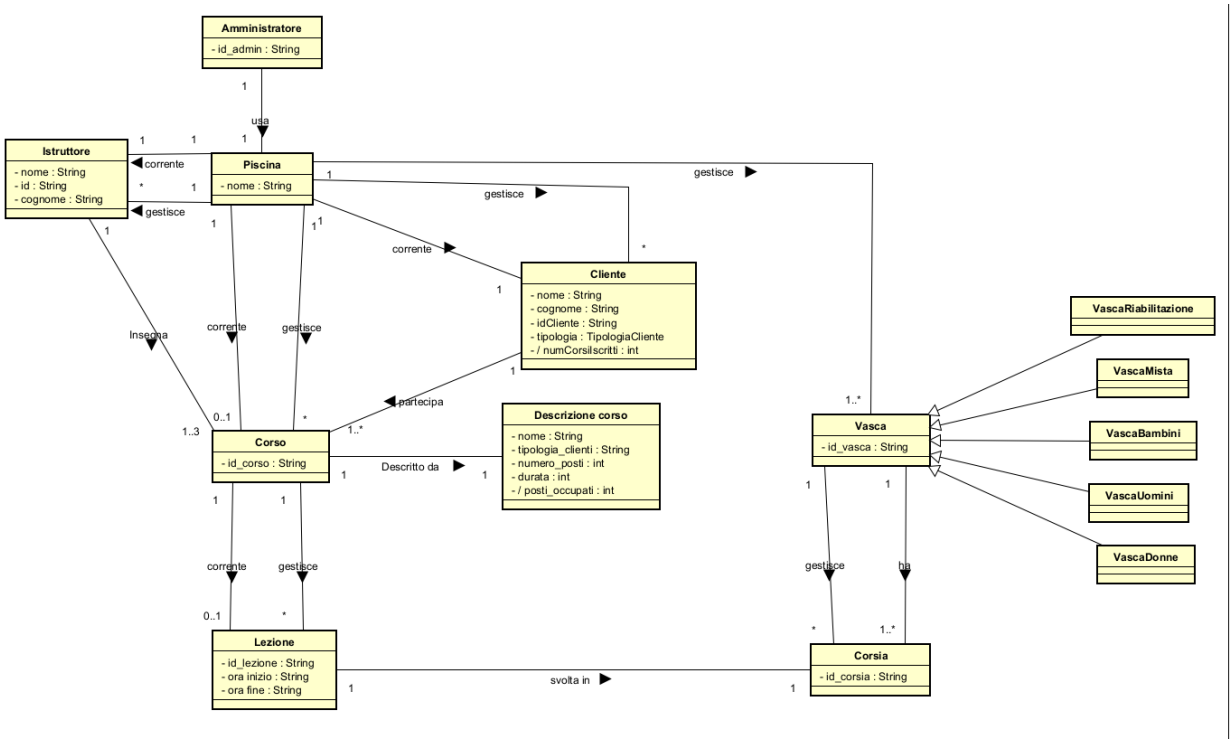
Eventuali ulteriori modifiche ai documenti saranno visionabili tramite la cronologia.

Fase di Analisi

Non sono presenti nuove classi concettuali identificate in questa iterazione, però è stato aggiornato:

- Il tipo dell'attributo tipologiaCliente in Descrizione Corso è passato dall'essere String al tipo "TipologiaVasca" già presente in vasca e definito opportunamente nel codice come enum.

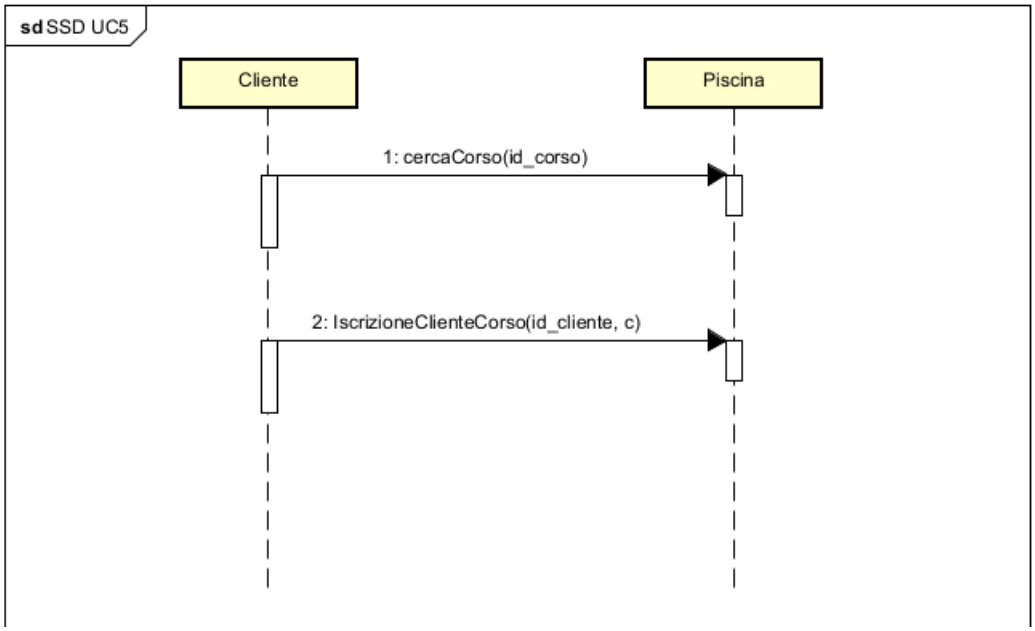
Si è arrivati al seguente Modello di Dominio:



2.1 Diagrammi di sequenza di sistema UC5

- UC5 ISCRIZIONE CLIENTE A CORSO

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra il cliente e il sistema Piscina.



2.1.1 Contratti delle operazioni

Riferimento:	UC5.1

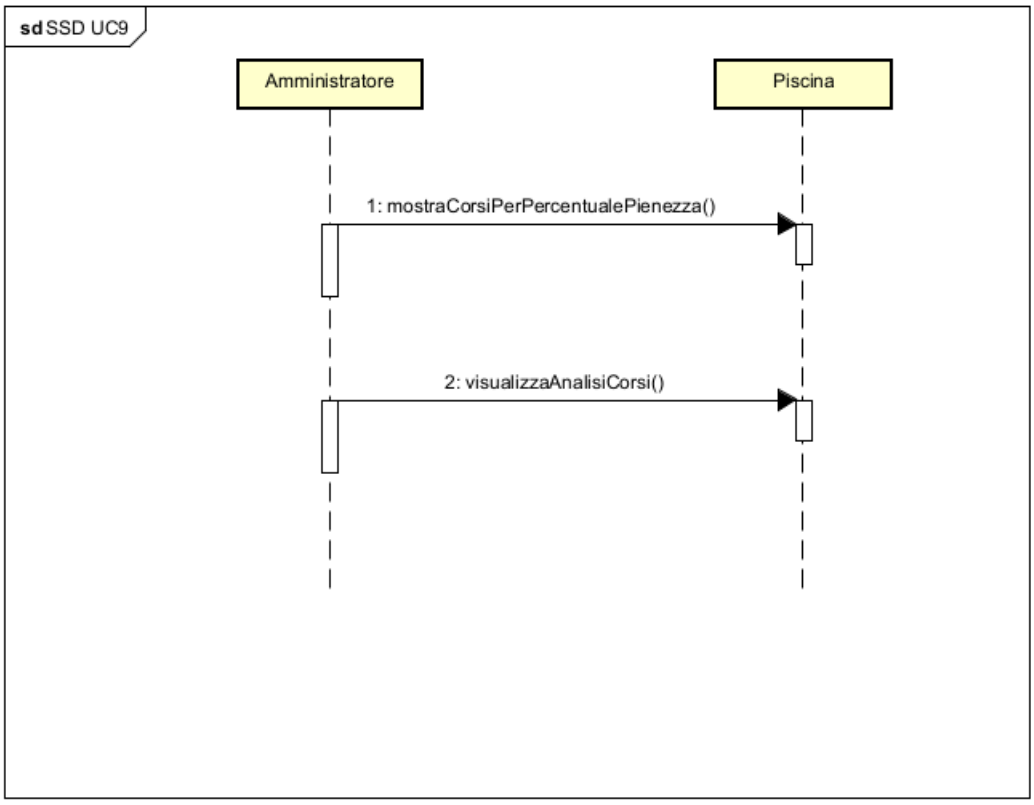
Operazione:	cercaCorso(id_corso)
Pre-condizioni:	• Nessuna
Post-condizioni:	• Se esiste viene recuperata l'istanza c di Corso

Riferimento:	UC5.2
Operazione:	IscrizioneClienteCorso(id_cliente, c)
Pre-condizioni:	• Esiste l'istanza c di Corso
Post-condizioni:	• Viene recuperata l'istanza cl di Cliente • Viene creata un'associazione tra l'istanza c di Corso e l'istanza cl di Cliente

2.2 Diagrammi di sequenza di sistema UC9

- UC9: VALUTAZIONE PIENEZZA CORSO

Il seguente Diagramma di Sequenza di Sistema (SSD) illustra il corso di eventi I/O tra l'amministratore e il sistema Piscina.



2.2.1 Contratti delle operazioni

Riferimento:	UC9.1
Operazione:	mostraCorsiPerPercentualePienezza()
Pre-condizioni:	• Nessuna
Post-condizioni:	• Nessuna

Riferimento:	UC9.2
Operazione:	visualizzaAnalisiCorsi()
Pre-condizioni:	• Nessuna
Post-condizioni:	• Viene creata la lista List<Corso> ListaDaEliminare • Viene creata la lista List<Corso>ListaDaAmpliare

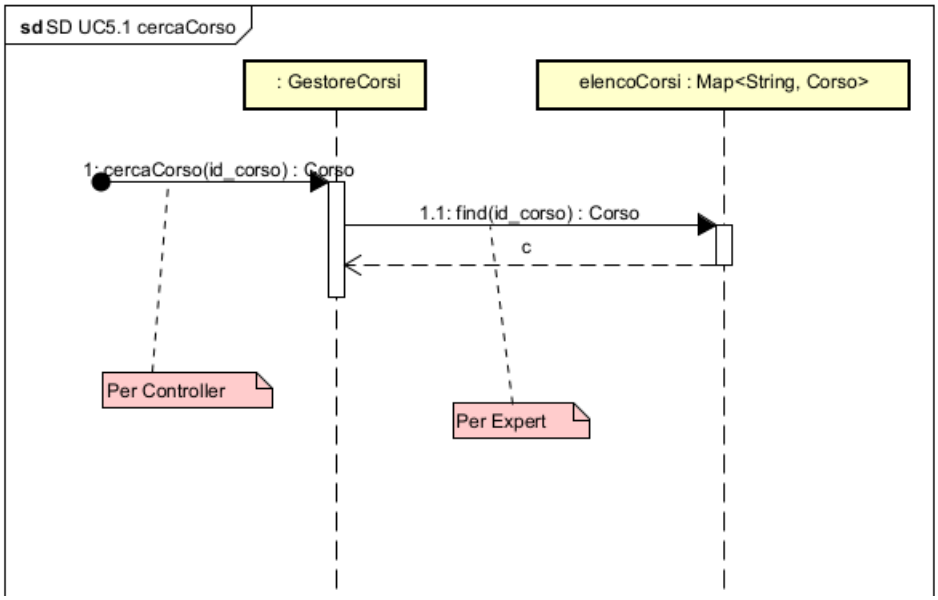
3.0 Fase di Progettazione

Gli elaborati principali della fase di progettazione sono i diagrammi di interazione, articolati in diagrammi di sequenza e diagrammi di interazione, con lo scopo di descrivere il comportamento del sistema da un punto di vista dinamico durante i casi d'uso presi in considerazione per questa seconda iterazione. Insieme a questi, il diagramma delle classi rappresenta il sistema da un punto di vista statico. Di seguito vengono riportati:

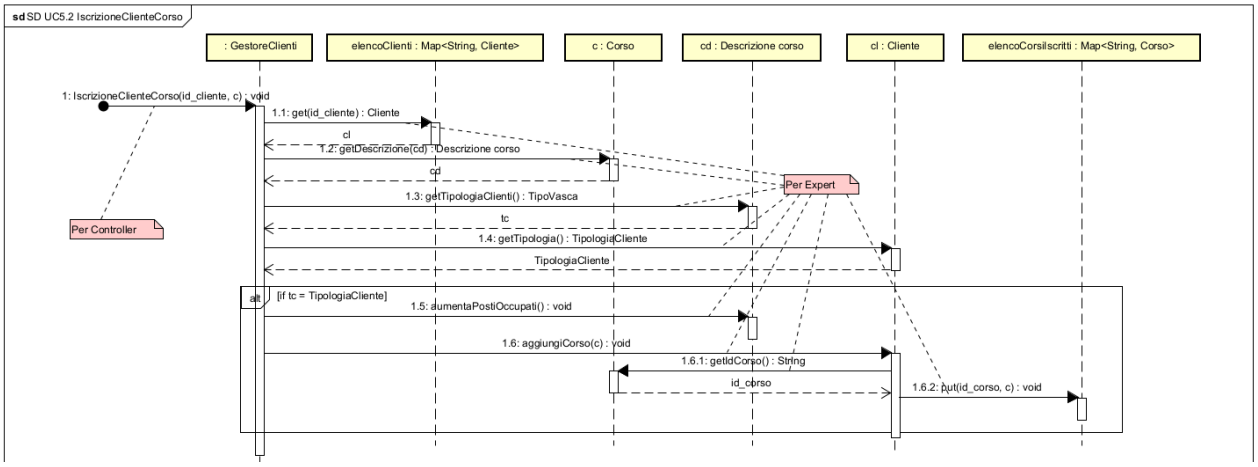
3.1 Diagrammi di Sequenza UC5

Si è utilizzato un Controller GestoreClienti per lavorare come interfaccia del sistema. Questo avrà il compito di creare l'associazione tra l'istanza del Cliente e l'istanza del corso a cui desidera iscriversi.

cercaCorso



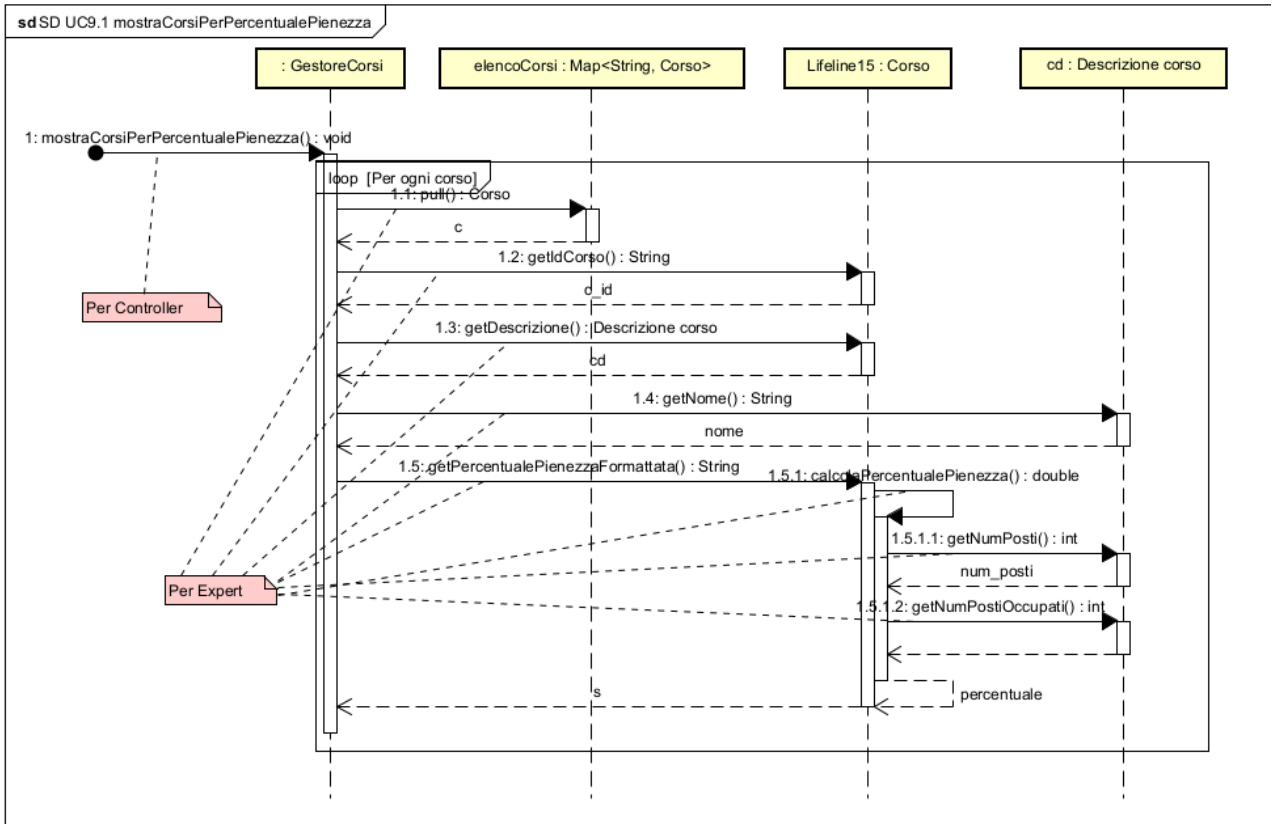
iscrizioneClienteCorso



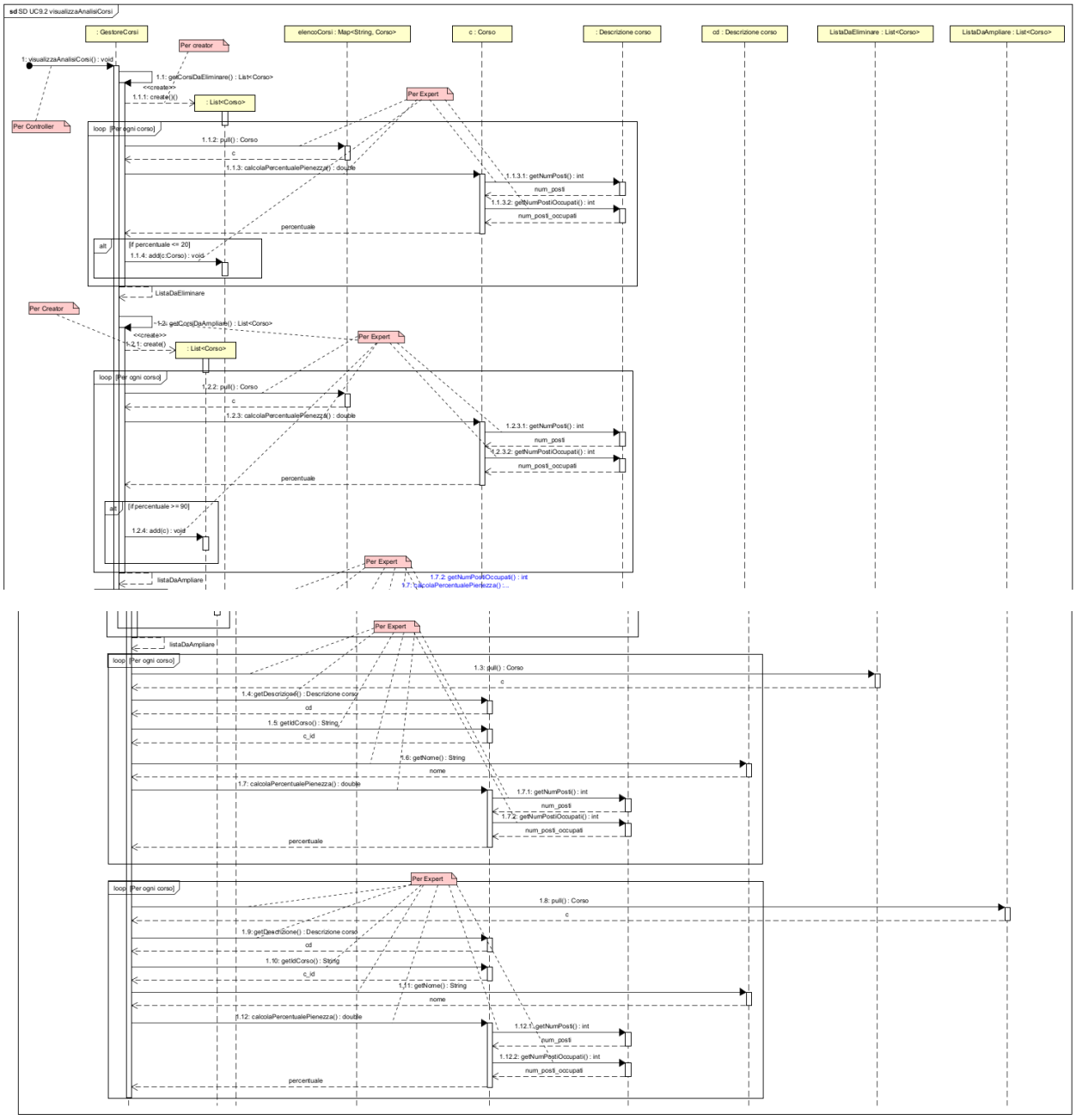
3.2 Diagrammi di Sequenza UC9

Si è utilizzato un Controller GestoreClienti per lavorare come interfaccia del sistema. Avrà il compito di visualizzare il report e creare le due nuove liste di corsi, una che consiglia quali eliminare, una che consiglia quali ampliare secondo le regole di dominio.

mostraCorsiPerPercentalePienezza

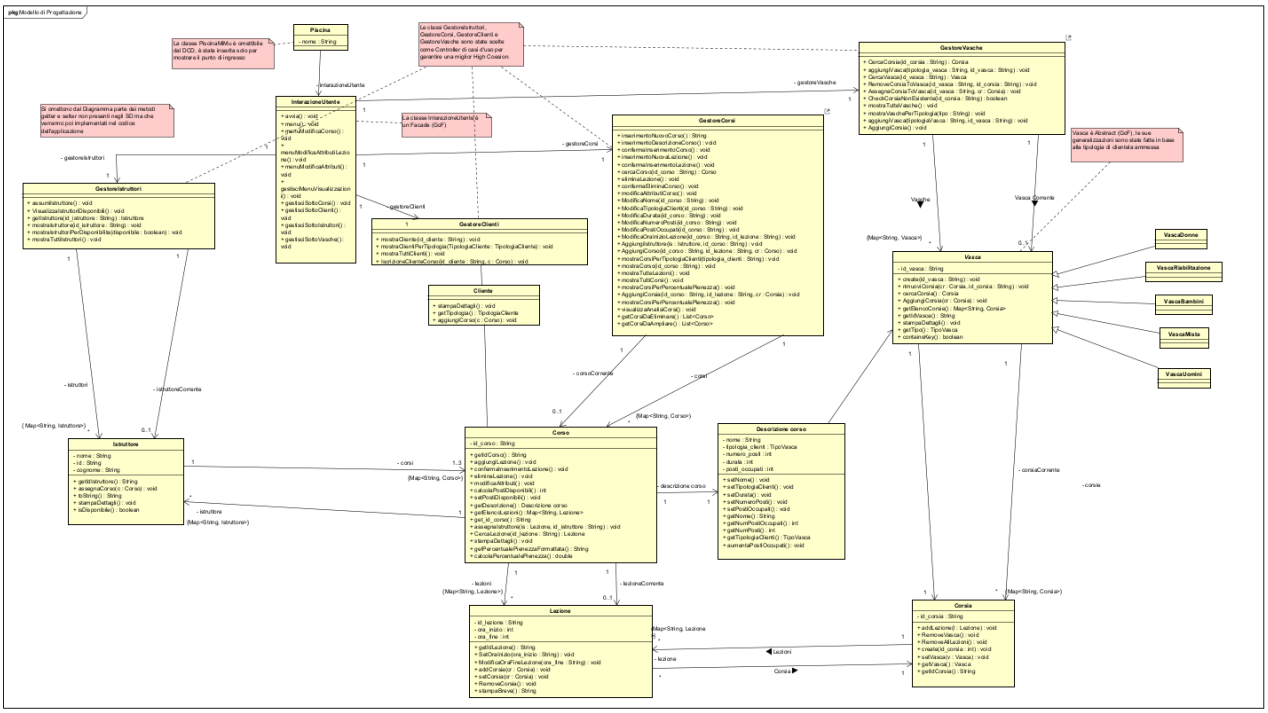

sd SD UC9.1 mostraCorsiPerPercentualePienezza
: GestoreCorsi
elencoCorsi : Map<String, Corso>
Lifeline15 : Corso
cd : Descrizione corso
1: mostraCorsiPerPercentualePienezza() : void
loop [Per ogni corso]
1.1: pull() : Corso
c
1.2: getIdCorso() : String
c_id
1.3: getDescrizione() : Descrizione corso
cd
1.4: getNome() : String
nome
1.5: getPercentualePienezzaFormattata() : String
1.5.1: calcolaPercentualePienezza() : double
1.5.1.1: getNumPosti() : int
num_posti
1.5.1.2: getNumPostiOccupati() : int
s
percentuale
Per Controller
Per Expert

visualizzaAnalisiCorsi



3.4 Diagramma delle Classi

Di seguito l'evoluzione del DCD:



5.0 Testing

5.1 Individuazione dei casi di test

In una fase preliminare al testing, si è svolta un'analisi del codice per individuare le classi e i metodi da testare. La strategia è stato quello di dare priorità ai metodi cruciali per il funzionamento del sistema relativamente ai casi d'uso implementati e al fixing di test precedenti per il già citato refactoring.

Di seguito sono elencati i nuovi metodi di test:

CorsoTest

1. **testCalcolaPercentualePienezza_standard**
   Verifica che la percentuale di posti occupati sia calcolata correttamente in condizioni standard.
2. **testCalcolaPercentualePienezza_zeroOccupati**
   Controlla che la percentuale sia 0 quando non ci sono posti occupati.
3. **testCalcolaPercentualePienezza_zeroPosti**
   Verifica che la percentuale gestisca correttamente il caso con zero posti totali evitando divisione per zero.
4. **testGetPercentualePienezzaFormattata**
   Controlla che la percentuale di pienezza sia restituita correttamente come stringa formattata con due decimali.

GestoreClientiTest

1. **testIscrizioneClienteCorso_ok**
   Verifica che un cliente possa iscriversi correttamente a un corso compatibile con la sua tipologia.

2. **testIscrizioneClienteCorso_TipologiaNonCorrispondente**
   Controlla che l'iscrizione fallisca se la tipologia del cliente non corrisponde a quella del corso, lanciando TipologiaNonCorrispondente.
3. **testIscrizioneClienteCorso_ClienteGiaIscrittoException**
   Verifica che l'iscrizione di un cliente già iscritto a un corso generi ClienteGiaIscrittoException.
4. **testIscrizioneClienteCorso_PostiPieniException**
   Controlla che l'iscrizione fallisca quando il corso ha raggiunto il numero massimo di posti disponibili, lanciando PostiPieniException.

GestoreCorsiTest

1. **testGetCorsiDaEliminare**
   Verifica che i corsi con pochi partecipanti vengano identificati correttamente come da eliminare.
2. **testGetCorsiDaAmpliare**
   Controlla che i corsi quasi pieni vengano identificati correttamente come da ampliare.