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.

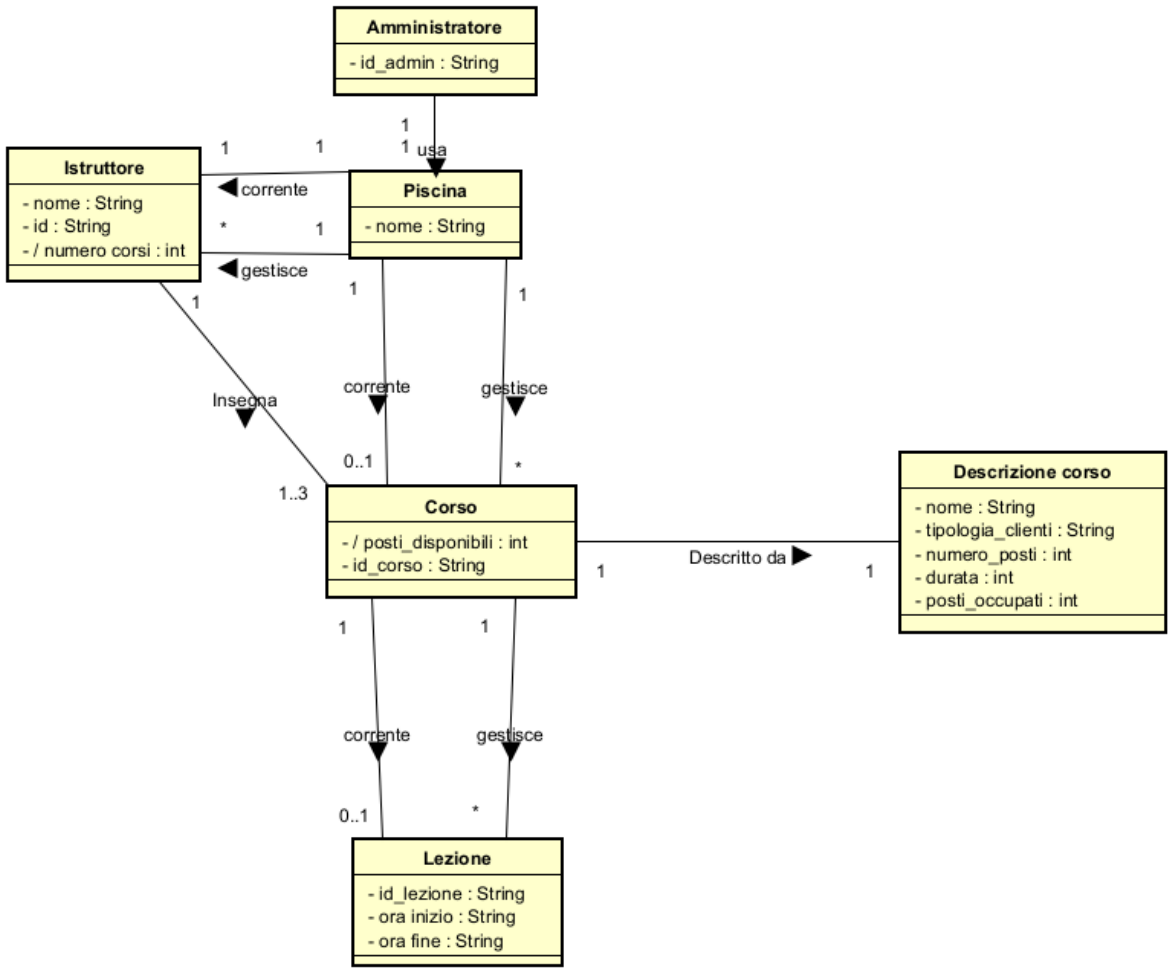
Eventuali ulteriori modifiche ai documenti saranno visionabili tramite la cronologia.

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

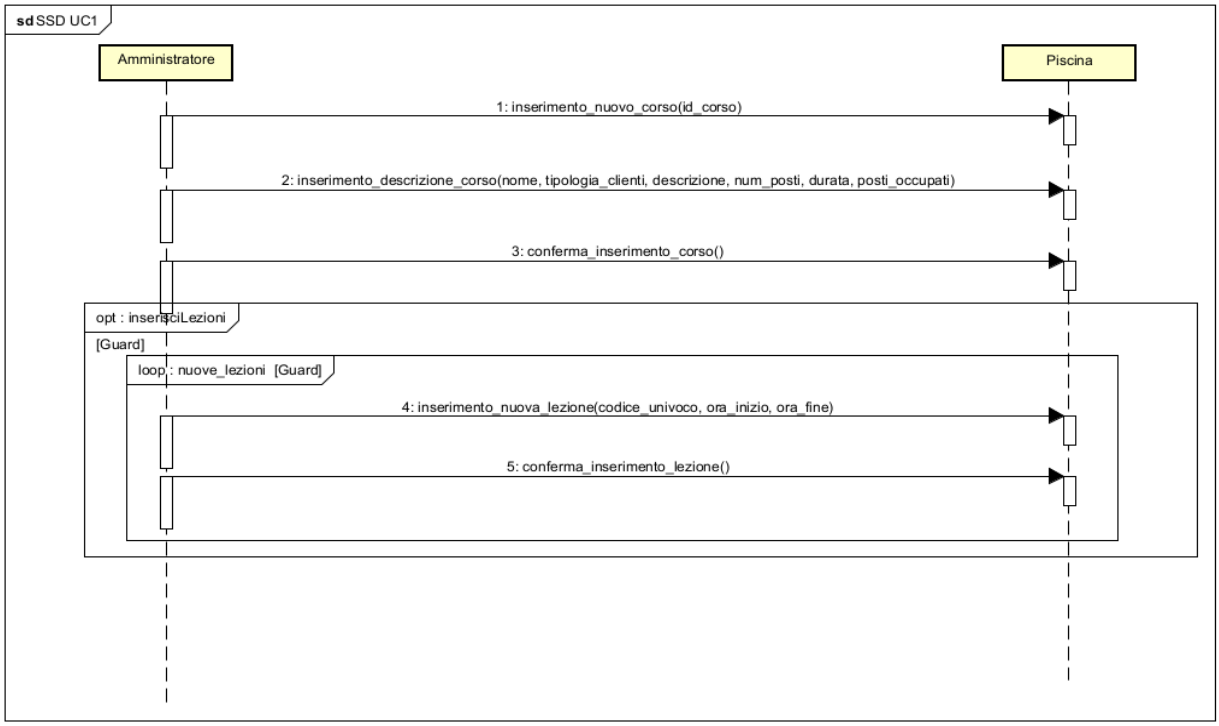
Da cui tenendo conto di associazioni e attributi, ottenuti analizzando i primi tre casi d'uso, è stato ricavato il seguente Modello di Dominio:



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.



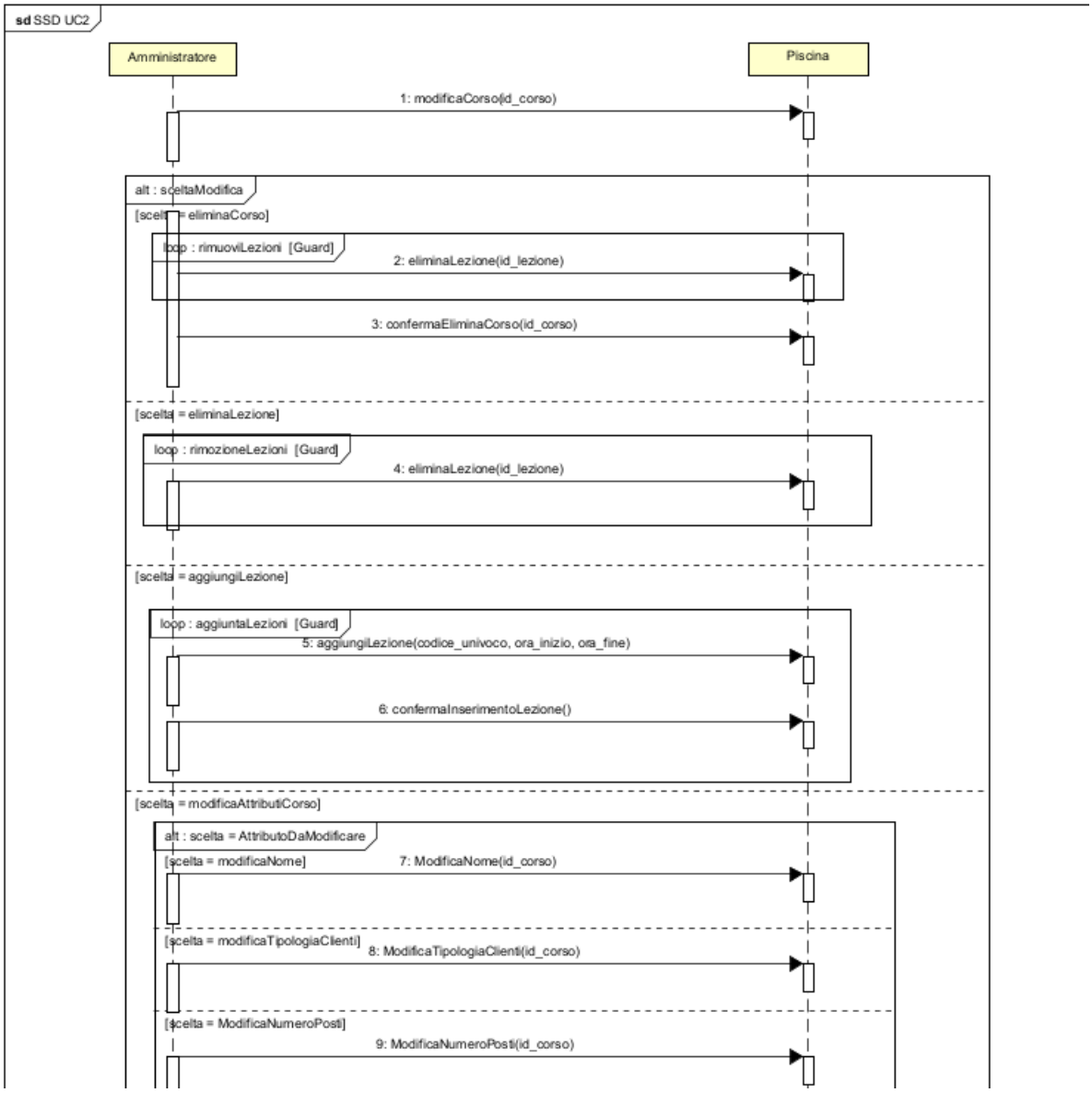
2.1.1 Contratti delle operazioni

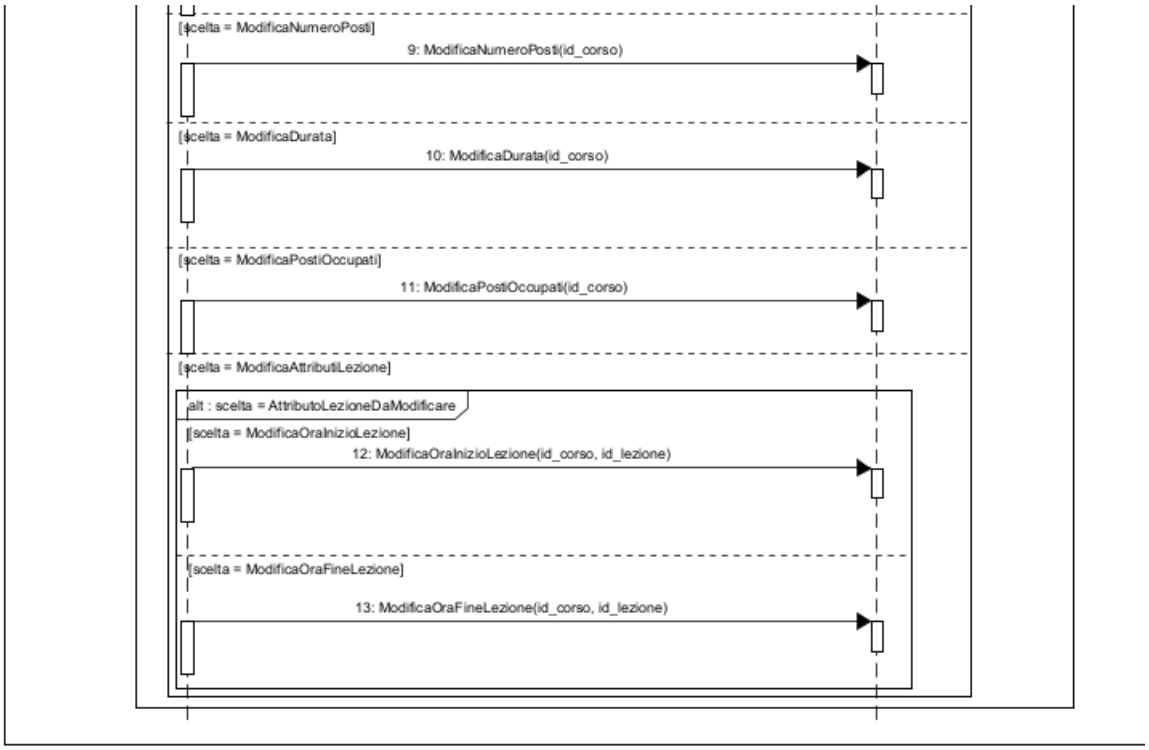
Non sono stati scritti contratti per questo caso d'uso.

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

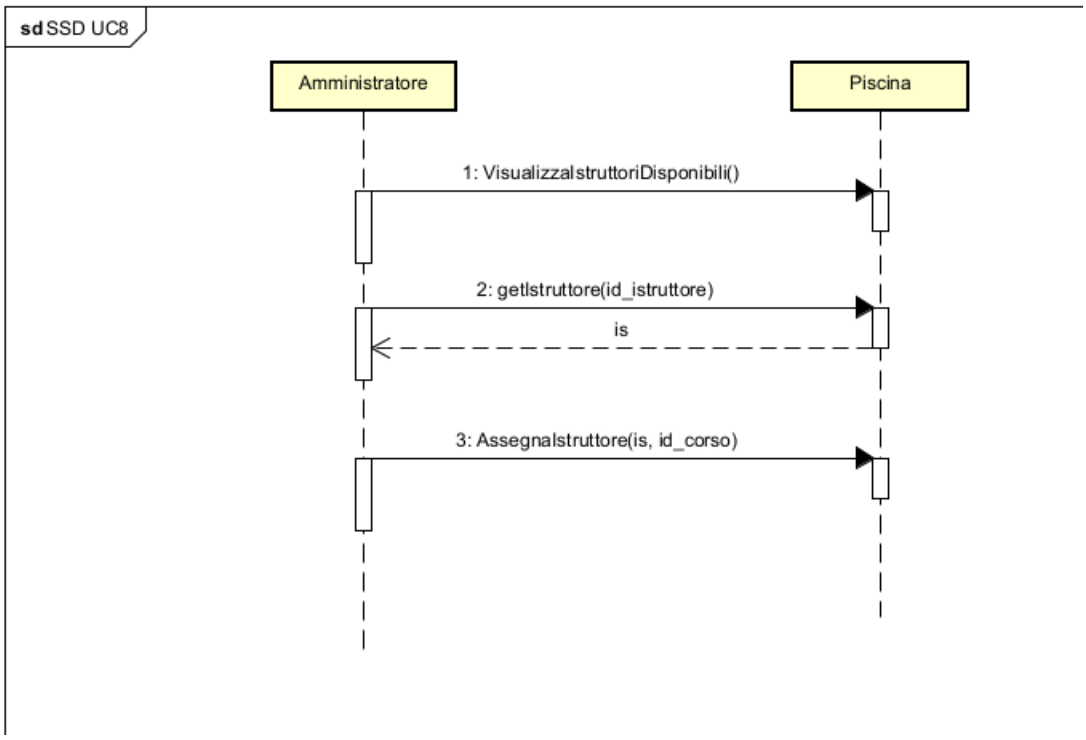
Riferimento:	UC2.12
Operazione:	ModificaOraInizioLezione(id_corso, id_lezione)
Pre-condizioni:	• Esiste un’istanza c di Corso • Esiste un’istanza l di Lezione associata all’istanza c di Corso
Post-condizioni:	• L’attributo ora_inizio di Lezione l viene modificato

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

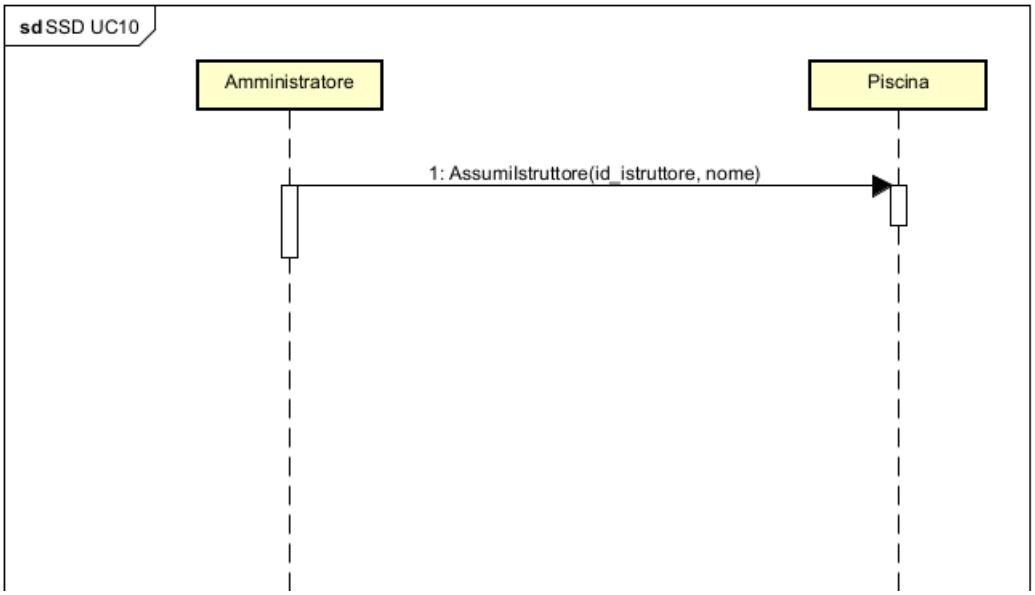
Riferimento:	UC8
Operazione:	AssegnaIstruttore(is, id_corso)
Pre-condizioni:	• Deve esistere un'istanza is di Istruttore • Deve esistere un'istanza c di Corso
Post-condizioni:	• È stata creata l'associazione tra l'istanza di corso c e l'istanza is di istruttore

	• 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

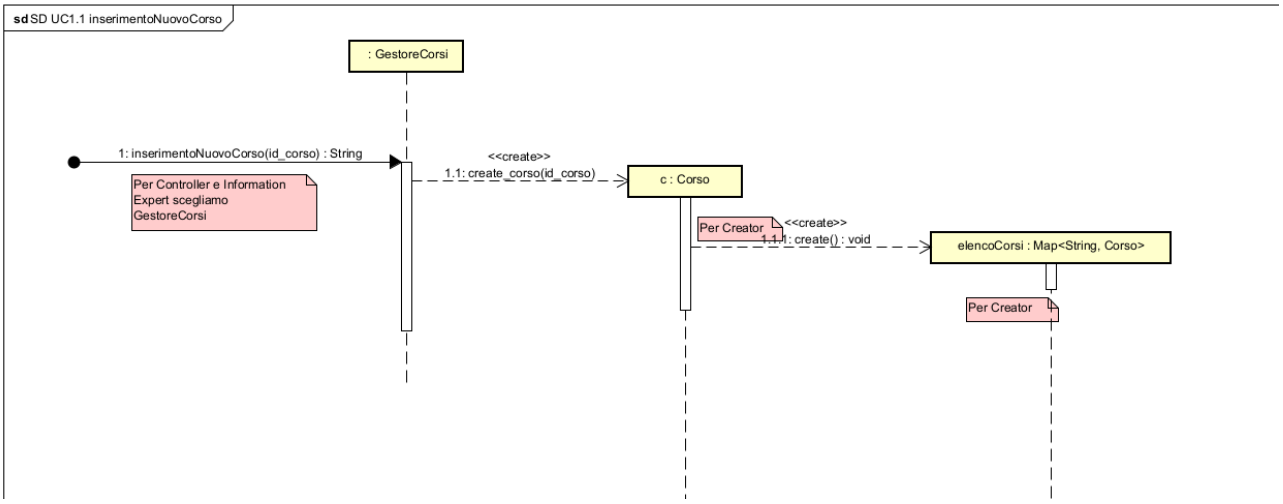
3.0 Fase di Progettazione

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:

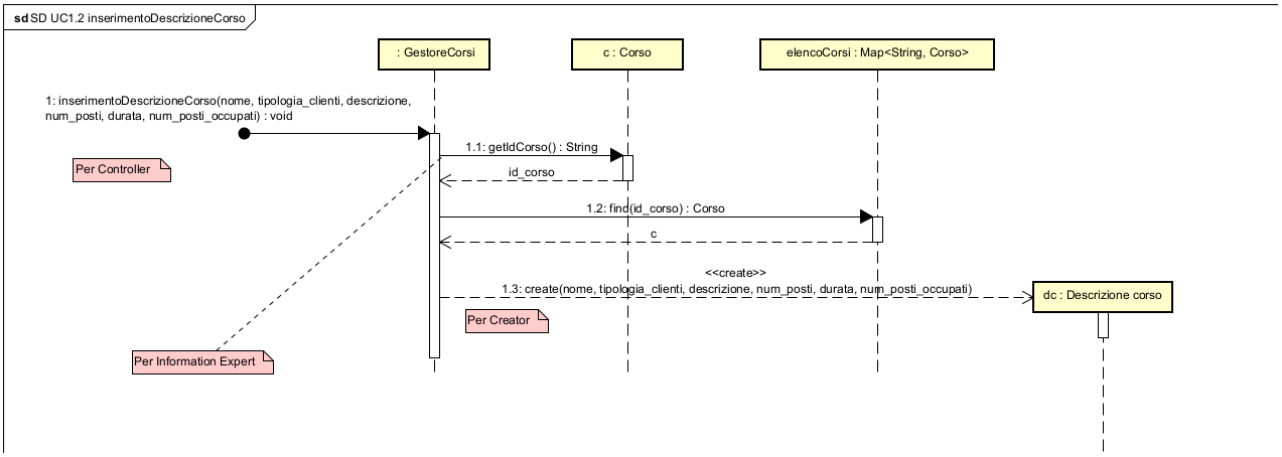
3.1 Diagrammi di Sequenza UC1

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.

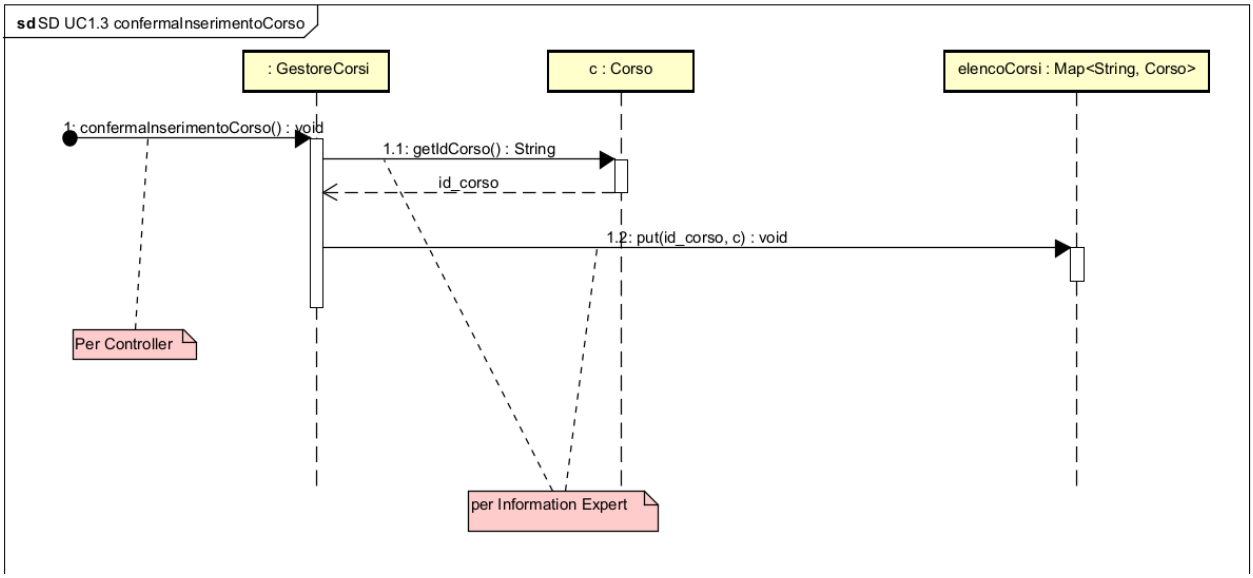
inserimentoNuovoCorso



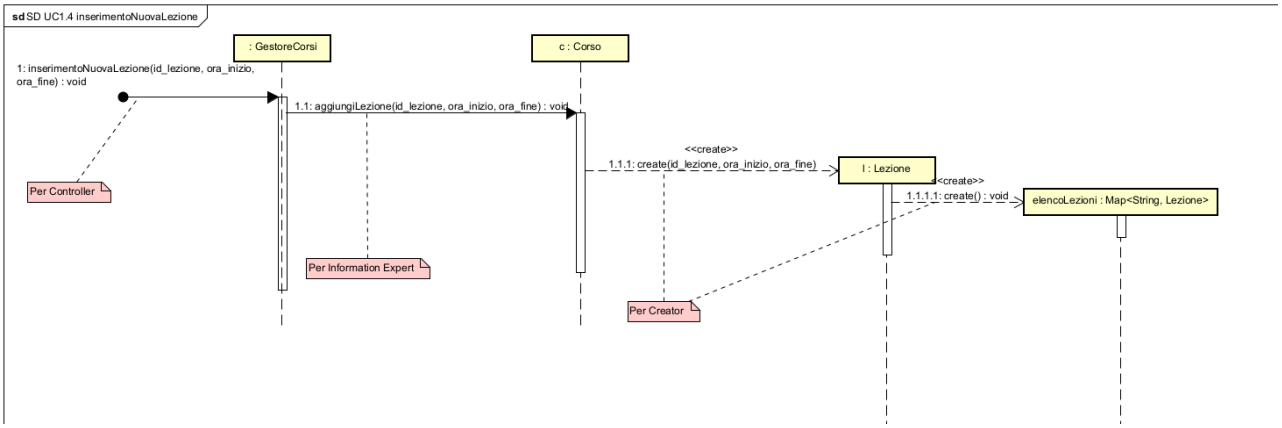
inserimentoDescrizioneCorso



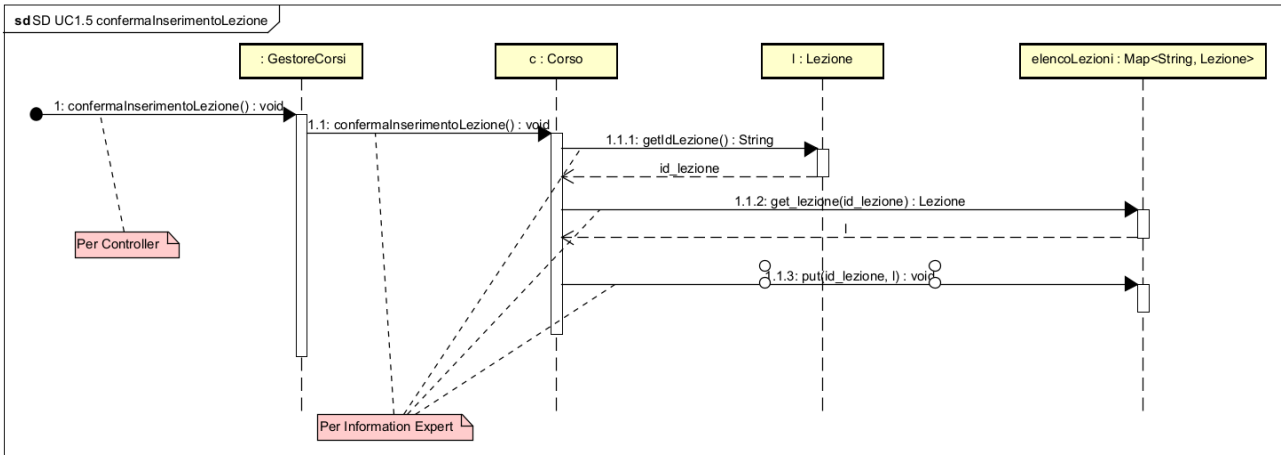
confermaInserimentoCorso



inserimentoNuovaLezione



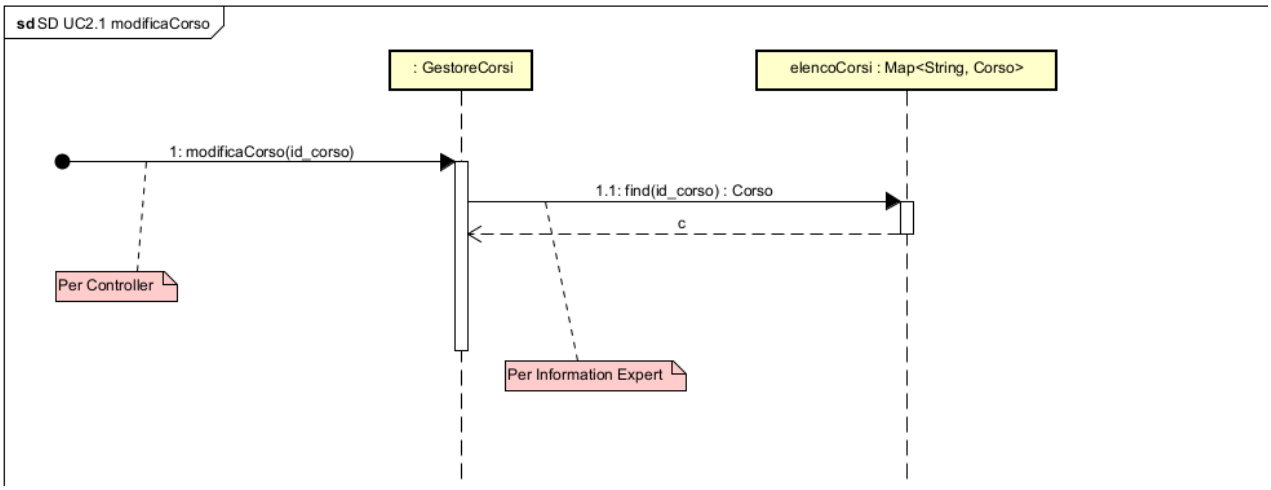
confermaInserimentoLezione



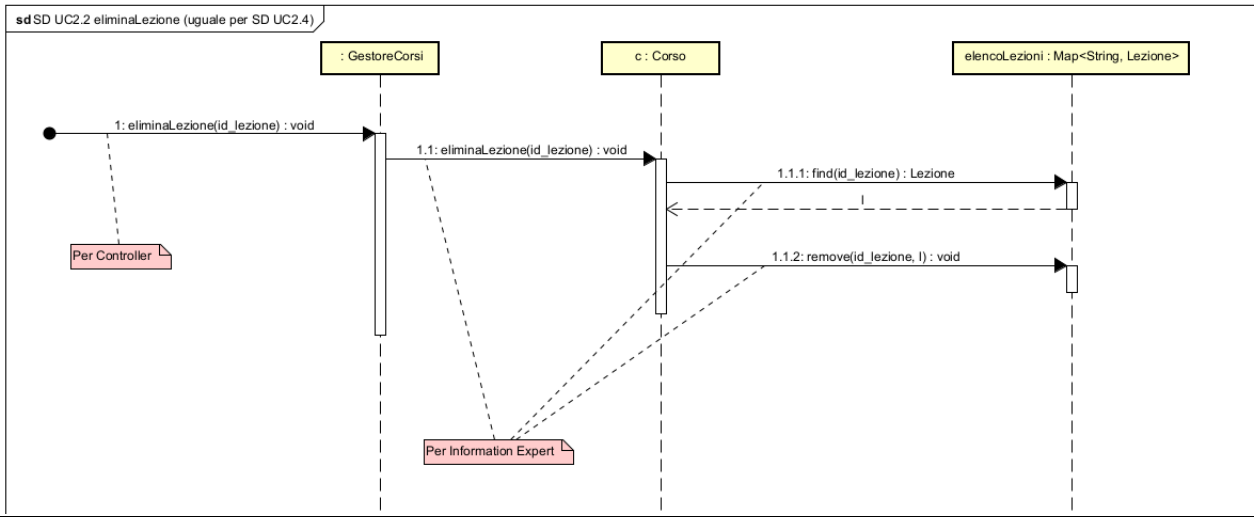
3.2 Diagrammi di Sequenza UC2

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.

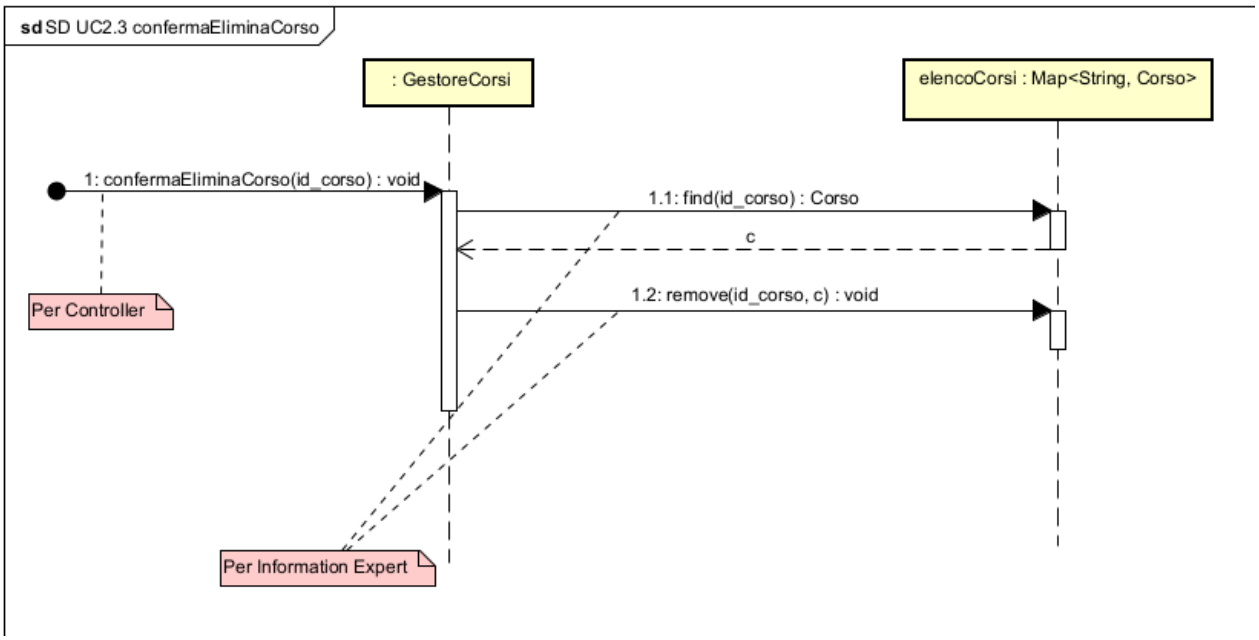
modificaCorso



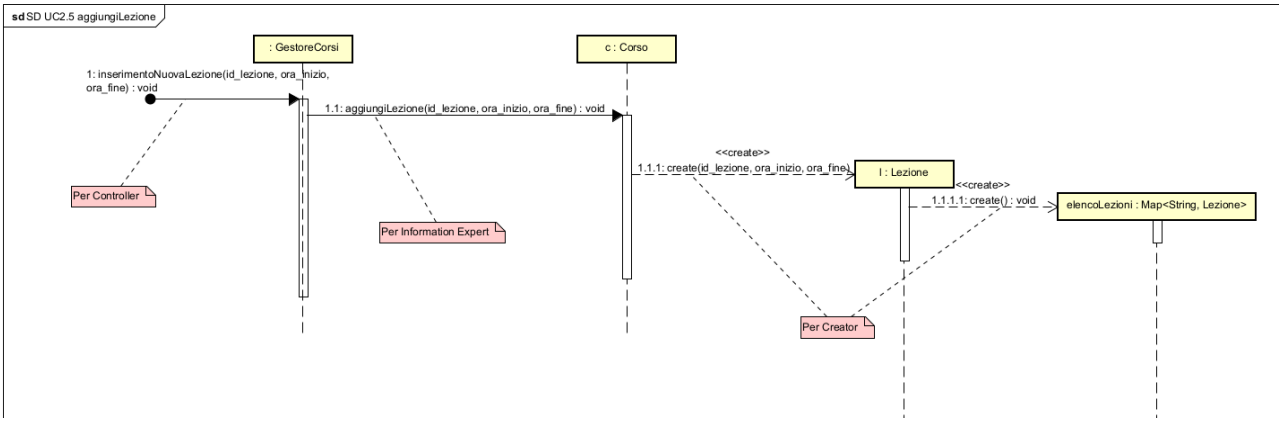
eliminaLezione



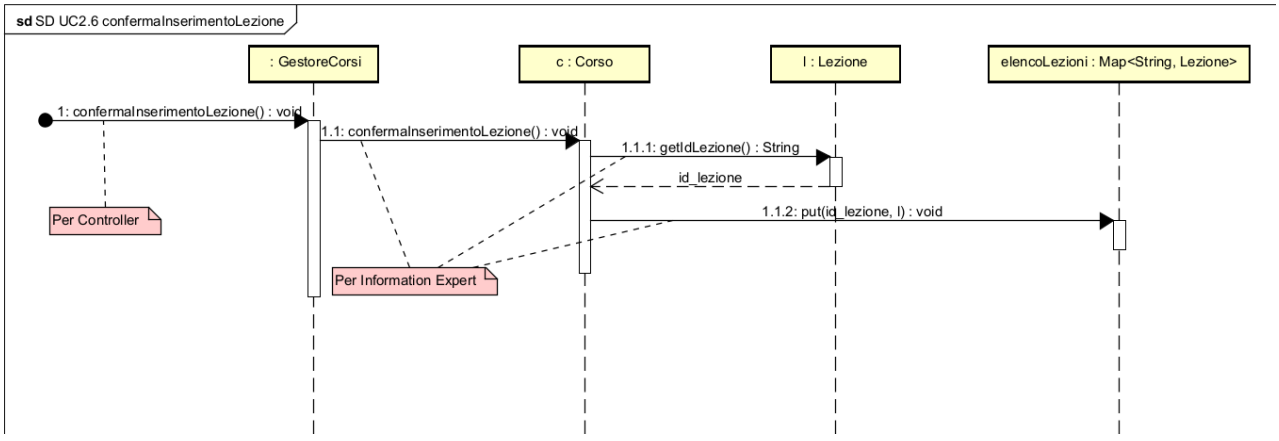
confermaEliminaCorso



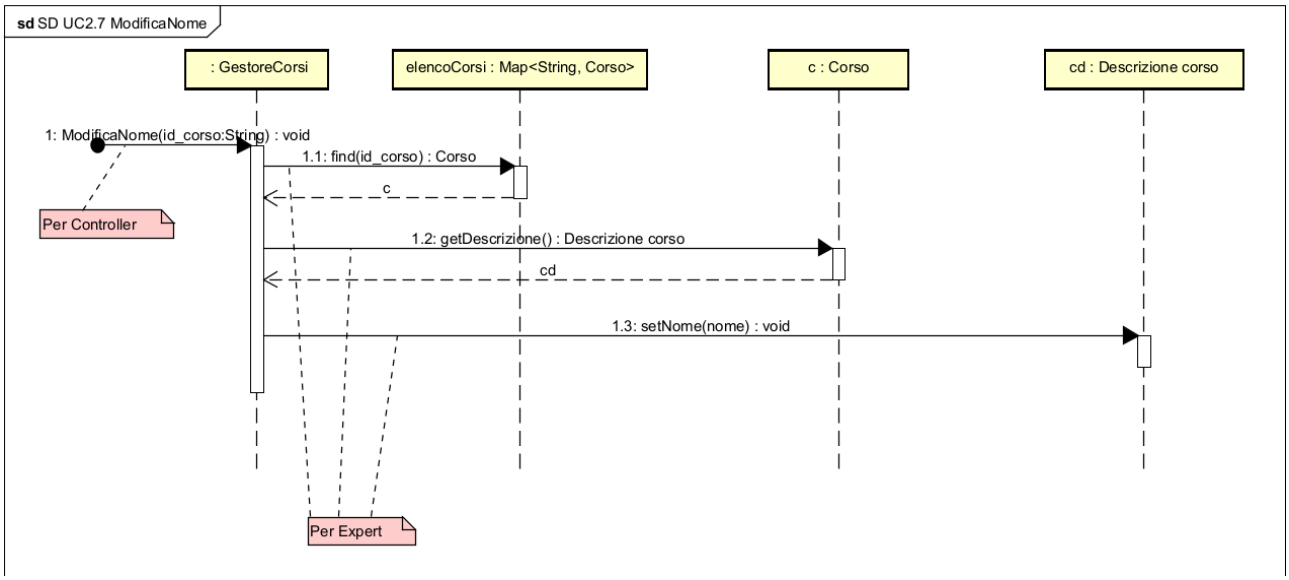
aggiungiLezione



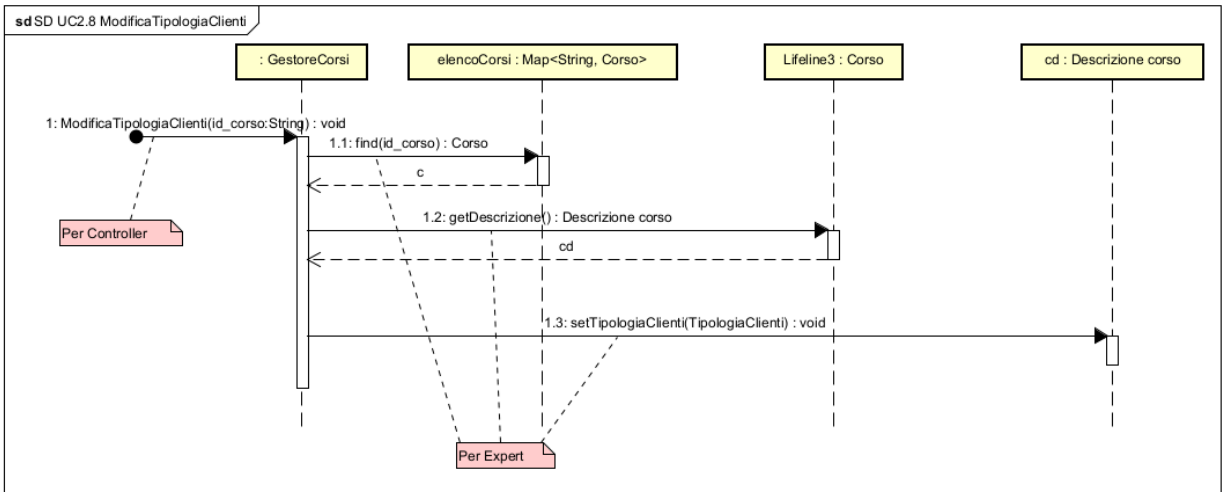
confermaInserimentoLezione



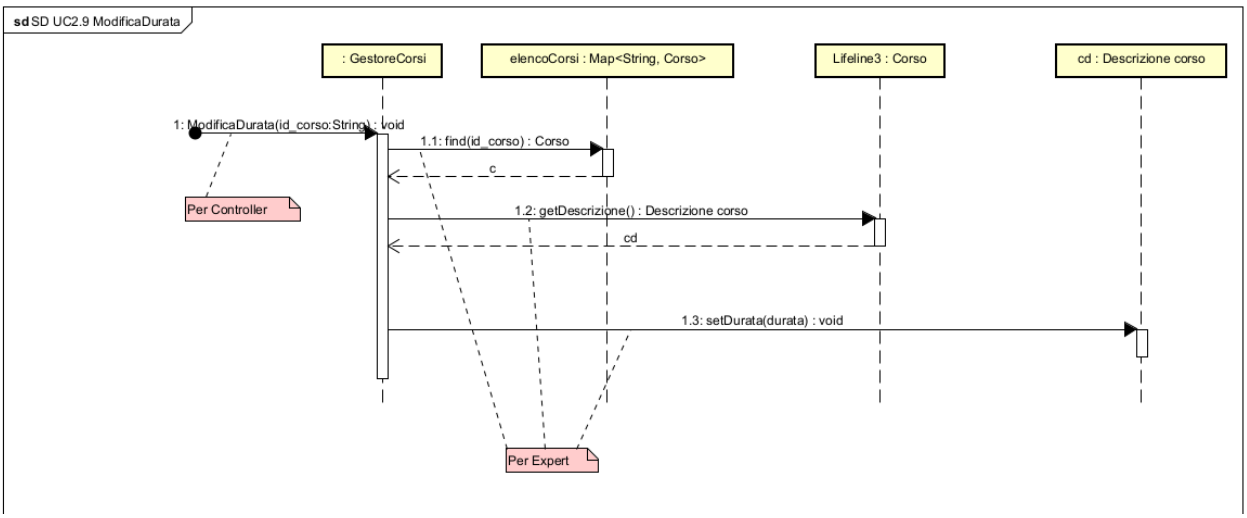
modificaNome



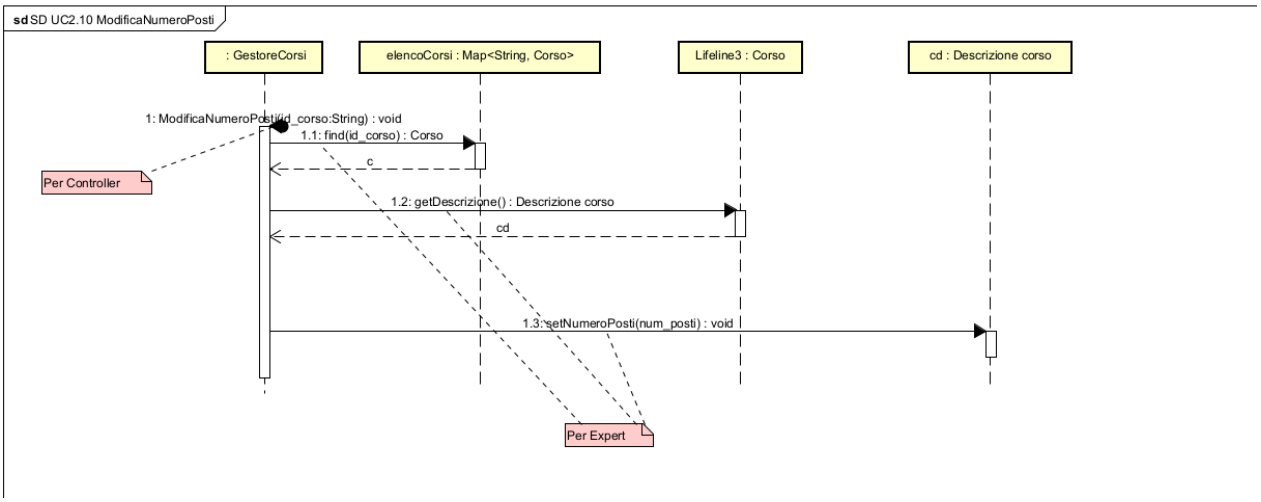
modificaTipologiaClienti



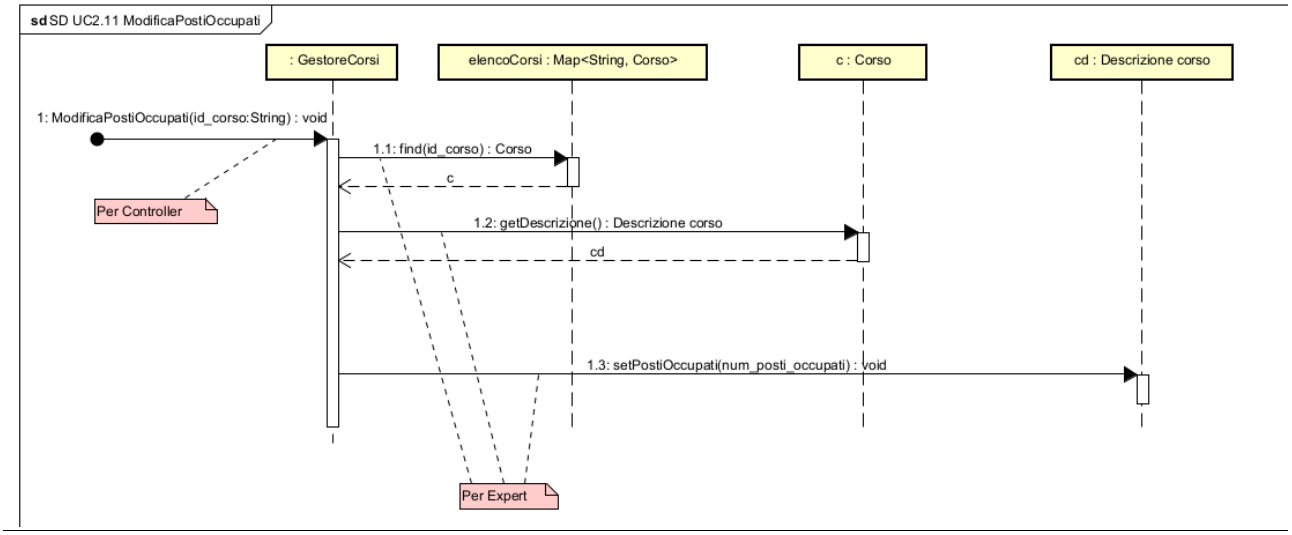
modificaDurata



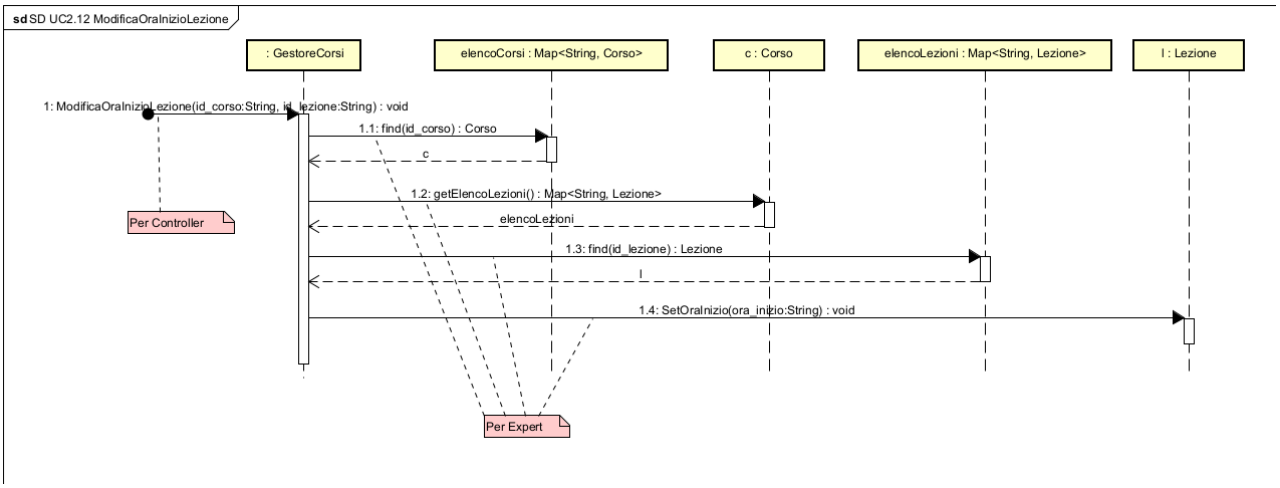
modificaNumeroPosti



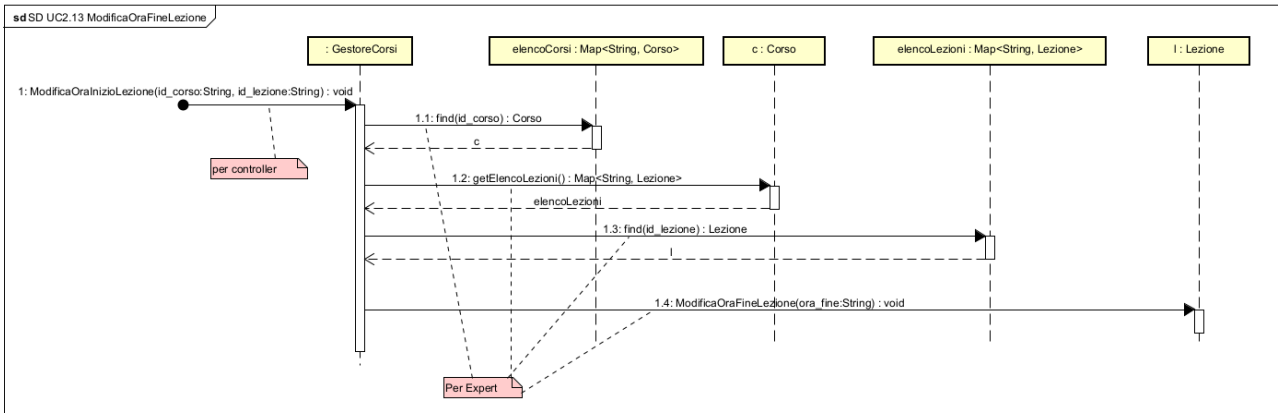
modificaPostiOccupati



modificaOraInizioLezione



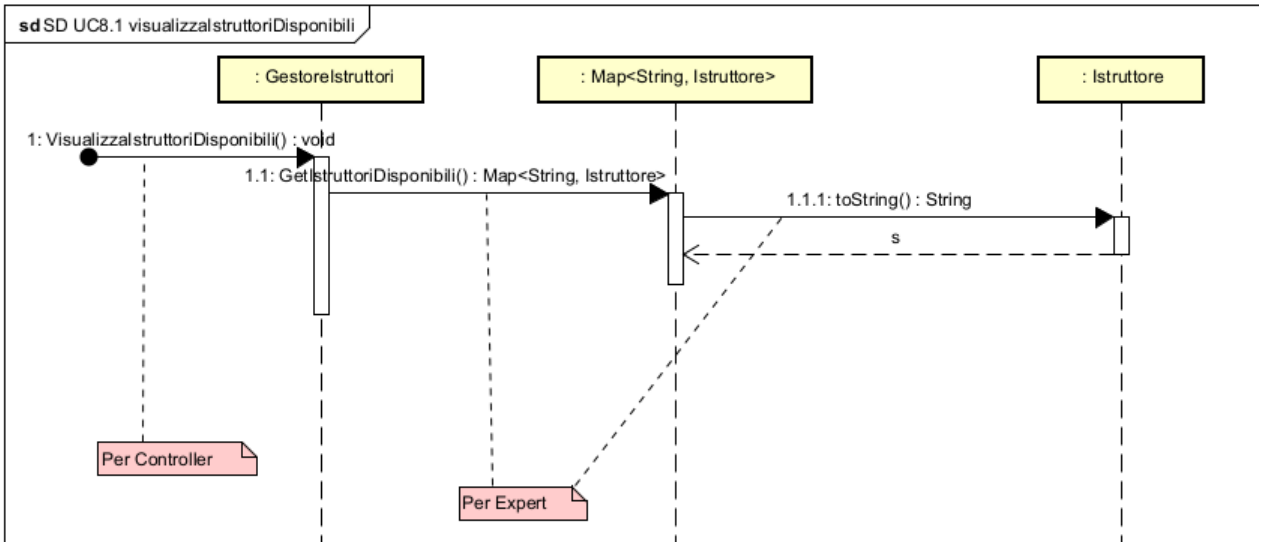
modificaOraFineLezione



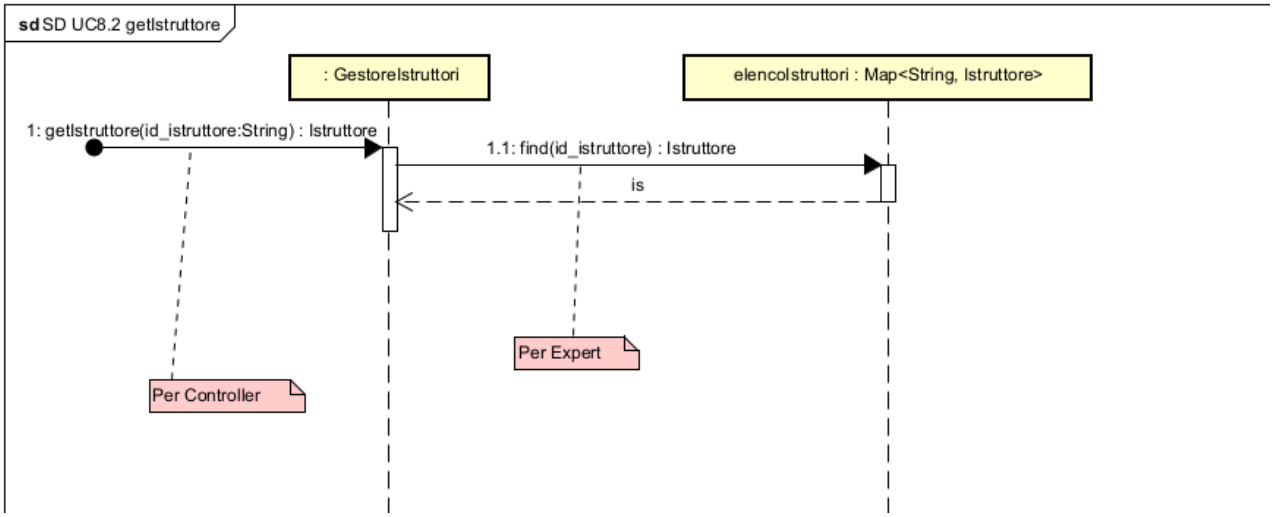
3.3 Diagrammi di Sequenza UC8

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.

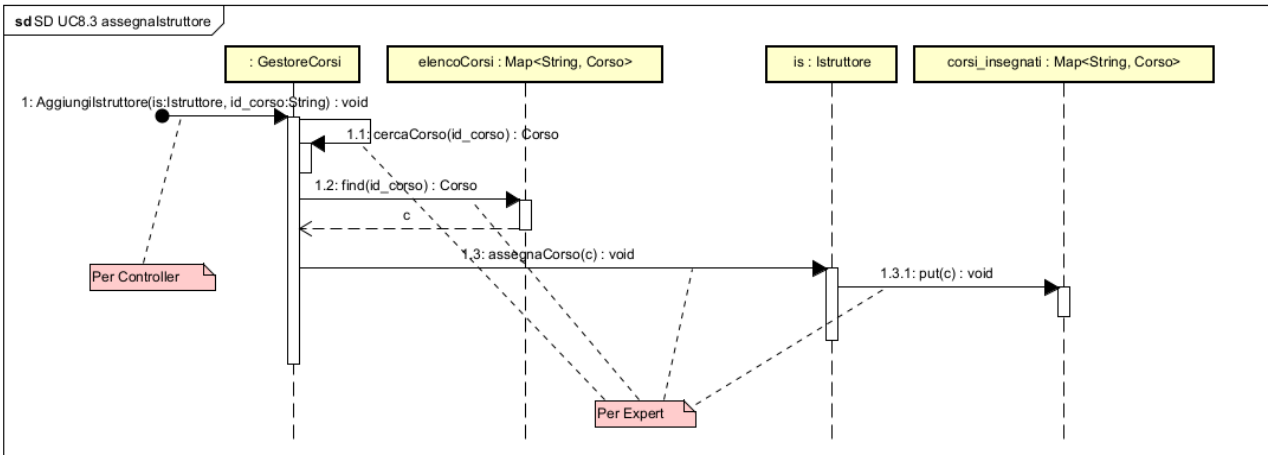
visualizzaIstruttoriDisponibili



getIstruttore



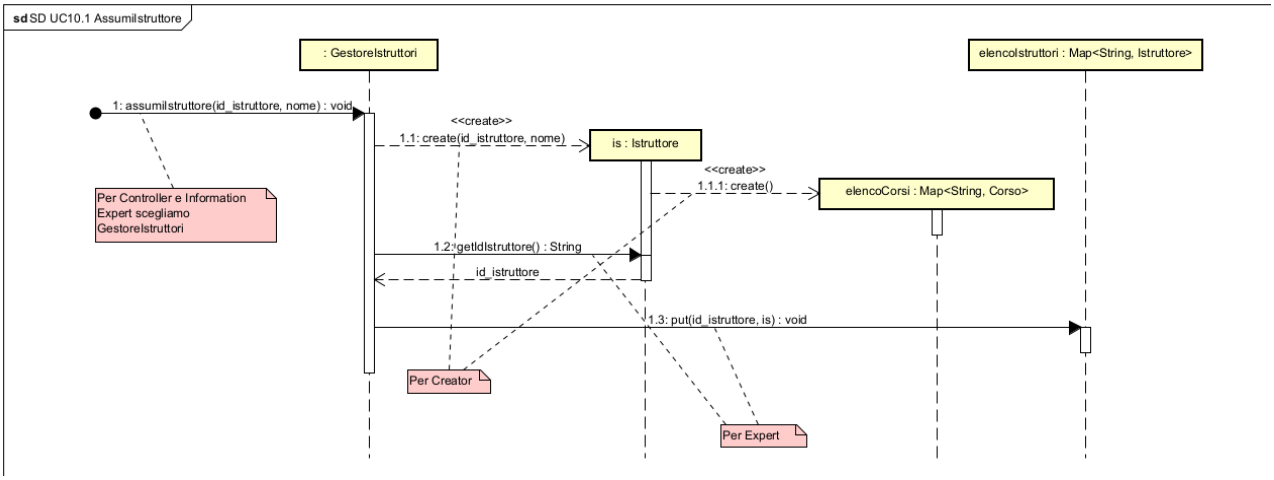
assegnaIstruttore



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

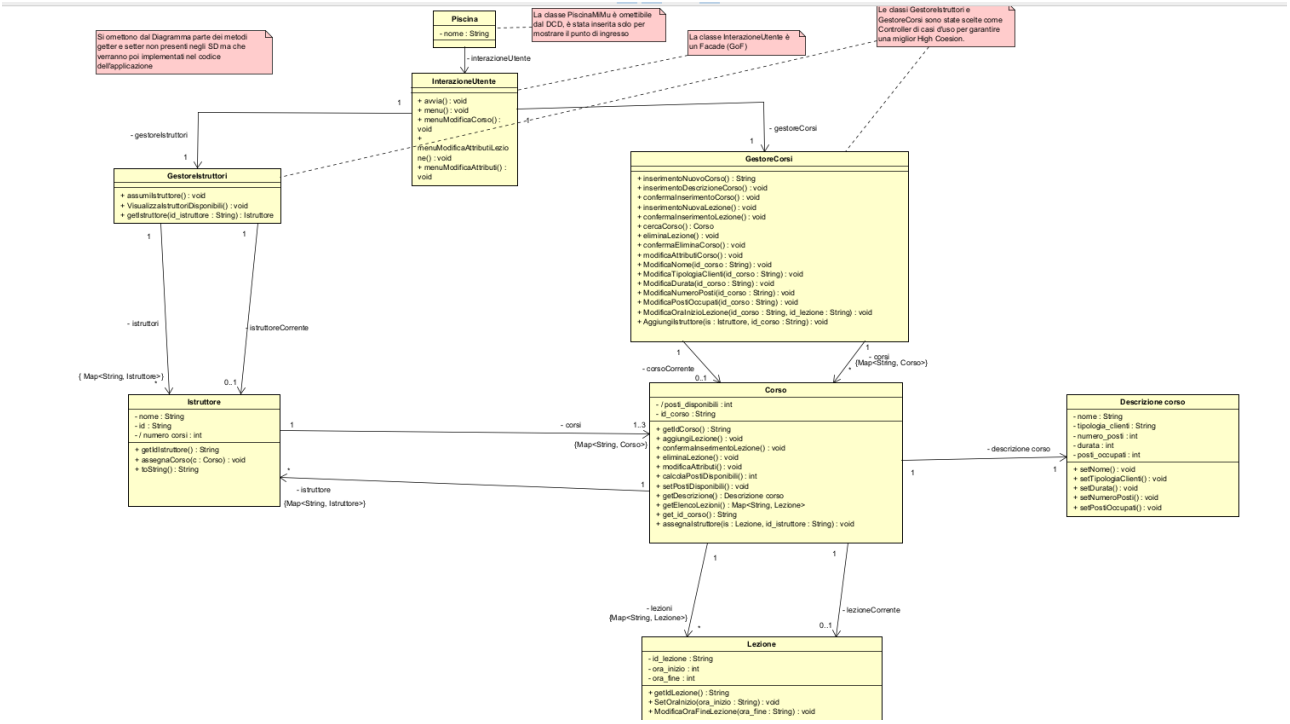


3.4 Diagramma delle Classi

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.