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

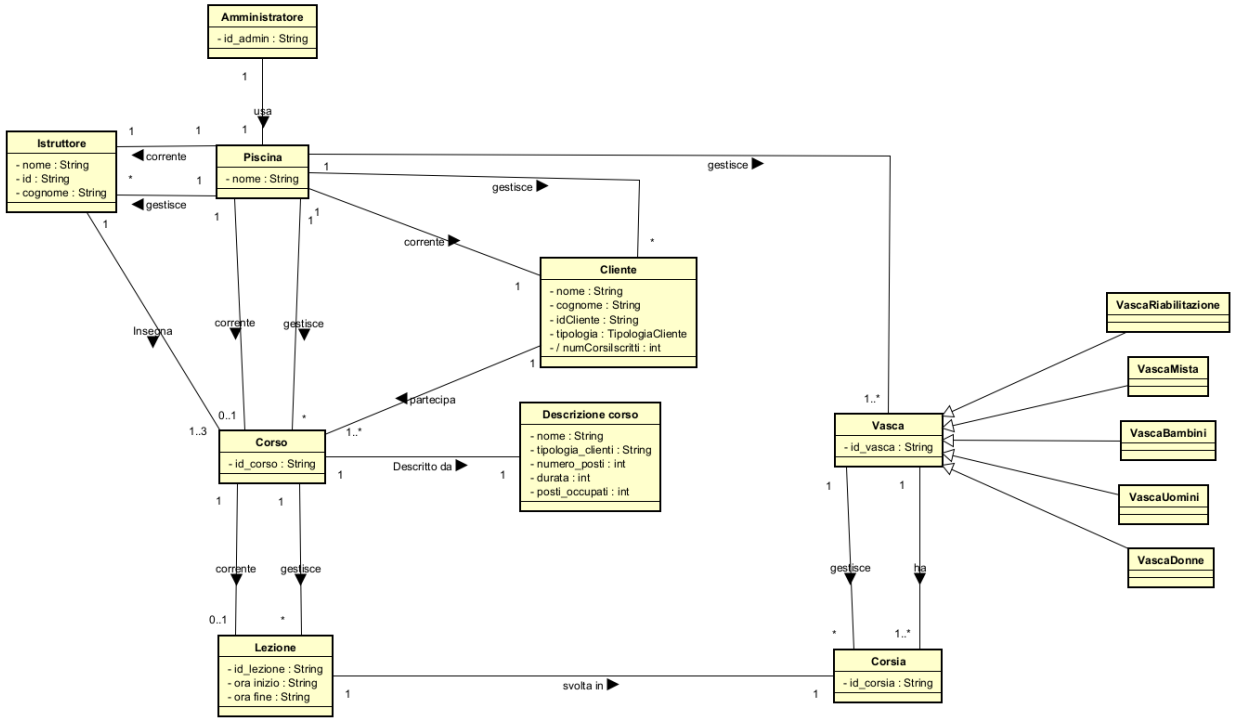
Le classi concettuali identificate durante questa seconda iterazione sono:

- Cliente
- Vasca
- Corsia

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

- Aggiunto l'attributo cognome su Istruttore

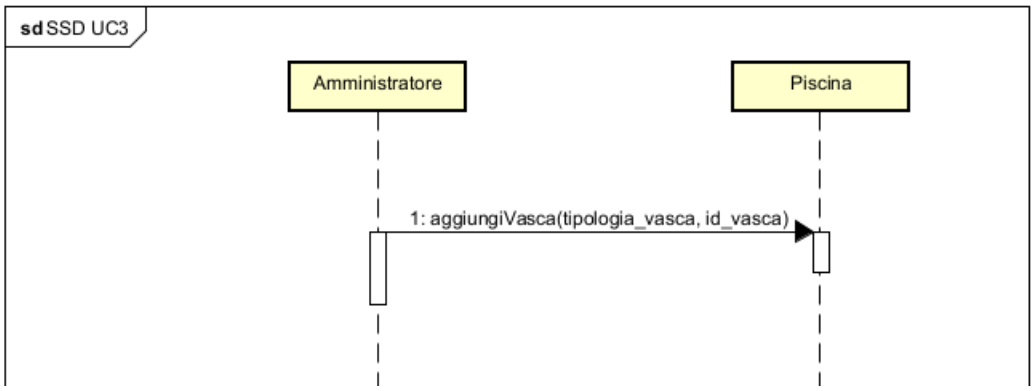
Si è arrivati al seguente Modello di Dominio:



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.



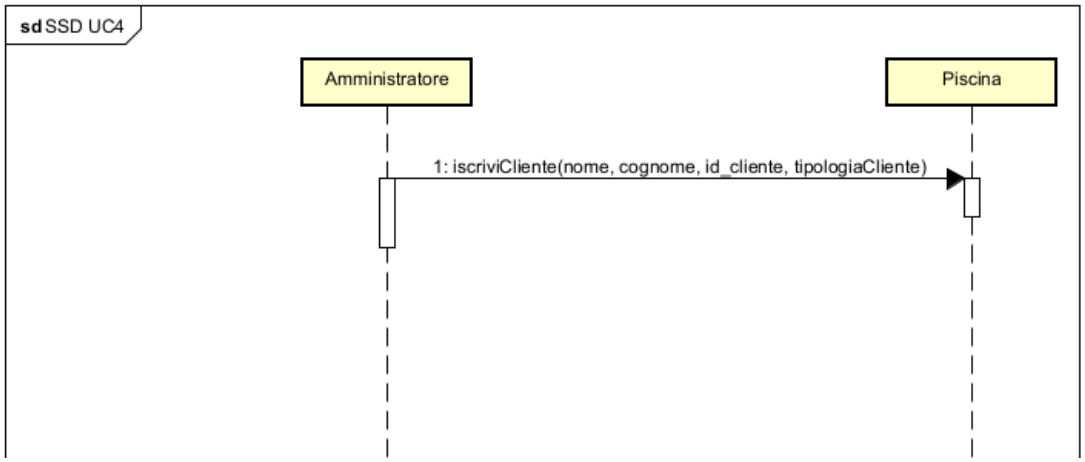
2.1.1 Contratti delle operazioni

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.



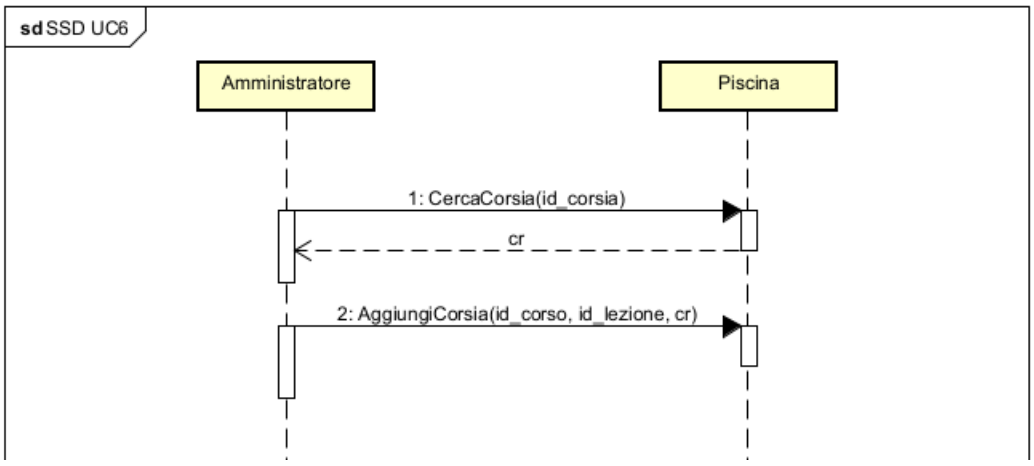
2.2.1 Contratti delle operazioni

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)

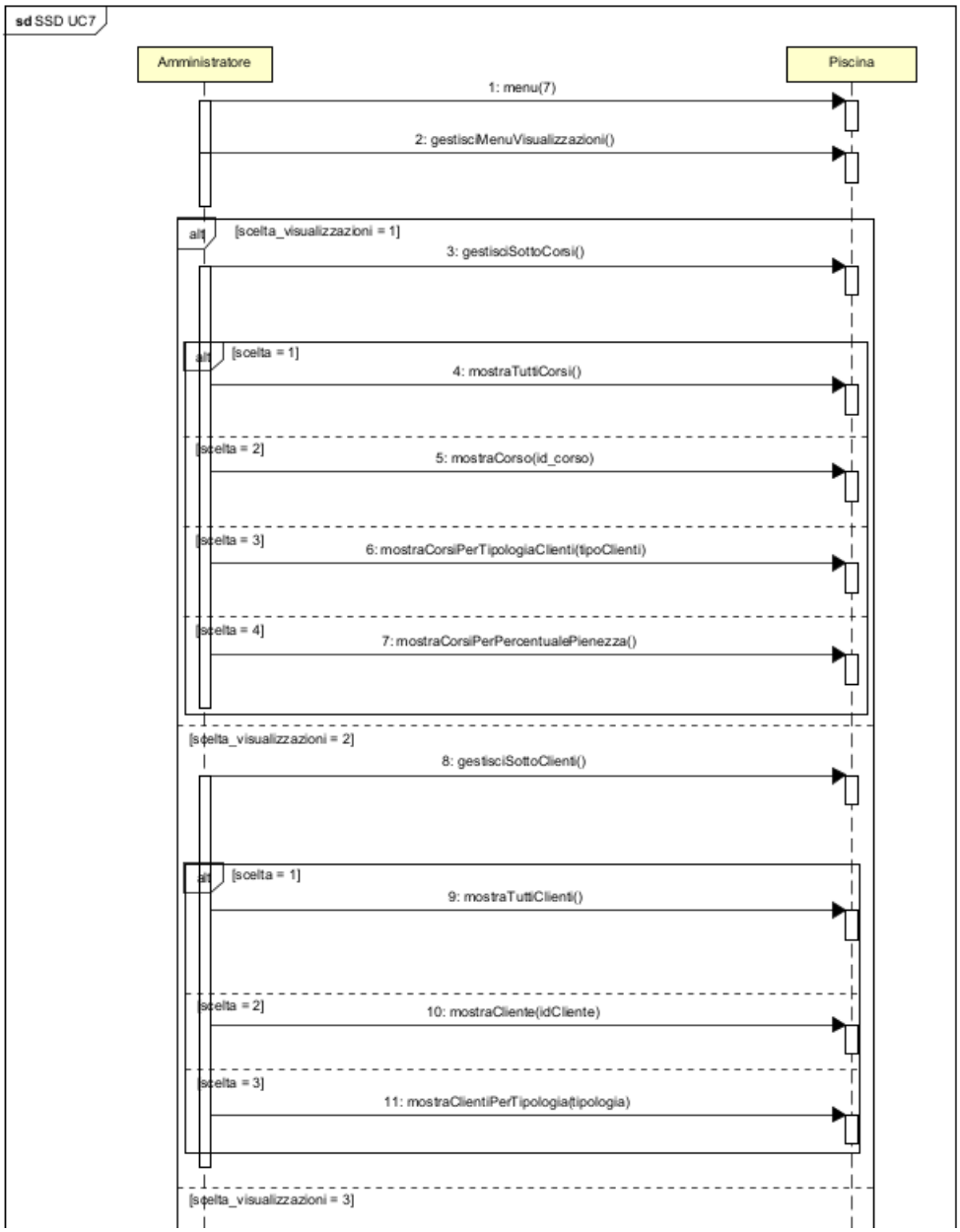
Pre-condizioni:	• Nessuna
Post-condizioni:	• Se esiste viene recuperata l'istanza cr di Corsia

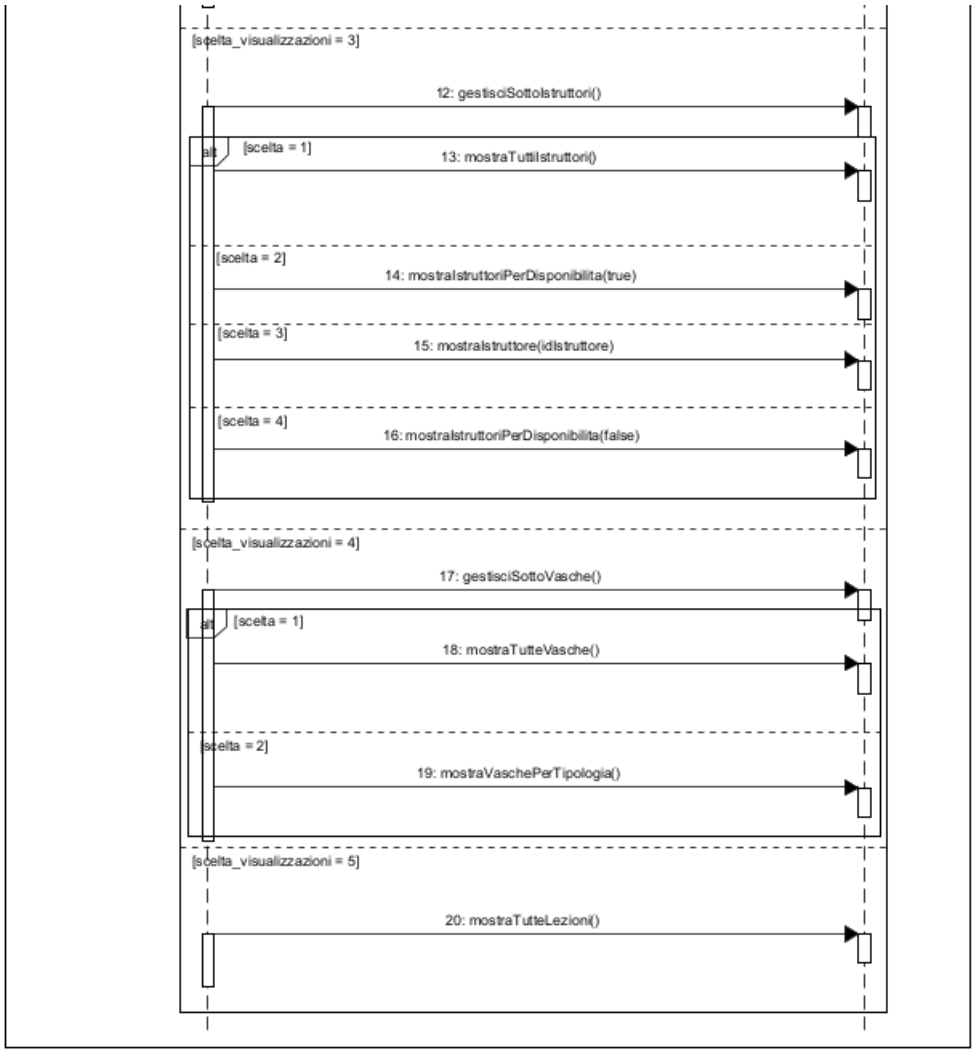
Riferimento:	UC6.2
Operazione:	AggiungiCorsia(id_corso, id_lezione, cr)
Pre-condizioni:	• Deve esistere un'istanza c di Corso • Deve esistere un'istanza l di Lezione associata all'istanza c di Corso • Deve esistere l'istanza cr di Corsia
Post-condizioni:	• Viene creata l'associazione tra l'istanza cr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



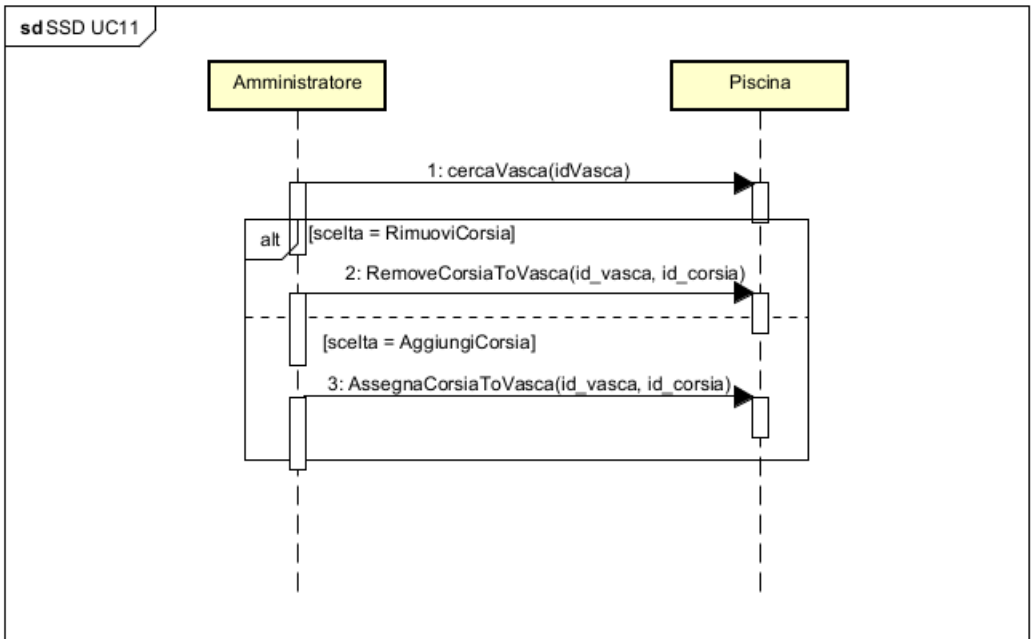
2.4.1 Contratti delle operazioni

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

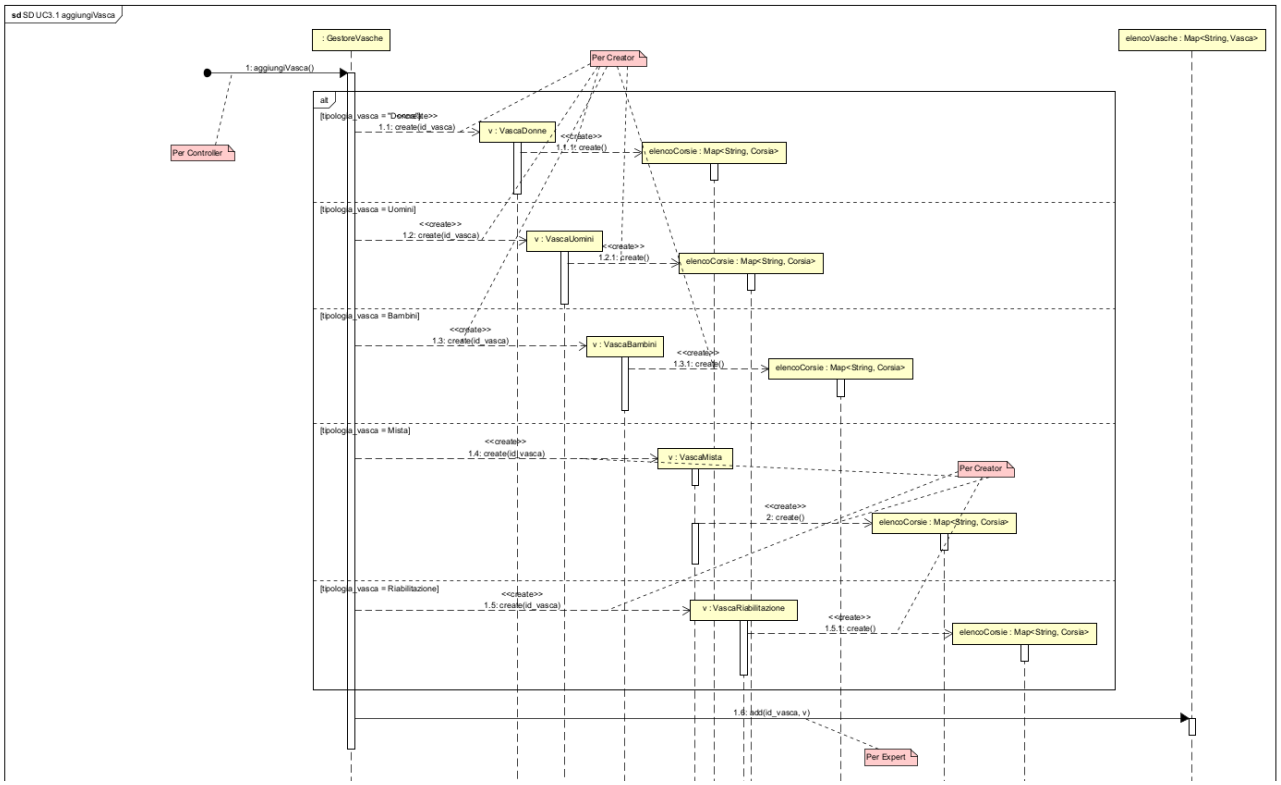
3.0 Fase di Progettazione

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.

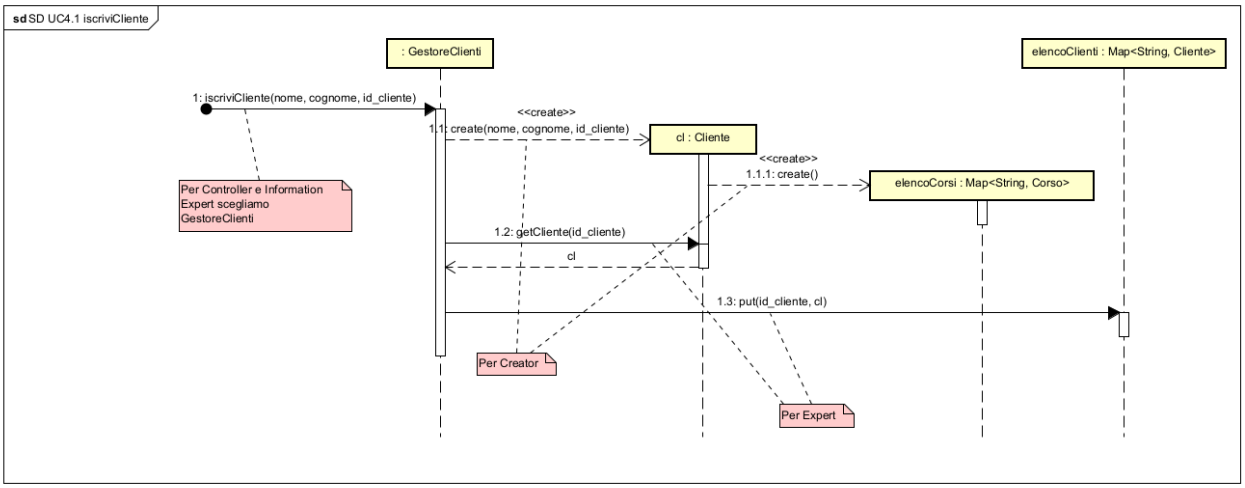
aggiungiVasca



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.

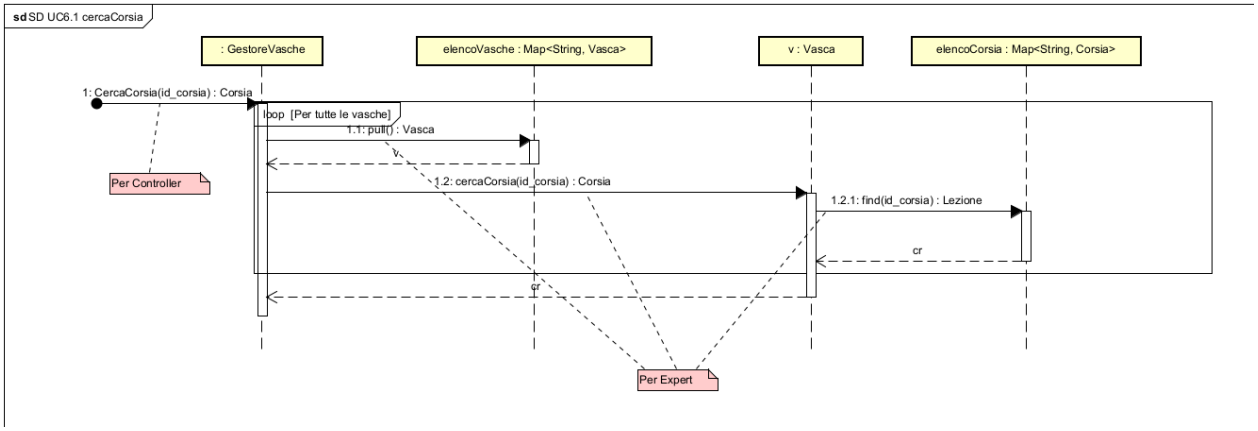
iscriviCliente



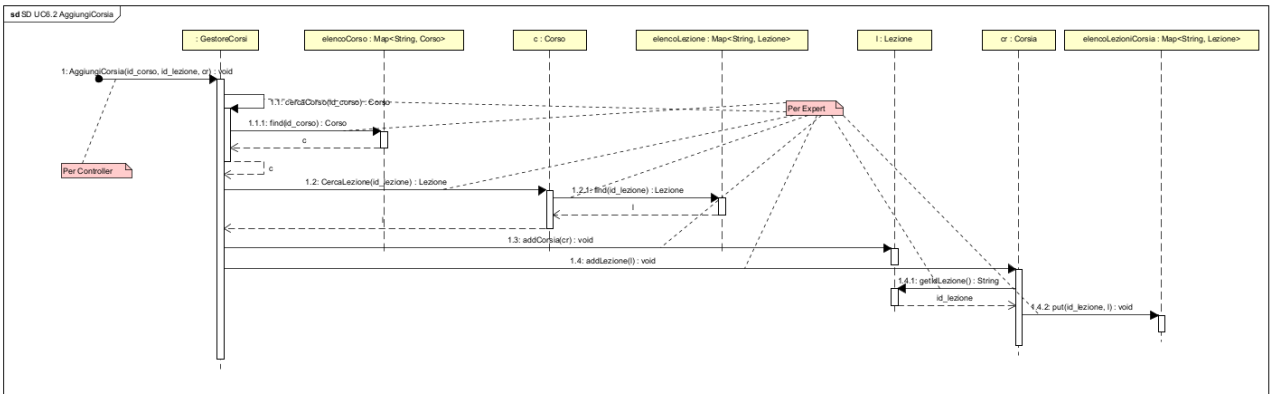
3.3 Diagrammi di Sequenza UC6

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



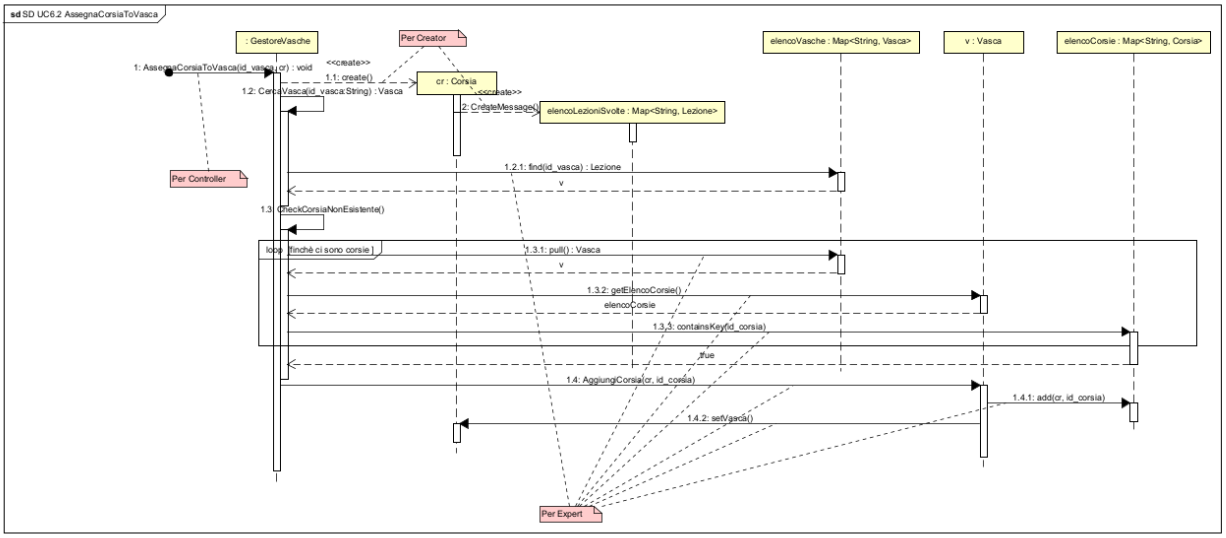
RemoveCorsiaToVasca



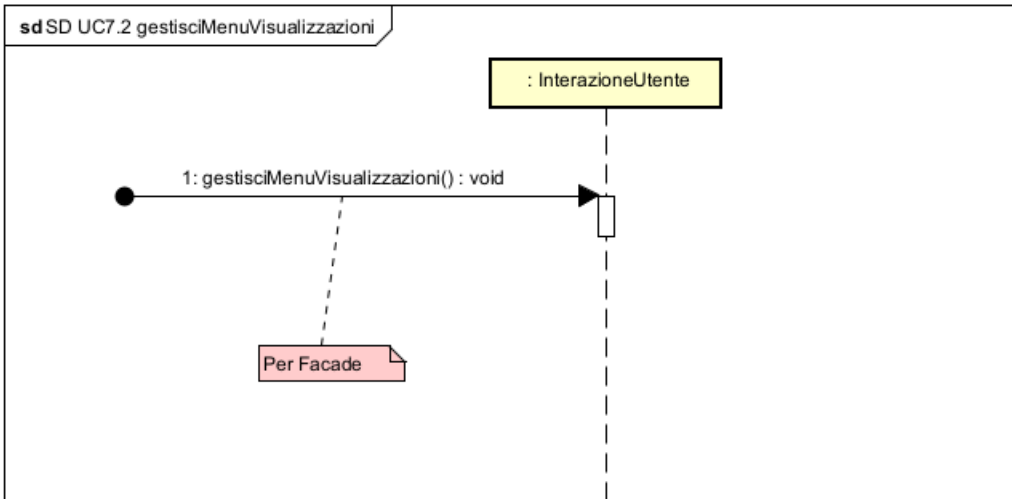
3.4 Diagrammi di Sequenza UC7

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.

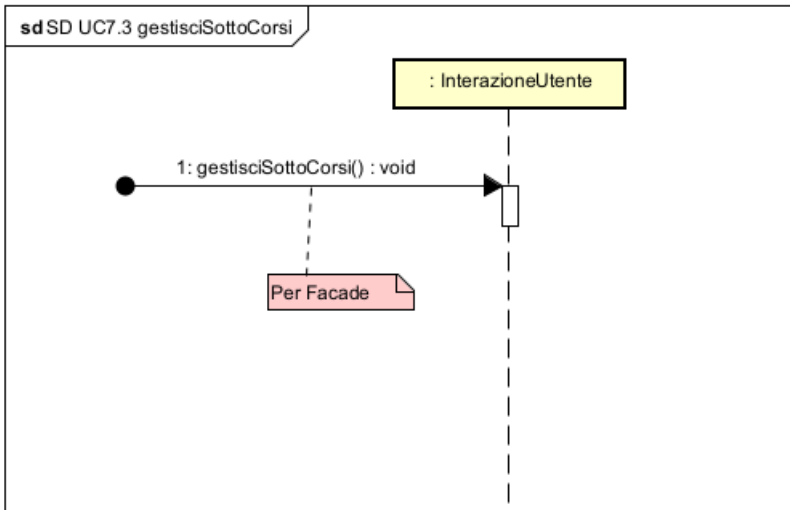
menu



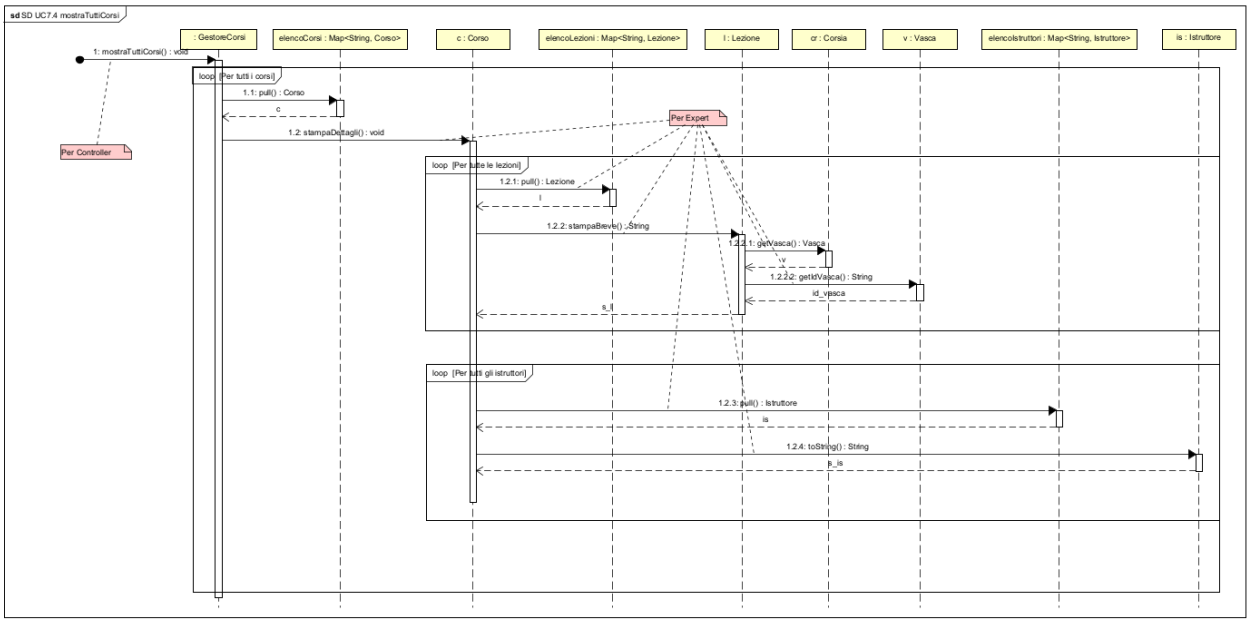
gestisciMenuVisualizzazioni



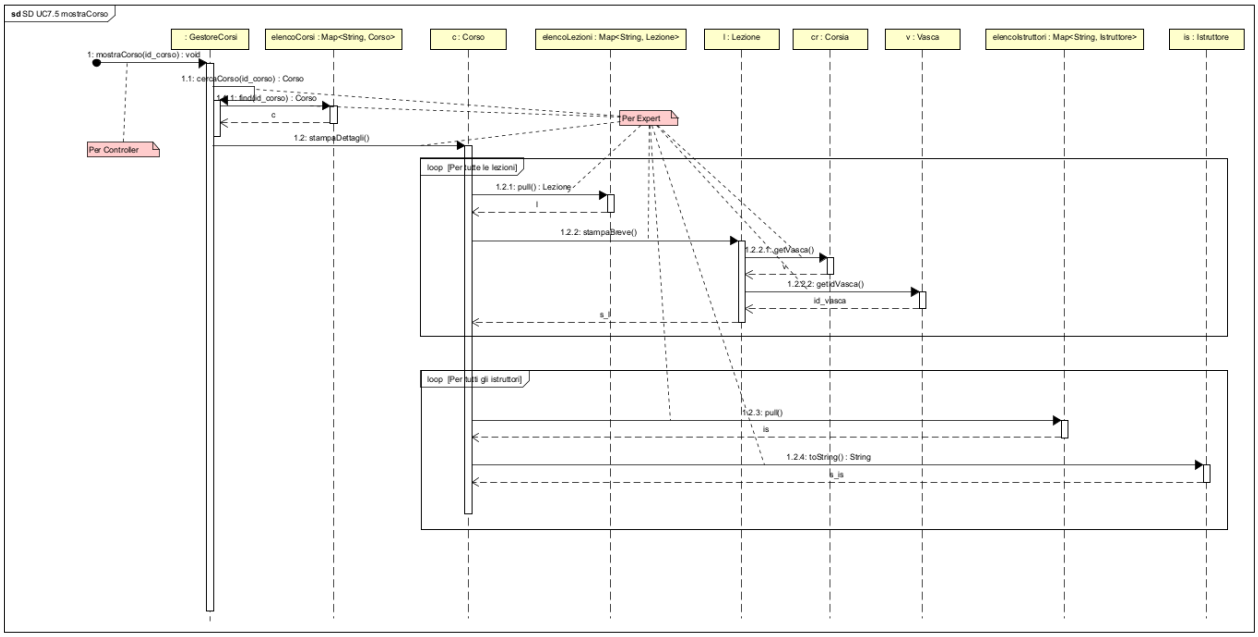
gestisciSottoCorsi



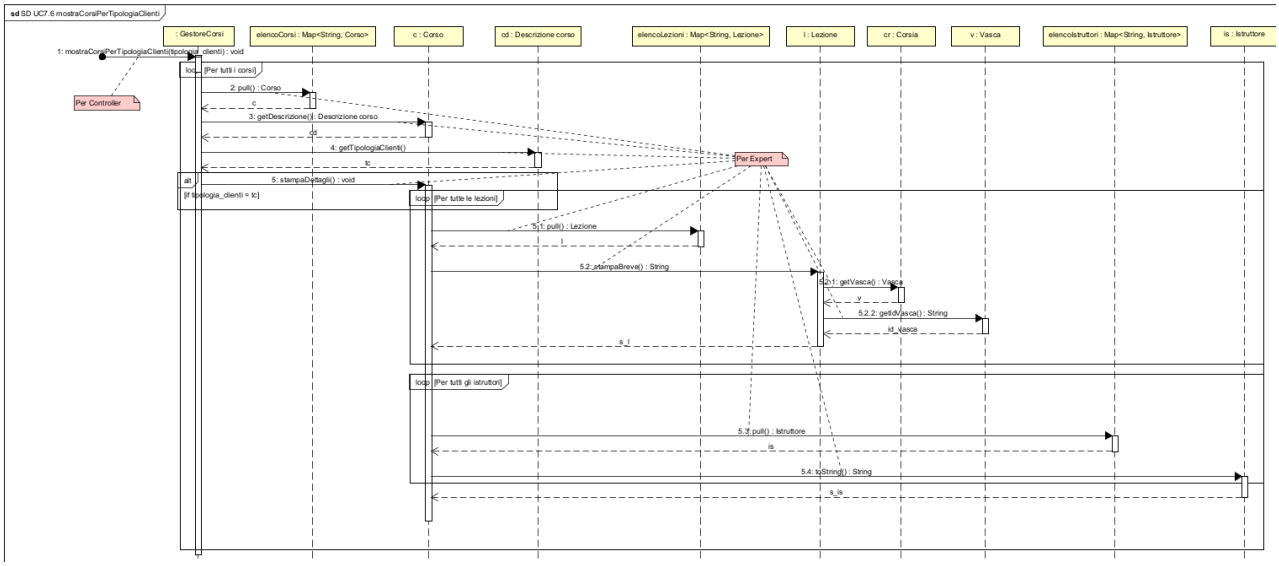
MostaTuttiCorsi



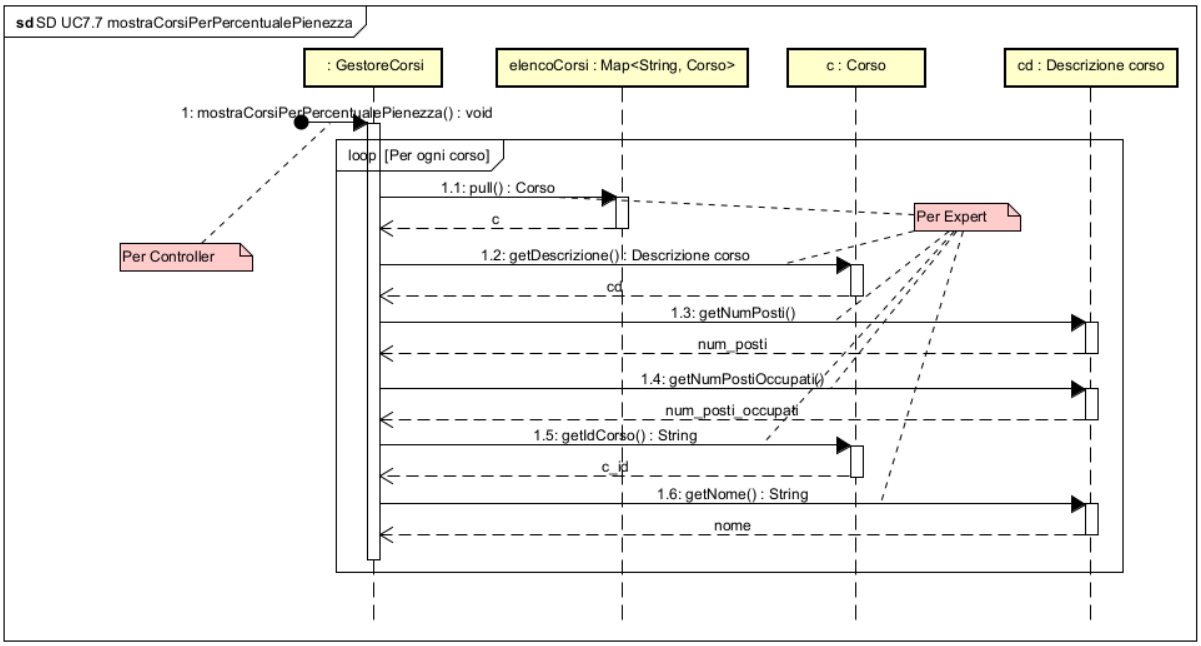
MostraCorso



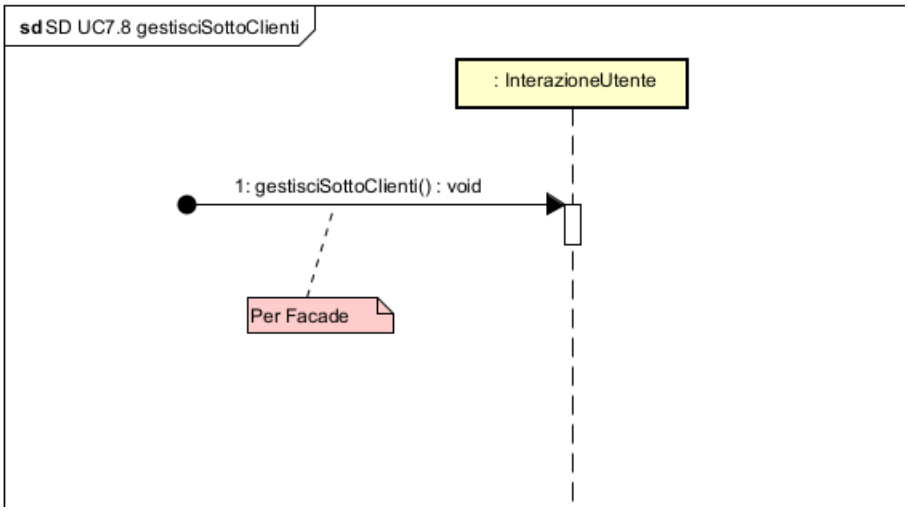
mostraCorsiPerTipologia



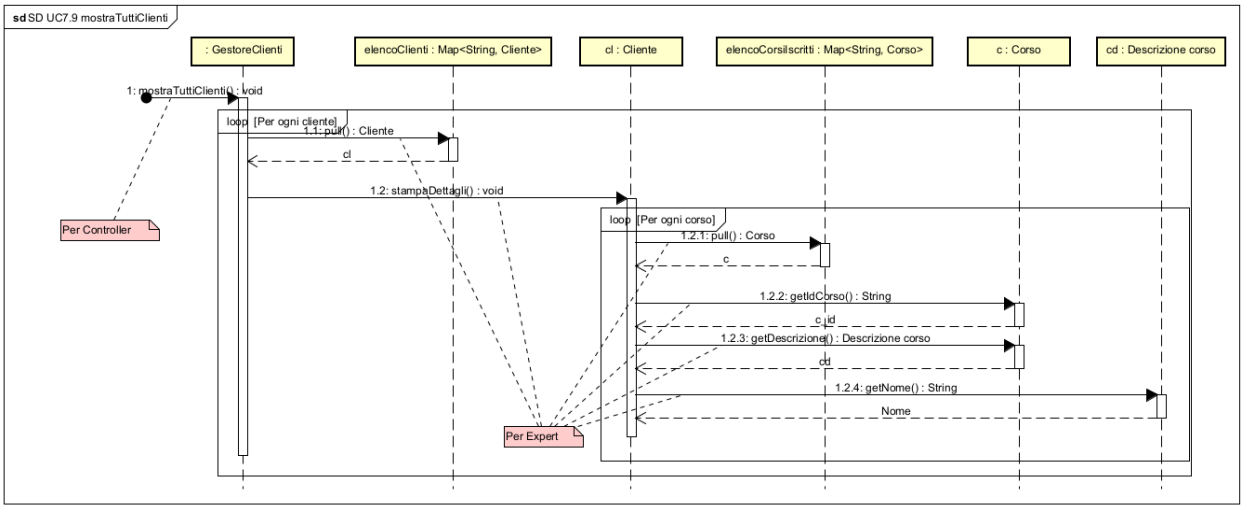
mostraCorsiPercentualePienezza



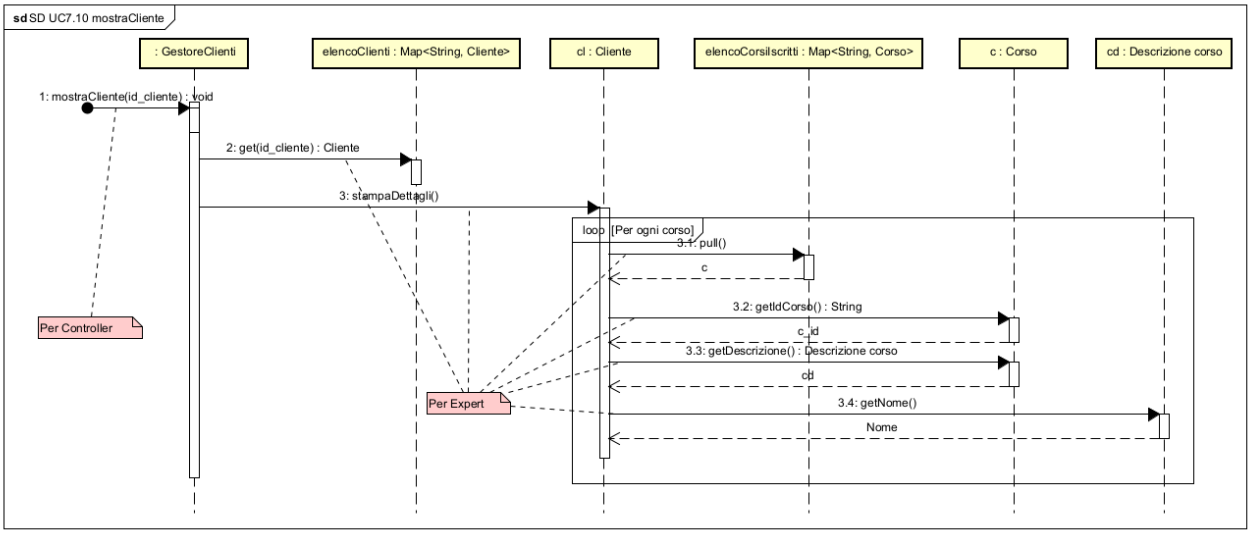
gestisciSottoClienti



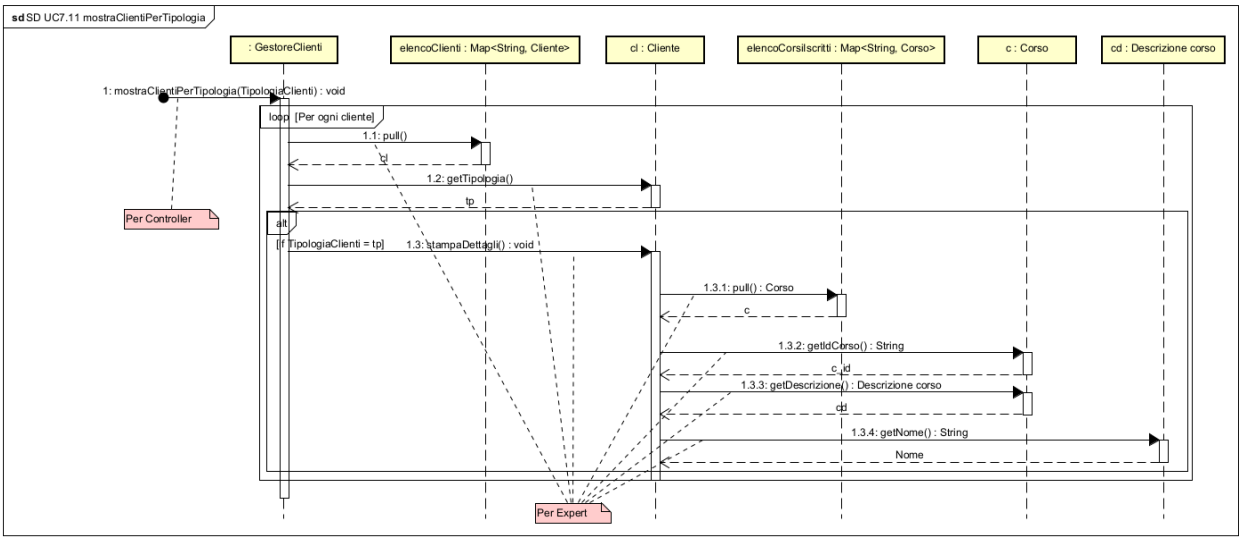
mostraTuttiClienti



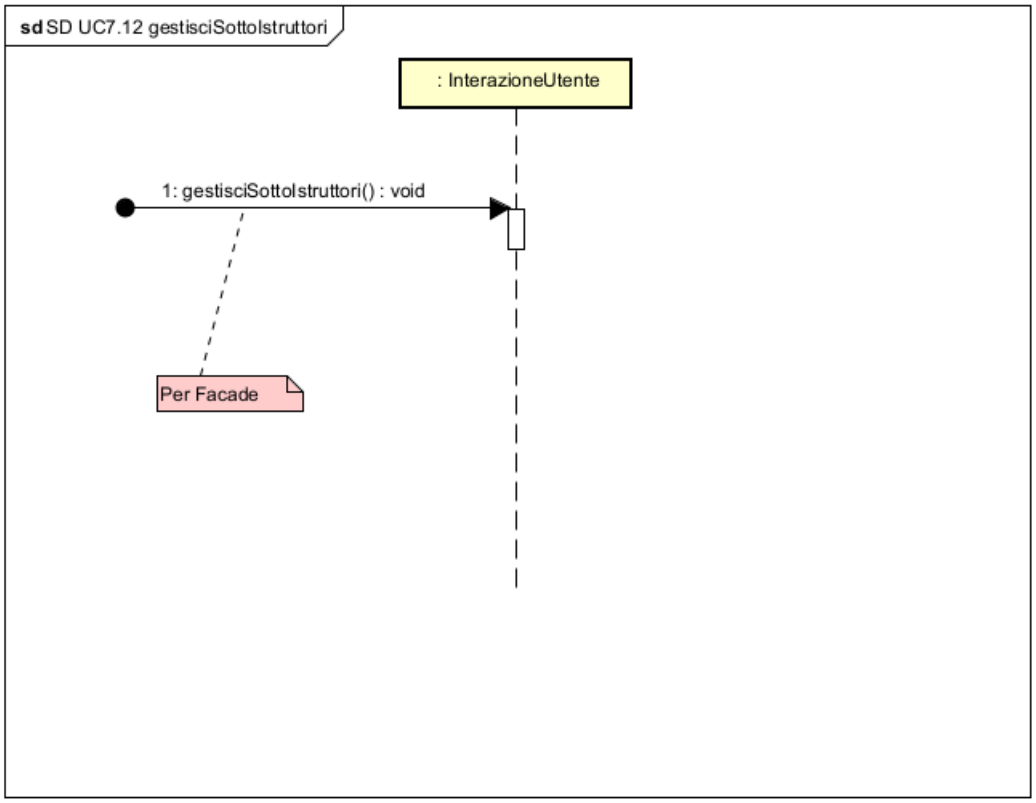
mostraCliente



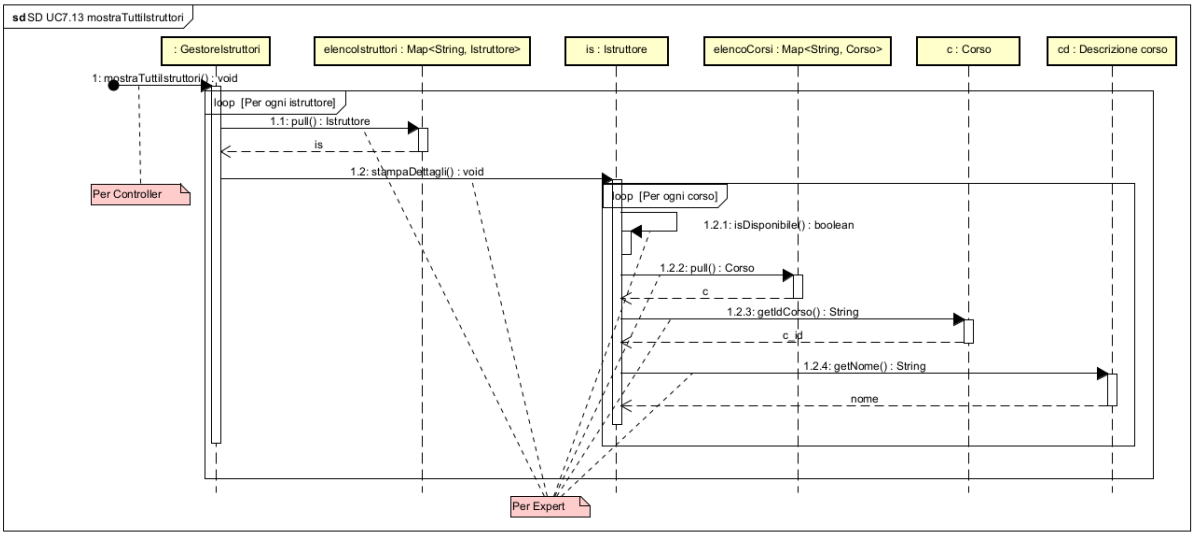
mostraClientiPerTipologia



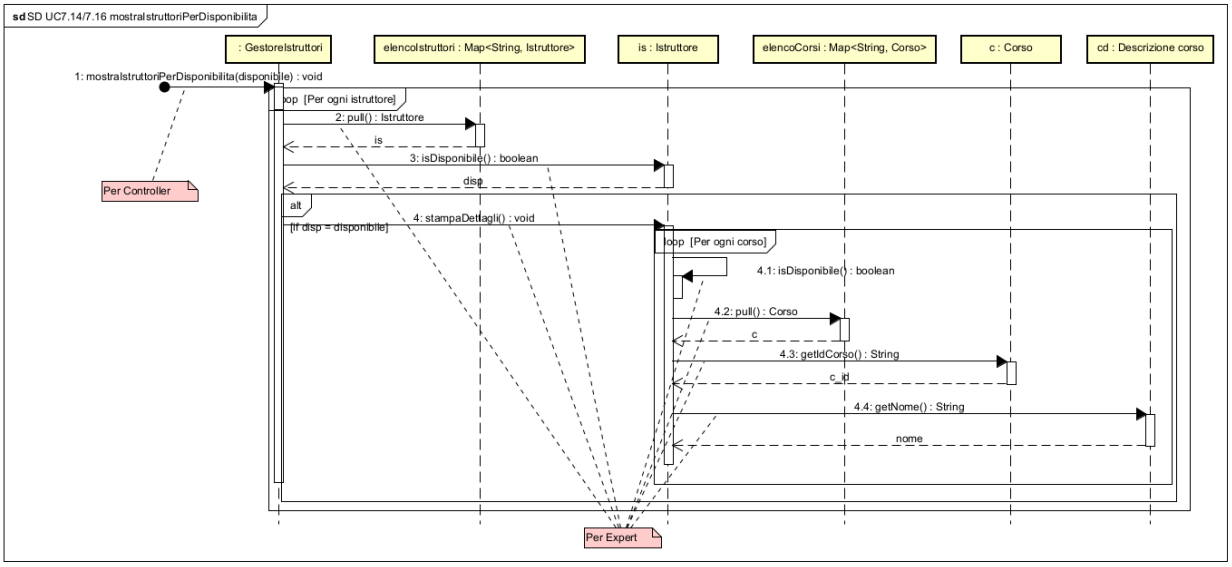
gestisciSottoIstruttori



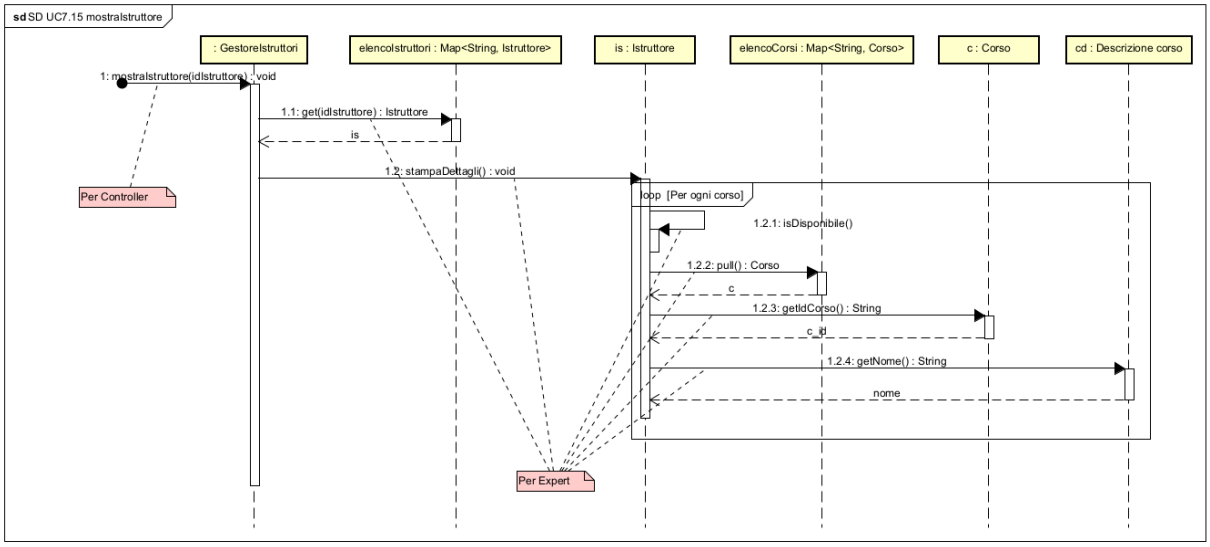
mostraTuttiIstruttori



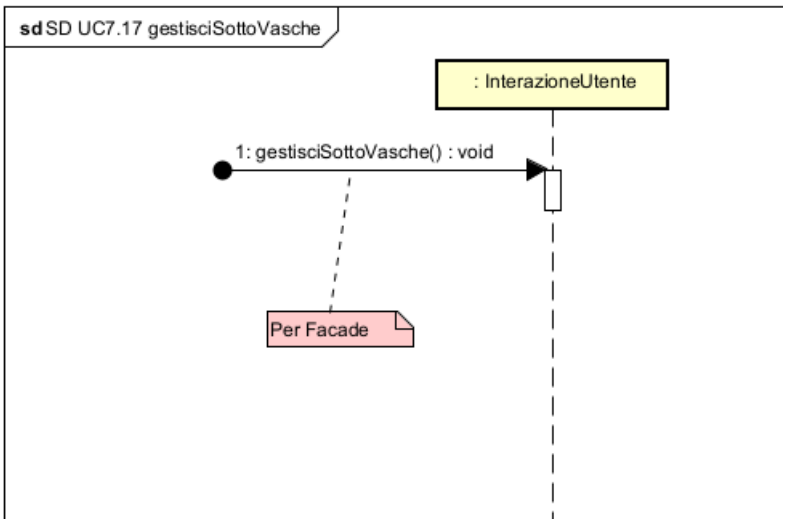
mostraIstruttoriPerDisponibilità



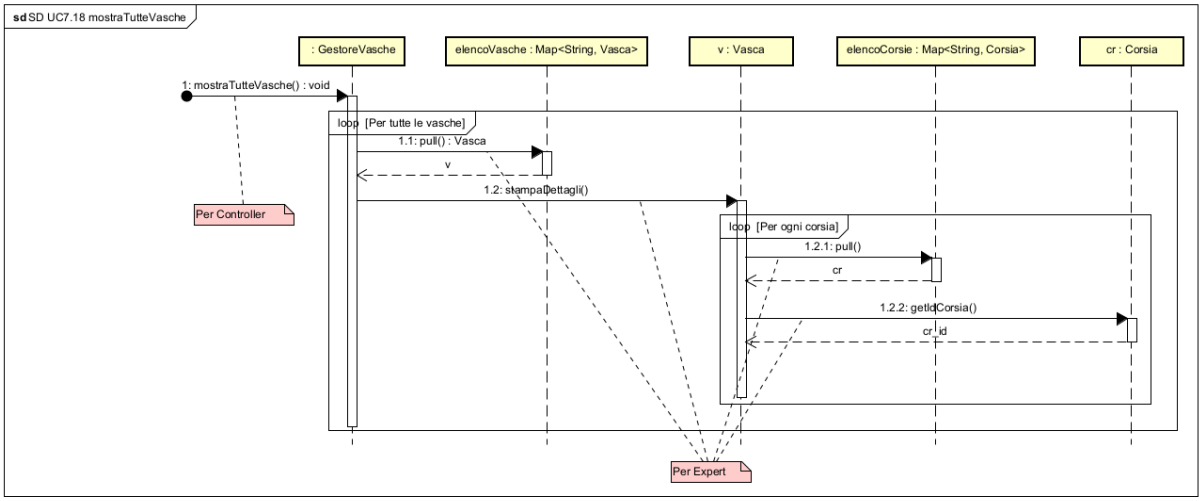
mostraIstruttore



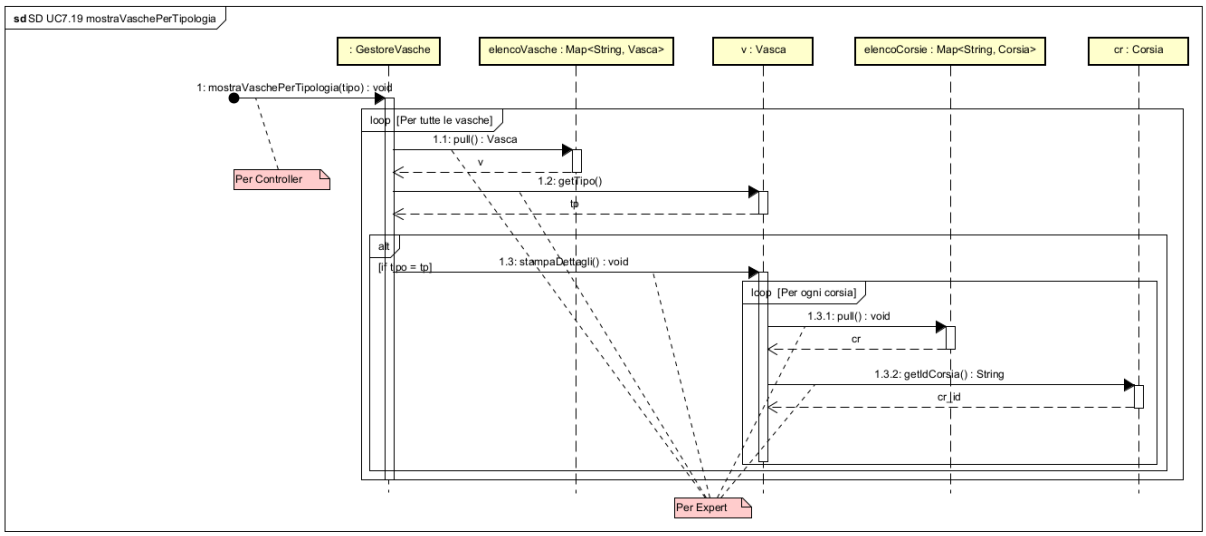
gestisciSottoVasche



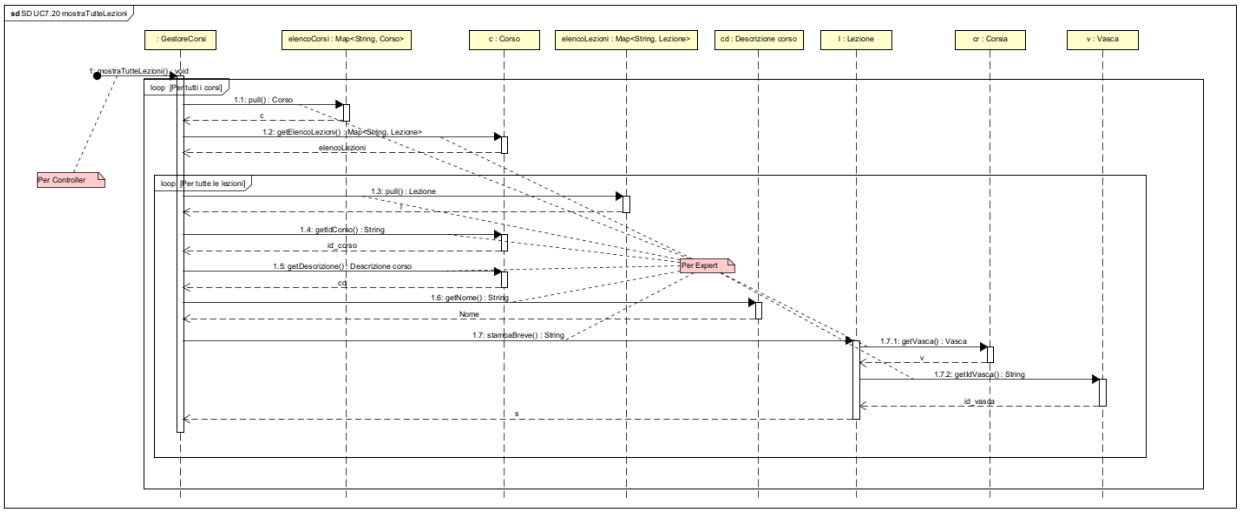
mostraTutteVasche



mostraVaschePerTipologia

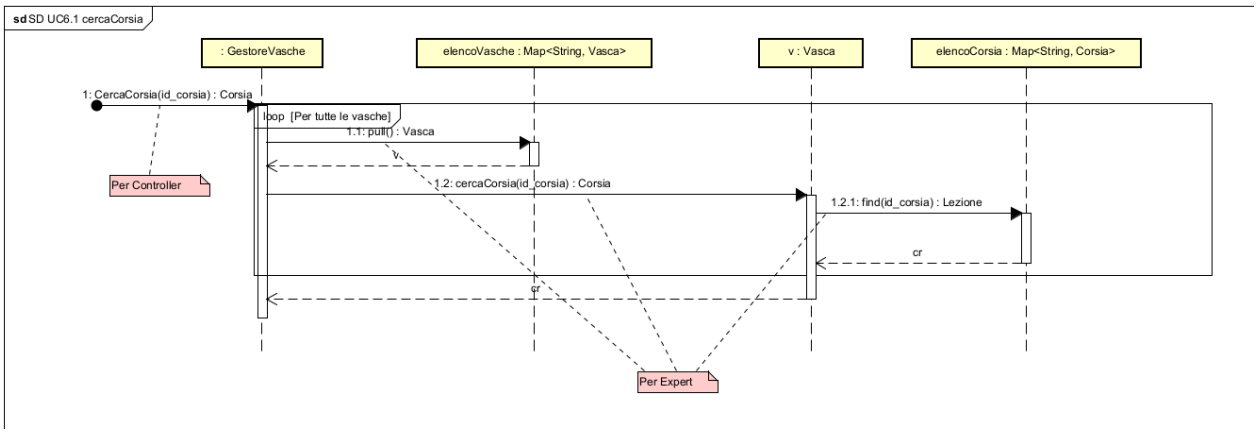


mostraTutteLezioni



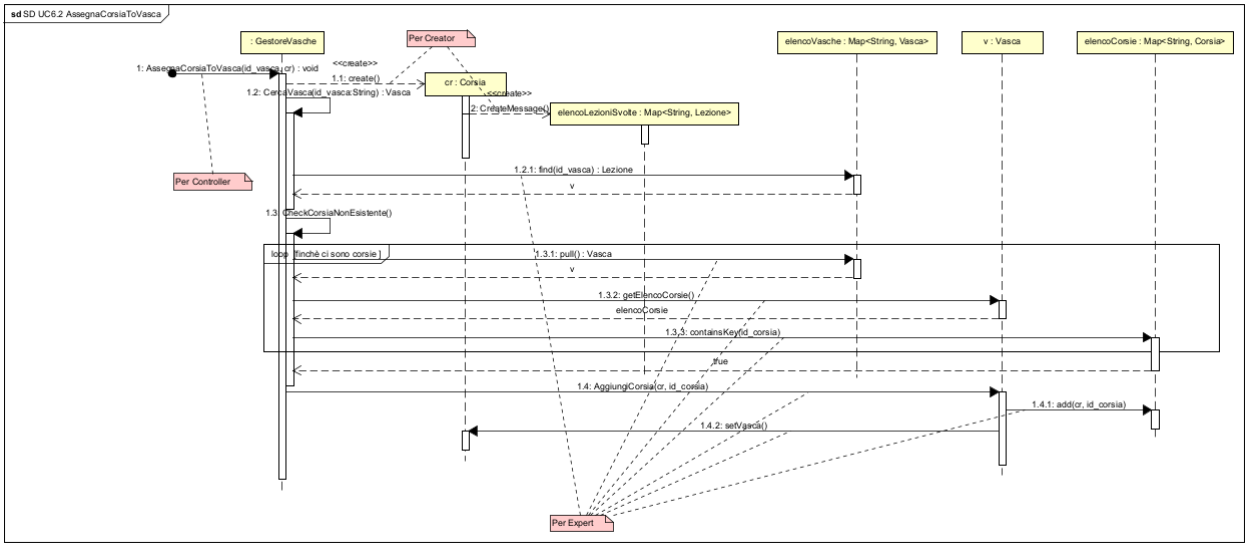
3.5 Diagrammi di Sequenza UC11

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

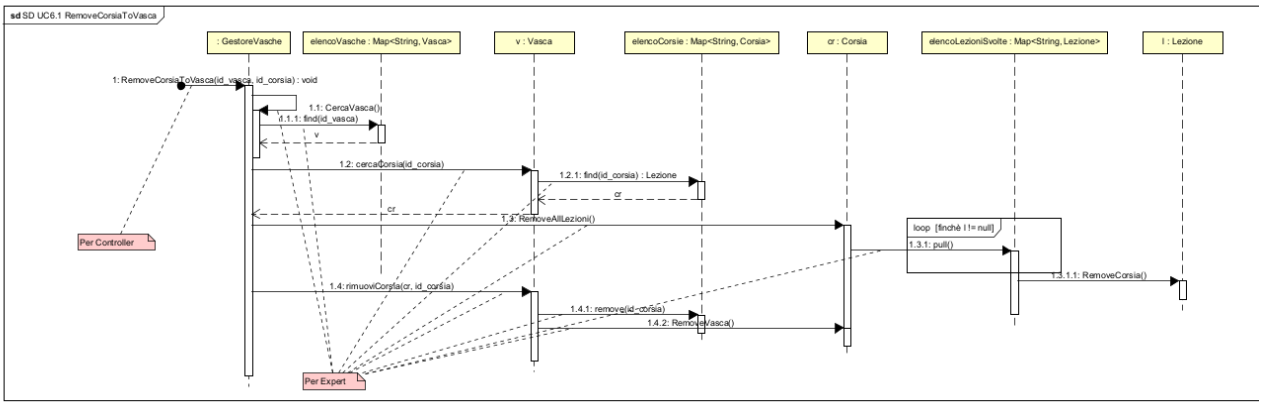


Vasca.

AssegnaCorsiaToVasca

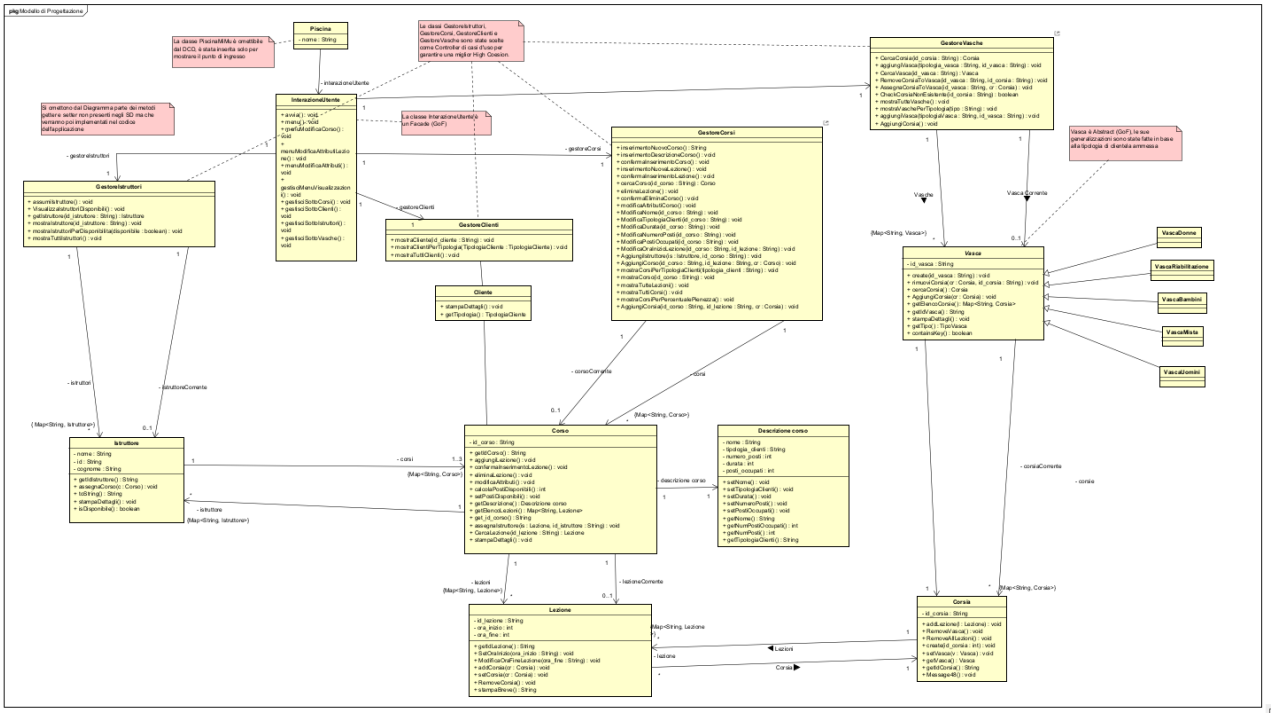


RemoveCorsiaToVasca



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.