UNIVERSITÀ
degli STUDI
di CATANIA

DIPARTIMENTO DI INGEGNERIA
ELETTRICA ELETTRONICA E INFORMATICA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

INGEGNERIA DEL SOFTWARE

SISTEMA BANCARIO

A.A. 2022/2023
Corso di Laurea Magistrale in Ingegneria Informatica

Alessandro Genovese
Davide Ferrauto

Prof. Orazio Tomarchio

Sommario

1. Requisiti 4
   1.1 Introduzione 4
   1.2 Casi d'uso 5
2. Iterazione 1, Analisi 7
   2.1. Introduzione 7
   2.2 Caso d'uso UC1, Modello di dominio 7
   2.3. Caso d'uso UC1, Diagramma di sequenza di sistema 9
3. Iterazione 2, Analisi 10
   3.1. Introduzione 10
   3.2. Caso d'uso UC1, Diagrammi di interazione 10
      3.2.1. inserisciNuovoCliente(...) 10
      3.2.2. inserisciConto(...) 11
      3.2.2. confermaOperazione() 12
   3.3. Caso d'uso UC1, Diagrammi delle classi di progetto 13
4. Iterazione 2, Analisi 14
   4.1. Introduzione 14
   4.2. Caso d'uso UC2/UC3, Modello di dominio 14
   4.3. Caso d'uso UC2, Diagramma di sequenza di sistema 15
   4.4. Caso d'uso UC3, Diagramma di sequenza di sistema 16
5. Iterazione 2, Progettazione 17
   5.1. Introduzione 17
   5.2. Caso d'uso UC2/UC3, Diagrammi di interazione 17
      5.2.1. UC2/UC3 verificaCarta(...) 17
      5.2.2. UC2 effettuaPrelievo(...) 18
      5.2.3. UC3 effettuaDeposito(...) 19
   5.3. Caso d'uso UC2/UC3, Diagrammi delle classi di progetto 19
6. Iterazione 3, Analisi 20
   6.1. Introduzione 20
   6.2. Caso d'uso UC4/UC5, Modello di dominio 20
   6.3. Caso d'uso UC4, Diagramma di sequenza di sistema 21
   6.4. Caso d'uso UC5, Diagramma di sequenza di sistema 22
7. Iterazione 3, Progettazione 23
   7.1. Introduzione 23
   7.2. Caso d'uso UC4/UC5, Diagrammi di interazione 23
      7.2.1. verificaCarta(...) vedi 5.2.1 23
      7.2.2. getCondizioni(...) 24

7.2.3. confermasPrestito(...) 24
7.2.4. richiediConsulente(...) 25
7.2.5. confermaConsulente(...) 25
7.3. Caso d'uso UC4/UC5, Diagrammi delle classi di progetto 26
8. Iterazione 4, Analisi 27
8.1. Introduzione 27
8.2. Caso d'uso UC6/UC7/UC8/UC9, Modello di dominio 27
8.3. Caso d'uso UC6, Diagramma di sequenza di sistema 28
8.4. Caso d'uso UC7, Diagramma di sequenza di sistema 28
8.5. Caso d'uso UC8a, Diagramma di sequenza di sistema 29
8.6. Caso d'uso UC8b, Diagramma di sequenza di sistema 29
8.7. Caso d'uso UC9, Diagramma di sequenza di sistema 30
9. Iterazione 4, Progettazione 31
9.1. Introduzione 31
9.2. Caso d'uso UC6/UC7/UC8/UC9, Diagrammi di interazione 31
9.2.1.UC6 stampaListaMovimenti(...) 31
9.2.2. UC7 stampa(...) 32
9.2.3. UC8a modificaTassoInteresse(...) 32
9.2.4. UC8b modificaMaxPrelevabile(...) 33
9.2.5. UC9 stampaPrestiti(...) 33
9.3. Caso d'uso UC6/UC7/UC8/UC9, Diagrammi delle classi di progetto 34
10. Implementazione 35
10.1. Introduzione 35
10.2. Strutture Dati 35
10.3. Test 36
10.3.1 Introduzione 36
10.3.2. Costruzione dei test 36
10.4. Git Practice e sviluppo iterativo (Agile) 37

1. Requisiti

1.1 Introduzione

Il progetto software che si vuole realizzare consiste nella gestione di un sistema bancario che memorizza le informazioni relative ai clienti e alle loro transazioni bancarie.
Un cliente ha la possibilità di aprire un conto corrente e decidere il tipo di conto a lui più conveniente, ad esempio un conto di tipo Gold o un conto di tipo Silver. Le differenze tra le varie tipologie sono le seguenti: canone annuo diverso, accesso a particolari servizi e la presenza di limitazioni al contante prelevabile in una giornata.
Ogni conto corrente ha un suo codice identificativo che si chiama IBAN (International Bank Account Number). L'IBAN identifica in maniera esclusiva il conto corrente e l'intermediario che lo detiene e consente di rendere pratica e veloce qualsiasi transazione finanziaria.
Il cliente ha la possibilità di effettuare un prelievo/deposito in contanti presso lo sportello della banca, inoltre è possibile effettuare un deposito tramite bonifico bancario.
Il cliente può richiedere il proprio estratto conto, con la lista dei movimenti degli ultimi quindici giorni, presso lo sportello della banca.
Il presente sistema bancario offre ai propri clienti la possibilità di scegliere un consulente finanziario per effettuare e gestire gli investimenti in borsa.
Ad ogni cliente verrà consegnata una carta di credito con cui poter effettuare i pagamenti nelle strutture abilitate e prelevare/depositare i contanti.
Il cliente può chiedere un prestito alla banca, la quale in base allo stipendio annuo del cliente deciderà se concedere o meno il prestito. Qualora venisse concesso la banca richiederà un tasso di interesse fisso in base al tipo di conto corrente del cliente. Ad esempio, per i clienti Silver del 5%, per quelli Gold del 4%, per quelli Platinum del 3%.

Oltre a questo documento, lo studio di caso è composto dal codice (workspace Intellij) di un'applicazione per la gestione di un sistema bancario, prodotto in diverse versioni. In particolare, è disponibile il codice per le varie iterazioni (1-2-3-4).

Tutte le versioni sono dotate di un'interfaccia utente a caratteri e i dati vengono memorizzati solo in memoria principale.

1.2 Casi d'uso

Si considerino i seguenti casi d'uso, di cui è di interesse solo lo scenario principale di successo.

- **UC1: Inserisci nuovo conto corrente**
  L'amministratore vuole gestire l'apertura di un nuovo conto corrente da parte di un nuovo cliente.

- **UC2: Gestisci prelievo**
  Il cassiere vuole gestire l'operazione bancaria "prelievo" da parte di un cliente.

- **UC3: Gestisci deposito**
  Il cassiere vuole gestire l'operazione bancaria "deposito" da parte di un cliente.

- **UC4: Gestisci prestito**
  Il cassiere vuole inserire nel sistema un nuovo prestito concesso dalla banca a un cliente.

- **UC5: Scelta consulente finanziario**
  Il cassiere vuole visualizzare la lista di tutti i consulenti finanziari, così da proporre un consulente finanziario al cliente in base al settore d'investimento scelto dal cliente.

- **UC6: Visualizza estratto conto**
  Il cassiere vuole che il sistema gli fornisca l'estratto conto del cliente, così da poterlo far visualizzare al cliente.

- **UC7: Visualizza conti correnti**
  L'amministratore del sistema vuole visualizzare le informazioni relative a tutti i conti correnti.

- **UC8: Modifica tassi d'interesse e tetto massimo al contante**
  L'amministratore del sistema vuole inserire/modificare i tassi d'interesse per i prestiti e il tetto massimo del contante prelevabile in una giornata, con riferimento alle tre tipologie di conto corrente.

- **UC9: Visualizza prestiti**
  L'amministratore del sistema vuole visualizzare tutti i prestiti che la banca ha concesso.

2. Iterazione 1, Analisi

2.1. Introduzione

Per l'iterazione 1, sono stati scelti i seguenti requisiti:

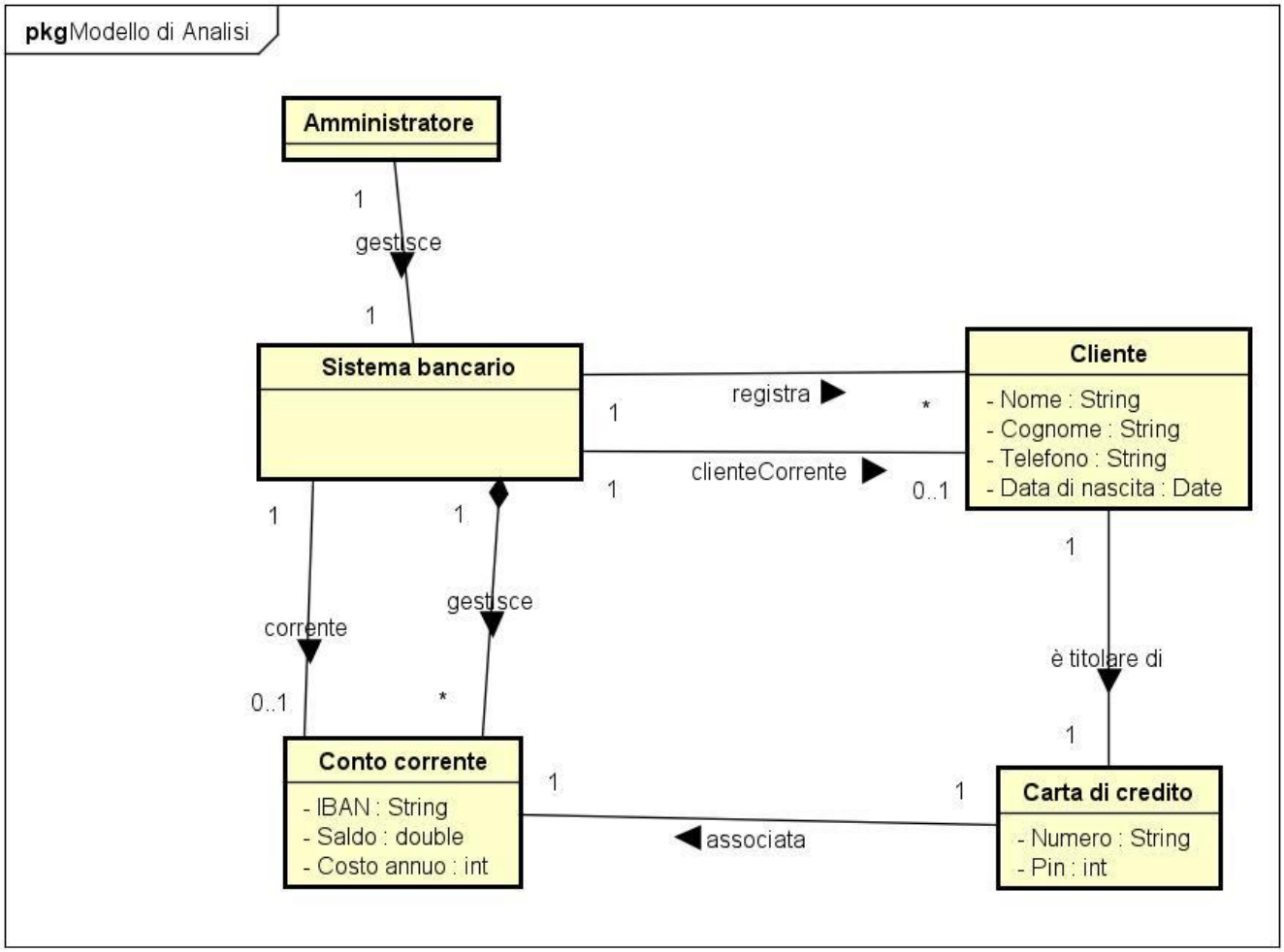
- lo scenario principale di successo del caso d'uso UC1 (Inserimento nuovo conto corrente);
- Caso d'uso d'avviamento;
- Dati solo in memoria principale.

Questo capitolo descrive l'analisi svolta nell'iterazione 1, mentre il capitolo successivo descrive l'attività di progettazione.

2.2 Caso d'uso UC1, Modello di dominio

In questa iterazione, del caso d'uso UC1 è di interesse lo scenario principale di successo, nella sua interezza, riportato nel Paragrafo 1.2. Da esso è possibile identificare le seguenti classi concettuali:

- Sistema Bancario: Rappresenta il sistema bancario
- Amministratore: Attore primario che interagisce direttamente con il sistema
- Cliente: Classe concettuale che rappresenta il cliente che vuole aprirsi un nuovo conto corrente
- Conto Corrente: Rappresenta il concetto conto corrente
- Carta di credito: Rappresenta la carta di credito relativa al conto corrente di un cliente

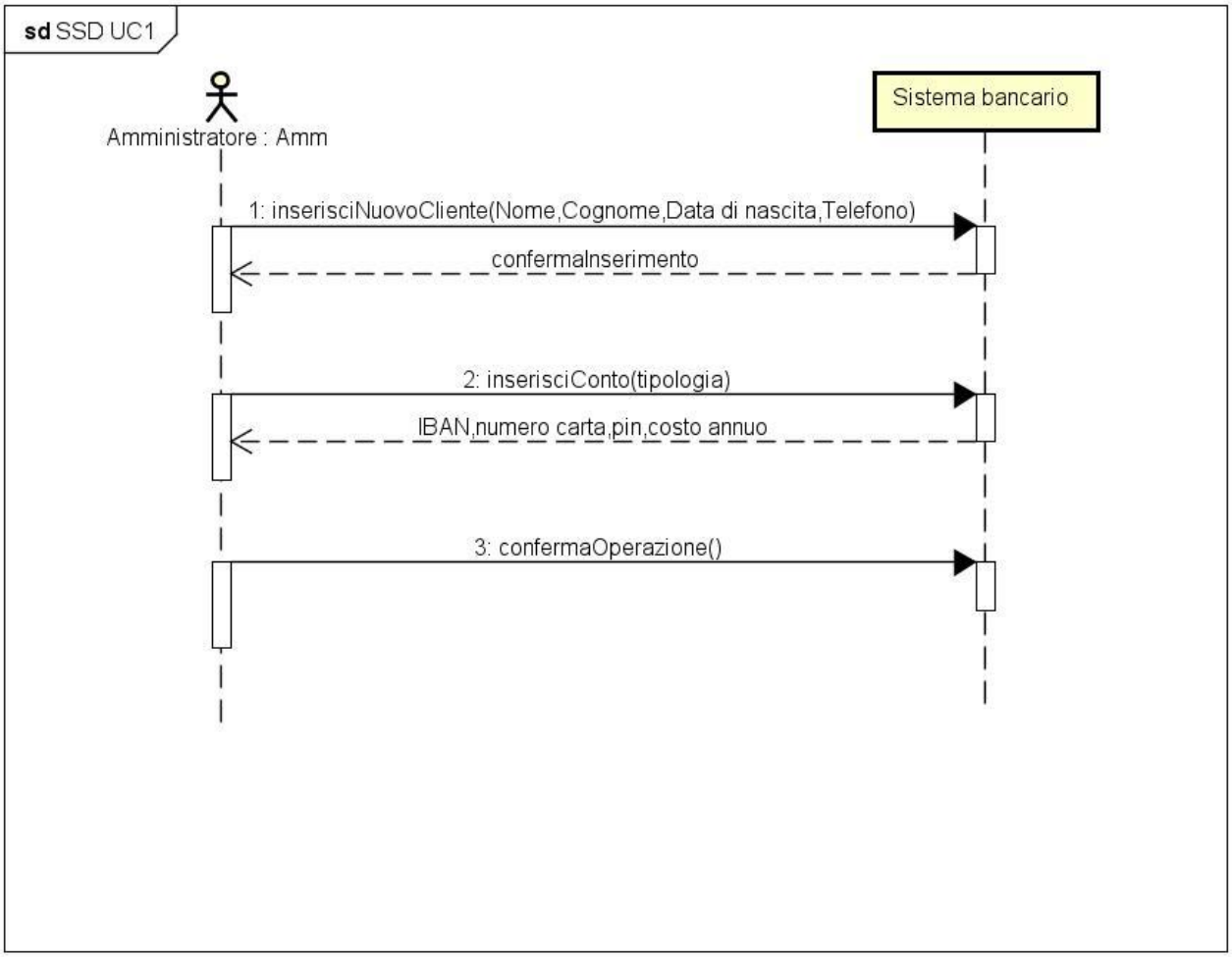


Il suddetto modello di dominio è una rappresentazione visuale di classi concettuali del mondo reale per il nostro dominio di interesse, che ci permette di mantenere basso il salto rappresentazionale. Quest'ultimo rappresenta la distanza tra il nostro modello mentale del dominio e la rappresentazione del dominio mediante il software.

L'associazione "corrente", tra le classi concettuali "Sistema Bancario" e "Conto corrente", rappresenta un legame logico tra le classi ed ha la funzione di creare un oggetto in più step, finché non viene confermato in Sistema Bancario.
Mentre l'associazione "gestisce" contiene i conti correnti precedentemente creati in una apposita struttura dati.

2.3. Caso d’uso UC1, Diagramma di sequenza di sistema

Il diagramma di sequenza di sistema per lo scenario principale di successo del caso d’uso UC1.



3. Iterazione 2, Analisi

3.1. Introduzione

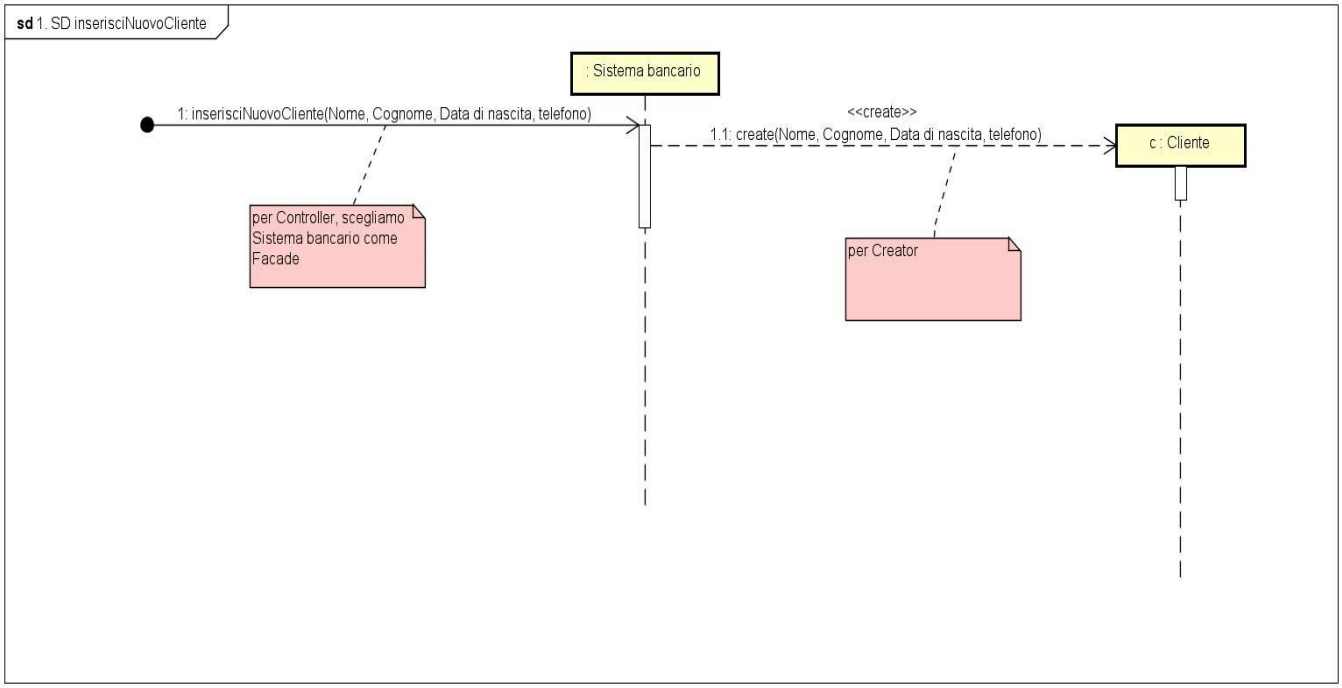
Prima di inizia la progettazione sono state prese le seguenti scelte:

- La classe concettuale amministratore non viene implementata perché aumenterebbe l'accoppiamento del sistema.

- Usare la classe sistema bancario come facade controller.

3.2. Caso d'uso UC1, Diagrammi di interazione

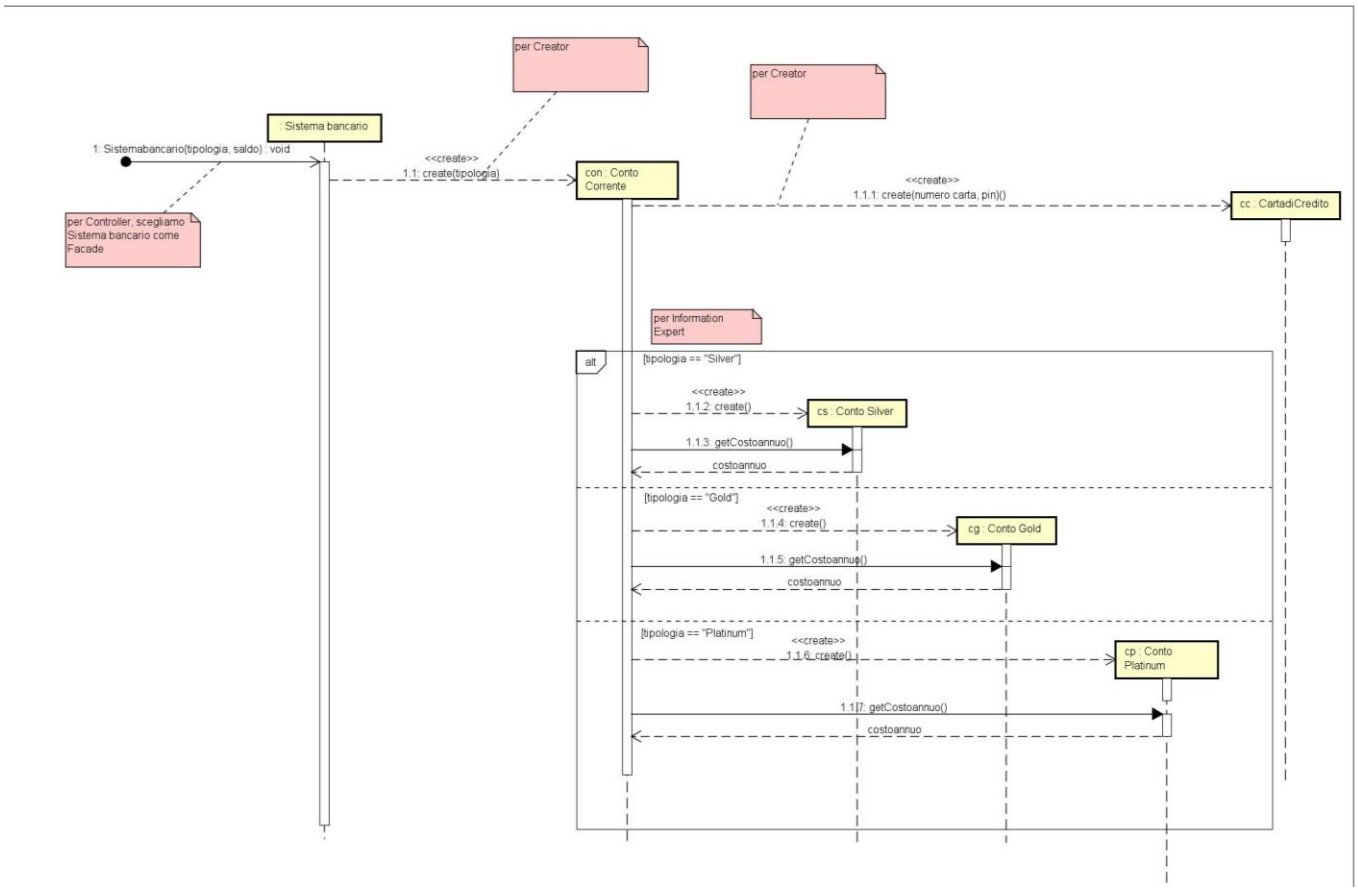
3.2.1. inserisciNuovoCliente(...)



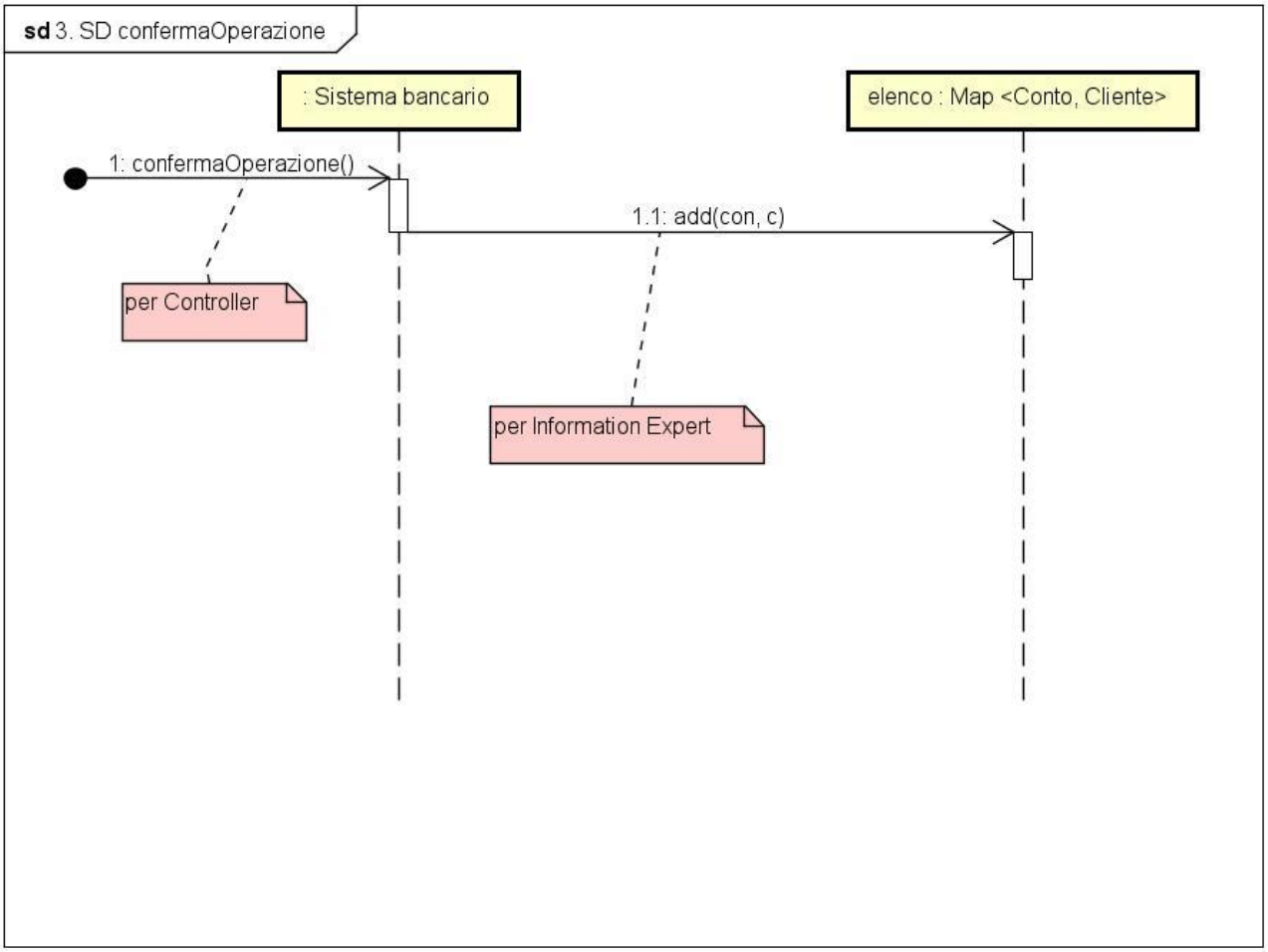
Si è utilizzato il **pattern GRASP Facade Controller** su "Sistema Bancario" perché quest'ultimo rappresenta il sistema complessivo, difatti è il primo oggetto a ricevere e gestire un'operazione di sistema.

Si è utilizzato il **pattern GRASP Creator**: "Sistema Bancario" ha la responsabilità di creare un oggetto Cliente sia perché possiede i dati per inizializzare Cliente e sia perché "Sistema Bancario" utilizza strettamente Cliente. Dunque, l'atto della creazione non incrementa l'accoppiamento in quanto "Cliente" deve essere già visibile a "Sistema Bancario".

3.2.2. inserisciConto(...)

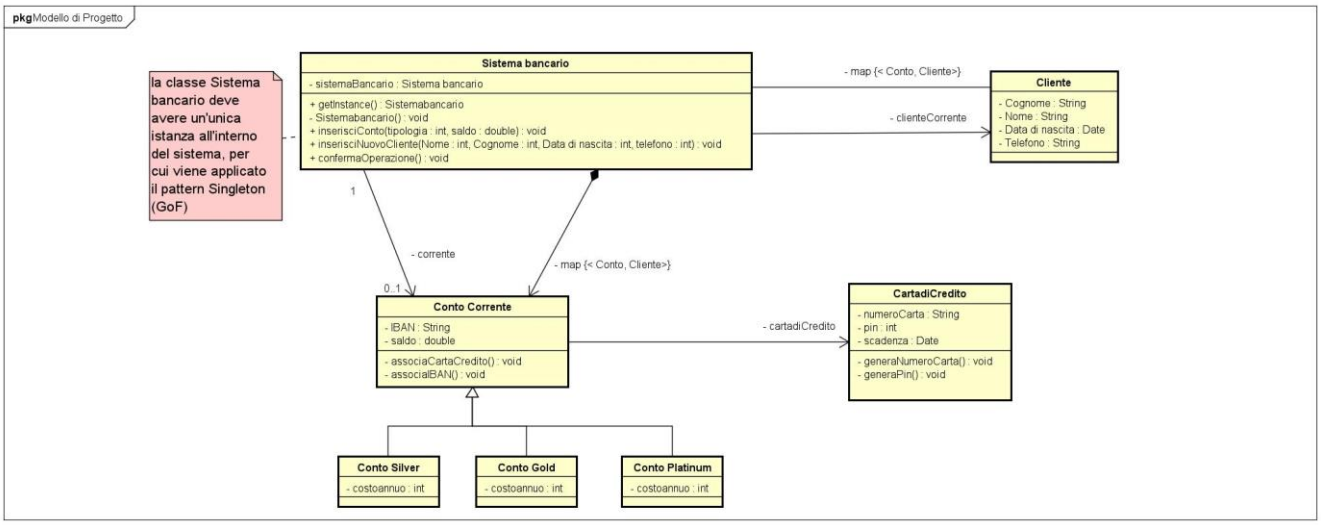


3.2.2. confermaOperazione ()



Si è utilizzato il **pattern GRASP Information Expert** su “Sistema Bancario” perché quest’ultimo possiede le informazioni per soddisfare l’aggiunta degli elementi nella struttura dati. Tale pattern supporta i principi di alta coesione e basso accoppiamento.

3.3. Caso d’uso UC1, Diagrammi delle classi di progetto



Si è utilizzato il **pattern GOF Singleton** su “Sistema Bancario” perché si vuole impedire la creazione di più istanze, in modo da avere un solo punto di accesso per l’esecuzione delle operazioni di sistema.

Viene considerata la specializzazione della classe “Conto corrente” in tre diverse sottoclassi, rispettivamente “Conto Silver”, “Conto Gold” e “Conto Platinum”.

4. Iterazione 2, Analisi

4.1. Introduzione

Per l'iterazione 2, sono stati scelti i seguenti requisiti:

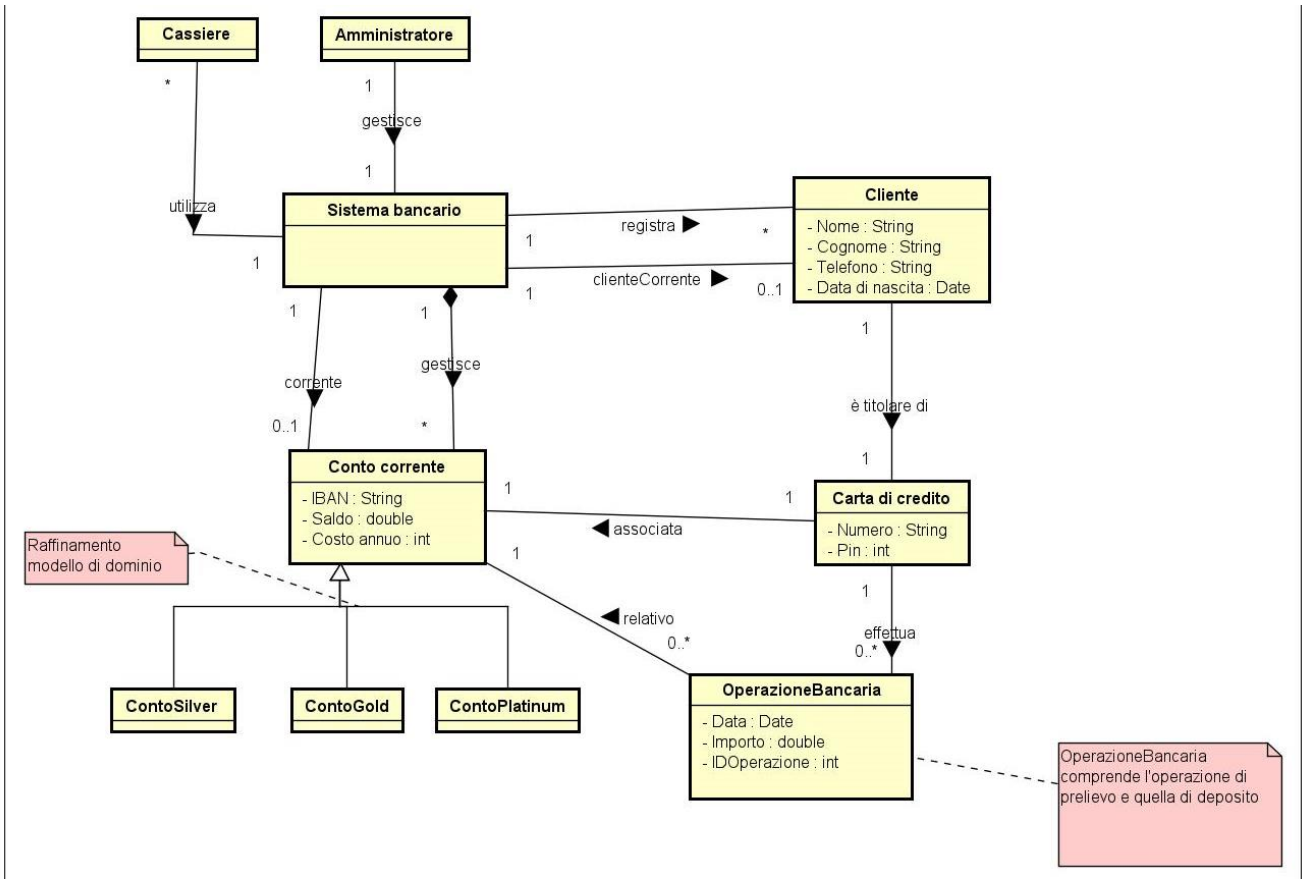
- lo scenario principale di successo del caso d'uso UC2 (gestisci prelievo) e del caso d'uso UC3(gestisci deposito);
- Dati solo in memoria principale.

Questo capitolo descrive l'analisi svolta nell'iterazione 2, mentre il capitolo successivo descrive l'attività di progettazione.

4.2. Caso d'uso UC2/UC3, Modello di dominio

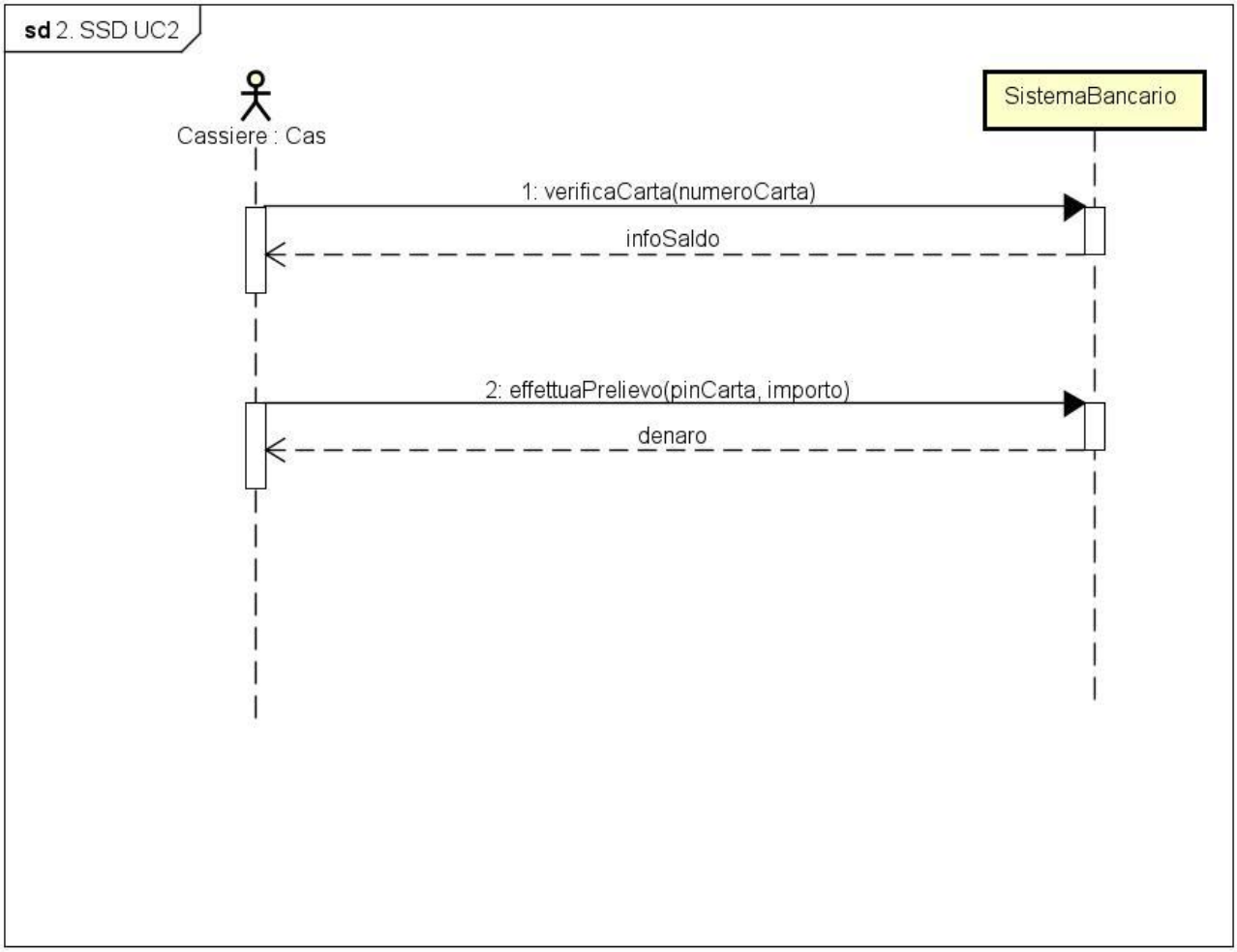
- Cassiere: Attore primario che interagisce direttamente con il sistema

- Operazione bancaria: Classe concettuale che rappresenta l'operazione che il sistema bancario può effettuare



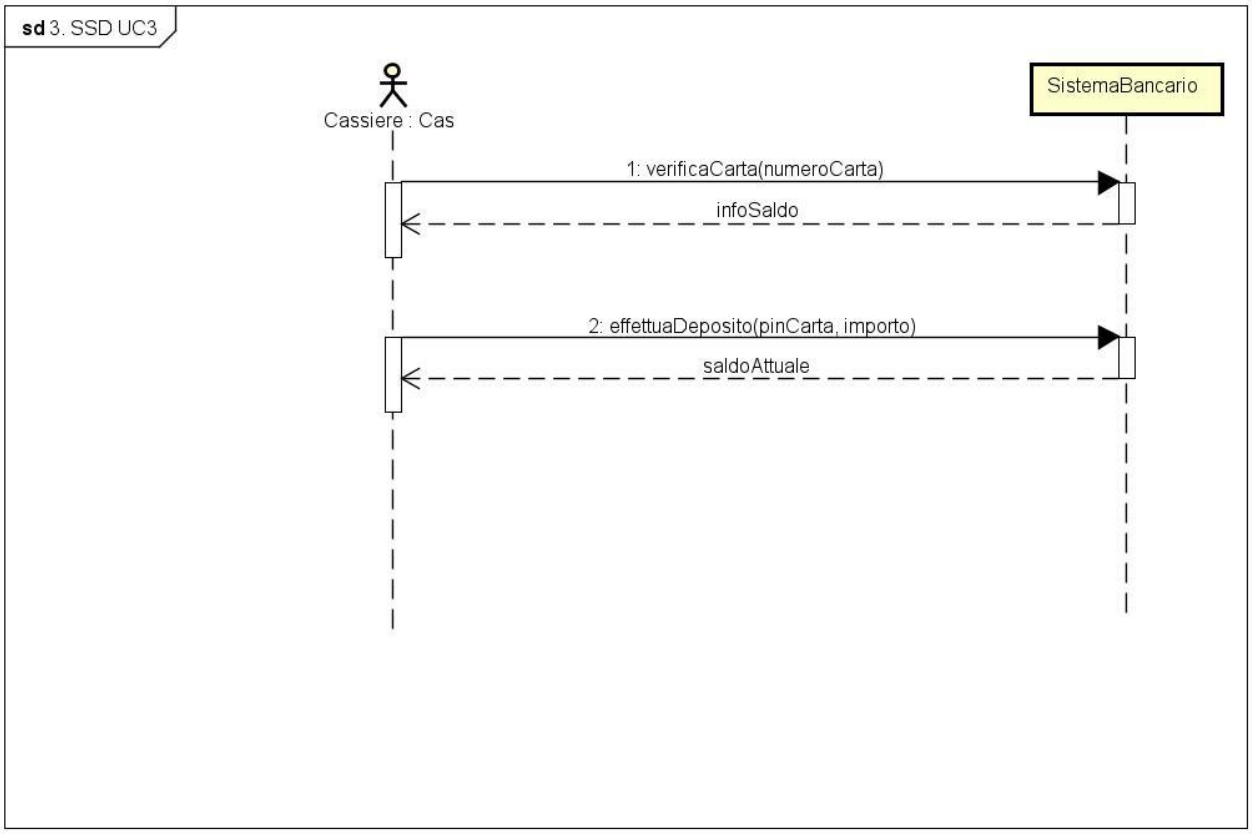
4.3. Caso d’uso UC2, Diagramma di sequenza di sistema

Il diagramma di sequenza di sistema per lo scenario principale di successo del caso d’uso UC2.



4.4. Caso d’uso UC3, Diagramma di sequenza di sistema

Il diagramma di sequenza di sistema per lo scenario principale di successo del caso d’uso UC3.



5. Iterazione 2, Progettazione

5.1. Introduzione

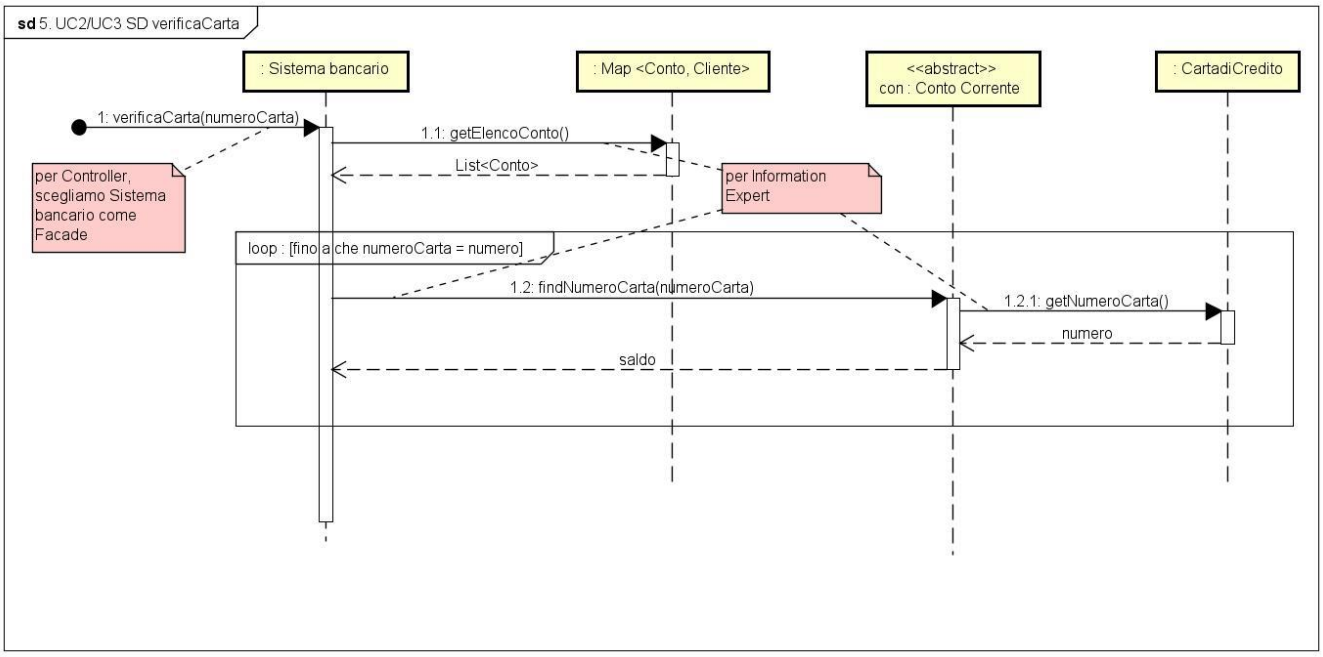
Prima di inizia la progettazione sono state prese le seguenti scelte:

- La classe concettuale cassiere non viene implementata perché aumenterebbe l'accoppiamento del sistema.

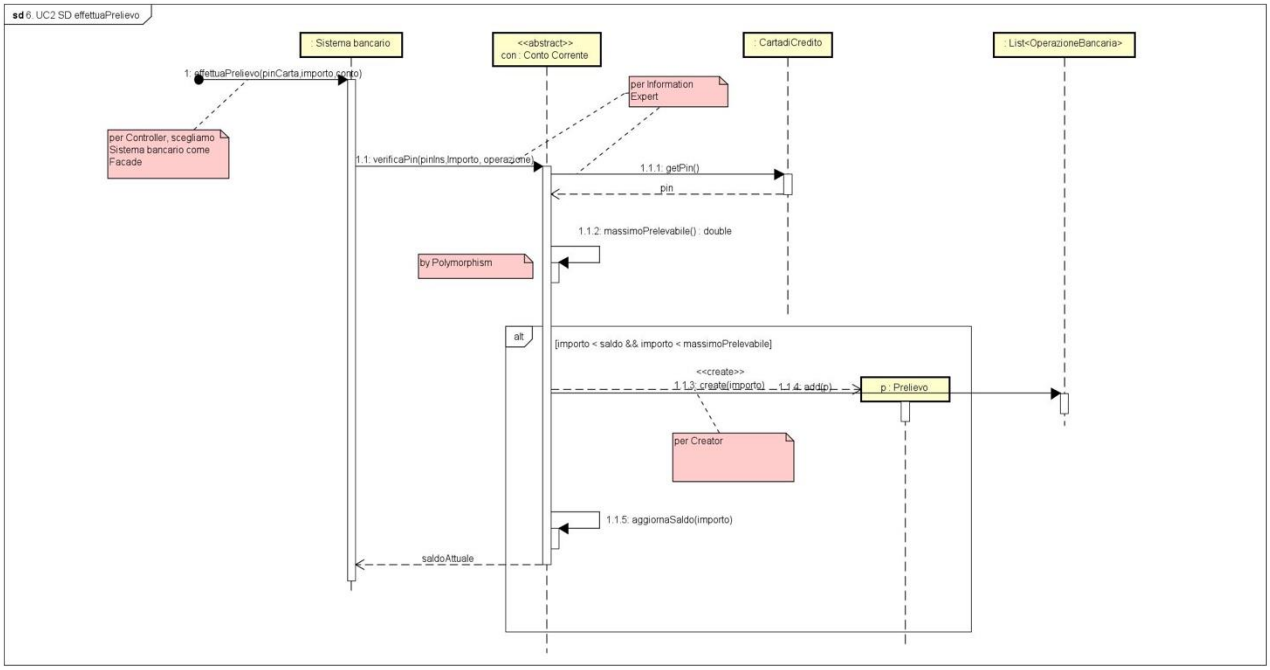
- Implementare la classe "OperazioneBancaria" come classe astratta e le relative sottoclassi "Prelievo", "Deposito".

5.2. Caso d'uso UC2/UC3, Diagrammi di interazione

5.2.1. UC2/UC3 verificaCarta(...)

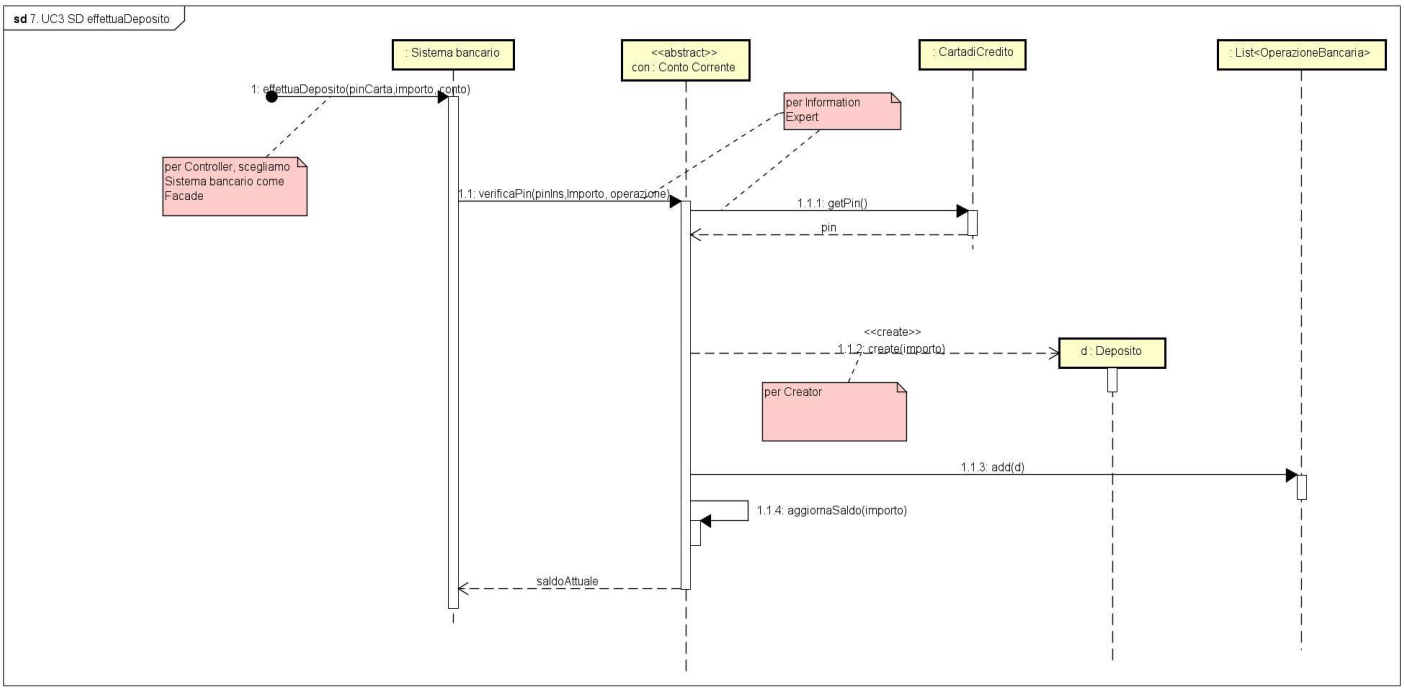


5.2.2. UC2 effettuaPrelievo(...)

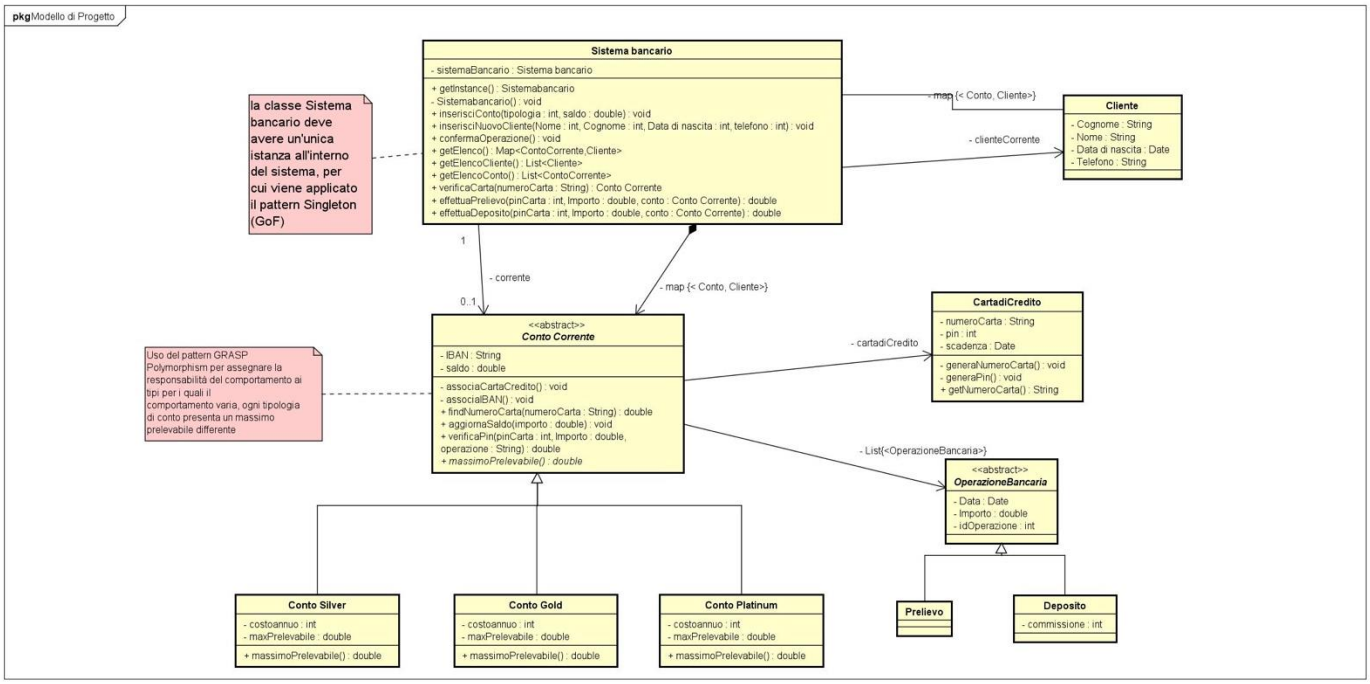


Si è utilizzato il **pattern GRASP Polymorphism** poiché il comportamento dell'operazione in questione varia sulla base del tipo di Conto Corrente da cui prelevare l'importo. Questo permette di rispettare le regole di dominio R4, R5 ed R6, le quali impongono che ogni tipologia di conto corrente abbia un importo massimo prelevabile diverso.
Per l'utilizzo di tale pattern è stata implementato un metodo abstract "massimoPrelevabile" nella classe abstract Conto Corrente.
Un ulteriore modo con cui era possibile gestire tale problema era quello di utilizzare delle istruzioni condizionali, tale soluzione però avrebbe portato degli svantaggi ovvero renderebbe difficoltosa, nella fase di manutenzione, l'aggiunta di nuove tipologie di conto corrente. Mentre l'utilizzo del pattern in questione permette di rendere più flessibile il programma sviluppato.

5.2.3. UC3 effettuaDeposito(...)



5.3. Caso d'uso UC2/UC3, Diagrammi delle classi di progetto



6. Iterazione 3, Analisi

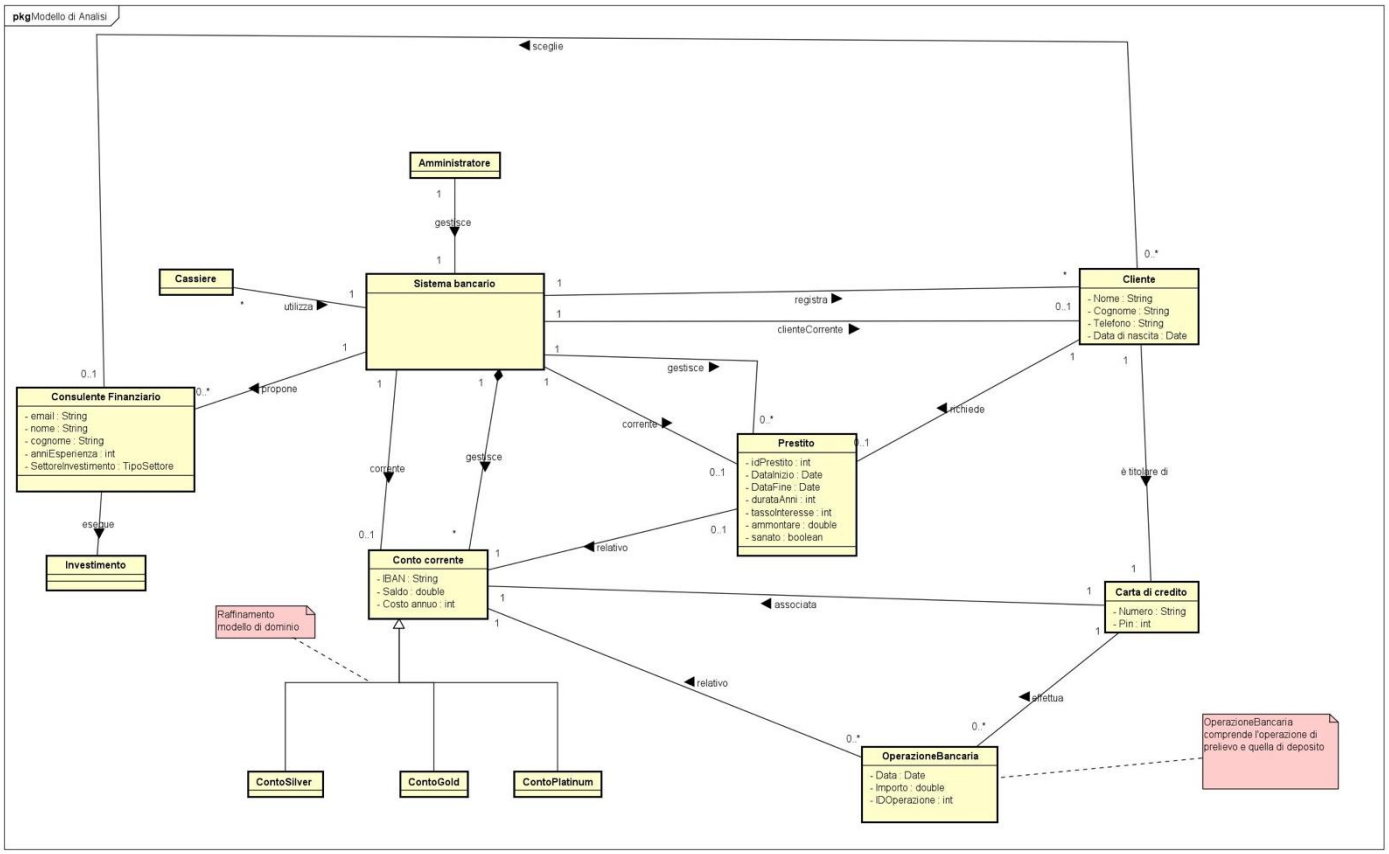
6.1. Introduzione

Per l'iterazione 3, sono stati scelti i seguenti requisiti:

- lo scenario principale di successo del caso d'uso UC4 (gestisci prestito) e del caso d'uso UC5(scelta consulente finanziario);
- Dati solo in memoria principale.

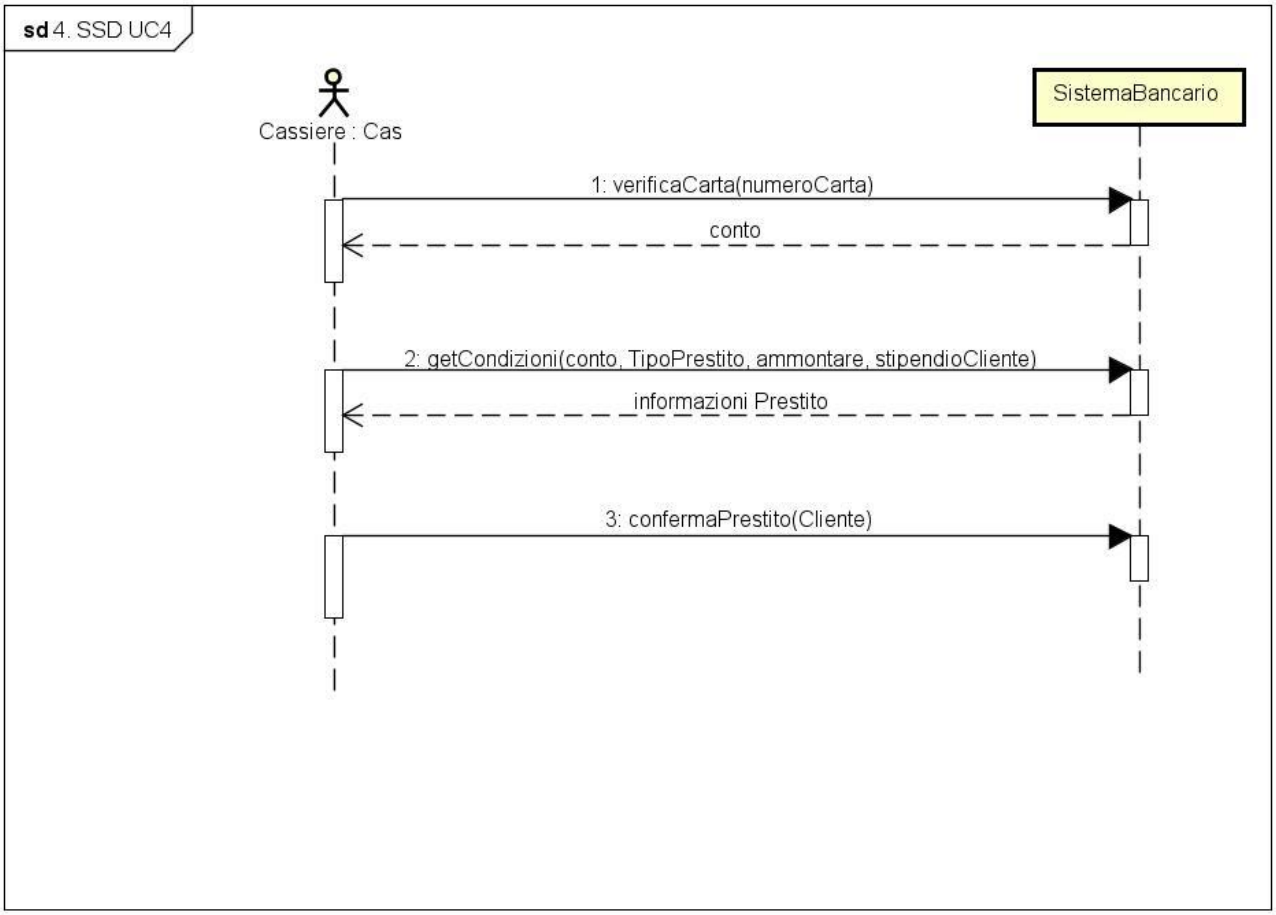
Questo capitolo descrive l'analisi svolta nell'iterazione 3, mentre il capitolo successivo descrive l'attività di progettazione.

6.2. Caso d'uso UC4/UC5, Modello di dominio



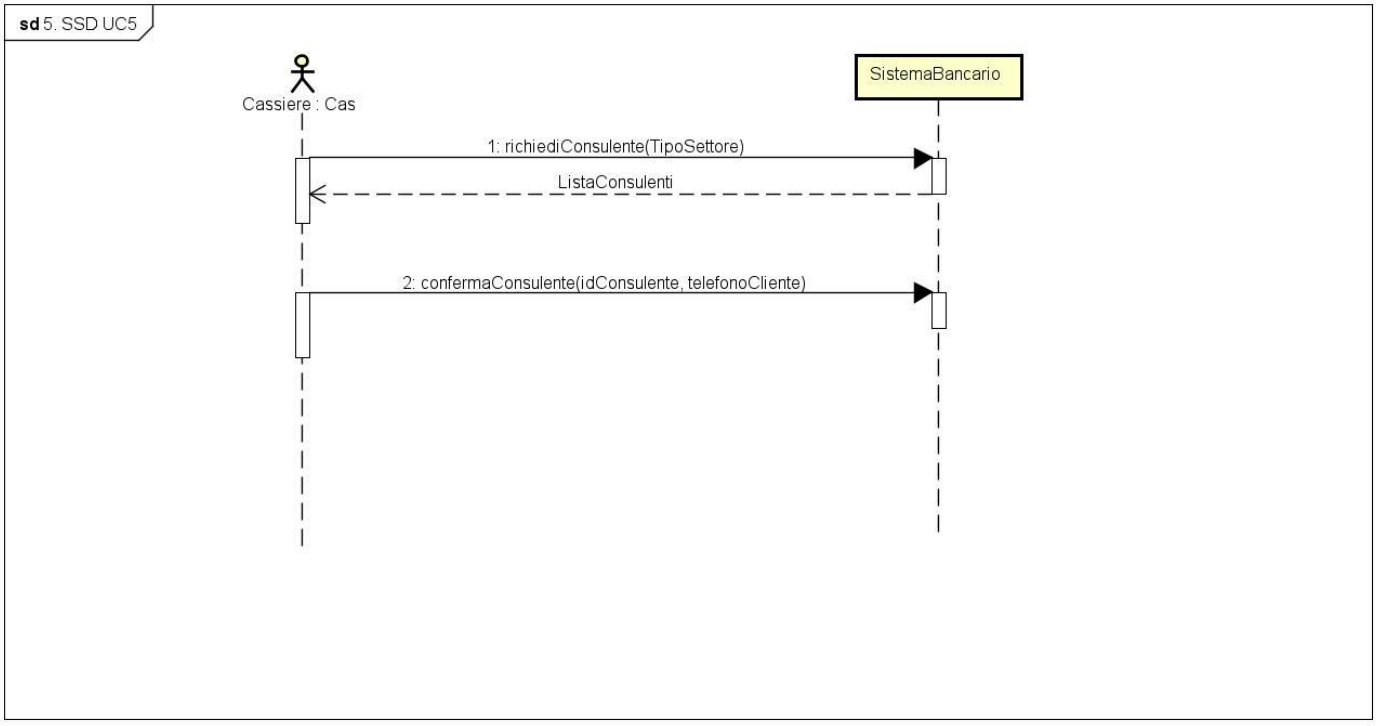
6.3. Caso d’uso UC4, Diagramma di sequenza di sistema

Il diagramma di sequenza di sistema per lo scenario principale di successo del caso d’uso UC4



6.4. Caso d’uso UC5, Diagramma di sequenza di sistema

Il diagramma di sequenza di sistema per lo scenario principale di successo del caso d’uso UC5



7. Iterazione 3, Progettazione

7.1. Introduzione

Prima di inizia la progettazione sono state prese le seguenti scelte:

- La classe concettuale investimento non viene implementata perché aumenterebbe l'accoppiamento del sistema e inoltre non è previsto nessun caso d'uso per la gestione degli investimenti.

- Implementare la classe ContoFactory a cui dare dare la responsabilità di istanziare le sottoclassi di ContoCorrente in funzione dei parametri ricevuti da SistemaBancario.

- Implementare la classe Controller Prestito a cui delegare gli oneri della gestione prestito anziché assegnare tali compiti a Sistema Bancario, il quale ha già tante responsabilità. Questo ci permette di avere Controller Prestito come una classe altamente coesa.

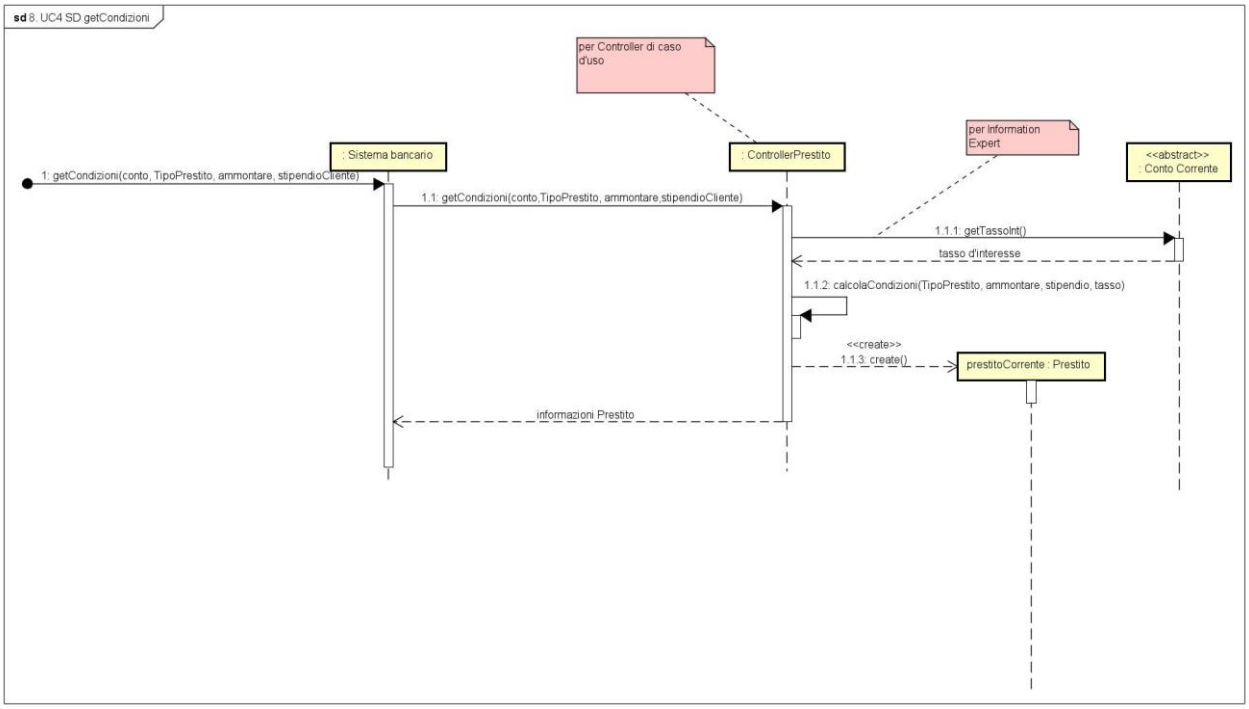
- Implementare il registro dei prestiti per avere un'organizzazione ordinata e strutturata della gestione dei prestiti. Anche in questo caso otteniamo una classe altamente coesa.

- Implementazione classe consulente finanziario.

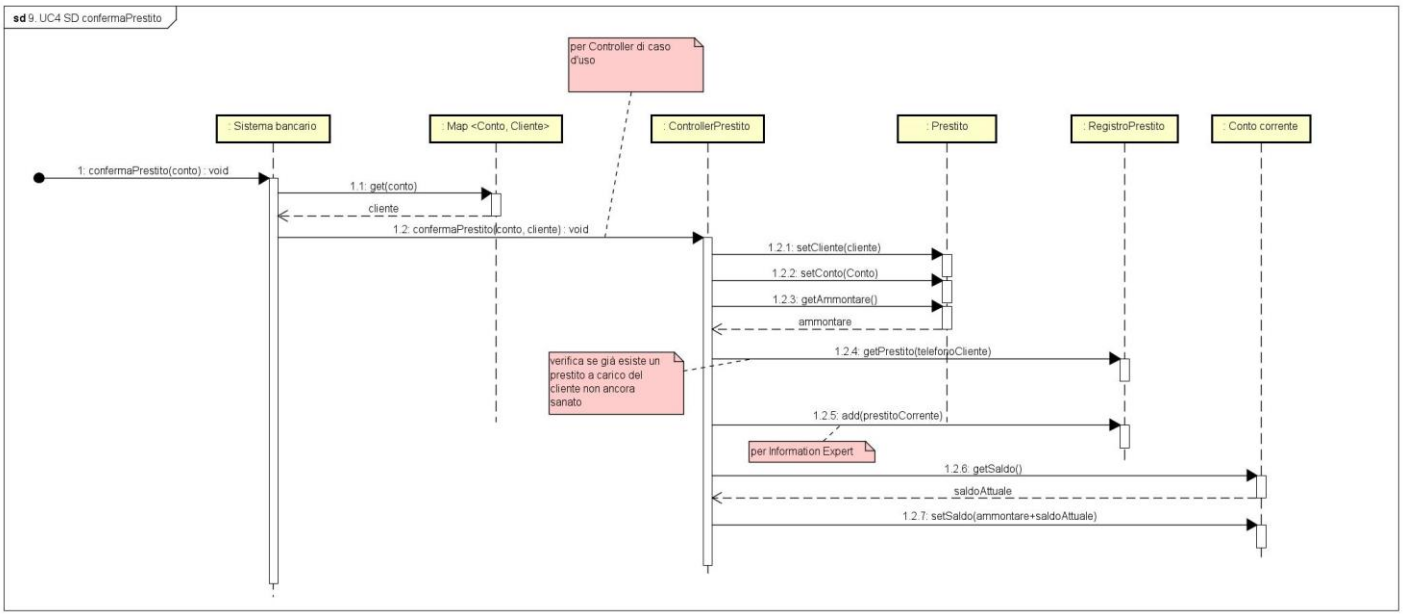
7.2. Caso d'uso UC4/UC5, Diagrammi di interazione

7.2.1. verificaCarta(...) vedi 5.2.1

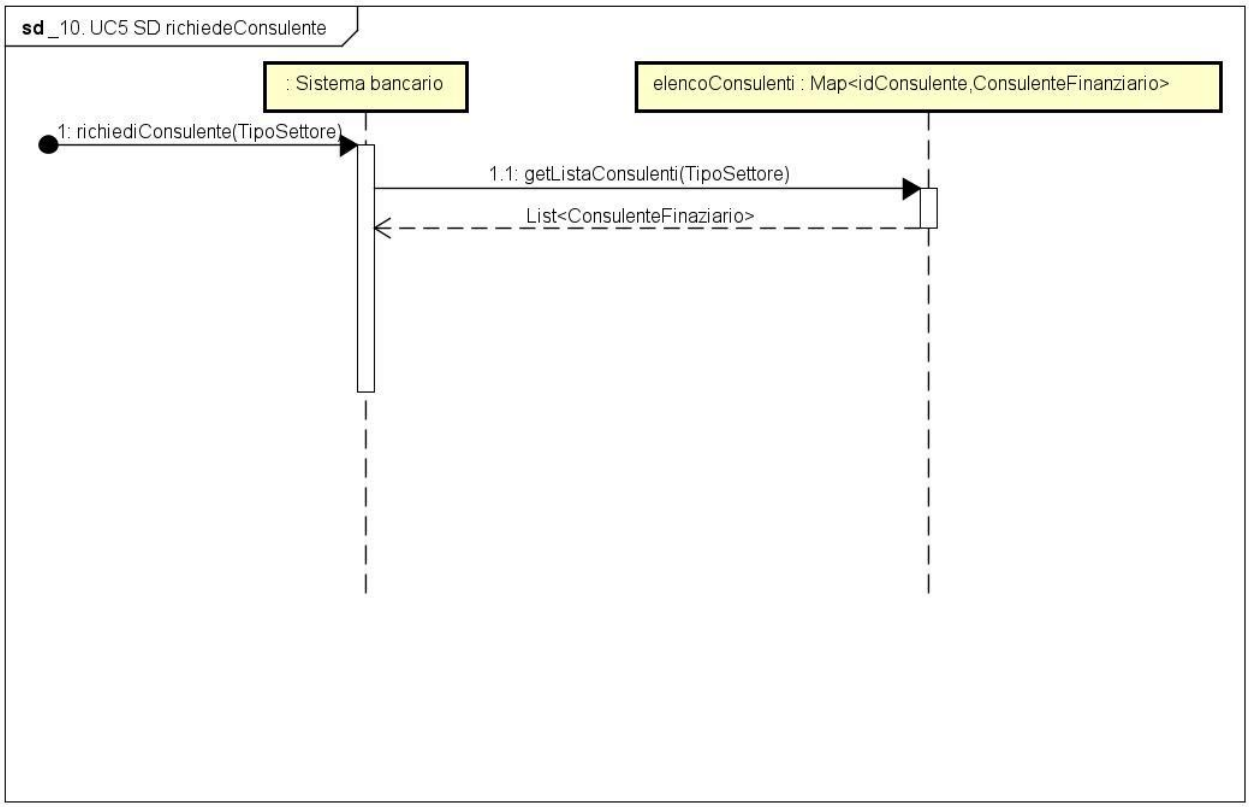
7.2.2. getCondizioni(...)



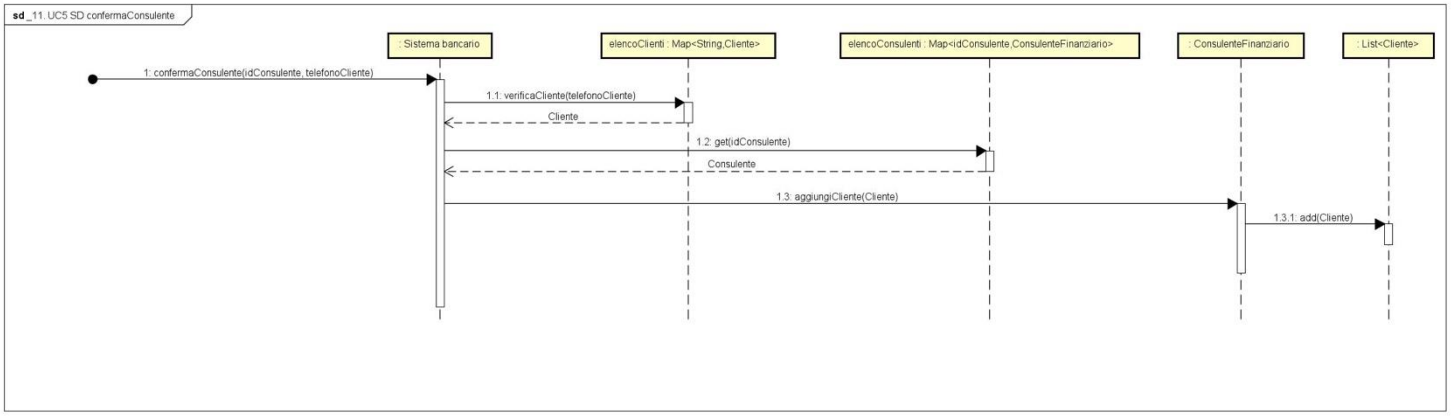
7.2.3. confermasPrestito(...)



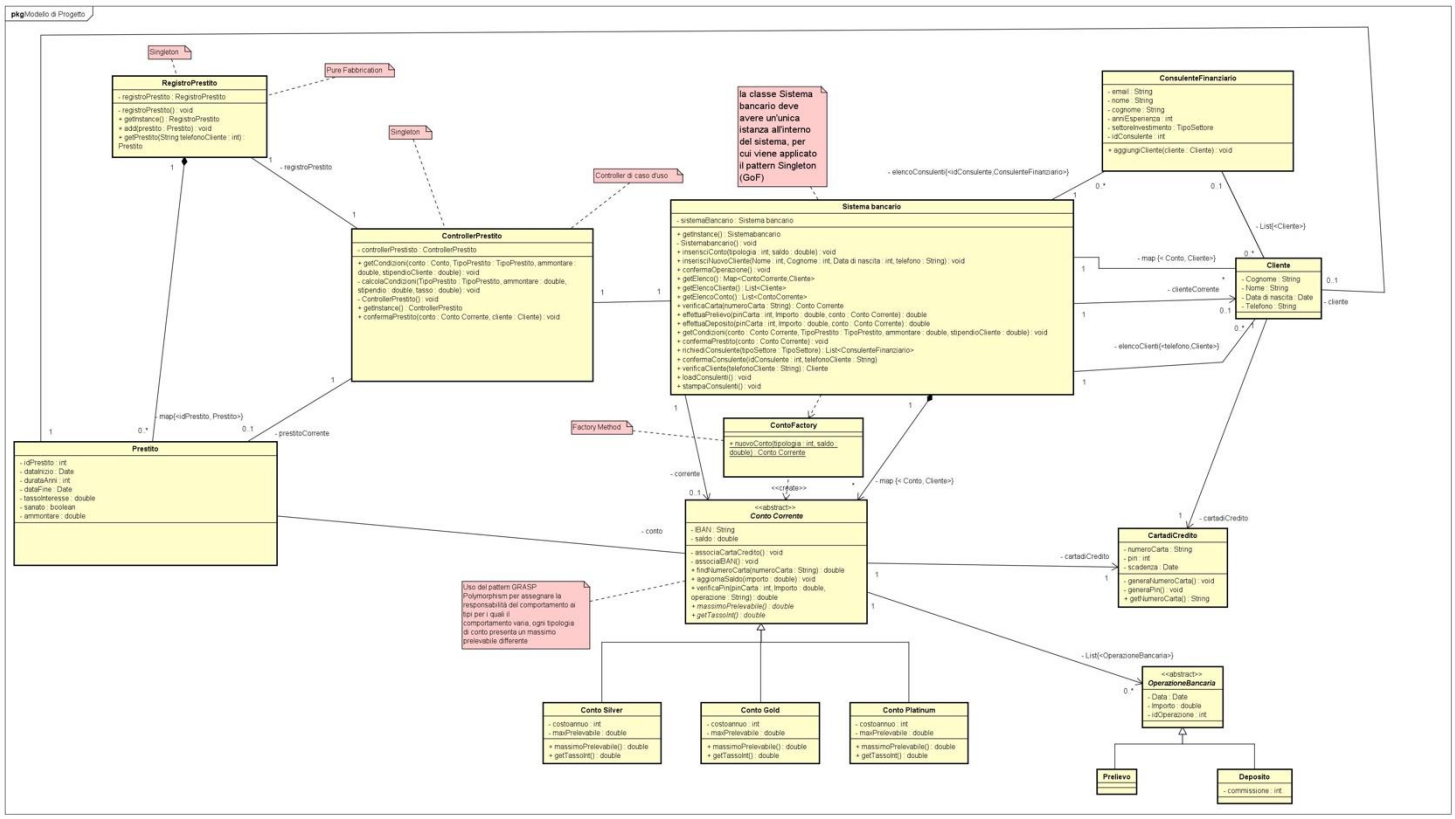
7.2.4. richiediConsulente(...)



7.2.5. confermaConsulente(...)



7.3. Caso d’uso UC4/UC5, Diagrammi delle classi di progetto



Si è utilizzato il **pattern Controller di caso d’uso**, poiché rifacendoci a quanto detto nell’introduzione del presente capitolo la classe Controller Prestito coordina tutte le operazioni del caso d’uso UC4 (Gestisci prestito). Il vantaggio di questa scelta è quello di poter ragionare solo sul caso d’uso in questione e inoltre incrementa il potenziale di riuso del codice.

Si è utilizzato il **pattern Pure Fabbrication** per la creazione della classe RegistroPrestito, la quale ha la sola responsabilità di gestire una struttura dati per l’elenco dei prestiti concessi dalla banca. In questo modo si garantisce una alta coesione, difatti tutto quello che riguardo l’inserimento, l’aggiornamento e la lettura dei prestiti è delegato al RegistroPrestito.

Si è utilizzato il **pattern Factory Method** per delegare l’istanziazione di una sottoclasse specifica del tipo conto (conto silver, conto gold, conto platinum). Esso indirizza il problema della creazione di oggetti senza specificarne l'esatta classe. Promuove l’accoppiamento libero eliminando la necessità di associare classi specifiche nel codice.

8. Iterazione 4, Analisi

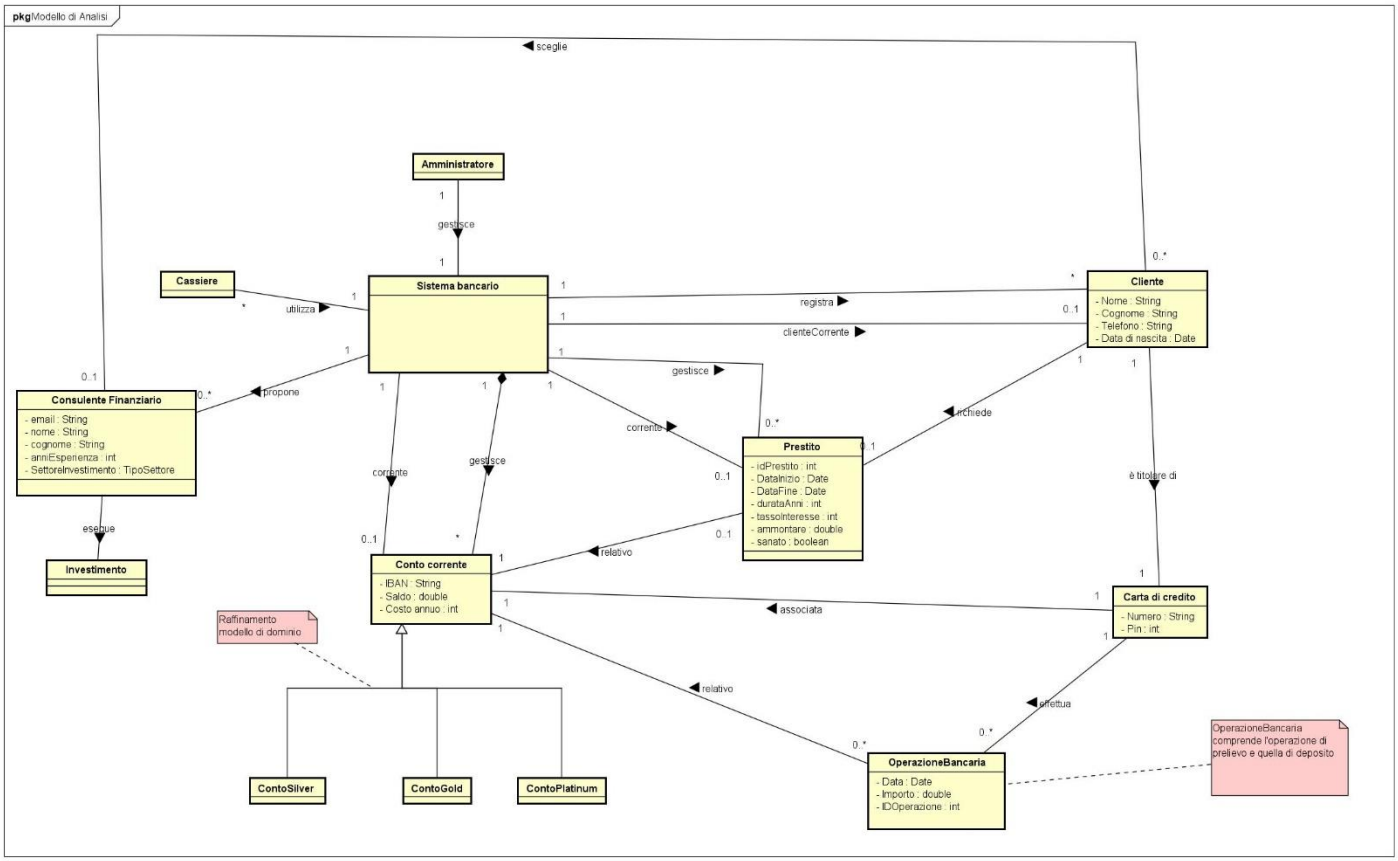
8.1. Introduzione

Per l'iterazione 4, sono stati scelti i seguenti requisiti:

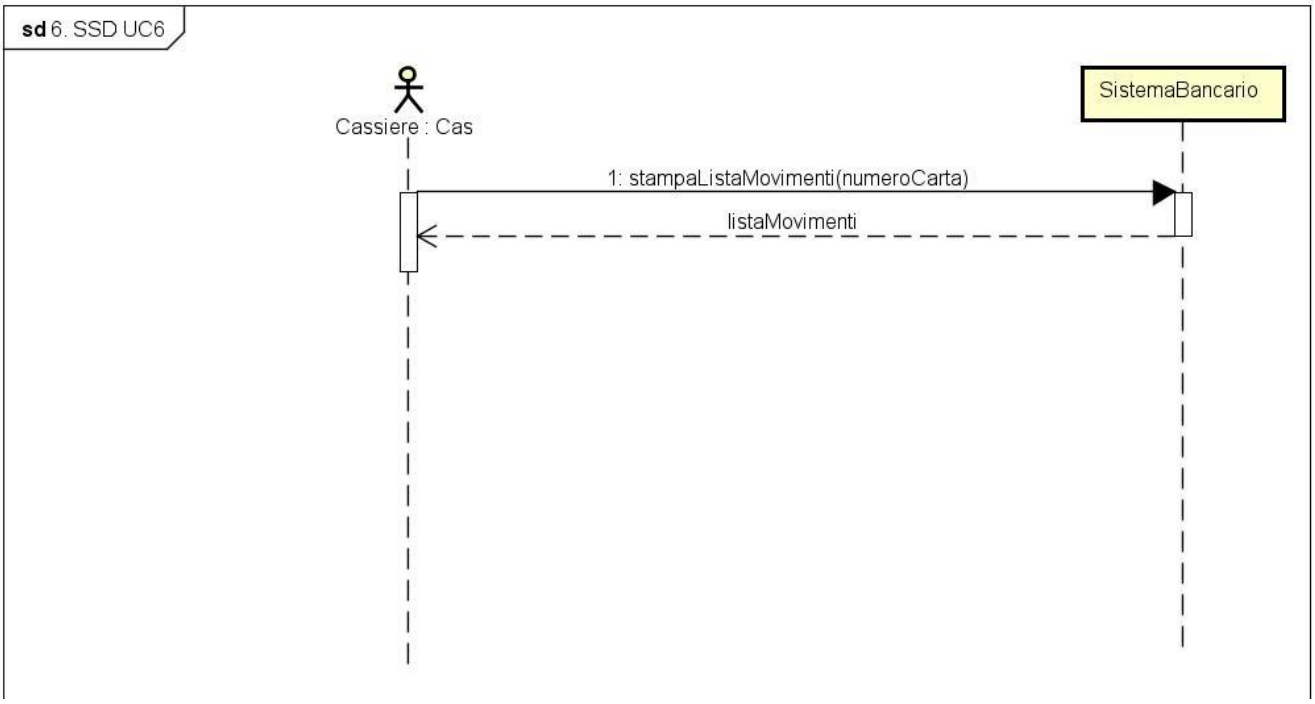
- lo scenario principale di successo del caso d'uso UC6 (Visualizza estratto conto), UC7 (visualizza conti correnti), UC8 (modifica tasso interesse e max prelevabile) e del caso d'uso UC9(Visualizza prestiti);
- Dati solo in memoria principale.

Questo capitolo descrive l'analisi svolta nell'iterazione 4, mentre il capitolo successivo descrive l'attività di progettazione.

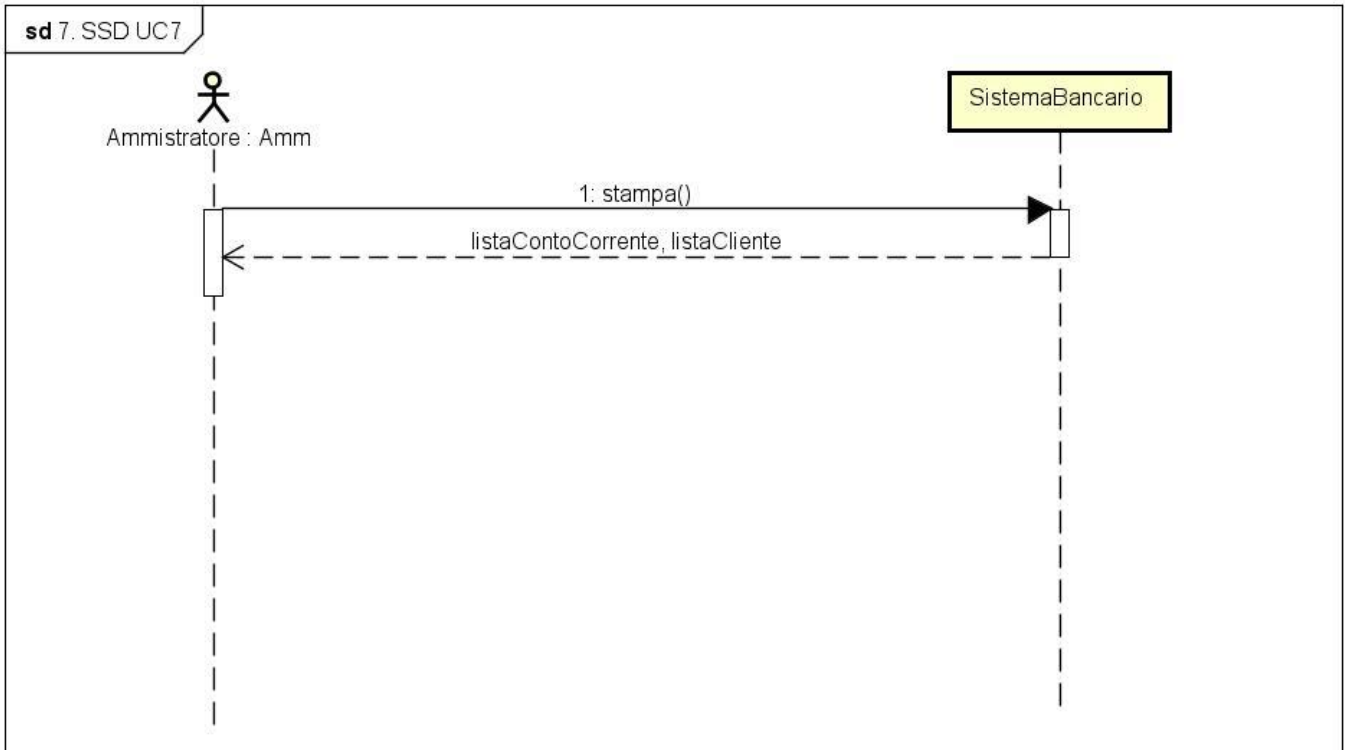
8.2. Caso d'uso UC6/UC7/UC8/UC9, Modello di dominio



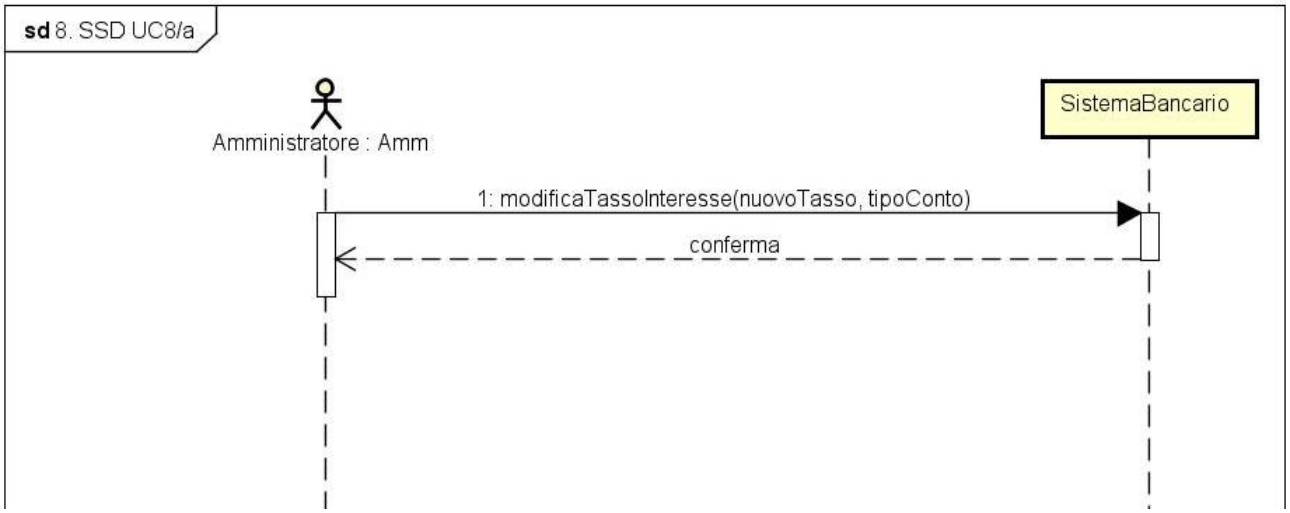
8.3. Caso d’uso UC6, Diagramma di sequenza di sistema



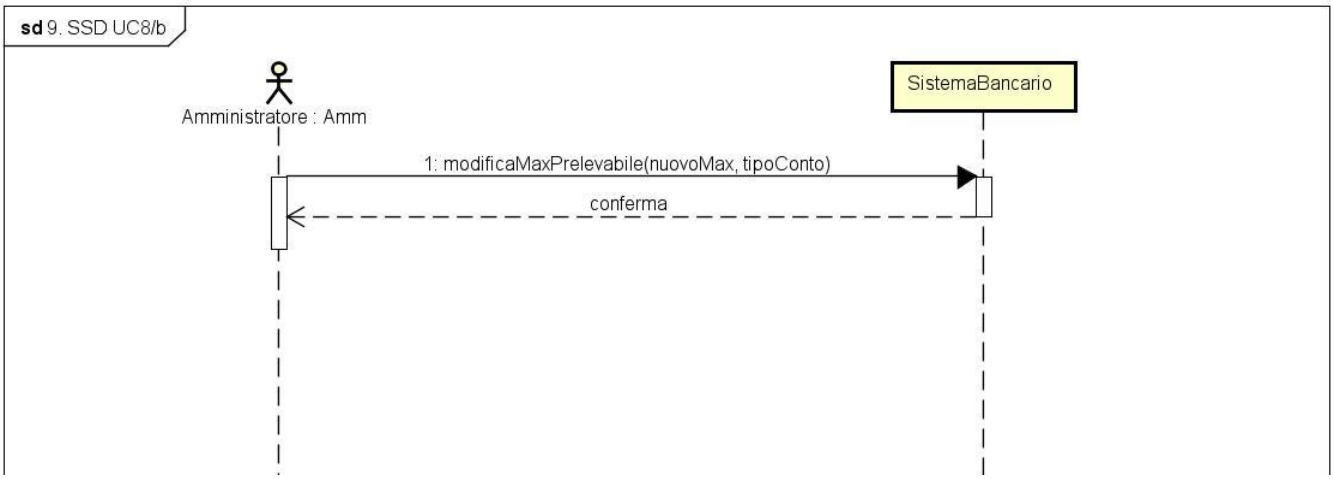
8.4. Caso d’uso UC7, Diagramma di sequenza di sistema



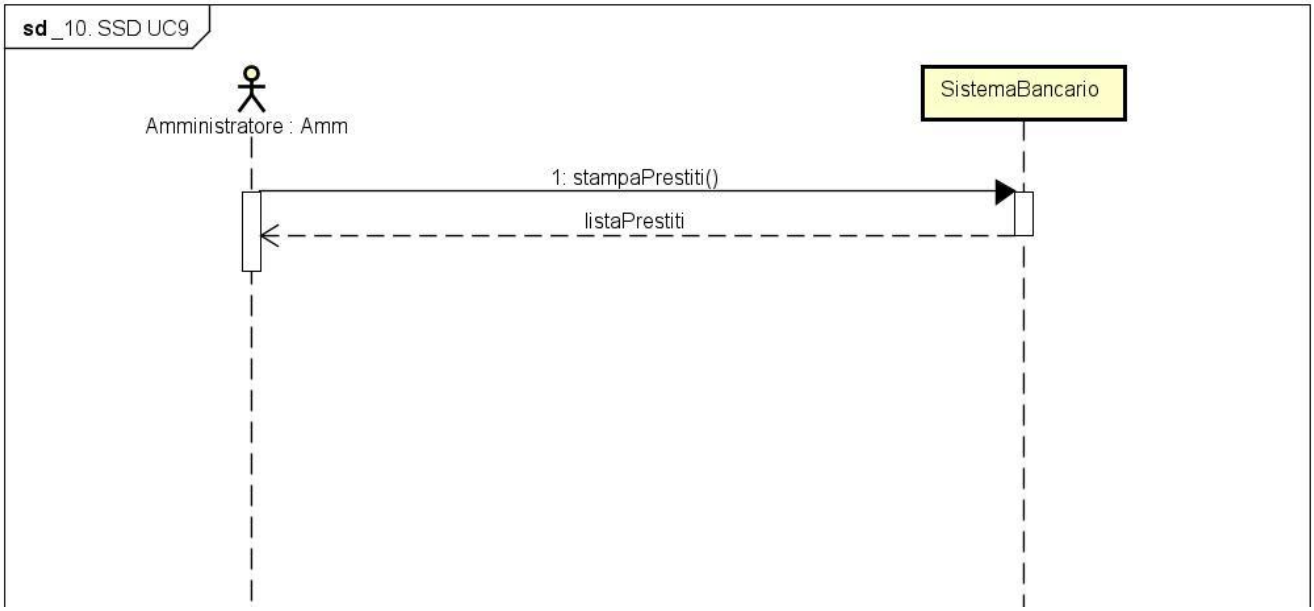
8.5. Caso d’uso UC8a, Diagramma di sequenza di sistema



8.6. Caso d’uso UC8b, Diagramma di sequenza di sistema



8.7. Caso d’uso UC9, Diagramma di sequenza di sistema



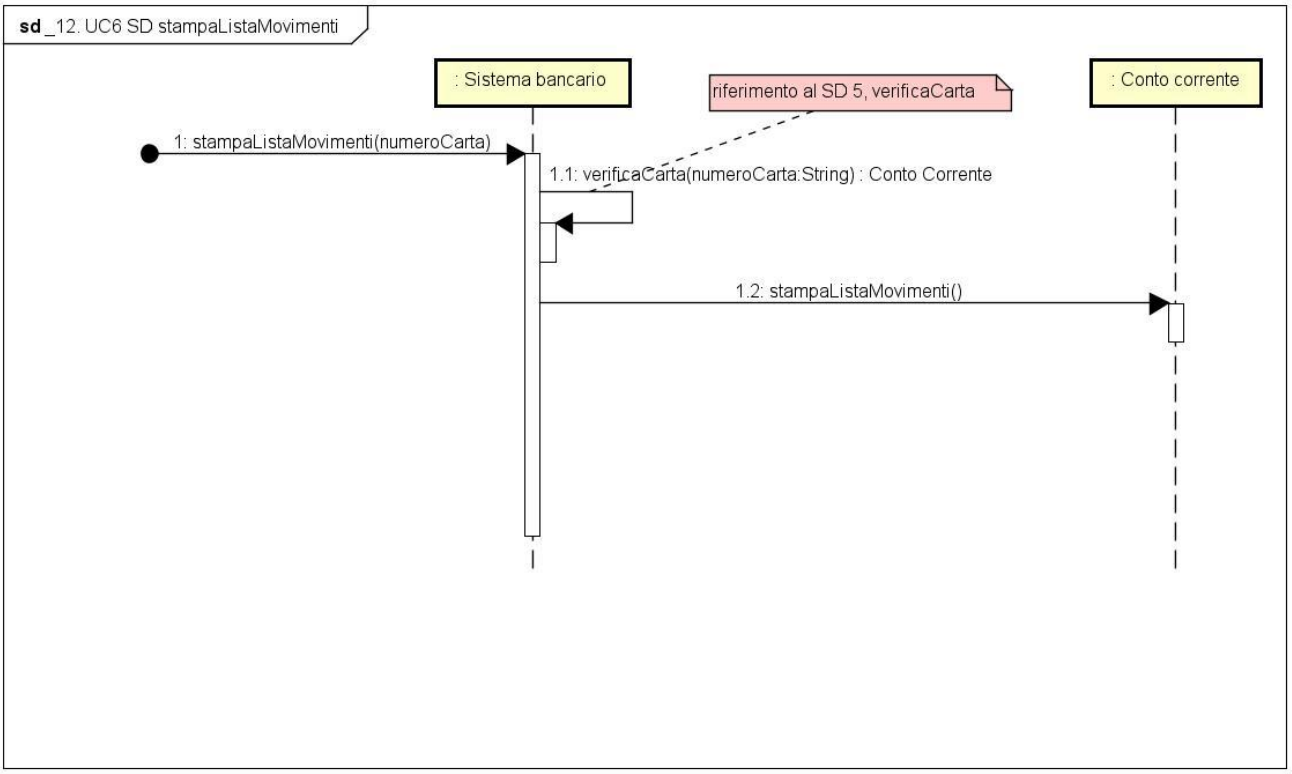
9. Iterazione 4, Progettazione

9.1. Introduzione

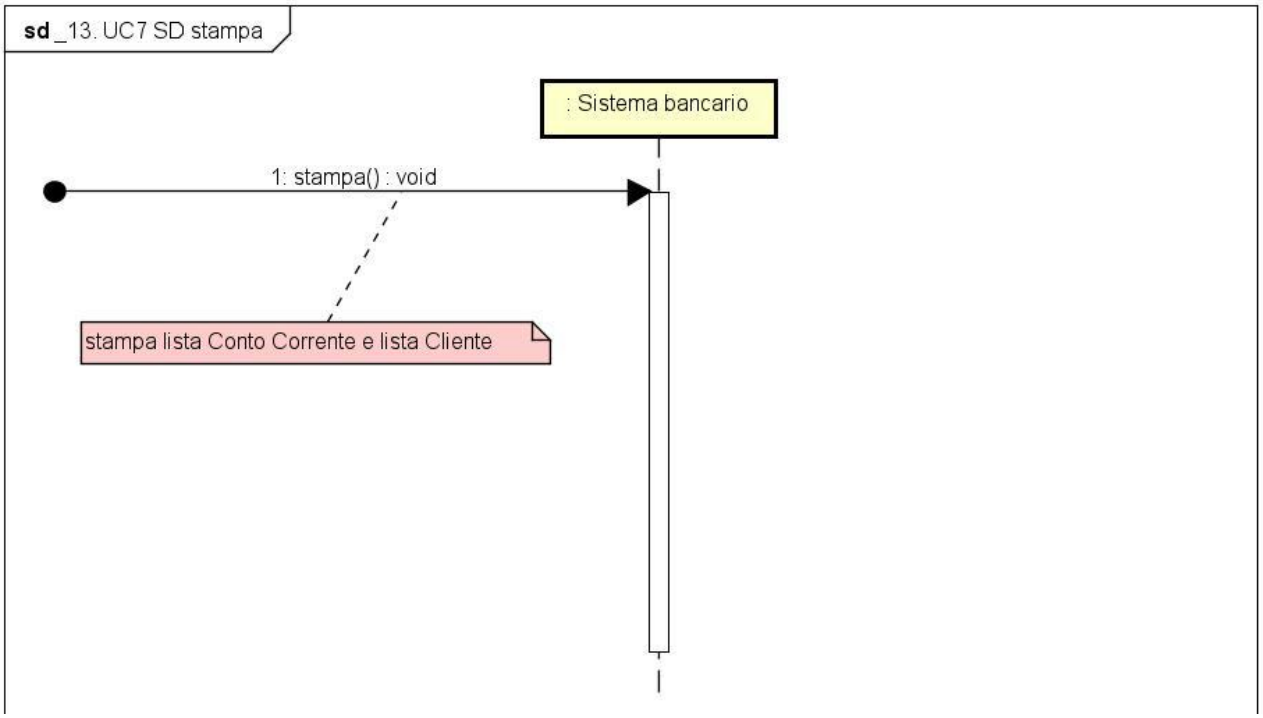
Per realizzare i casi d'uso dell'iterazione quattro non è stato necessario intervenire a livello progettuale nel diagramma delle classi.

9.2. Caso d'uso UC6/UC7/UC8/UC9, Diagrammi di interazione

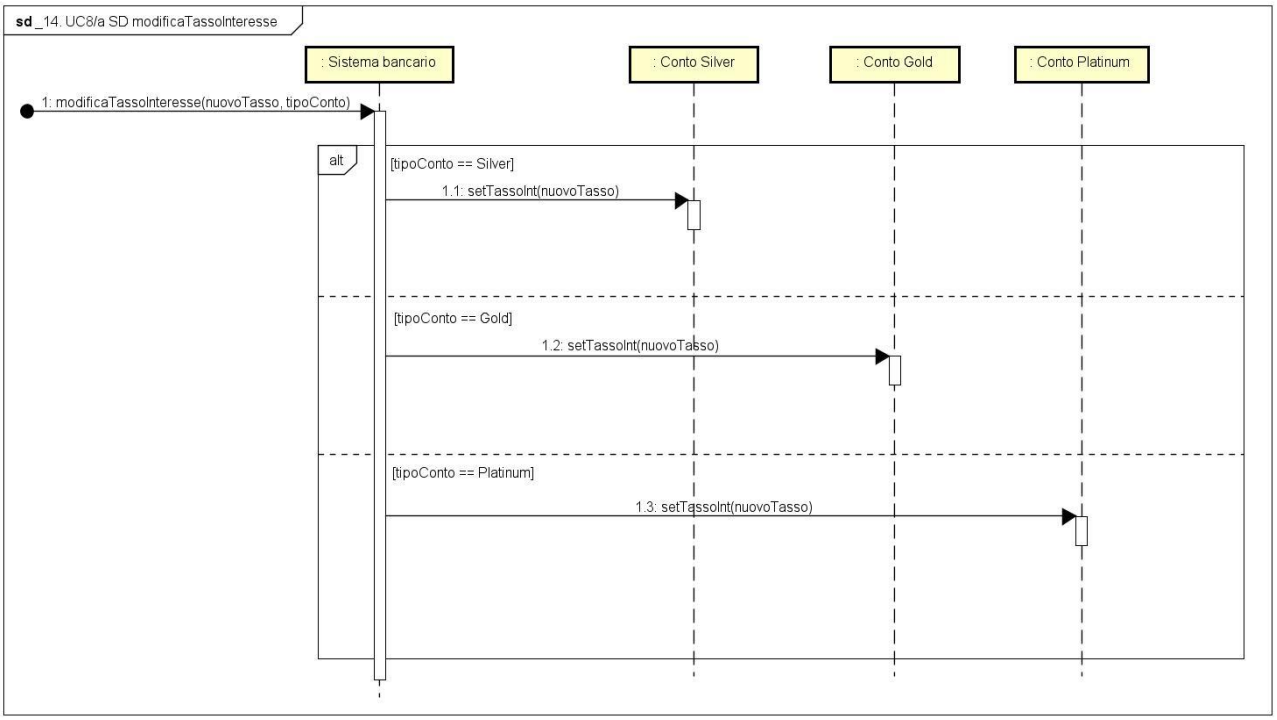
9.2.1.UC6 stampaListaMovimenti(...)



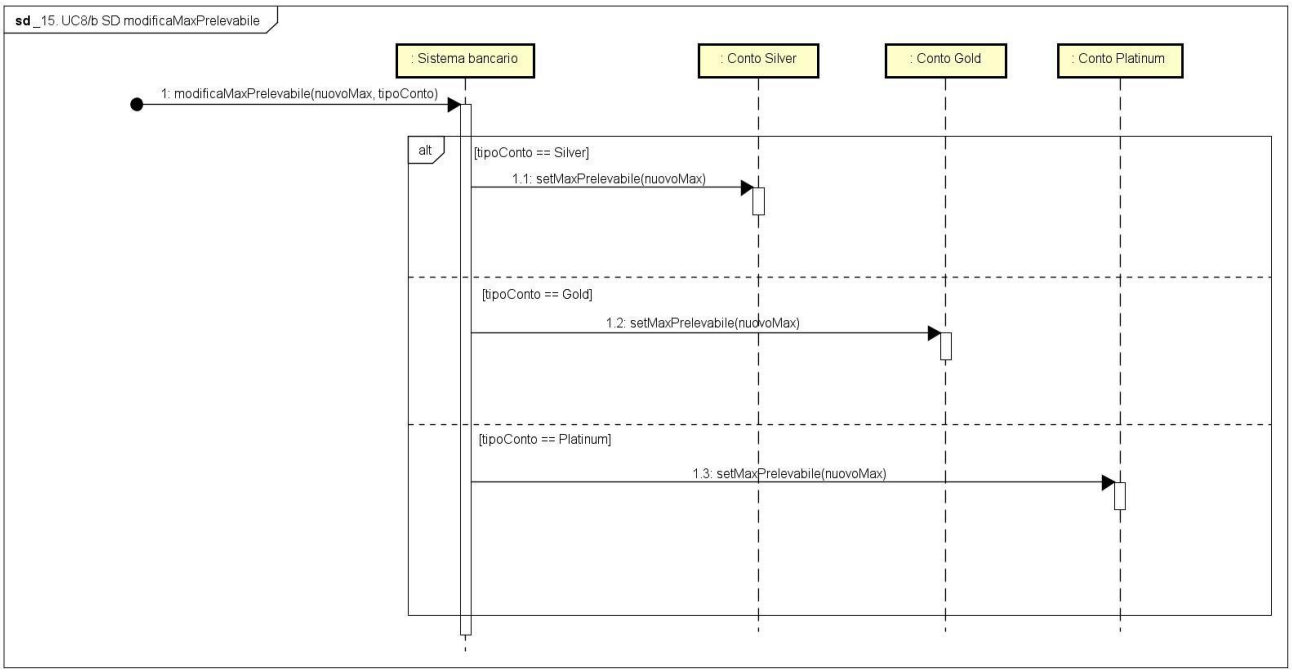
9.2.2. UC7 stampa(...)



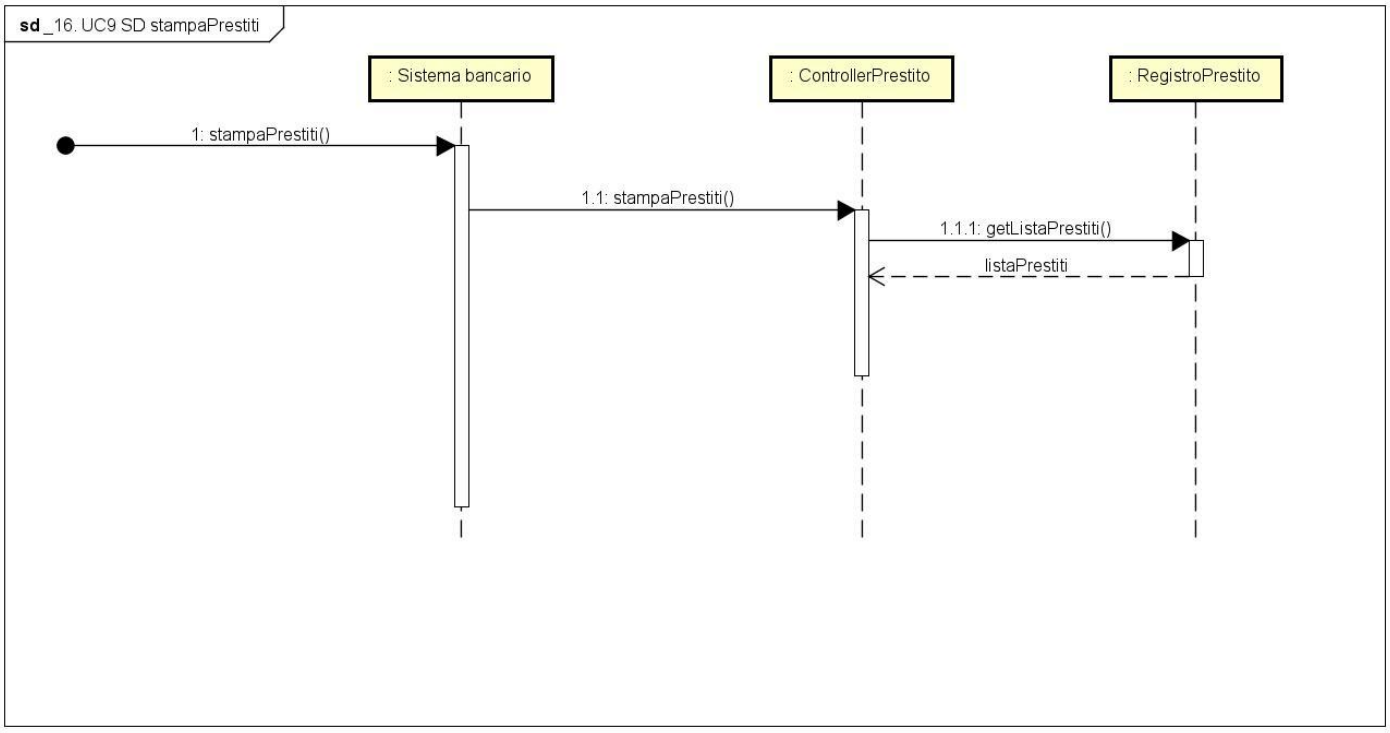
9.2.3. UC8a modificaTassoInteresse(...)



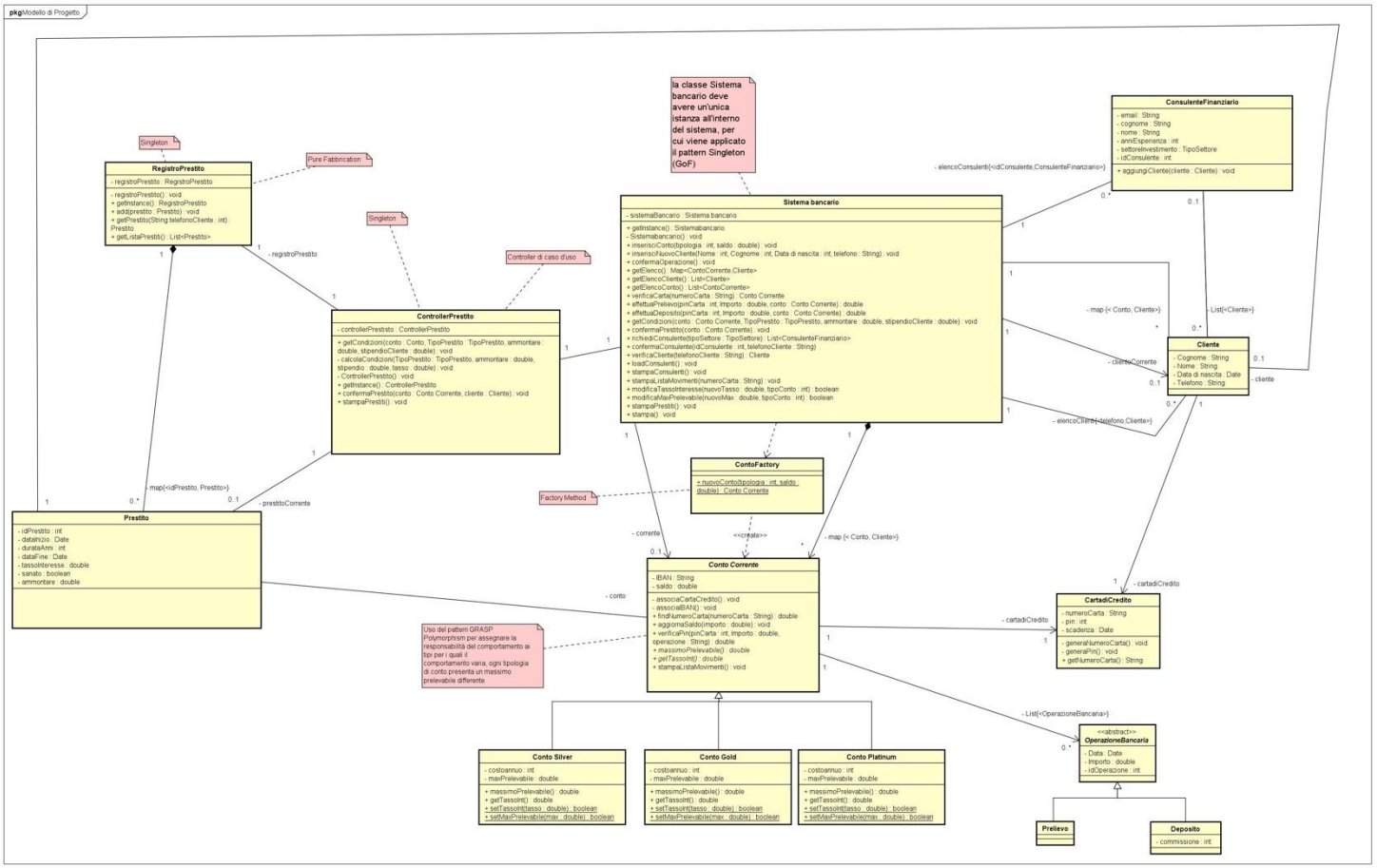
9.2.4. UC8b modificaMaxPrelevabile(...)



9.2.5. UC9 stampaPrestiti(...)



9.3. Caso d’uso UC6/UC7/UC8/UC9, Diagrammi delle classi di progetto



10. Implementazione

10.1. Introduzione

Per l'implementazione del presente progetto software è stato utilizzato l'IDE "Intellij" con linguaggio di programmazione Java. È stato scelto Java perché rispetta il paradigma di programmazione orientata agli oggetti e inoltre per la nostra familiarità con tale linguaggio.

10.2. Strutture Dati

```
private List<Cliente> listaClienti;
```

Classe: Consulente Finanziario
Utilità: Serve a mantenere l'insieme dei clienti che hanno scelto un determinato consulente finanziario.

```
private List<OperazioneBancaria> lista_movimenti;
```

Classe: Conto Corrente
Utilità: Serve a mantenere traccia di tutte le operazioni bancarie effettuate in un determinato conto corrente.

```
private final Map<Integer, Prestito> elencoPrestiti;
```

Classe: Registro prestito
Utilità: Serve a mantenere l'insieme di tutti i prestiti concessi dalla banca.

```
private final Map<ContoCorrente,Cliente>elenco;
```

Classe: Sistema Bancario
Utilità: Permette di associare ad ogni ContoCorrente il relativo cliente. Il presente progetto, nel rispetto della regola di dominio R9, permette di associare a ogni cliente un solo conto corrente.
Se tale regola di dominio dovesse subire variazioni nel tempo e permettere a un cliente di possedere più di un conto corrente, la struttura dati riuscirebbe comunque a rimanere allineata e valida, perché ogni conto corrente ha un attributo "IBAN", il quale è diverso per ogni conto e difatti identifica in modo univoco un determinato conto corrente.
La generazione del codice IBAN è un'operazione, eseguita dalla classe conto corrente, che sfrutta una variabile static, la quale viene incrementa ad ogni istanza creata di conto corrente per renderla univoca.

```
private final Map<String, Cliente> elencoClienti;
```
Classe: Sistema bancario
Utilità: Serve a mantenere l'insieme di tutti i clienti che hanno aperto un conto corrente presso la banca. Come chiave viene usato il numero di telefono del Cliente, il quale identifica uno e un solo cliente.

```
private final Map<Integer, ConsulenteFinanziario>
elencoConsulenti;
```
Classe: Sistema bancario
Utilità: Serve a mantenere l'insieme di tutti i consulenti finanziari che lavorano alla banca. Come chiave viene usato l'id del consulente finanziario, il quale identifica uno e un solo cliente.

10.3. Test

10.3.1 Introduzione

Il presente progetto è stato testato grazie all'utilizzo del framework open source JUnit 5.8.1., il quale offre diverse funzionalità di test driver che permettono di eseguire determinati metodi con certi input e verificare se la loro computazione è corretta o meno, confrontando il risultato ottenuto dal metodo da testare con l'output desiderato (assertXxx).

I test sono stati implementati seguendo i principi di scomponibilità (i moduli sono indipendenti tra loro e possono essere testati indipendentemente) e semplicità (esistono solo le funzioni necessarie) per migliore la collaudabilità del software.

Black-box: funzionale ai casi di test determinati in base a ciò che il componente deve fare e la sua specifica

10.3.2. Costruzione dei test

Nella costruzione di un test sono stati seguiti tre step:

1. Setup: tramite le annotations @BeforeEach e @BeforeAll, associate ai relativi metodi, è stato inizializzato il SUT (System Under Test)

2. Exercise and Verify: per l'implementazione dei test è stata usata l'annotation

@Test. Per ogni test sono stati considerati i valori desiderati e i valori attuali confrontandoli tra di loro con dei metodi assert. Viene quindi creato un esito del test.

3. Teardown: è un metodo che viene chiamato alla fine di ogni metodo di test per eseguire le operazioni di pulizia o ripristino dello stato del sistema alla situazione iniziale, in modo da garantire l'indipendenza tra i vari test. Viene eseguito alla fine di ogni metodo di test per effettuare le operazioni di pulizia o ripristino dello stato del sistema alla situazione iniziale, in modo da garantire l'indipendenza tra i vari test. Nella programmazione di test unitari con JUnit, il metodo di teardown viene solitamente definito come un metodo annotato con "@After" e viene eseguito automaticamente alla fine di ogni metodo di test marcato con l'annotazione "@Test".

10.4. Git Practice e sviluppo iterativo (Agile)

Per la realizzazione del progetto, il team ha usufruito del servizio di hosting offerto da “Github” per condividere il codice e rimanere allineati su eventuali sviluppi del team.
Si è adottata la git practice di avere 2 linee di sviluppo (Branches) che sono:

- Develop: Branch di sviluppo in cui il team “crea” il software e prova funzionalità.
- Main: Branch principale in cui vengono caricate le versioni stabili e funzionanti del progetto, iterazione dopo iterazione.

Il software ha seguito uno sviluppo iterativo e incrementale, in cui il sistema è cresciuto in funzionalità, iterazione dopo iterazione, seguendo i principi di Agile.
Ciò ha permesso di:

- Accogliere eventuali cambiamenti e adattarli al meglio allo scopo del sistema.
- Operare su un piccolo insieme di requisiti per volta.
- Fornire un feedback rapido dall’utente e dai test, senza aspettare che sia totalmente finito.
- Ridurre i rischi grazie al feedback e sfruttarlo come opportunità per far crescere il sistema.
- Gestire la complessità
- Miglioramento continuo del processo
- Osservare il progresso