



UNIVERSITÁ POLITECNICA DELLE MARCHE

Facoltà di Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione



Progetto Ingegneria del Software

Software per la gestione di un museo

Morelli Valerio
Sbattella Mattia
Achilli Nicolò
Sumcutean Sara

Luglio 2022 - Ottobre 2022

Indice

1	Introduzione	3
1.1	Organizzazione Generale	4
1.1.1	Clientela	4
1.1.2	Biglietti	5
1.1.3	Tasse e costi	5
1.2	Struttura Interna	6
1.2.1	La Reception	6
1.2.2	La Segreteria	7
1.2.3	L'Amministrazione	7
1.3	Il Sistema	8
1.4	Glossario di Progetto	9
2	Analisi dei Requisiti	10
2.1	Package dei requisiti	11
2.2	Requisiti Funzionali	12
2.3	Requisiti non Funzionali	14
3	Diagrammi dei casi d'uso	15
3.1	Package dei casi d'uso	16
3.2	Attori	17
3.3	Gestione Cliente	18
3.4	Gestione Segreteria	29
3.5	Gestione Amministrazione	34
3.6	Gestione Sistema	44
3.7	Gestione Statistiche	48
4	Matrice di mapping	57
4.1	Matrice di mapping	58
5	Diagramma delle classi (analisi)	59
5.1	Package delle classi	60
5.2	Diagramma delle classi	61
6	Diagrammi di sequenza (analisi)	62
6.1	Gestione Amministrazione	64
6.2	Gestione Cliente	67
6.3	Gestione Segreteria	76
6.4	Gestione Sistema	81
6.5	Gestione Statistiche	85
7	Diagrammi di attività (analisi)	87
7.1	Gestione Amministrazione	89
7.2	Gestione Cliente	92
7.3	Gestione Segreteria	102
7.4	Gestione Sistema	106
7.5	Gestione Statistiche	111
8	Diagrammi delle classi (progettazione)	114
8.1	Package delle classi	115
8.1.1	Backend	115
8.1.2	Frontend	116
8.2	Diagramma delle classi	117
8.2.1	Visione generale	117
8.2.2	Backend-HighLevel	118
8.2.3	Backend-LowLevel	120

8.2.4	Frontend	121
9	Diagrammi di sequenza raffinati	123
9.1	CompraBiglietto Raffinato	125
9.2	CompraOpera Raffinato	127
9.3	VerificaAbbonamento Raffinato	130
9.4	AbbonaCliente Raffinato	132
10	Diag. delle macchine a stati	135
10.1	Museo	136
10.2	Biglietto	137
10.3	Abbonamento	138
10.4	RichiestaDonazione	139
10.5	Dipendente (Login)	140
10.6	Dipendente (Lavoro)	141
11	Diagramma dei componenti	142
11.1	Diagramma dei componenti completo	143
11.2	Diagramma dei componenti backend	144
11.3	Diagramma dei componenti frontend	145
12	Mockup	146
12.1	Mockups delle Home e dell'account	147
12.2	Mockups da HomeReception	149
12.3	Mockups da HomeSegreteria	151
12.4	Mockups da HomeAmministrazione	152
12.5	Altri mockups	157
13	Diagramma di deployment	158
13.1	Diagramma di Deployment	159
14	Implementazione	160
14.1	Flusso di lavoro	161
14.2	Motivazioni	162
14.3	Directories di implementazione	163
14.4	Requisiti	164
15	Test	165
15.1	Test delle classi di alto livello	166
15.1.1	TestAbbonamento	166
15.1.2	TestDipendente	167
15.1.3	TestRichiestaDonazione	169
15.2	Test delle classi di basso livello	170
15.2.1	TestIOTextFile	170
15.2.2	TestSerializzazionePickle	171
15.2.3	TestDropBoxAPI	172
15.2.4	TestSHA256Hashing	174
15.2.5	TestQRCodeEncoding	174
16	Conclusioni	175
16.1	Link Utili	176
16.2	Dimostrazioni di funzionamento del software	176

1 Introduzione

Il progetto proposto consiste nella realizzazione di un software gestionale per il **Museo Statale Omero di Ancona**.

Segue una dettagliata descrizione della logica di business appresa a seguito di alcune interviste effettuate agli *stakeholders*, nelle quali sono state poste domande volte alla comprensione della **struttura generale interna al museo** e delle **esigenze dei dipendenti**.

La necessità del personale intervistato era quella di avere un software a supporto, per facilitare le operazioni quotidiane svolte dai dipendenti quali la **stampo dei biglietti**, la **compravendita delle opere** e la **redazione del report degli incassi**.

1.1 Organizzazione Generale

La struttura museale si compone di **due** parti:

- una dedicata al museo, nella quale sono presenti 32 opere **fisse**, quindi **non vendibili**;
- e un'altra dedicata alle **mostre**, nella quale le opere d'arte sono temporaneamente esposte per tutta la durata della mostra stessa. Ciascuna di essa può contenere un massimo di 80 opere. Tutte le opere appartenenti alla mostra **possono essere vendute al cliente interessato all'acquisto**.

1.1.1 Clientela

Il cliente gioca un ruolo fondamentale all'interno della struttura museale, egli può **visitare il museo permanente**, le **mostre temporanee**, partecipare a **tour guidati**, **acquistare o rinnovare abbonamenti**, **comprare opere e donarle** al museo.

Essi sono organizzati nelle due seguenti categorie:

- Cliente non abbonato (alias *Visitatore*)
- Cliente abbonato (alias *Cliente*)

Cliente non abbonato (**Visitatore**)

Il cliente non abbonato è colui che entra al museo senza essere in possesso di un abbonamento. Il sinonimo con il quale ci si riferirà ad esso è *"Visistatore"*. Questo termine **verrà utilizzato anche come accezione generale** alla persona che usufruisce dei servizi offerti dal museo, indipendentemente dall'essere abbonata o meno.

Seppur sprovvisto di abbonamento, egli può comunque visitare il museo scegliendo il reparto di interesse; a tal fine può certamente **acquistare biglietti**. All'acquisto di quest'ultimi, l'operatore al pubblico (ovvero colui che gestisce la reception e si interfaccia direttamente con la clientela), **gli chiederà la disponibilità a rilasciare informazioni quali sesso, Paese di provenienza e luogo di nascita**. Questi dati verranno salvati dal sistema e serviranno per **generare statistiche** sulla tipologia di clientela che visita il museo.

Per poter usufruire di **tariffe ridotte o esenzione dal pagamento** del biglietto, il visitatore dovrà dimostrare di rientrare in una delle seguenti formule:

- Possono usufruire della tariffa **"Ridotto"**:
 - I ragazzi **al di sotto dei 18 anni** d'età;
 - Gli adulti **al di sopra dei 59 anni** d'età;
 - Tutti coloro che risultino **iscritti a scuole** di ogni ordine e grado, comprese **università**.
- Sono **esenti** dal pagamento:
 - Bambini **al di sotto dei 6 anni**;
 - Docenti di ogni scuola di ordine e grado e università;
 - **Possessori di Abbonamento** (non scaduto);
 - **Possessori di certificato di invalidità o handicap** (legge 104/92).

Cliente abbonato

Il visitatore, tra le altre cose, può acquistare un abbonamento; ci riferiremo al visitatore in possesso di abbonamento con il termine "*Cliente*".

Il museo offre la possibilità di abbonarsi per un periodo di tempo che va da un **minimo di un mese** al **massimo di un anno**.

Il cliente sarà dunque libero, esibendo il suo abbonamento e previa la sua validità, di visitare sia il museo che le mostre quante volte egli desideri **senza costi aggiuntivi**. Inoltre, avrà la possibilità di **acquistare le opere in vendita** presenti nel catalogo delle opere, **facoltà quest'ultima, non concessa al semplice visitatore**.

1.1.2 Biglietti

Al momento dell'acquisto di un biglietto, questo viene stampato e utilizzato come scontrino per ogni visitatore. Al fine di distinguerlo dagli altri e di essere riconosciuto dal sistema, al momento della creazione il sistema gli assegna **un codice identificativo univoco**, il quale verrà poi stampato sul fronte sotto forma di codice bidimensionale **QR**, ed **utilizzato per la convalida all'ingresso** del visitatore.

Il visitatore ha la possibilità **prenotare una guida** e anche di **scegliere l'orario del turno**, tra quelli non pieni della giornata, per la mostra alla quale desidera partecipare o per la visita al reparto fisso del museo. Se il visitatore rientra in una delle categorie per le quali si è esenti dal pagamento, per poter accedere, esso è tenuto comunque ad esibire per la convalida il biglietto.

1.1.3 Tariffe e costi

L'accesso al reparto fisso del museo è totalmente gratuito.

Per la mostra il costo del biglietto può variare a seconda dei diversi fattori sopra citati. Dunque le tariffe all'acquisto di un biglietto possono essere riassunte nella seguente tabella:

Offerte	Prezzo (€)
Museo fisso	Gratis
Intero Mostra	€8
Ridotto Mostra	€5
Abbonati	Gratis
Guida	€5 a persona

1.2 Struttura Interna

La struttura organizzativa interna è suddivisa nei seguenti 3 reparti:

- Reception;
- Segreteria;
- Amministrazione.

Ogni dipendente nella struttura ha possibilità di accedere al portale dedicato al reparto di competenza, inserendo le sue credenziali nel sistema.

1.2.1 La Reception

La reception è il reparto dove vengono accolti i visitatori; qui lavorano gli *Operatori al Pubblico*, che quindi **si interfacciano direttamente con essi**.

Gli operatori sono incaricati di accogliere ed illustrare alla clientela le varie soluzioni per l'accesso alla struttura. Nello specifico i loro compiti possono essere riassunti nelle seguenti operazioni:

- **Vendita e convalida** dei biglietti;
- **Vendita di opere** al cliente interessato;
- **Compilazione di richieste di donazioni**;
- Gestione di **tour guidati**.

Il primo compito di un operatore al pubblico è quindi quello di **vendere il biglietto di accesso al visitatore**; una volta scelta la tariffa (tra quelle sopra riportate) e scelto l'eventuale turno guida, l'operatore si assicura di richiedere al cliente, nel caso in cui egli non abbia mostrato un abbonamento, i **dati facoltativi per le statistiche anonime**. Se invece il cliente ha mostrato il suo abbonamento, sarà il sistema ad ottenere questi dati dall'abbonamento stesso e quindi non sarà necessaria nessun'altra operazione da parte dell'operatore. Infine l'operatore provvede a convalidare il biglietto del cliente al momento dell'accesso eseguendo una scannerizzazione del codice qr e ottenendo dunque risposta dal sistema.

Un'altra responsabilità dell'operatore al pubblico è quella di mostrare il **catalogo delle opere in vendita** al cliente che lo richiede, dopo aver verificato il possesso di un valido abbonamento, e di portare a termine dunque la vendita dell'eventuale opera scelta.

Si occupa inoltre di **creare richieste per le donazioni** ricevute da parte dei visitatori volenterosi. Una volta compilate le specifiche dell'opera oggetto di donazione, tra i quali dimensioni, tipologia e fotografia, sceglie una **località provvisoria** dove depositare temporaneamente l'opera fisica. Una volta richiesto un **canale di comunicazione** (email o numero telefonico) al donante, che verrà usato per comunicare l'esito, la richiesta viene inoltrata alla **segreteria** che la prenderà in carico successivamente.

A seconda del proprio turno, gli operatori svolgono anche il **ruolo di guida**. È presente un tour guidato **ogni ora** e ogni tour ha una **durata variabile** a seconda del contesto. I turni delle guide **sono definiti dall'amministrazione** e gli operatori sono tenuti ad assistere il gruppo che a loro è stato assegnato. Essi possono visualizzare i loro turni di guide accedendo al software. Le guide possono accompagnare gruppi di **massimo 20 persone**, senza necessità di raggiungere un numero minimo. Se un turno ha raggiunto la capienza massima, risulterà non disponibile e non verrà visualizzato fra i disponibili dall'operatore.

1.2.2 La segreteria

La *segreteria* si occupa principalmente del **lato servizio-cliente**. Nello specifico le responsabilità di questo reparto possono essere riassunte nelle seguenti operazioni:

- **Vendita, rinnovo e convalida** degli abbonamenti;
- Gestione delle **richieste di donazioni**.

Nel momento in cui un visitatore decide di volersi abbonare, si reca in segreteria. Qui il *segretario* gli comunica le possibili soluzioni per la sottoscrizione dell'abbonamento, che interessano **la durata** dello stesso. La scelta della tipologia è limitata alle seguenti offerte:

Durata	Prezzo (€)
1 mese	€9.99
3 mesi	€24.99
6 mesi	€49.99
12 mesi	€99.99

Dopodiché al cliente viene richiesto di fornire i propri dati (si veda la descrizione del **CU 11** per tutti i dettagli) che vengono registrati nel sistema.

Ogni nuovo abbonamento, per la stessa motivazione sopra fornita riguardo al biglietto, **ha un identificativo univoco**, stampato sotto forma di codice QR, che verrà utilizzato dal cliente abbonato per la convalida e quindi per il riconoscimento del diritto alla tariffa gratuita per l'accesso alla struttura. Alla scadenza dell'abbonamento, **al fine di mantenere la sua validità, il cliente ha la possibilità di rinnovarlo**.

Come detto, l'altro compito della segreteria è quello di occuparsi della **gestione delle donazioni di opere**, che potranno poi essere esposte nelle successive mostre organizzate. Quando una richiesta di donazione viene presa in carico, essa può essere **accettata o rifiutata**; in entrambi i casi **il donante verrà notificato dell'esito** via il canale di comunicazione rilasciato al momento della donazione.

1.2.3 L'Amministrazione

L'Amministrazione si occupa delle questioni interne al museo, che nello specifico possono essere riassunte nei seguenti punti:

- **Allestimento delle mostre;**
- **Acquisto** delle opere;
- Gestione del **personale interno**.
- Redazione **statistiche e report incassi**.
- Gestione dei **turni delle guide**.

Nel momento in cui l'amministrazione decide di voler allestire una mostra, si impegna a cercare opere. Essa può sceglierle fra **quelle già possedute** o può **acquistarne di nuove**. Subito dopo un acquisto, si impegna a **registrare nel sistema l'opera con i suoi dati** quali nome, autore, provenienza, data di acquisto e costo.

Terminata la fase di acquisto, l'amministrazione procede ad **organizzare la mostra**. Per prima cosa **sceglie le opere** da allestire nella mostra; dopodiché programma l'evento decidendo una **data di inizio e una di fine**. Come detto in precedenza, tutte le opere che i clienti osservano nelle mostre **possono essere acquistate** dagli stessi, a patto di disporre di un valido abbonamento.

Un'altra responsabilità dell'amministrazione è quella di **gestire il personale interno**, dunque è sua facoltà quella di **assumere** nuovo personale o **licenziare** quello già presente. Ai dipendenti assunti **verranno assegnati dei ruoli** all'interno della struttura museale, ma questi **non saranno necessariamente fissi**, bensì potranno cambiare nel tempo su decisione dell'amministrazione.

Inoltre, l'amministrazione si occupa di controllare **il lato finanziario**, valutando le **statistiche** fatte dal sistema sui visitatori, e il **report degli incassi** a fine mese. Le statistiche verranno dunque redatte sulla base delle informazioni richieste in forma anonima da parte degli operatori al pubblico ai visitatori entranti nella struttura museale.

Infine l'amministrazione ha la responsabilità di gestire e organizzare gli **orari dei turni delle guide**; dovrà assegnare agli operatori al pubblico dei turni di lavoro come guida assicurandosi però di lasciare sempre disponibile almeno uno di essi alla **reception** in modo da accogliere i nuovi visitatori.

1.3 Il Sistema

Tenendo presente quanto ho detto finora, il sistema che proponiamo deve **supportare tutti gli enti coinvolti** nella struttura museale nel loro lavoro, dovrà quindi fornire un sistema di **memorizzazione dei dati** quali quelli riguardanti i clienti, i biglietti, gli abbonamenti, ma anche le informazioni interne al museo come le opere i dipendenti stessi.

Il presente sistema gestionale, inoltre, deve fornire una serie di **azioni automatizzate** che facilitano i lavori più comuni, quali quelli di stampare un biglietto da parte dell'operatore al pubblico, o quelli di redigere statistiche a fine mese per gli amministratori.

Per quanto riguarda invece la **sicurezza**, il sistema dovrà accertarsi di effettuare un **backup** ogni sera alle 23:59 su un file e di mantenerne una copia sia localmente sia su un cloud dedicato; in questo modo sarà garantita la permanenza delle informazioni in caso di guasti o imprevisti.

1.4 Glossario di Progetto

Il glossario di progetto è il dizionario che contiene i **principali termini coinvolti nella logica di business** comprensivi della loro descrizione. Inoltre evidenzia eventuali sinonimie. Viene riportato anche il tipo del termine.

Termine	Significato	Tipo	Sinonimi
Abbonamento	Documento che viene rilasciato al cliente che ne abbia fatto richiesta e che riserva servizi aggiuntivi, come l'acquisto delle opere.	Business	Nessuno
Amministratore	Gestore del museo. Si occupa della parte finanziaria e dell'organizzazione interna di museo e mostre come la ricerca delle opere.	Business	Nessuno
Backup	Copia di sicurezza dei dati contenuti nel sistema.	Technical	Nessuno
Reception	Luogo all'ingresso del museo, dove gli operatori al pubblico accolgono la clientela e stampano biglietti	Business	Nessuno
Biglietto	Oggetto cartaceo il cui possesso dà diritto all'entrata al museo o alla mostra. Il biglietto ridotto si differenzia per una variazione di prezzo dal biglietto standard.	Business	Nessuno
Catalogo	Lista di opere vendibili possedute dalla struttura.	Business	Nessuno
Visitatore	Colui che fruisce dei servizi offerti dalla struttura museale	Business	Nessuno
Cliente	Cliente munito di abbonamento al quale sono riservati privilegi all'interno della struttura museale.	Business	Cliente abbonato
Mostra	Disposizione temporanea che permette ai clienti di poter visionare opere d'arte e oggetti.	Business	Nessuno
Museo	Raccolta fissa di opere d'arte ed oggetti esposti al pubblico. Allo stesso tempo anche l'edificio destinato ad ospitarle.	Business	Nessuno
Opera	Qualunque prodotto, manufatto come dipinti, scultura nato dalla creatività, abilità, tecnica e passione dell'uomo.	Business	Nessuno
Operatore al Pubblico	Utenti che gestiscono i servizi per i clienti, la loro entrata al museo e il servizio di tour guidato.	Business	Guida
Pagamento	Trasferimento di una somma di denaro come corrispettivo di una prestazione..	Business	Nessuno
Segreteria	Sportello che gestisce il lato servizio-cliente come prenotazioni e sottoscrizione di abbonamenti.	Business	Nessuno
Statistiche	Elaborazione dei dati acquisiti sulle visite e sui visitatori al fine di analizzare l'andamento dell'attività e creare report.	Business	Nessuno
Struttura Museale	Struttura che ospita il museo e la mostra.	Business	Struttura
Tour Guidato	Visita di gruppo del museo, accompagnata da una guida che ne illustra i contenuti.	Business	Tour
QR-code	Codice scansionabile presente su biglietto e sull'abbonamento contenente informazioni per l'accesso alla struttura.	Business	Codice QR

2 Analisi dei Requisiti

Di seguito verranno mostrati tutti i **requisiti che il sistema deve soddisfare** per poter adempiere alle funzioni richieste dal personale intervistato.

Essi sono divisi in requisiti **funzionali** e requisiti **non funzionali**, a seconda che siano relativi a concetti legati alla **logica di business** dettata dal **dominio del problema** o ad aspetti più tecnici, legati quindi al **dominio della soluzione**.

2.1 Package dei requisiti

In una visione generale, i package che contengono i requisiti, suddivisi in **funzionali** e **non funzionali**, sono i seguenti:

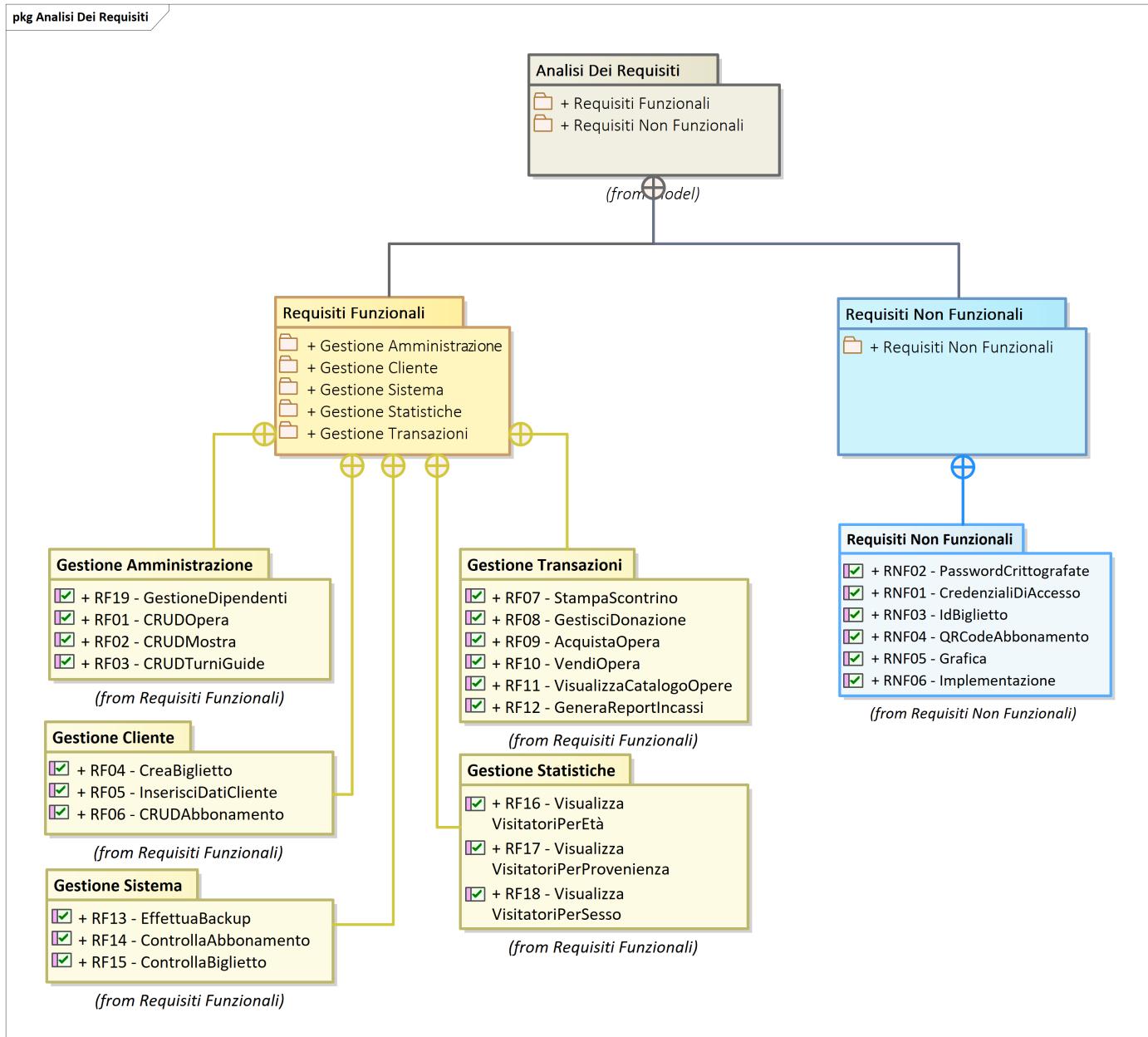


Figura 1: Package: **Analisi dei Requisiti**

2.2 Requisiti Funzionali

I requisiti funzionali raccolgono tutti i requisiti di business richiesti dagli stakeholders, rappresentano i punti chiave del programma gestionale da realizzare. Si è scelto di utilizzare il metodo **MoSCoW** per l'assegnamento della **priorità** a ciascuno dei requisiti funzionali.

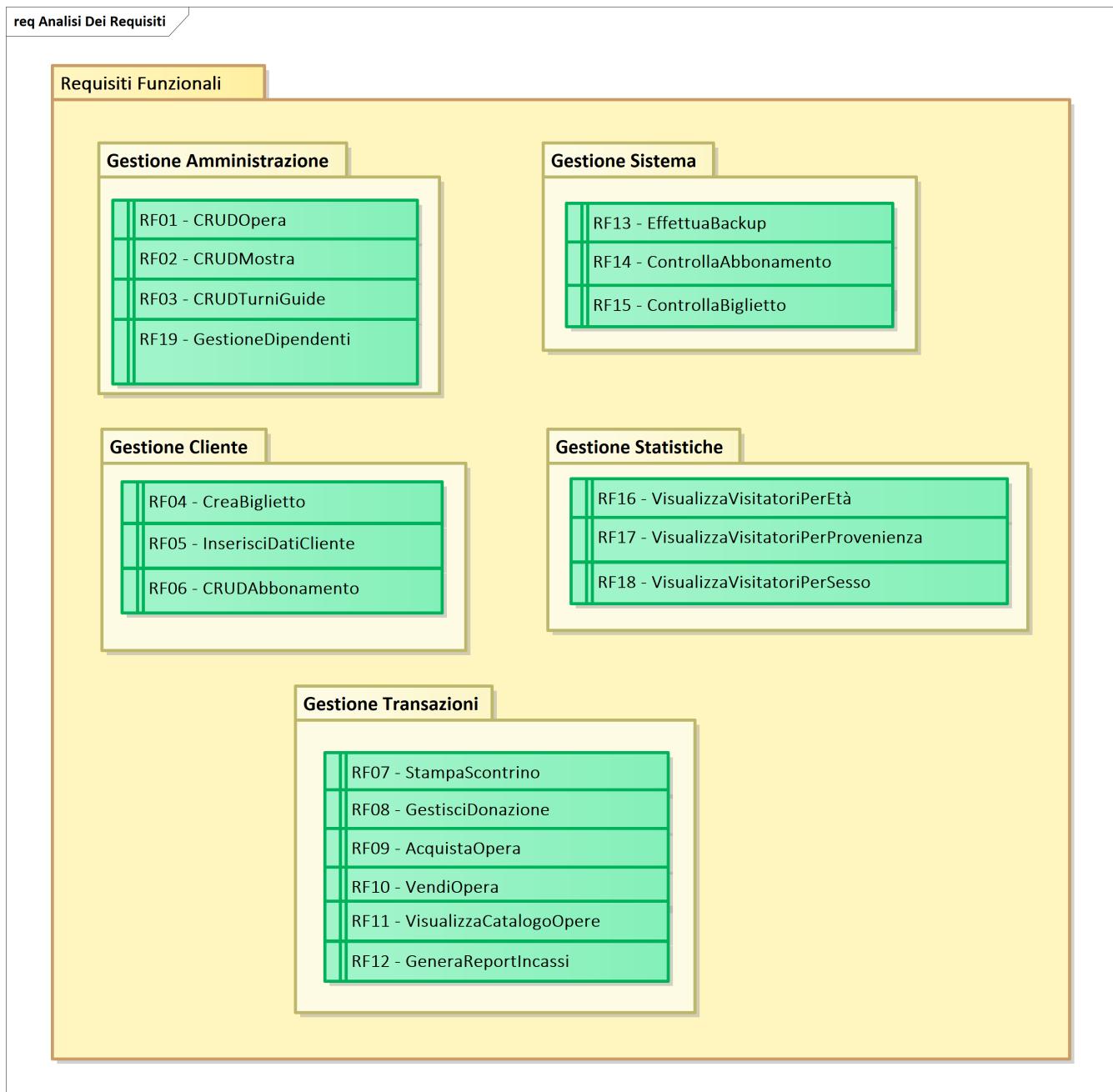


Figura 2: Package: **Requisiti Funzionali**

Gestione Amministrazione

- RF1 – **CRUDOpera**: Il sistema dovrà gestire le attività CRUD delle opere. ***Must have***
- RF2 – **CRUDMostra**: Il sistema dovrà gestire le attività CRUD delle mostre.
- RF3 – **CRUDMostra**: Il sistema dovrà gestire le attività CRUD dei turni delle guide. ***Must have***
- RF4 – **CreaBiglietto**: Il sistema dovrà permettere di creare il biglietto. ***Must have***

Gestione Cliente

- RF5 – **Inserisci dati Cliente**: Il sistema dovrà permettere di inserire i dati del cliente. ***Must have***
- RF6 – **CRUDAbbonamento**: Il sistema dovrà gestire le attività CRUD dell'abbonamento. ***Must have***

Gestione Transazioni

- RF7 – **StampaScontrino**: Il sistema dovrà permettere la stampa dello scontrino. ***Could have***
- RF8 – **GestisciDonazione**: Il sistema dovrà gestire le donazioni. ***Must have***
- RF9 – **CRUDAcquistaOpera**: Il sistema dovrà gestire le attività CRUD dell'acquisto delle opere. ***Should have***
- RF10 – **CRUDVendiOpera**: Il sistema dovrà gestire le attività CRUD della vendita delle opere. ***Should have***
- RF11 – **VisualizzaCatalogoOpere**: Il sistema dovrà permettere di visualizzare il catalogo delle opere. ***Must have***
- RF12 – **GeneraReportIncassi**: Il sistema dovrà generare il report degli incassi. ***Must have***

Gestione Sistema

- RF13 – **EffettuaBackup**: Il sistema dovrà effettuare un backup periodico dei dati. ***Must have***
- RF14 – **ControllaAbbonamento**: Il sistema dovrà controllare la validità dell'abbonamento e inviare un avviso in caso di prossima scadenza. ***Should have***
- RF15 – **ControllaBiglietto**: Il sistema dovrà controllare la validità del biglietto. ***Must have***

Gestione Statistiche

- RF16 – **VisualizzaStatistiche**: Il sistema dovrà permettere di visualizzare le statistiche. ***Must have***
- RF17 – **VisualizzaVisitatoriPerEtà**: Il sistema dovrà permettere di visualizzare i visitatori per età. ***Should have***
- RF18 – **VisualizzaVisitatoriPerNazionalità**: Il sistema dovrà permettere di visualizzare i visitatori per nazionalità. ***Should have***
- RF19 – **VisualizzaVisitatoriPerSesso**: Il sistema dovrà permettere di visualizzare i visitatori per sesso. ***Should have***

2.3 Requisiti non Funzionali

I requisiti non funzionali raccolgono tutti i requisiti relativi agli aspetti più tecnici del progetto. Una parte importante del flusso di lavoro è stata dedicata all'**interfaccia grafica**, tassello fondamentale per un software che sia usabile con semplicità e il più possibile *user-friendly*.

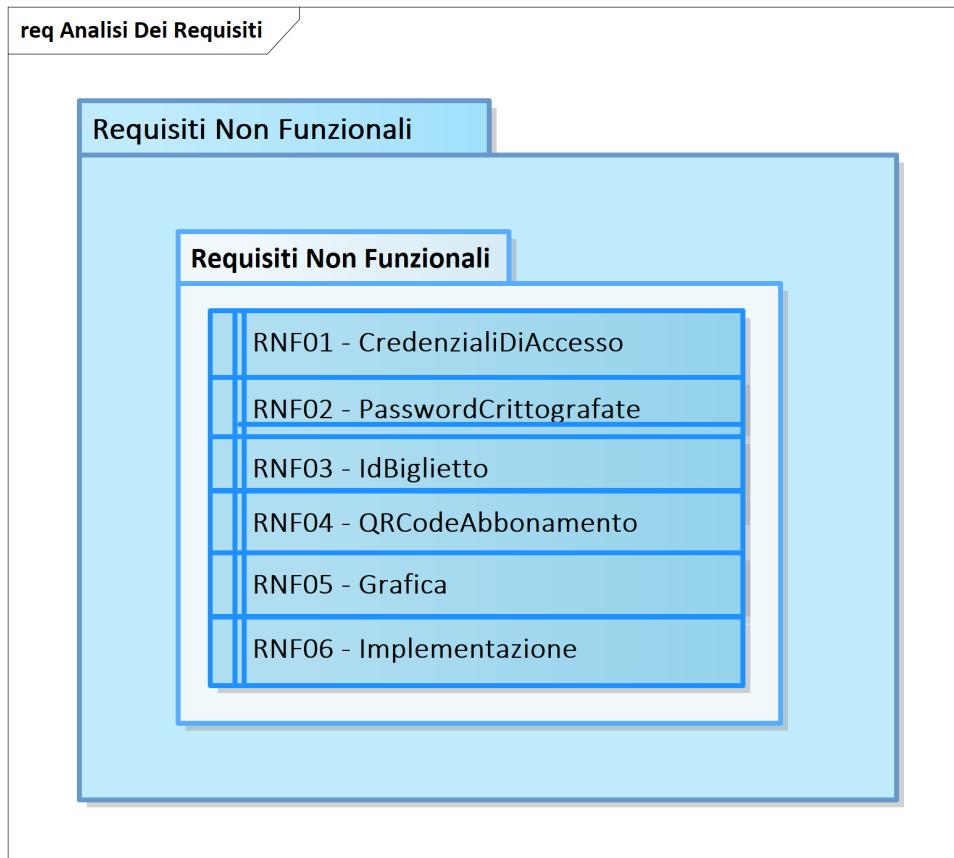


Figura 3: Package: **Requisiti Non Funzionali**

- RNF1 – **IdOpere**: Il sistema dovrà fornire un id univoco per ogni opera.
- RNF2 – **IdBiglietto**: Il sistema dovrà fornire un id univoco per ogni biglietto.
- RNF3 – **QRCodeAbbonamento**: Il sistema dovrà fornire un QRCode per ogni abbonamento.
- RNF4 – **Grafica**: Il sistema dovrà essere fornito di interfaccia grafica facilmente intuibile dall'utilizzatore anche inesperto.
- RNF5 – **Implementazione**: Il sistema dovrà essere implementato in Python versione 3. L'interfaccia grafica dovrà essere realizzata con la libreria Qt.

3 Diagrammi dei casi d'uso

Seguono ora i **diagrammi dei casi d'uso** e per ciascuno di essi una dettagliata descrizione dei casi d'uso presenti. Per rendere il più possibile comoda la lettura da parte del lettore, gli autori hanno scelto di **ordinare la descrizione dei casi d'uso secondo il loro identificativo numerico crescente**. Tale identificativo è stato riportato anche sulle rappresentazioni grafiche dei diagrammi.

Si tenga inoltre presente che, al fine di non appesantire le rappresentazioni grafiche, sono state omesse le relazioni di associazione tra attore e caso d'uso qualora esse fossero state già esplicitate in altri diagrammi e ripeterle nell'attuale **non avrebbe contribuito in modo indispensabile alla comprensione** della logica del sistema. Sempre nel perseguitamento di tale scopo si è scelto di colorare in grigio i casi d'uso presenti in un diagramma, ma originari di un altro diagramma.

3.1 Package dei casi d'uso

Il diagramma dei casi d'uso è composto da 5 *packages* oltre al *package* degli attori; nel dettaglio essi sono:

- Gestione Cliente (CU 01) - (CU 10)
- Gestione Segreteria (CU 11) - (CU 15)
- Gestione Amministrazione (CU 16) - (CU 23)
- Gestione Sistema (CU 24) - (CU 27)
- Gestione Statistiche (CU 28) - (CU 35)

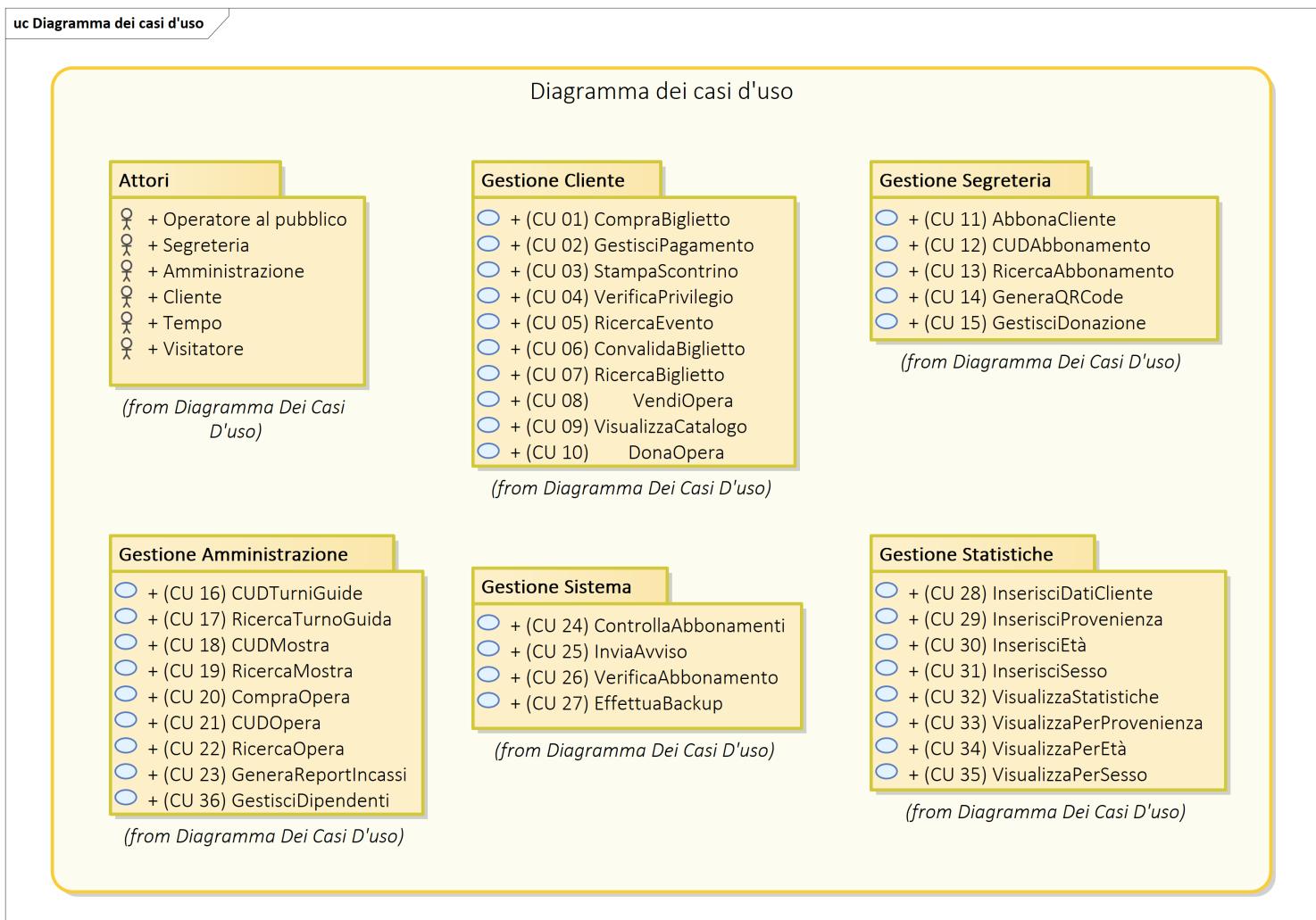


Figura 4: diagramma dei casi d'uso: **Diagramma dei casi d'uso**

3.2 Attori

Il package degli attori contiene i 'soggetti' del dominio del problema in esame, ovvero coloro che interagiranno con il presente sistema al fine di svolgere il loro lavoro. Nel dettaglio i 6 attori sono:

- Visitatore
- Cliente
- Operatore al pubblico
- Segreteria
- Amministrazione
- Tempo

Si noti la **relazione di generalizzazione** che prende posto tra l'attore *Visitatore* e l'attore *Cliente*. Mentre del secondo il sistema possiede le informazioni anagrafiche lasciate al momento della sua registrazione, non possiede che dati in forma anonima del *Visitatore*, ovvero di colui che usufruisce dei servizi offerti dal museo senza essere in possesso di un abbonamento. In altre parole, poiché il presente sistema **non obbliga il *Visitatore* a registrarsi per accedere alla struttura** e usufruire dei suoi servizi, gli unici dati che il sistema memorizza su di lui sono quelli (facoltativi) rilasciati in forma anonima per le statistiche. Questo concetto verrà certamente ripreso nel dettaglio a livello di analisi.

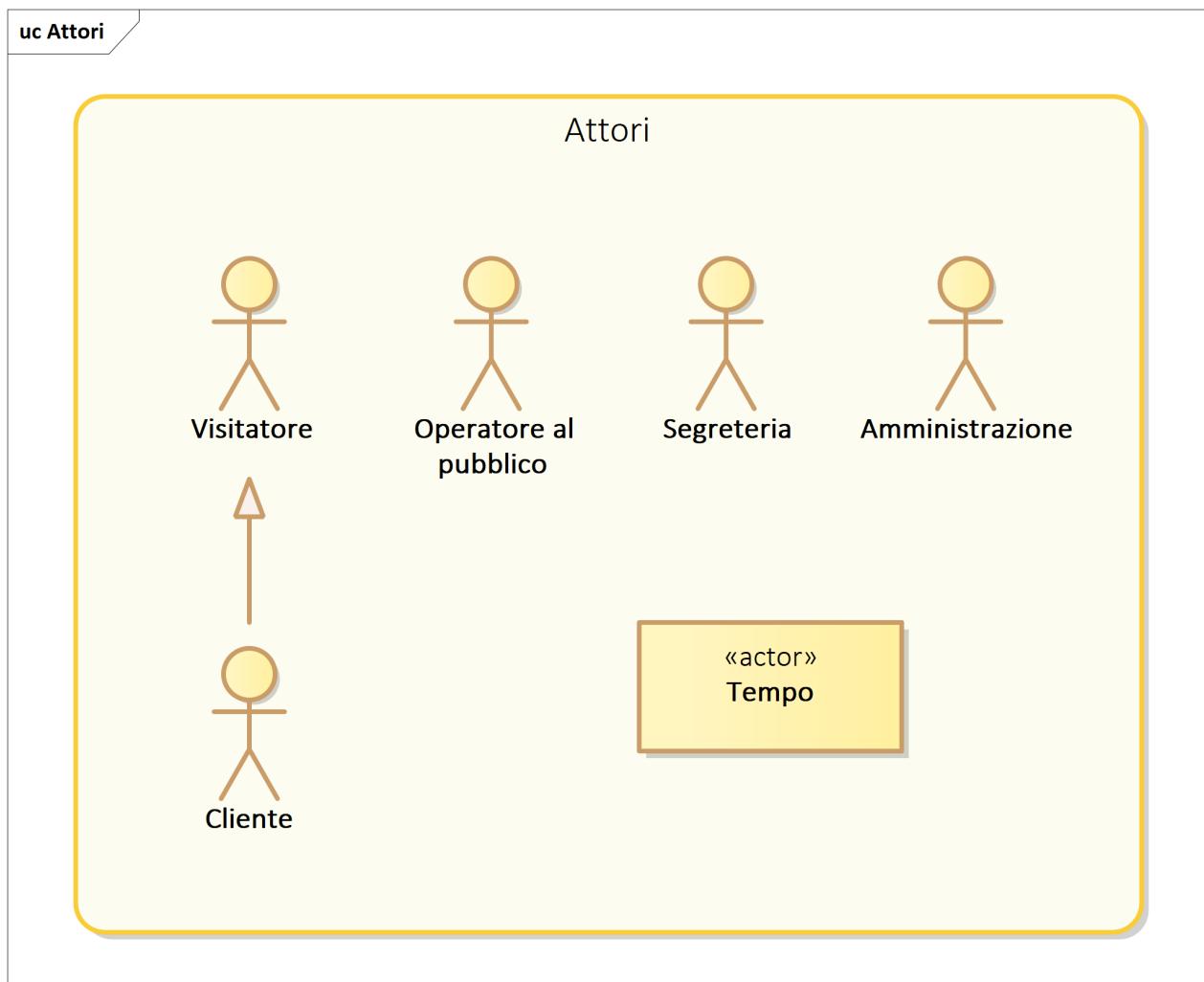


Figura 5: Package: **Attori**

3.3 Gestione Cliente

Il diagramma dei casi d'uso **Gestione Cliente** racchiude i casi d'uso volti a fornire servizi al cliente. Nel dettaglio, i casi d'uso di pertinenza sono:

- CompraBiglietto (CU 01)
- GestisciPagamento (CU 02)
- StampaScontrino (CU 03)
- VerificaPrivilegio (CU 04)
- RicercaEvento (CU 05)
- ConvalidaBiglietto (CU 06)
- RicercaBiglietto (CU 07)
- VendOpera (CU 08)
- VisualizzaCatalogo (CU 09)
- DonaOpera (CU 10)

Gli attori coinvolti in questi casi d'uso sono il *Cliente*, generalizzato nell'attore *Visitatore*, e l'attore *Operatore al pubblico*.

Si noti che, poiché questo sistema informatico **non prevede terminali sui quali il visitatore si può interfacchiare**, i casi d'uso istanziabili dal visitatore hanno come attore secondario l'operatore al pubblico, il quale, una volta compresa la necessità del visitatore, avvia la procedura interfacciandosi con il sistema.

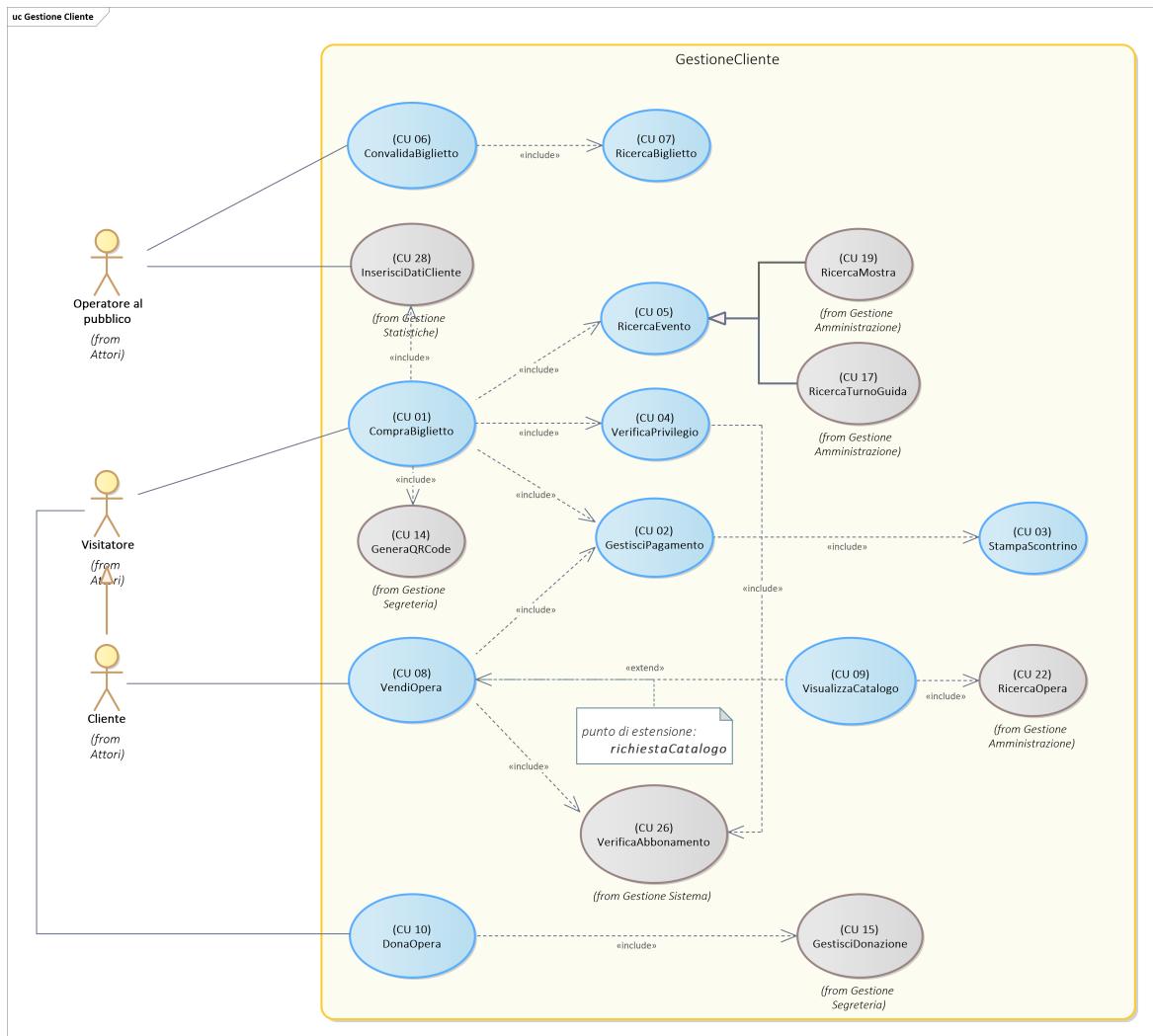


Figura 6: diagramma dei casi d'uso: **Gestione Cliente**

CompraBiglietto (CU 01)

Questo caso d'uso consente l'acquisto di un biglietto per l'accesso al museo o ad una mostra.

Attori primari: Visitatore.

Attori secondari: Operatore al pubblico.

Precondizioni: Nessuna.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario esprime la volontà di acquistare un biglietto.
2. L'attore secondario prende coscienza della volontà dell'attore primario e lo assiste nell'operazione.
3. L'attore secondario effettua l'accesso con le sue credenziali se richiesto.
4. **if** l'attore primario vuole accedere alla mostra:
 - 4.1. L'attore secondario seleziona la mostra attuale.
5. L'attore secondario inserisce nel sistema i dati relativi alla tipologia del biglietto.
6. **if** l'attore primario richiede anche il servizio guida:
 - 6.1. `include(RicercaTurnoGuida)`.
 - 6.2. L'attore secondario comunica all'attore primario gli orari dei turni delle guide disponibili.
 - 6.3. **if** c'è almeno un turno disponibile **and** l'attore primario ne sceglie uno:
 - 6.3.1. L'attore secondario inserisce la scelta dell'attore primario nel sistema.
7. **if** l'attore primario comunica all'attore secondario di voler usufruire della tariffa ridotta:
 - 7.1. `include(VerificaPrivilegio)`.
8. Il sistema memorizza le informazioni e calcola il costo totale del biglietto, mostrando in un opportuno il resoconto dell'operazione.
9. `include(InserisciDatiCliente)`.
10. `include(GestisciPagamento)`.
11. Il sistema genera un identificativo univoco per il biglietto.
12. `include(GeneraQRCode)`.
13. Il sistema applica il codice QR sul fronte del biglietto e lo stampa.

Sequenza eventi alternativa: *PrivilegioNonVerificato*

1. La sequenza di eventi alternativa inizia dopo il passo 6.1. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore secondario la non validità del privilegio necessario per poter usufruire della tariffa ridotta.

Sequenza eventi alternativa: *AbbonamentoScaduto*

1. La sequenza di eventi alternativa inizia dopo il passo 6.1. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore secondario la non validità dell'abbonamento inserito.

Sequenza eventi alternativa: *PagamentoRifiutato*

1. La sequenza di eventi alternativa inizia dopo il passo 8. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore primario l'errore avvenuto durante il pagamento ed esortandolo a riprovare.

GestisciPagamento (CU 02)

Questo caso d'uso permette di gestire il pagamento di un servizio offerto dal museo.

Attori primari: Segreteria, Operatore al pubblico.

Attori secondari: Visitatore.

Precondizioni:

1. Il sistema sta elaborando la richiesta da parte del visitatore di acquisto di un biglietto, di un abbonamento **or** di un'opera.
2. L'attore primario ha comunicato al sistema il bene che il visitatore desidera acquistare.

Postcondizioni:

1. Il sistema ha registrato il pagamento.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario deve gestire il pagamento di un acquisto effettuato da parte del visitatore.
2. Il sistema comunica all'attore primario il costo totale dell'acquisto.
3. L'attore primario comunica all'attore secondario il totale da pagare e gli chiede la modalità di pagamento che desidera seguire.
4. **if** l'attore secondario comunica di voler pagare in contanti:
 - 4.1. L'attore primario gestisce manualmente il pagamento usufruendo della cassa per erogare l'eventuale resto e riporre il denaro.
 - 4.2. L'attore primario comunica l'avvenuto pagamento al sistema.
5. **else if** l'attore secondario comunica di voler pagare utilizzando il *Point Of Sale*:
 - 5.1. L'attore primario consegna all'attore secondario il *Point Of Sale*.
 - 5.2. L'attore secondario inserisce la sua carta di credito, di debito **or** prepagata, e segue i passaggi richiesti per autenticarla.
 - 5.3. Il sistema riceve l'esito della transazione dal *Point Of Sale*.
6. Il sistema registra l'avvenuto pagamento.
7. **include(StampaScontrino)**

Sequenza eventi alternativa: *TransazioneNonRiuscita*

1. La sequenza di eventi alternativa inizia dopo il passo 5.3. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore primario l'errore avvenuto durante il pagamento ed esortandolo a ripetere l'operazione.

StampaScontrino (CU 03)

Questo caso d'uso permette di stampare lo scontrino con l'importo pagato.

Attori primari: Segreteria, Operatore al pubblico.

Attori secondari: Visitatore.

Precondizioni:

1. Il processo di pagamento terminato ed è andato a buon fine.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario deve stampare lo scontrino per l'avvenuto pagamento.
2. Il sistema comunica al registratore di cassa telematico le informazioni del pagamento.
3. L'attore primario attende l'avvenuta stampa e consegna all'attore secondario.

Sequenza eventi alternativa: *ConnessioneAssente*

1. La sequenza di eventi alternativa inizia dopo il passo 2. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore primario l'impossibilità di connessione da parte del registratore di cassa telematico, e lo esorta a ripovare **or** a riportare a mano lo scontrino su carta.

VerificaPrivilegio (CU 04)

Questo caso d'uso permette di verificare il privilegio di un visitatore ad aderire al costo ridotto o di essere esente dal pagamento per l'acquisto di un biglietto.

Attori primari: Operatore al pubblico.

Attori secondari: Visitatore.

Precondizioni:

1. Si è verificata la volontà da parte del visitatore di acquistare un biglietto.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario desidera verificare se il visitatore può usufruire dei privilegi di costo ridotto o nullo.
2. L'attore primario chiede l'attore secondario quale privilegio desidera sfruttare.
3. **if** l'attore secondario chiede di utilizzare il privilegio sull'età :
 - 3.1. L'attore primario chiede all'attore secondario di mostrare un documento d'identità.
 - 3.2. **if** l'età dell'attore secondario è minore o uguale a 5 anni:
 - 3.2.1. L'attore primario conferma l'esenza dal pagamento.
 - 3.3. **else if** L'età dell'attore secondario è minore di 18 anni **or** è maggiore di 59 anni:
 - 3.3.1. L'attore primario conferma l'aderenza alla tariffa ridotta.
4. **else if** l'attore secondario chiede di utilizzare il privilegio sull'occupazione :
 - 4.1. L'attore primario chiede all'attore secondario di mostrare il badge di studente **or** il badge di professore.
 - 4.2. **if** l'attore secondario ha consegnato il badge di studente **and** l'attore secondario ritiene valido il badge consegnatogli:
 - 4.2.1. L'attore primario conferma l'aderenza alla tariffa ridotta.
 - 4.3. **else if** l'attore secondario ha consegnato il badge di professore **and** l'attore secondario ritiene valido il badge consegnatogli:
 - 4.3.1. L'attore primario conferma l'esenza dal pagamento.
5. **else if** l'attore secondario chiede di utilizzare il privilegio sul possedimento di certificato di invalidità:
 - 5.1. L'attore primario chiede all'attore secondario di mostrare il certificato di invalidità.
 - 5.2. L'attore primario inserisce nel sistema il codice del certificato di invalidità.
 - 5.3. **if** il sistema conferma la validità del certificato:
 - 5.3.1. L'attore primario conferma l'esenza dal pagamento.
6. **else if** l'attore secondario chiede di utilizzare il privilegio sul possedimento di abbonamento:
 - 6.1. L'attore primario chiede all'attore secondario di mostrare il suo abbonamento.
 - 6.2. **include(VerificaAbbonamento)**
 - 6.3. **if** l'abbonamento non è scaduto:
 - 6.3.1. L'attore primario conferma l'esenza dal pagamento.
7. **else** l'attore primario non applica alcuna riduzione al costo del biglietto.

Sequenza eventi alternativa: Nessuna.

RicercaEvento (CU 05)

Questo caso d'uso consente di ricercare un evento.

Attori primari: Operatore al pubblico.

Attori secondari: Nessuno.

Precondizioni: Nessuna.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario vuole effettuare la ricerca di un evento **or** il sistema deve effettuare la ricerca di un evento.
2. **if** il caso d'uso è stato avviato dall'attore:
 - 2.1. Il sistema chiede i criteri di ricerca.
 - 2.2. L'attore inserisce i criteri di ricerca richiesti.
3. Il sistema registra i parametri di ricerca.
4. **for each** evento nel database:
 - 4.1. Il sistema controlla se l'evento soddisfa i criteri di ricerca.
 - 4.2. **if** l'evento soddisfa il criterio:
 - 4.2.1. Il sistema aggiunge l'evento alla lista dei risultati.
5. **if** la lista dei risultati contiene almeno un evento:
 - 5.1. Il sistema mostra la lista dei risultati con tutte le loro informazioni.
6. **else:**
 - 6.1. Il sistema comunica all'attore che nessun evento soddisfa il criterio specificato.

Sequenza eventi alternativa: Nessuna.

ConvalidaBiglietto (CU 06)

Questo caso d'uso consente di verificare la validità di un biglietto mediante il suo identificativo stampato sul fronte sotto forma di codice QR.

Attori primari: Operatore al pubblico.

Attori secondari: Nessuno.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.
2. L'attore primario dispone di un biglietto con codice QR.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario vuole eseguire la convalida di un biglietto.
2. Il sistema chiede il codice identificativo del biglietto.
3. **while** il codice identificativo non corrisponde ad alcun biglietto memorizzato nel sistema:
 - 3.1. L'attore primario inserisce il codice identificativo eseguendo una scannerizzazione del codice QR stampato sul fronte del biglietto.
 - 3.2. **include(RicercaBiglietto)**
4. **if** la data attuale conincide con il giorno di validità del biglietto:
 - 4.1. Il sistema mostra un opportuno messaggio di successo dell'operazione.
5. **else:**
 - 5.1. Il sistema mostra un opportuno messaggio di errore.

Sequenza eventi alternativa: *IdentificativoNonTrovato*

1. La sequenza di eventi alternativa inizia dopo il passo 3.2. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore l'inesistenza dell'identificativo all'interno del database ed esortandolo a ripetere il processo di scannerizzazione.
3. La sequenza di eventi principale prosegue dal punto 3.

RicercaBiglietto (CU 07)

Questo caso d'uso consente di ricercare un biglietto nel database del sistema mediante il suo identificativo.

Attori primari: Operatore al pubblico.

Attori secondari: Nessuno.

Precondizioni:

1. L'attore primario ha inserito il codice identificativo del biglietto o la data di ricerca nel sistema.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario vuole effettuare la ricerca di un biglietto **or** il sistema deve effettuare la ricerca di un biglietto.
2. Il sistema ottiene il parametro di ricerca fornito.
3. **for each** biglietto nel database dei biglietti:
 - 3.1. **if** il parametro di ricerca è l'ID del biglietto:
 - 3.1.1. **if** l'identificativo del biglietto coincide con l'identificativo fornito al sistema:
 - 3.1.1.1. Il sistema raccoglie tutte le informazioni del biglietto.
 - 3.2. **else if** il parametro di ricerca è la data di emissione del biglietto:
 - 3.2.1. **if** la data di emissione del biglietto appartiene al range fornito:
 - 3.2.1.1. Il sistema raccoglie tutte le informazioni del biglietto.
4. **if** la ricerca non ha prodotto alcun risultato:
 - 4.1. Il sistema mostra un opportuno messaggio di errore.
5. **else if** il caso d'uso è stato avviato dall'attore primario:
 - 5.1. Il sistema mostra un opportuno messaggio di successo dell'operazione.
 - 5.2. Il sistema mostra, in una schermata opportuna, tutte le informazioni del biglietto, tra le quali la sua tipologia e l'eventuale presenza del servizio di guida.

Sequenza eventi alternativa: Nessuna.

VendiOpera (CU 08)

Questo caso d'uso consente la vendita di un'opera da parte del museo.

Attori primari: Cliente.

Attori secondari: Operatore al pubblico.

Precondizioni:

1. L'attore primario deve disporre di un abbonamento valido.

Postcondizioni:

1. Nel caso in cui l'operazione sia andata a buon fine, il sistema ha segnato come vendute le opere oggetto di transazione.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario esprime la volontà di acquistare un'opera di una mostra.
2. L'attore secondario riceve la richiesta e chiede all'attore primario di mostrare il suo abbonamento.
3. **include(VerificaAbbonamento)**
4. **extention point: richiestaCatalogo**
5. **for each** opera scelta dall'attore primario:
 - 5.1. L'attore secondario aggiunge l'opera al carrello.
6. L'attore secondario comunica al sistema la conferma dell'operazione.
7. **include(GestisciPagamento)**
8. **for each** opera venduta:
 - 8.1. Il sistema segna come venduta l'opera.

Sequenza eventi alternativa: OperazioneAnnullata

1. La sequenza di eventi alternativa inizia in qualunque momento.
2. L'attore primario annulla l'acquisto delle opere.

Sequenza eventi alternativa: AbbonamentoScaduto

1. La sequenza di eventi alternativa inizia dopo il passo 3. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore secondario la non validità dell'abbonamento inserito.

Sequenza eventi alternativa: PagamentoRifiutato

1. La sequenza di eventi alternativa inizia dopo il passo 7. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore primario l'errore avvenuto durante il pagamento ed esortandolo a riprovare.

VisualizzaCatalogo (CU 09)

Questo caso d'uso permette di generare ed eventualmente stampare una lista delle opere appartenenti alle mostre del museo ed in quanto tali vendibili al visitatore richiedente.

Attori primari: Operatore al pubblico.

Attori secondari: Nessuno.

Precondizioni del segmento 1:

1. L'attore secondario ha richiesto esplicitamente la volonta di visualizzare il catalogo delle opere vendibili.

Postcondizioni del segmento 1: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario riceve la richiesta di visualizzare il catalogo delle opere vendibili.
2. **for each** mostra nel database delle mostre:
 - 2.1. **include(RicercaMostra)**
 - 2.2. Il sistema genera una presentazione della mostra.
 - 2.3. **for each** opera nella mostra:
 - 2.3.1. **include(RicercaOpera)**
 - 2.3.2. Il sistema genera una presentazione dell'opera e la aggiunge alla presentazione della mostra.
 - 2.4. Il sistema aggiunge la presentazione della mostra al catalogo.
3. Il sistema mostra a schermo il catalogo generato, offrendo all'attore primario la possibilità di mandarlo in stampa.

Sequenza eventi alternativa del segmento 1: *NessunaOperaVendibile*

1. La sequenza di eventi alternativa inizia dopo il passo 3. della sequenza di eventi principale.
2. Il sistema comunica all'attore primario l'assenza di opere vendibili e lo esorta a portare a termine l'operazione di vendita.

DonaOpera (CU 10)

Questo caso d'uso consente la donazione di un'opera da parte di un visitatore.

Attori primari: Visitatore.

Attori secondari: Operatore al pubblico, Segreteria.

Precondizioni: Nessuna.

Postcondizioni:

1. Il sistema ha memorizzato la richiesta insieme a tutti i dati dell'opera in fase di donazione.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario esprime la volontà di donare un'opera al museo.
2. L'attore secondario *Operatore al pubblico* riceve la richiesta ed inserisce nel sistema il nome, l'autore, le dimensioni, il tipo e l'immagine dell'opera.
3. L'attore secondario *Operatore al pubblico* chiede all'attore primario anche un indirizzo di posta elettronica attraverso il quale verrà comunicata la scelta della segreteria.
4. L'attore secondario *Operatore al pubblico* ripone l'opera in un magazzino appropriato in attesa dell'esito della richiesta.
5. Il sistema registra le informazioni e notifica l'attore *Segreteria* della richiesta.
6. `include(GestisciDonazione)`

Sequenza eventi alternativa: Nessuna.

3.4 Gestione Segreteria

Il diagramma dei casi d'uso **Gestione Segreteria** racchiude i casi d'uso necessari all'attore *Segreteria* per svolgere le sue operazioni orientate a migliorare il lato servizio-cliente. Nel dettaglio, i casi d'uso di pertinenza sono:

- **AbbonaCliente** (CU 11)
- **CUDAbbonamento** (CU 12)
- **RicercaAbbonamento** (CU 13)
- **GeneraQRCode** (CU 14)
- **GestisciDonazione** (CU 15)

L'attore coinvolto in questi casi d'uso è la *Segreteria*.

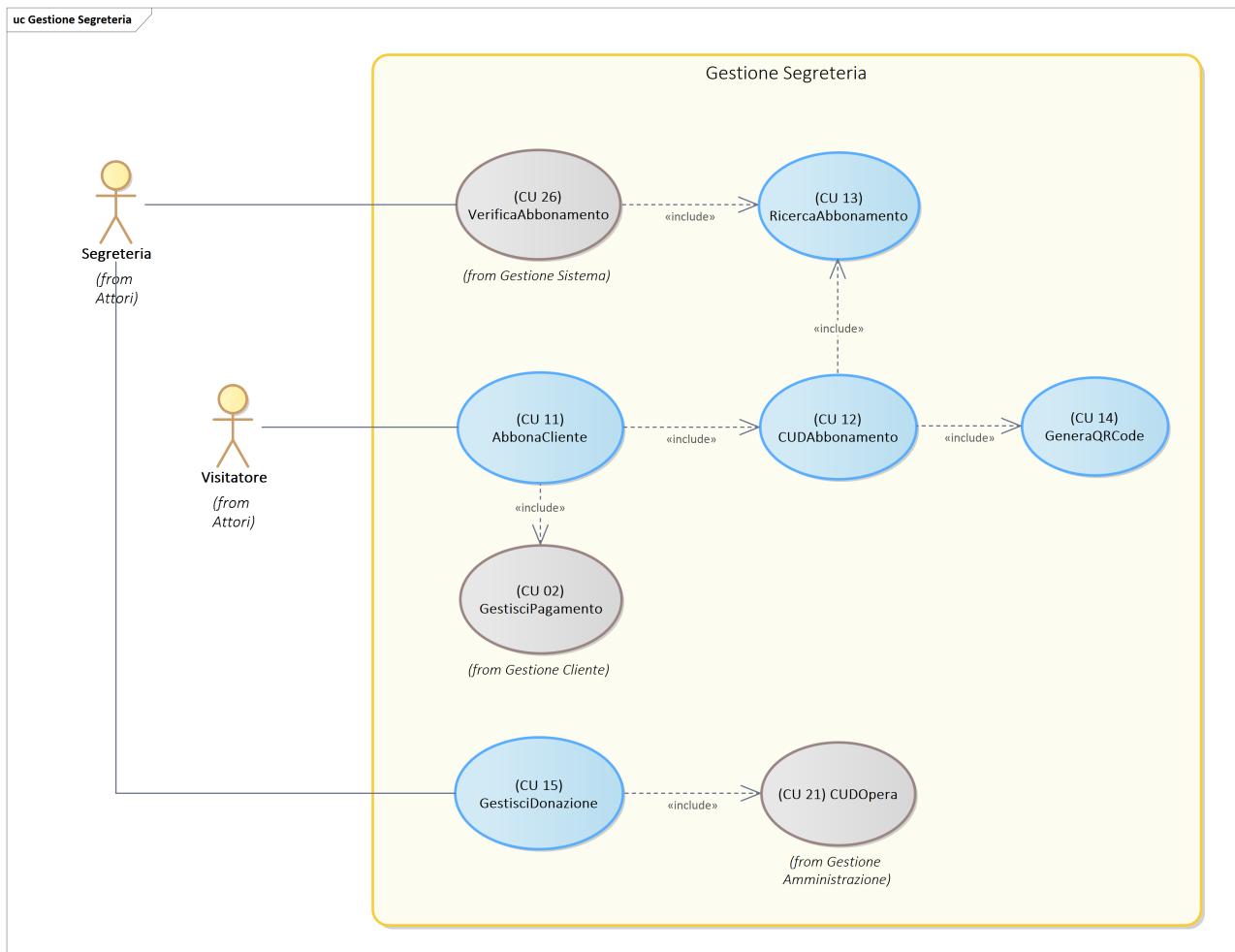


Figura 7: diagramma dei casi d'uso: **Gestione Segreteria**

AbbonaCliente (CU 11)

Questo caso d'uso consente la creazione di un abbonamento per un visitatore e il rinnovo di un abbonamento per un cliente abbonato.

Attori primari: Cliente.

Attori secondari: Segreteria.

Precondizioni: Nessuno.

Postcondizioni:

1. Il sistema ha memorizzato il nuovo abbonamento nel database degli abbonamenti.
2. Il sistema ha memorizzato il pagamento ricevuto.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario esprime la volontà di abbonarsi o di rinnovare il suo abbonamento.
2. L'attore secondario prende coscienza della volontà dell'attore primario e lo assiste nell'operazione.
3. **if** l'attore primario vuole abbonarsi:
 - 3.1. L'attore secondario richiede all'attore primario i dati anagrafici e li registra nel sistema.
4. **else if** l'attore primario vuole rinnovare il suo abbonamento:
 - 4.1. L'attore secondario chiede all'attore primario di mostrare il suo abbonamento.
 - 4.2. L'attore secondario inserisce il codice identificativo eseguendo una scannerizzazione del codice QR stampato sul fronte dell'abbonamento.
 - 4.3. **include(RicercaAbbonamento)**
5. L'attore secondario informa l'attore primario sulle tipologie di abbonamento disponibili in base alla loro durata e al loro costo, quindi chiede all'attore secondario i dati necessari per l'abbonamento.
6. L'attore primario sceglie una tipologia di abbonamento e comunica all'attore secondario i dati richiesti.
7. L'attore secondario inserisce i dati nel sistema.
8. **include(CUDAbbonamento)**
9. L'attore secondario comunica all'attore primario l'importo da pagare.
10. **include(GestisciPagamento)**
11. L'attore secondario comunica all'attore primario l'esito positivo dell'operazione.
12. Il sistema applica il codice QR sul fronte dell'abbonamento e lo stampa.

Sequenza eventi alternativa: *OperazioneAnnullata*.

1. La sequenza di eventi alternativa inizia in qualunque momento.
2. L'attore primario annulla la richiesta di abbonarsi.

Sequenza eventi alternativa: *PagamentoRifiutato*.

1. La sequenza di eventi alternativa inizia dopo il passo 9. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore primario l'errore avvenuto ed esortandolo a riprovare.

CUDAbbonamento (CU 12)

Questo caso d'uso consente di eseguire la creazione, la modifica o l'eliminazione di un abbonamento.

Attori primari: Segreteria.

Attori secondari: Nessuno.

Precondizioni: Nessuna.

Postcondizioni:

1. Il sistema ha memorizzato le eventuali modifiche nel database degli abbonamenti.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario vuole eseguire un'operazione CUD relativa all'abbonamento.
2. **if** l'attore primario vuole creare un nuovo abbonamento:
 - 2.1. L'attore primario inserisce i dati nel sistema.
 - 2.2. Il sistema provvede a memorizzare i dati.
 - 2.3. Il sistema genera un identificativo univoco per l'abbonamento.
 - 2.4. **include(GeneraQRCode)**
 - 2.5. Il sistema inserisce il codice QR nell'abbonamento.
 - 2.6. Il sistema mostra a schermo l'abbonamento creato e un messaggio contenente l'esito dell'operazione.
3. **else if** l'attore vuole modificare l'abbonamento esistente:
 - 3.1. **include(RicercaAbbonamento)**
 - 3.2. **if** l'abbonamento viene trovato:
 - 3.2.1. l'attore primario aggiorna i dati dell'abbonamento trovato.
 - 3.2.2. Il sistema provvede a salvare le modifiche.
4. **else if** l'attore primario **or** il sistema vuole eliminare l'abbonamento:
 - 4.1. **if** il caso d'uso è stato attivato dall'attore primario
 - 4.1.1. L'attore secondario chiede al cliente di mostrare il suo abbonamento.
 - 4.1.2. L'attore secondario inserisce il codice identificativo eseguendo una scannerizzazione del codice QR stampato sul fronte dell'abbonamento.
 - 4.1.3. **include(RicercaAbbonamento)**
 - 4.1.4. Il sistema mostra un messaggio di conferma.
 - 4.1.5. **if** l'attore primario conferma l'operazione:
 - 4.1.5.1. Il sistema provvede ad eliminare l'abbonamento.
 - 4.2. **else if** il caso d'uso è stato attivato dal sistema:
 - 4.2.1. **include(RicercaAbbonamento)**
 - 4.2.2. Il sistema provvede ad eliminare l'abbonamento.
 - 4.3. Il sistema notifica a schermo l'esito positivo dell'operazione.

Sequenza eventi alternativa: *AbbonamentoNonTrovato*.

1. La sequenza di eventi alternativa inizia dopo i passi 3.1., 4.1.3. **or** 4.2.1. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore primario di non aver trovato un abbonamento corrispondente a quello ricercato.

RicercaAbbonamento (CU 13)

Questo caso d'uso consente di eseguire la ricerca di un abbonamento all'interno del database.

Attori primari: Segreteria.

Attori secondari: Nessuno.

Precondizioni: Nessuna.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario desidera effettuare la ricerca di un abbonamento **or** il sistema deve effettuare la ricerca.
2. **if** il caso d'uso è stato attivato dall'attore primario
 - 2.1. L'attore primario inserisce il codice identificativo dell'abbonamento da ricercare.
3. **for each** abbonamento nel database degli abbonamenti:
 - 3.1. **if** il codice identificativo dell' abbonamento corrisponde a quello inserito nel sistema:
 - 3.1.1. Il sistema mostra in un'opportuna finestra l'abbonamento trovato con tutte le relative informazioni.
4. **if** la ricerca non ha prodotto un risultato:
 - 4.1. il sistema comunica di non aver trovato l'abbonamento.

Sequenza eventi alternativa: Nessuna.

GeneraQRCode (CU 14)

Questo caso d'uso consente di generare un codiceQR a partire da un identificativo fornito.

Attori primari: Segreteria.

Attori secondari: Nessuno.

Precondizioni:

1. Il sistema ha generato un identificativo.
2. Tale identificativo è univoco all'interno del database.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando il sistema deve generare un codice QR.
2. Il sistema utilizza l'identificativo fornito per generare il codice QR.

Sequenza eventi alternativa: Nessuna.

GestisciDonazione (CU 15)

Questo caso d'uso consente di gestire una richiesta di donazione di un'opera da parte di un cliente.

Attori primari: Segreteria.

Attori secondari: Nessuno.

Precondizioni: Nel sistema è presente almeno una richiesta di donazione di un'opera.

Postcondizioni:

1. Il sistema ha autenticato l'attore primario.
2. Nel caso in cui la richiesta sia stata accettata, il sistema ha provveduto a memorizzare la nuova opera nel database delle opere.
3. Il sistema ha mandato un avviso comunicante l'esito della scelta al visitatore donante mediante la posta elettronica.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario prende in carico la donazione.
2. Il sistema mostra in un'opportuna schermata tutti i dettagli della richiesta.
3. L'attore primario riceve le informazioni mostrate e decide se accettare la donazione.
4. L'attore primario inserisce la scelta nel sistema.
5. **if** il sistema ha ricevuto una conferma:
 - 5.1. **include(CUDOpera)**
6. Il sistema genera un messaggio di notifica in base alla scelta effettuata e la invia all'indirizzo di posta elettronica allegato alla richiesta.

3.5 Gestione Amministrazione

Il diagramma dei casi d'uso **Gestione Amministrazione** racchiude i casi d'uso necessari all'attore *Amministrazione* per svolgere le sue operazioni che riguardano la gestione interna e organizzativa del museo. Nel dettaglio, i casi d'uso di pertinenza sono:

- CUDTurniGuide (CU 16)
- RicercaTurnoGuida (CU 17)
- CUDMostra (CU 18)
- RicercaMostra (CU 19)
- CompraOpera (CU 20)
- CUDOOpera (CU 21)
- RicercaOpera (CU 22)
- GeneraReportIncassi (CU 23)
- GestisciDipendenti (CU 36)

Gli attori coinvolti in questi casi d'uso sono l' *Amministrazione* e il *Cliente*.

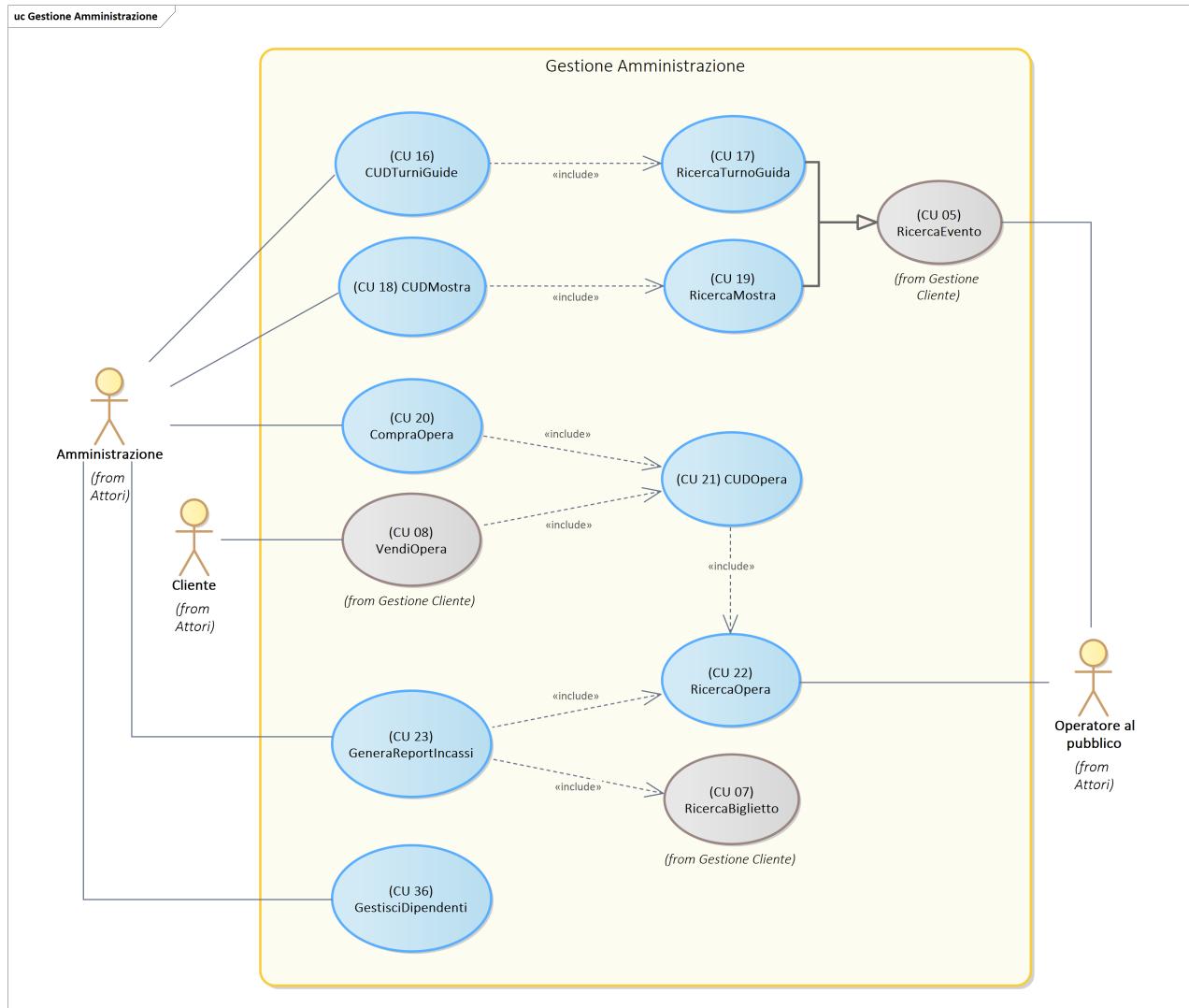


Figura 8: diagramma dei casi d'uso: **Gestione Amministrazione**

CUDTurniGuide (CU 16)

Questo caso d'uso consente l'inserimento, la modifica e la rimozione dei turni relativi alle guide che effettueranno i tour della mostra.

Attori primari: Amministrazione.

Attori secondari: Nessuno.

Precondizioni:

- Il sistema ha autenticato l'attore primario.

Postcondizioni:

- Il sistema ha memorizzato tutte le modifiche nel database dei turni guide.

Sequenza eventi principale:

- Il caso d'uso inizia quando l'attore primario vuole effettuare un'operazione CUD relativa ai turni delle guide.
- if** l'attore primario vuole inserire un nuovo turno:
 - L'attore inserisce l'ora del nuovo turno e il nome dell'operatore al pubblico.
 - include(RicercaTurnoGuida)**
 - Il sistema provvede ad aggiungere il nuovo turno.
- else if** l'attore primario vuole modificare un turno già esistente:
 - L'attore primario inserisce l'ora del turno da modificare e il nome dell'operatore al pubblico.
 - include(RicercaTurnoGuida)**
 - L'attore primario inserisce l'ora dove spostare il turno.
 - include(RicercaTurnoGuida)**
 - Il sistema provvede a modificare l'orario del turno.
- else if** l'attore primario vuole eliminare un turno:
 - L'attore primario inserisce l'ora del turno da eliminare e il nome dell'operatore al pubblico.
 - include(RicercaTurnoGuida)**
 - Il sistema mostra un messaggio di conferma.
 - if** l'attore primario conferma l'operazione:
 - Il sistema provvede ad eliminare il turno.

Sequenza eventi alternativa: TurnoGiàEsistente

- La sequenza di eventi alternativa inizia dopo il passo 2.2. **or** 3.4. della sequenza di eventi principale.
- Il sistema mostra un opportuno messaggio di errore informando che per l'operatore al pubblico indicato è già programmato un turno nell'orario scelto.

Sequenza eventi alternativa: TurnoNonTrovato

- La sequenza di eventi alternativa inizia dopo il passo 3.2. **or** 4.2. della sequenza di eventi principale.
- Il sistema mostra un opportuno messaggio di errore informando che non è stato trovato un turno relativo ai parametri di ricerca.

RicercaTurnoGuida (CU 17)

Questo caso d'uso consente di ricercare un turno di una guida.

ID padre: (CU 05).

Attori primari: Amministrazione, Operatore al pubblico.

Attori secondari: Nessuno.

Precondizioni: Nessuna.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. (o 1.) Il caso d'uso inizia quando l'attore primario vuole effettuare la ricerca di un turno di una guida **or** il sistema deve effettuare la ricerca di un turno di una guida.
2. (2.) **if** il caso d'uso è stato avviato dall'attore:
 - 2.1. (2.1.) Il sistema chiede i criteri di ricerca.
 - 2.2. (o 2.2.) **if** l'attore non inserisce la data **nor** il nome della guida:
 - 2.2.1. Il sistema usa la data attuale come parametro di ricerca.
3. (3.) Il sistema registra i parametri di ricerca.
4. (o 4.) **for each** turno nel database dei turni delle guide:
 - 4.1. (o 4.1.) Il sistema controlla se il turno soddisfa i criteri di ricerca.
 - 4.2. (o 4.2.) **if** il turno soddisfa il criterio:
 - 4.2.1. (o 4.2.1.) Il sistema aggiunge il turno alla lista dei risultati.
5. (o 5.) **if** la lista dei risultati contiene almeno un turno:
 - 5.1. (5.1.) Il sistema mostra la lista dei risultati con tutte le loro informazioni.
6. (6.) **else:**
 - 6.1. (o 6.1.) Il sistema comunica all'attore che nessun turno di alcuna guida soddisfa il criterio specificato.

Sequenza eventi alternativa: Nessuna.

CUDMostra (CU 18)

Questo caso d'uso consente l'inserimento, la modifica e la rimozione di una mostra.

Attori primari: Amministrazione.

Attori secondari: Nessuno.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.

Postcondizioni:

1. Il sistema ha memorizzato tutte le modifiche nel database delle mostre.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario vuole effettuare un'operazione CUD relativa alla mostra.
2. **if** l'attore primario vuole inserire una nuova mostra:
 - 2.1. Il sistema chiede all'attore primario la data nella quale egli desideri organizzare una mostra.
 - 2.2. **include(RicercaMostra)**
 - 2.3. **for each** opera nel database:
 - 2.3.1. Il sistema aggiunge l'opera alla schermata.
 - 2.4. Il sistema chiede all'attore di selezionare le opere da aggiungere alla mostra.
 - 2.5. Il sistema provvede a creare la nuova mostra.
3. **else if** l'attore vuole modificare una mostra esistente:
 - 3.1. Il sistema chiede all'attore primario la data nella quale egli desideri modificare la mostra.
 - 3.2. **include(RicercaMostra)**
 - 3.3. **for each** opera nella mostra **and for each** opera nel database, ma non nella mostra:
 - 3.3.1. Il sistema aggiunge l'opera alla schermata.
 - 3.4. L'attore rimuove e aggiunge le opere alla mostra.
 - 3.5. Il sistema provvede a salvare le modifiche.
4. **else if** l'attore primario vuole eliminare una mostra esistente:
 - 4.1. Il sistema chiede all'attore primario la data nella quale egli desideri rimuovere la mostra.
 - 4.2. **include(RicercaMostra)**
 - 4.3. Il sistema mostra un messaggio di conferma.
 - 4.4. **if** l'attore primario conferma l'operazione:
 - 4.4.1. Il sistema provvede ad eliminare la mostra.

Sequenza eventi alternativa: MostraGiàEsistente

1. La sequenza di eventi alternativa inizia dopo il passo 2.2. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore informando che è già programmata una mostra nella data specificata.

Sequenza eventi alternativa: MostraNonTrovata

1. La sequenza di eventi alternativa inizia dopo il passo 3.2. **or** 4.2. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore informando che non è stata trovata alcuna mostra nella data specificata.

RicercaMostra (CU 19)

Questo caso d'uso consente di ricercare una mostra.

ID padre: (CU 05).

Attori primari: Amministrazione, Operatore al pubblico.

Attori secondari: Nessuno.

Precondizioni: Nessuna.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. (o 1.) Il caso d'uso inizia quando l'attore primario vuole effettuare la ricerca di una mostra **or** il sistema deve effettuare la ricerca di una mostra.
2. (2.) **if** il caso d'uso è stato avviato dall'attore:
 - 2.1. (2.1.) Il sistema chiede i criteri di ricerca.
 - 2.2. (o 2.2.) **if** l'attore non inserisce la data **nor** il nome della mostra:
 - 2.2.1. Il sistema usa la data attuale come parametro di ricerca.
3. (3.) Il sistema registra i parametri di ricerca.
4. (o 4.) **for each** mostra nel database delle mostre:
 - 4.1. (o 4.1.) Il sistema controlla se la mostra soddisfa i criteri di ricerca.
 - 4.2. (o 4.2.) **if** la mostra soddisfa il criterio:
 - 4.2.1. (o 4.2.1.) Il sistema aggiunge la mostra alla lista dei risultati.
5. (o 5.) **if** la lista dei risultati contiene almeno una mostra:
 - 5.1. (5.1.) Il sistema mostra la lista dei risultati con tutte le loro informazioni.
6. (6.) **else:**
 - 6.1. (o 6.1.) Il sistema comunica all'attore che nessuna mostra soddisfa il criterio specificato.

Sequenza eventi alternativa: Nessuna.

CompraOpera (CU 20)

Questo caso d'uso permette di comprare opere da enti esterni, di gestire le relative transazioni di denaro, e di aggiungerle al database delle opere.

Attori primari: Anmministrazione.

Attori secondari: Nessuno.

Precondizioni:

- Il sistema ha autenticato l'attore primario.

Postcondizioni:

- Il sistema ha memorizzato la nuova opera nel database insieme a tutte le sue informazioni.
- Il sistema ha memorizzato la transazione effettuata per l'acquisto.

Sequenza eventi principale:

- Il caso d'uso inizia quando l'attore primario desidera comprare una nuova opera.
- if** l'attore primario ha già concluso un affare con un ente esterno privatamente:
 - Il sistema chiede all'attore primario di inserire tutti i dettagli dell'opera acquistata.
 - Il sistema chiede anche le informazioni sul costo della transazione.
- else**
 - Il sistema si collega al sito web per la compravendita di opere tra enti privati.
 - for each** opera disponibile per l'acquisto:
 - Il sistema preleva tutte le sue informazioni e aggiunge l'opera alla schermata.
 - l'attore primario seleziona un'opera alle quale è interessato.
- Il sistema mostra un messaggio di conferma.
- if** l'attore primario conferma l'operazione:
 - Il sistema provvede a completare l'acquisto memorizzando la transazione.
 - include(CUDOpera)**

Sequenza eventi alternativa: *OperazioneAnnullata*.

- La sequenza di eventi alternativa inizia in qualunque momento.
- L'attore primario annulla l'operazione di acquisto.

Sequenza eventi alternativa: *ConnessioneNonRiuscita*.

- La sequenza di eventi alternativa inizia dopo il passo 3.1. della sequenza di eventi principale.
- Il sistema comunica all'attore primario, in un opportuno messaggio, l'indisponibilità del sito web per la compravendita di opere e lo esorta ad attendere prima di ripetere l'operazione.

CUDOpera (CU 21)

Questo caso d'uso consente l'inserimento, la modifica e la rimozione di un'opera.

Attori primari: Amministrazione.

Attori secondari: Nessuno.

Precondizioni:

- Il sistema ha autenticato l'attore primario.

Postcondizioni:

- Il sistema ha memorizzato tutte le modifiche nel database delle opere.

Sequenza eventi principale:

- Il caso d'uso inizia quando l'attore primario vuole effettuare un'operazione CUD relativa ad un'opera.
- if** l'attore primario vuole inserire una nuova opera:
 - L'attore primario inserisce i dati relativi all'opera.
 - include(RicercaOpera)**
 - Il sistema provvede ad aggiungere la nuova opera.
- else if** l'attore vuole modificare un'opera già esistente:
 - L'attore primario inserisce i dati dell'opera da ricercare.
 - include(RicercaOpera)**
 - L'attore aggiorna i dati dell'opera.
 - Il sistema provvede a salvare le modifiche.
- else if** l'attore primario vuole eliminare un'opera:
 - L'attore primario inserisce i dati dell'opera da eliminare.
 - include(RicercaOpera)**
 - Il sistema mostra un messaggio di conferma.
 - if** l'attore primario conferma l'operazione:
 - Il sistema provvede ad eliminare l'opera.

Sequenza eventi alternativa: *OperaGiàEsistente*

- La sequenza di eventi alternativa inizia dopo il passo 2.2. della sequenza di eventi principale.
- Il sistema mostra un opportuno messaggio di errore informando che l'opera è già presente nel database.

Sequenza eventi alternativa: *OperaNonTrovata*

- La sequenza di eventi alternativa inizia dopo il passo 3.2. **or** 4.2. della sequenza di eventi principale.
- Il sistema mostra un opportuno messaggio di errore informando l'attore primario che non è stata trovata alcuna opera conforme ai parametri di ricerca inseriti.

RicercaOpera (CU 22)

Questo caso d'uso permette di ricercare opere all'interno del database delle opere.

Attori primari: Amministrazione, Operatore al pubblico.

Attori secondari: Nessuno.

Precondizioni: Nessuna.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando uno degli attori primari vuole effettuare la ricerca di un'opera **or** il sistema deve ricercare un'opera.
2. **if** il caso d'uso è stato avviato da uno degli attori primari:
 - 2.1. Il sistema chiede i criteri di ricerca.
 - 2.2. L'attore inserisce i criteri di ricerca richiesti.
3. Il sistema preleva i parametri di ricerca.
4. **for each** opera nel database:
 - 4.1. Il sistema confronta i parametri di ricerca con i parametri dell'opera.
 - 4.2. **if** l'opera rispetta i parametri richiesti:
 - 4.2.1. Il sistema aggiunge l'opera alla lista del risultato di ricerca.
5. **if** il sistema ha trovato almeno un'opera corrispondente ai parametri di ricerca:
 - 5.1. Il sistema fornisce tutti i dati relativi alle opere trovate.
6. **else**
 - 6.1. Il sistema informa di non aver trovato nessuna opera corrispondente ai parametri ricercati.

Sequenza eventi alternativa: Nessuna.

GeneraReportIncassi (CU 23)

Questo caso d'uso consente di generare il report degli incassi su tutte le transazioni effettuate da inizio mese fino al momento della richiesta oppure in mesi precedenti.

Attori primari: Anmministrazione.

Attori secondari: Nessuno.

Precondizioni:

- Il sistema ha autenticato l'attore primario.

Postcondizioni: Nessuna.

Sequenza eventi principale:

- Il caso d'uso inizia quando l'attore primario desidera generare il report degli incassi.
- Il sistema chiede all'attore primario il mese nel quale desidera generare il report degli incassi.
- if** l'attore primario inserisce un mese antecedente il mese corrente:
 - Il sistema utilizza il mese inserito come criterio di ricerca delle transazioni.
- else if** l'attore primario non inserisce alcuna data:
 - Il sistema utilizza il mese corrente come criterio di ricerca delle transazioni.
- for each** biglietto nel database dei biglietti:
 - include(RicercaBiglietto)**
 - if** il biglietto è stato venduto nel mese selezionato:
 - Il sistema aggiunge la transazione alla lista delle transizioni.
- for each** abbonamento nel database degli abbonamenti:
 - include(RicercaAbbonamento)**
 - if** l'abbonamento è stato venduto **or** è stato rinnovato nel mese selezionato :
 - Il sistema aggiunge la transazione alla lista delle transizioni.
- for each** opera nel database delle opere:
 - include(RicercaOpera)**
 - if** l'opera è stata venduta **or** l'opera è stata acquistata nel mese selezionato :
 - Il sistema aggiunge la transazione alla lista delle transizioni.
- Il sistema emette il report con tutte le transazioni recuperate e calcola il totale degli incassi come differenza tra le entrate e le uscite.

Sequenza eventi alternativa: *Report Vuoto*

- La sequenza di eventi alternativa inizia dopo il passo 8. della sequenza di eventi principale.
- Il sistema mostra un opportuno messaggio di errore informando l'attore primario dell'assenza di transazione nel mese selezionato.

GestisciDipendenti (CU 36)

Questo caso d'uso consente di gestire i dipendenti che lavorano all'interno del museo.

Attori primari: Amministrazione.

Attori secondari: Nessuno.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.

Postcondizioni:

1. Il sistema ha modificato tutte le modifiche effettuate.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario desidera gestire i dipendenti del museo.
2. Il sistema chiede all'attore primario il reparto da gestire.
3. **if** l'attore primario vuole gestire la reception:
 - 3.1. Il sistema mostra all'attore primario una lista degli operatori al pubblico dipendenti del museo.
 - 3.2. L'attore apporta le modifiche necessarie al personale degli operatori al pubblico.
4. **else if** l'attore primario vuole gestire la segreteria:
 - 4.1. Il sistema mostra all'attore primario una lista dei segretari dipendenti del museo.
 - 4.2. L'attore apporta le modifiche necessarie al personale dei segretari.
5. **else if** l'attore primario vuole gestire l'amministrazione:
 - 5.1. Il sistema mostra all'attore primario una lista degli amministratori dipendenti del museo.
 - 5.2. L'attore apporta le modifiche necessarie al personale degli amministratori.
6. Il sistema memorizza tutte le modifiche effettuate al personale.

3.6 Gestione Sistema

Il diagramma dei casi d'uso **Gestione Sistema** racchiude i casi d'uso istanziabili direttamente solo dal sistema stesso, che permettono di eseguire azioni di **scheduling**, quali effettuare backup locale e su cloud e avvisare i possessori di abbonamenti prossimi alla scadenza. Nel dettaglio, i casi d'uso di pertinenza sono:

- **ControllaAbbonamenti** (CU 24)
- **InviaAvviso** (CU 25)
- **VerificaAbbonamento** (CU 26)
- **EffettuaBackup** (CU 27)

Gli attori coinvolti in questi casi d'uso sono l'attore *Tempo*, capace di eseguire lo **scheduling** delle azioni e l'attore *Amministrazione* che ha la facoltà di eseguire un backup anche fuori dall'orario prestabilito dallo scheduler.

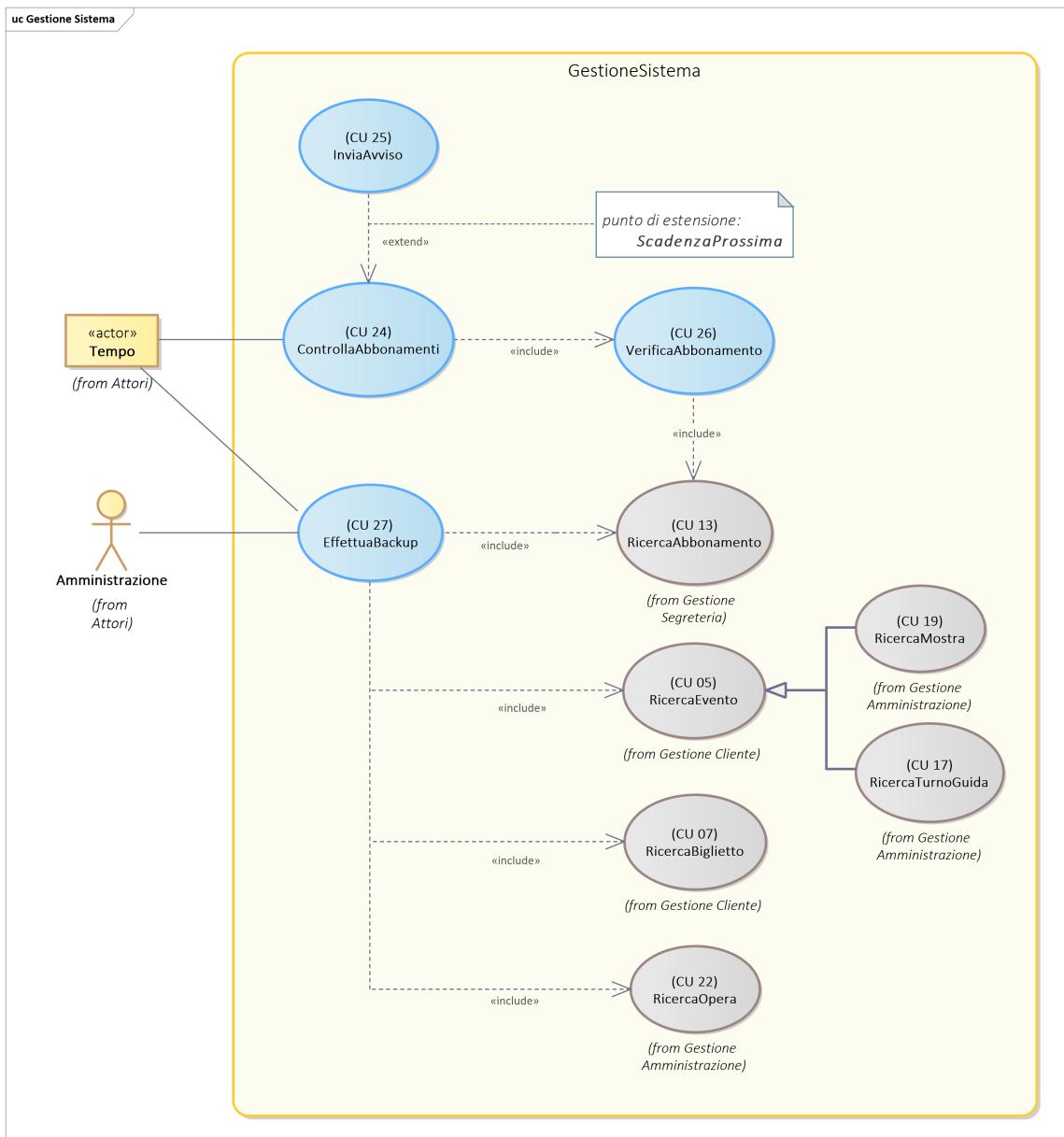


Figura 9: diagramma dei casi d'uso: **Gestione Sistema**

ContollaAbbonamenti (CU 24)

Questo caso d'uso consente di controllare gli abbonamenti attivi e segnalare con un avviso i possessori di abbonamenti con scadenza imminente.

Attori primari: Tempo.

Attori secondari: Nessuno.

Precondizioni: Nessuna.

Postcondizioni:

1. Gli abbonamenti sono stati controllati.

Sequenza eventi principale:

1. Il caso d'uso inizia ogni giorno alle ore 23:59 secondo il fuso orario locale.

2. **for each** abbonamento nel database degli abbonamenti:

- 2.1. **include(VerificaAbbonamento)**
- 2.2. **extention point: scadenzaProssima**

- 2.3. Il sistema aggiorna la data di ultimo controllo dell'abbonamento con la data attuale.

Sequenza eventi alternativa: Nessuna.

InviaAvviso (CU 25)

Segmento 1: Questo caso d'uso permette al sistema di inviare un avviso ai clienti possessori di un abbonamento in fase di scadenza.

Attori primari: Tempo.

Attori secondari: Nessuno.

Precondizioni del segmento 1:

1. La data di scadenza dell'abbonamento antecede la data corrente di meno di 5 giorni.

Postcondizioni del segmento 1:

1. Il sistema ha inviato una comunicazione via posta elettronica al possessore dell'abbonamento.

Sequenza eventi principale del segmento 1:

1. Il caso d'uso inizia quando il sistema deve mandare un messaggio di avviso per rinnovare un abbonamento.
2. Il sistema preleva l'indirizzo di posta elettronica dall'abbonamento.
3. Il sistema genera un messaggio di avviso di rinnovo dell'abbonamento personalizzato e lo invia all'indirizzo di posta elettronica prelevato.

Sequenza eventi alternativa del segmento 1: Nessuna.

VerificaAbbonamento (CU 26)

Questo caso d'uso permette di controllare se un abbonamento è valido, scaduto o in fase di scadenza.

Attori primari: Segreteria, Operatore al Pubblico, Tempo.

Attori secondari: Nessuno.

Precondizioni:

1. Il sistema ha ricevuto l'identificativo dell'abbonamento da controllare.

Postcondizioni:

1. Il sistema ha verificato lo stato dell'abbonamento.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario vuole verificare lo stato di un abbonamento **or** il sistema deve verificare lo stato di un abbonamento.
2. **include(RicercaAbbonamento)**
3. Il sistema legge la data di scadenza dell'abbonamento e la confronta con la data corrente.
4. **if** la data di scadenza dell'abbonamento precede la data corrente:
 - 4.1. Il sistema informa l'attore primario che l'abbonamento è scaduto.
5. **else if** la data di scadenza dell'abbonamento antecede la data corrente di meno di 5 giorni:
 - 5.1. Il sistema informa l'attore primario che l'abbonamento è in fase di scadenza.

Sequenza eventi alternativa: *AbbonamentoNonTrovato*.

1. La sequenza di eventi alternativa inizia dopo il passo 2. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore comunicando all'attore primario di non aver trovato alcun abbonamento avente lo stesso codice identificativo inserito, e lo esorta a ripetere il processo di scannerizzazione del codice QR sul fronte dell'abbonamento.

Sequenza eventi alternativa: *OperazioneAnnullata*.

1. La sequenza di eventi alternativa inizia in qualunque momento.
2. L'attore primario annulla l'operazione verifica dell'abbonamento.

EffettuaBackup (CU 27)

Questo caso d'uso consente di effettuare il backup dei dati relativi al sistema informativo di riferimento.

Attori primari: Amministrazione, Tempo.

Attori secondari: Amministrazione.

Precondizioni:

1. Se il caso d'uso è stato avviato dall'attore primario *Amministrazione*, Il sistema lo ha autenticato.

Postcondizioni: I dati sono stati copiati.

Sequenza eventi principale:

1. Il caso d'uso inizia ogni sera alle 23:59 secondo il fuso orario locale.
2. **for each** biglietto nel database dei biglietti:
 - 2.1. **include(RicercaBiglietto).**
 - 2.2. Il sistema preleva tutte le informazioni riguardanti quel biglietto.
 - 2.3. Il sistema copia sul disco i dati del biglietto.
3. **for each** abbonamento nel database degli abbonamenti:
 - 3.1. **include(RicercaAbbonamento).**
 - 3.2. Il sistema preleva tutte le informazioni riguardanti quell'abbonamento.
 - 3.3. Il sistema copia sul disco i dati dell'abbonamento.
4. **for each** opera nel database delle opere:
 - 4.1. **include(RicercaOpera).**
 - 4.2. Il sistema preleva tutte le informazioni riguardanti quell'opera.
 - 4.3. Il sistema copia sul disco i dati dell'opera.
5. **for each** evento nel database degli eventi:
 - 5.1. **include(RicercaEvento).**
 - 5.2. Il sistema preleva tutte le informazioni riguardanti quell'evento.
 - 5.3. Il sistema copia sul disco i dati dell'evento.
6. **for each** dato sul visitatore nel database dei dati sui visitatori:
 - 6.1. Il sistema preleva tutte le informazioni riguardanti quel visitatore.
 - 6.2. Il sistema copia sul disco i dati del visitatore.
7. Il sistema organizza in una cartella adeguata i dati copiati.
8. Il sistema effettua l'upload sul cloud utilizzato dal museo del backup appena effettuato.

Sequenza eventi alternativa: *DiscoPieno.*

1. La sequenza di eventi alternativa inizia dopo il passo 6. della sequenza di eventi principale.
2. Il sistema invia un avviso all'attore secondario comunicante l'impossibilità di effettuare il backup sul disco locale per via della mancanza di spazio sufficiente, e lo esorta a risolvere il problema.

Sequenza eventi alternativa: *UploadNonRiuscito.*

1. La sequenza di eventi alternativa inizia dopo il passo 7. della sequenza di eventi principale.
2. Il sistema notifica all'attore secondario l'impossibilità di effettuare l'upload del backup sul cloud e visualizza l'errore.

3.7 Gestione Statistiche

Il diagramma dei casi d'uso **Gestione Statistiche** racchiude i casi d'uso necessari per la raccolta dei dati sui clienti in forma anonima e la generazione delle statistiche in base alla provenienza, all'età e al sesso del visitatore. Nel dettaglio, i casi d'uso di pertinenza sono:

- **InserisciDatiCliente** (CU 28)
- **InserisciProvenienza** (CU 29)
- **InserisciEtà** (CU 30)
- **InserisciSesso** (CU 31)
- **VisualizzaStatistiche** (CU 32)
- **VisualizzaPerProvenienza** (CU 33)
- **VisualizzaPerEtà** (CU 34)
- **VisualizzaPerSesso** (CU 35)

Gli attori coinvolti in questi casi d'uso sono l'*Operatore al pubblico* per la popolazione dei dati sui quali inferire le statistiche e l'*Amministrazione* per la visualizzazione delle stesse.

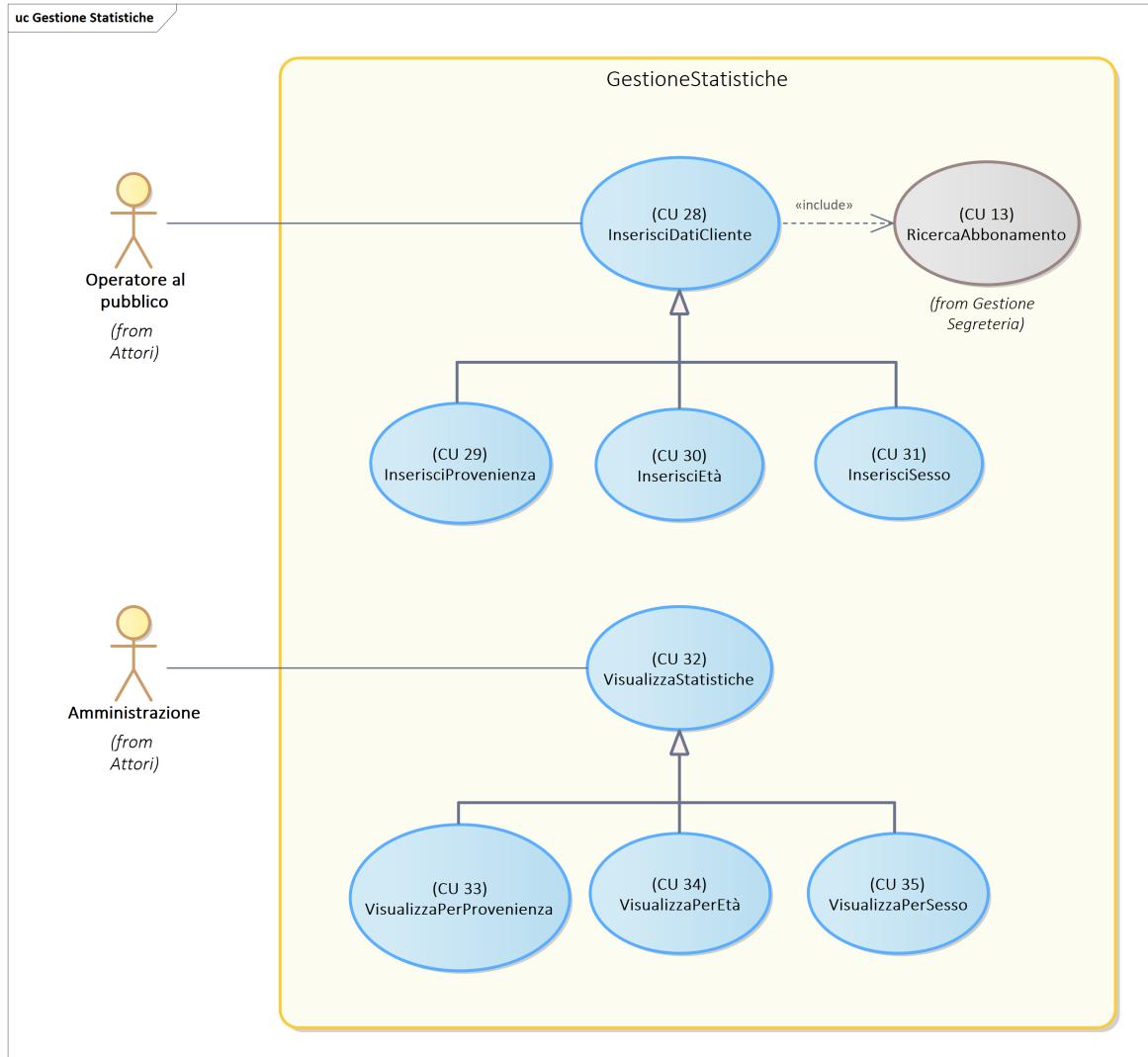


Figura 10: diagramma dei casi d'uso: **Gestione Statistiche**

InserisciDatiCliente (CU 28)

Questo caso d'uso consente di inserire i dati del visitatore al momento dell'acquisto del biglietto.

Attori primari: Operatore al pubblico.

Attori secondari: Visitatore.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.
2. Nel caso in cui il visitatore disponga di un abbonamento, l'attore primario ha inserito nel sistema l'identificativo dell'abbonamento eseguendo una scannerizzazione del codice QR.

Postcondizioni:

1. Il sistema ha memorizzato i dati inseriti in forma anonima.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario vuole memorizzare i dati di un visitatore.
2. **if** il visitatore dispone di un abbonamento:
 - 2.1. Il sistema recupera i dati necessari dal suo abbonamento.
 - 2.2. **include(RicercaAbbonamento)**
3. **else:**
 - 3.1. L'attore primario chiede al visitatore i dati necessari.
 - 3.2. L'attore primario inserisce i dati nel sistema.
4. Il sistema memorizza i dati.
5. Il sistema produce un messaggio con l'esito dell'operazione.

Sequenza eventi alternativa: *DatiNonFornitiDalCliente*

1. La sequenza di eventi alternativa inizia dopo il passo 3.1. della sequenza di eventi principale.
2. L'attore secondario si rifiuta di fornire le informazioni.
3. L'attore primario annulla l'operazione.
4. Il sistema termina l'operazione senza memorizzare alcun dato riguardante il visitatore.

InserisciProvenienza (CU 29)

Questo caso d'uso consente di inserire lo Stato di nascita del visitatore.

Id padre: (CU 28)

Attori primari: Operatore al pubblico.

Attori secondari: Visitatore.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.
2. Nel caso in cui il visitatore disponga di un abbonamento, l'attore primario ha inserito nel sistema l'identificativo dell'abbonamento eseguendo una scannerizzazione del codice QR.

Postcondizioni:

1. Il sistema ha memorizzato i dati inseriti in forma anonima.

Sequenza eventi principale:

1. (o1) Il caso d'uso inizia quando l'attore primario vuole memorizzare il Paese d'origine del visitatore.
2. (2.) **if** il visitatore dispone di un abbonamento:
 - 2.1. (2.1.) Il sistema recupera i dati necessari dal suo abbonamento.
 - 2.2. (2.2.) **include(RicercaAbbonamento)**
3. (3.) **else:**
 - 3.1. (o3.1) L'attore primario chiede al visitatore il suo Paese d'origine.
 - 3.2. (o3.2.) L'attore primario inserisce il Paese d'origine del visitatore nel sistema.
4. (4.) Il sistema memorizza i dati.
5. (5.) Il sistema produce un messaggio con l'esito dell'operazione.

Sequenza eventi alternativa: *DatiNonFornitiDalCliente*

1. (1.) La sequenza di eventi alternativa inizia dopo il passo 3.1. della sequenza di eventi principale.
2. (o2.) L'attore secondario si rifiuta di fornire l'informazione del suo Paese di origine.
3. (3.) L'attore primario annulla l'operazione.
4. (4.) Il sistema termina l'operazione senza memorizzare alcun dato riguardante il visitatore.

InserisciEtà (CU 30)

Questo caso d'uso consente di inserire l'età del visitatore.

Id padre: (CU 28)

Attori primari: Operatore al pubblico.

Attori secondari: visitatore.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.
2. Nel caso in cui il visitatore disponga di un abbonamento, l'attore primario ha inserito nel sistema l'identificativo dell'abbonamento eseguendo una scannerizzazione del codice QR.

Postcondizioni:

1. Il sistema ha memorizzato i dati inseriti in forma anonima.

Sequenza eventi principale:

1. (o1) Il caso d'uso inizia quando l'attore primario vuole memorizzare la data di nascita del visitatore.
2. (2.) **if** il visitatore dispone di un abbonamento:
 - 2.1. (2.1.) Il sistema recupera i dati necessari dal suo abbonamento.
 - 2.2. (2.2.) **include(RicercaAbbonamento)**
3. (3.) **else:**
 - 3.1. (o3.1) L'attore primario chiede al visitatore la sua data di nascita.
 - 3.2. (o3.2.) L'attore primario inserisce la data di nascita del visitatore nel sistema.
4. (4.) Il sistema memorizza i dati.
5. (5.) Il sistema produce un messaggio con l'esito dell'operazione.

Sequenza eventi alternativa: *DatiNonFornitiDalCliente*

1. (1.) La sequenza di eventi alternativa inizia dopo il passo 3.1. della sequenza di eventi principale.
2. (o2.) L'attore secondario si rifiuta di fornire l'informazione della sua data di nascita.
3. (3.) L'attore primario annulla l'operazione.
4. (4.) Il sistema termina l'operazione senza memorizzare alcun dato riguardante il visitatore.

InserisciSesso (CU 31)

Questo caso d'uso consente di inserire il genere del visitatore.

Id padre: (CU 28)

Attori primari: Operatore al pubblico.

Attori secondari: Visitatore.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.
2. Nel caso in cui il visitatore disponga di un abbonamento, l'attore primario ha inserito nel sistema l'identificativo dell'abbonamento eseguendo una scannerizzazione del codice QR.

Postcondizioni:

1. Il sistema ha memorizzato i dati inseriti in forma anonima.

Sequenza eventi principale:

1. (o1) Il caso d'uso inizia quando l'attore primario vuole memorizzare il genere del visitatore.
2. (2.) **if** il visitatore dispone di un abbonamento:
 - 2.1. (2.1.) Il sistema recupera i dati necessari dal suo abbonamento.
 - 2.2. (2.2.) **include(RicercaAbbonamento)**
3. (3.) **else:**
 - 3.1. (o3.1) L'attore primario chiede al visitatore il suo genere.
 - 3.2. (o3.2.) L'attore primario inserisce il genere del visitatore nel sistema.
4. (4.) Il sistema memorizza i dati.
5. (5.) Il sistema produce un messaggio con l'esito dell'operazione.

Sequenza eventi alternativa: *DatiNonFornitiDalCliente*

1. (1.) La sequenza di eventi alternativa inizia dopo il passo 3.1. della sequenza di eventi principale.
2. (o2.) L'attore secondario si rifiuta di fornire l'informazione del suo genere.
3. (3.) L'attore primario annulla l'operazione.
4. (4.) Il sistema termina l'operazione senza memorizzare alcun dato riguardante il visitatore.

VisualizzaStatistiche (CU 32)

Questo caso d'uso consente di generare statistiche in un particolare mese sulla base dei dati sui clienti raccolti in forma anonima dal sistema.

Attori primari: Amministrazione.

Attori secondari: Nessuno.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. Il caso d'uso inizia quando l'attore primario vuole visualizzare le statistiche.
2. Il sistema chiede all'attore primario il mese nel quale desidera generare le statistiche.
3. **if** l'attore primario inserisce un mese antecedente il mese corrente:
 - 3.1. Il sistema utilizza il mese inserito come criterio di ricerca.
4. **else if** l'attore primario non inserisce alcuna data:
 - 4.1. Il sistema utilizza il mese corrente come criterio di ricerca.
5. **for each** dato sul visitatore nel database dei dati sui clienti:
 - 5.1. **if** il dato sul visitatore è stato registrato nel sistema nel mese selezionato:
 - 5.1.1. Il sistema aggiunge il dato sul visitatore alla lista dei dati raccolti.
6. Il sistema organizza i dati sulla base dei quali genera un grafico e lo mostra a schermo.
7. Il sistema calcola deviazione standard campionaria sulla base dei dati raccolti e mostra a schermo i risultati.
8. Il sistema mostra a schermo anche il numero di visite avvenute dall'inizio del mese, fino al momento della richiesta.

Sequenza eventi alternativa: *NessunDato*

1. La sequenza di eventi alternativa inizia dopo il passo 6. della sequenza di eventi principale.
2. Il sistema mostra un opportuno messaggio di errore informando l'attore primario dell'assenza di dati sui clienti nel mese selezionato.

VisualizzaPerProvenienza (CU 33)

Questo caso d'uso consente di eseguire la ricerca di un abbonamento all'interno del database.

Id padre: (CU 32)

Attori primari: Amministrazione.

Attori secondari: Nessuno.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. (1.) Il caso d'uso inizia quando l'attore primario vuole visualizzare le statistiche.
2. (2.) Il sistema chiede all'attore primario il mese nel quale desidera generare le statistiche.
3. (3.) **if** l'attore primario inserisce un mese antecedente il mese corrente:
 - 3.1. (3.1.) Il sistema utilizza il mese inserito come criterio di ricerca.
4. (4.) **else if** l'attore primario non inserisce alcuna data:
 - 4.1. (4.1.) Il sistema utilizza il mese corrente come criterio di ricerca.
5. (5.) **for each** dato sul visitatore nel database dei dati sui clienti:
 - 5.1. (5.1.) **if** il dato sul visitatore è stato registrato nel sistema nel mese selezionato:
 - 5.1.1. (5.1.1.) Il sistema aggiunge il Paese di origine del visitatore alla lista dei dati raccolti.
6. (6.) Il sistema organizza i dati sulla base dei quali genera un grafico a torta e lo mostra a schermo.
7. (7.) Il sistema calcola la deviazione standard campionaria sulla base dei dati raccolti e mostra a schermo i risultati.
8. (8.) Il sistema mostra a schermo anche il numero di visite avvenute dall'inizio del mese, fino al momento della richiesta.

Sequenza eventi alternativa: *NessunDato*

1. (1.) La sequenza di eventi alternativa inizia dopo il passo 6. della sequenza di eventi principale.
2. (2.) Il sistema mostra un opportuno messaggio di errore informando l'attore primario dell'assenza di dati sui clienti nel mese selezionato.

VisualizzaPerEtà (CU 34)

Questo caso d'uso consente di eseguire la ricerca di un abbonamento all'interno del database.

Id padre: (CU 32)

Attori primari: Amministrazione.

Attori secondari: Nessuno.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. (1.) Il caso d'uso inizia quando l'attore primario vuole visualizzare le statistiche.
2. (2.) Il sistema chiede all'attore primario il mese nel quale desidera generare le statistiche.
3. (3.) **if** l'attore primario inserisce un mese antecedente il mese corrente:
 - 3.1. (3.1.) Il sistema utilizza il mese inserito come criterio di ricerca.
4. (4.) **else if** l'attore primario non inserisce alcuna data:
 - 4.1. (4.1.) Il sistema utilizza il mese corrente come criterio di ricerca.
5. (5.) **for each** dato sul visitatore nel database dei dati sui clienti:
 - 5.1. (5.1.) **if** il dato sul visitatore è stato registrato nel sistema nel mese selezionato:
 - 5.1.1. (5.1.1) Il sistema raccoglie la data di nascita del visitatore e la utilizza per calcolare l'età.
 - 5.1.2. (5.1.2.) Il sistema aggiunge l'età del visitatore alla lista dei dati raccolti.
6. (6.) Il sistema organizza i dati sulla base dei quali genera un grafico istogramma e lo mostra a schermo.
7. (7.) Il sistema calcola la media campionaria e la deviazione standard campionaria sulla base dei dati raccolti e mostra a schermo i risultati.
8. (8.) Il sistema mostra a schermo anche il numero di visite avvenute dall'inizio del mese, fino al momento della richiesta.

Sequenza eventi alternativa: *NessunDato*

1. (1.) La sequenza di eventi alternativa inizia dopo il passo 6. della sequenza di eventi principale.
2. (2.) Il sistema mostra un opportuno messaggio di errore informando l'attore primario dell'assenza di dati sui clienti nel mese selezionato.

VisualizzaPerSesso (CU 35)

Questo caso d'uso consente di eseguire la ricerca di un abbonamento all'interno del database.

Id padre: (CU 32)

Attori primari: Amministrazione.

Attori secondari: Nessuno.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.

Postcondizioni: Nessuna.

Sequenza eventi principale:

1. (1.) Il caso d'uso inizia quando l'attore primario vuole visualizzare le statistiche.
2. (2.) Il sistema chiede all'attore primario il mese nel quale desidera generare le statistiche.
3. (3.) **if** l'attore primario inserisce un mese antecedente il mese corrente:
 - 3.1. (3.1.) Il sistema utilizza il mese inserito come criterio di ricerca.
4. (4.) **else if** l'attore primario non inserisce alcuna data:
 - 4.1. (4.1.) Il sistema utilizza il mese corrente come criterio di ricerca.
5. (5.) **for each** dato sul visitatore nel database dei dati sui clienti:
 - 5.1. (5.1.) **if** il dato sul visitatore è stato registrato nel sistema nel mese selezionato:
 - 5.1.1. (5.1.1.) Il sistema aggiunge il genere del visitatore alla lista dei dati raccolti.
6. (6.) Il sistema organizza i dati sulla base dei quali genera un grafico a torta e lo mostra a schermo.
7. (7.) Il sistema calcola la deviazione standard campionaria sulla base dei dati raccolti e mostra a schermo i risultati.
8. (8.) Il sistema mostra a schermo anche il numero di visite avvenute dall'inizio del mese, fino al momento della richiesta.

Sequenza eventi alternativa: *NessunDato*

1. (1.) La sequenza di eventi alternativa inizia dopo il passo 6. della sequenza di eventi principale.
2. (2.) Il sistema mostra un opportuno messaggio di errore informando l'attore primario dell'assenza di dati sui clienti nel mese selezionato.

4 Matrice di mapping

La **matrice di mapping** è un ottimo strumento per verificare che tutti i requisiti proposti dagli *stakeholders* sono stati **soddisfatti dai casi d'uso** proposti. La relazione che si viene a creare non è però una biezione, può capitare per alcuni requisiti di essere coperti da più casi d'uso.

4.1 Matrice di mapping

Segue la matrice di mapping completa, dove si possono osservare i **casi d'uso sulle righe**, i **requisiti funzionali sulle colonne** e delle frecce in corrispondenza di un'occorrenza di **dipendenza** tra due di essi:

	RF01 - CRUDOpera	RF02 - CRUDMostra	RF03 - CRUDTurniGuide	RF04 - Creabiglietto	RF05 - InserisciDatiCliente	RF06 - CRUDAbbonamento	RF07 - StampaScontrino	RF08 - GestisciDonazione	RF09 - AcquistaOpera	RF10 - VendOpera	RF11 - VisualizzaCatalogoOpere	RF12 - GeneraReportIncassi	RF13 - EffettuaBackup	RF14 - ControllaAbbonamento	RF15 - ControllaBiglietto	RF16 - VisualizzaVisitatoriPerEtà	RF17 - VisualizzaVisitatoriPerProvenienza	RF18 - VisualizzaVisitatoriPerSesso	RF19 - GestioneDipendenti
(CU 01) CompraBiglietto				↑															
(CU 02) GestisciPagamento					↑	↑													
(CU 03) StampaScontrino						↑													
(CU 04) VerificaPrivilegio				↑															
(CU 05) RicercaEvento	↑	↑												↑					
(CU 06) ConvalidaBiglietto																↑			
(CU 07) RicercaBiglietto													↑	↑	↑				
(CU 08) VendOpera	↑									↑									
(CU 09) VisualizzaCatalogo											↑								
(CU 10) DonaOpera								↑											
(CU 11) AbbonaCliente						↑													
(CU 12) CUDAbbonamento						↑													
(CU 13) RicercaAbbonamento						↑													
(CU 14) GeneraQRCode				↑	↑														
(CU 15) GestisciDonazione								↑											
(CU 16) CUDTurniGuide		↑																	
(CU 17) RicercaTurnoGuida			↑																
(CU 18) CUDMostra	↑																		
(CU 19) RicercaMostra	↑																		
(CU 20) CompraOpera	↑								↑										
(CU 21) CUDOpera	↑																		
(CU 22) RicercaOpera	↑																		
(CU 23) GeneraReportIncassi										↑									
(CU 24) ControllaAbboname...														↑					
(CU 25) InviaAvviso														↑					
(CU 26) VerificaAbbonamento														↑					
(CU 27) EffettuaBackup													↑						
(CU 28) InserisciDatiCliente					↑														
(CU 29) InserisciProvenienza					↑														
(CU 30) InserisciEtà					↑														
(CU 31) InserisciSesso					↑														
(CU 32) VisualizzaStatistiche														↑	↑	↑			
(CU 33) VisualizzaPerProvenie...																↑			
(CU 34) VisualizzaPerEtà															↑				
(CU 35) VisualizzaPerSesso																	↑		
(CU 36) GestisciDipendenti																			↑

Figura 11: Matrice di mapping tra requisiti funzionali e casi d'uso.

5 Diagramma delle classi (analisi)

Il **Diagramma delle classi** in analisi offre un primo spunto di modellazione statica della struttura museale a partire da ciò che è stato appreso nella fase di dialogo con gli stakeholders e la raccolta dei requisiti. Il modello è **destinato agli stessi** e **non vuole proporre certo una visione molto dettagliata né definitiva** delle entità di business e le loro relazioni. Sarà poi nella fase di progettazione, che seguendo un approccio top-down questo diagramma verrà raffinato, ed eventualmente modificato.

5.1 Package delle classi

Le classi del diagramma delle classi di analisi sono state "smistate" nei seguenti package:

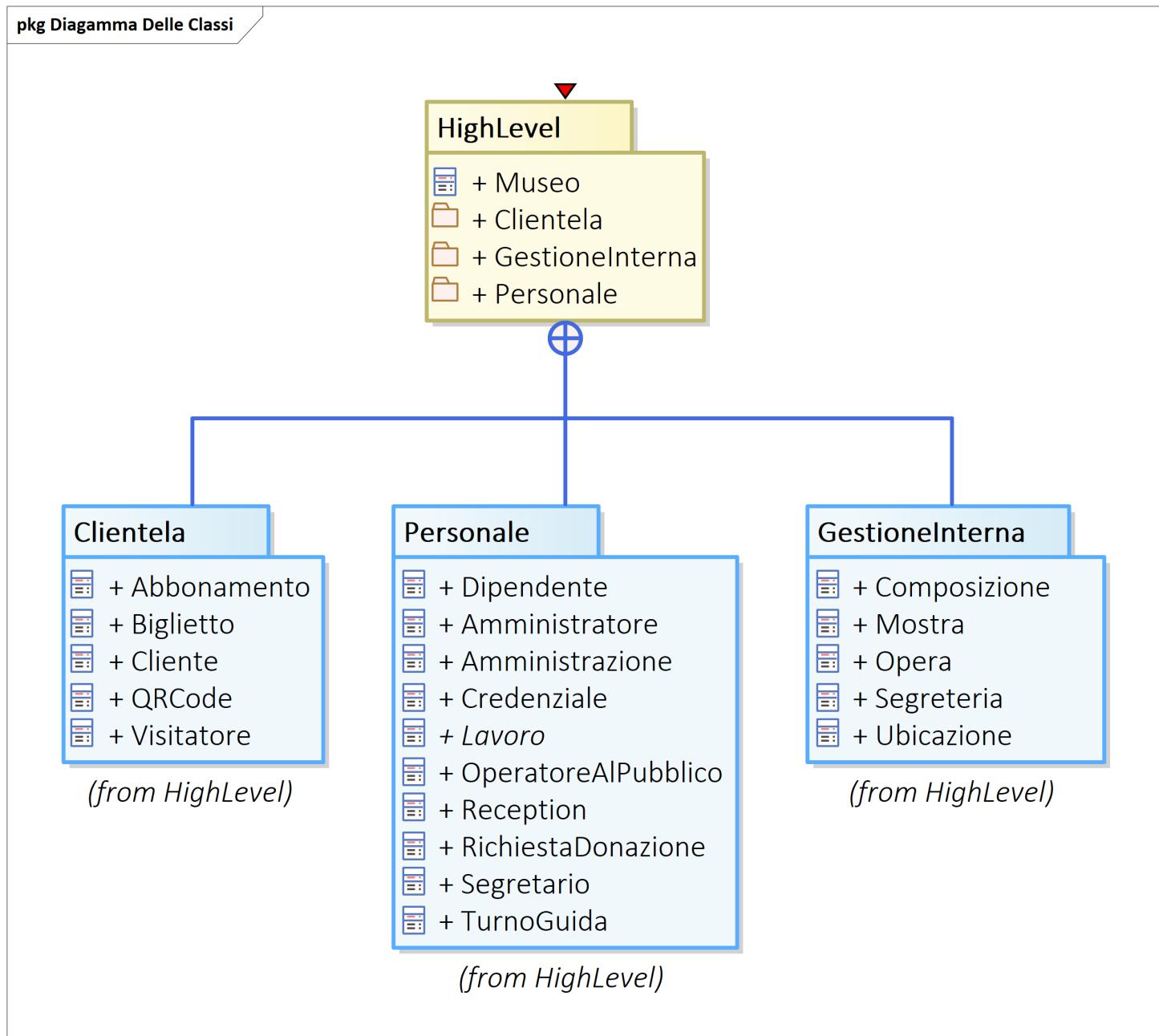


Figura 12: package delle classi

5.2 Diagramma delle classi

Segue quindi il diagramma completo delle classi:

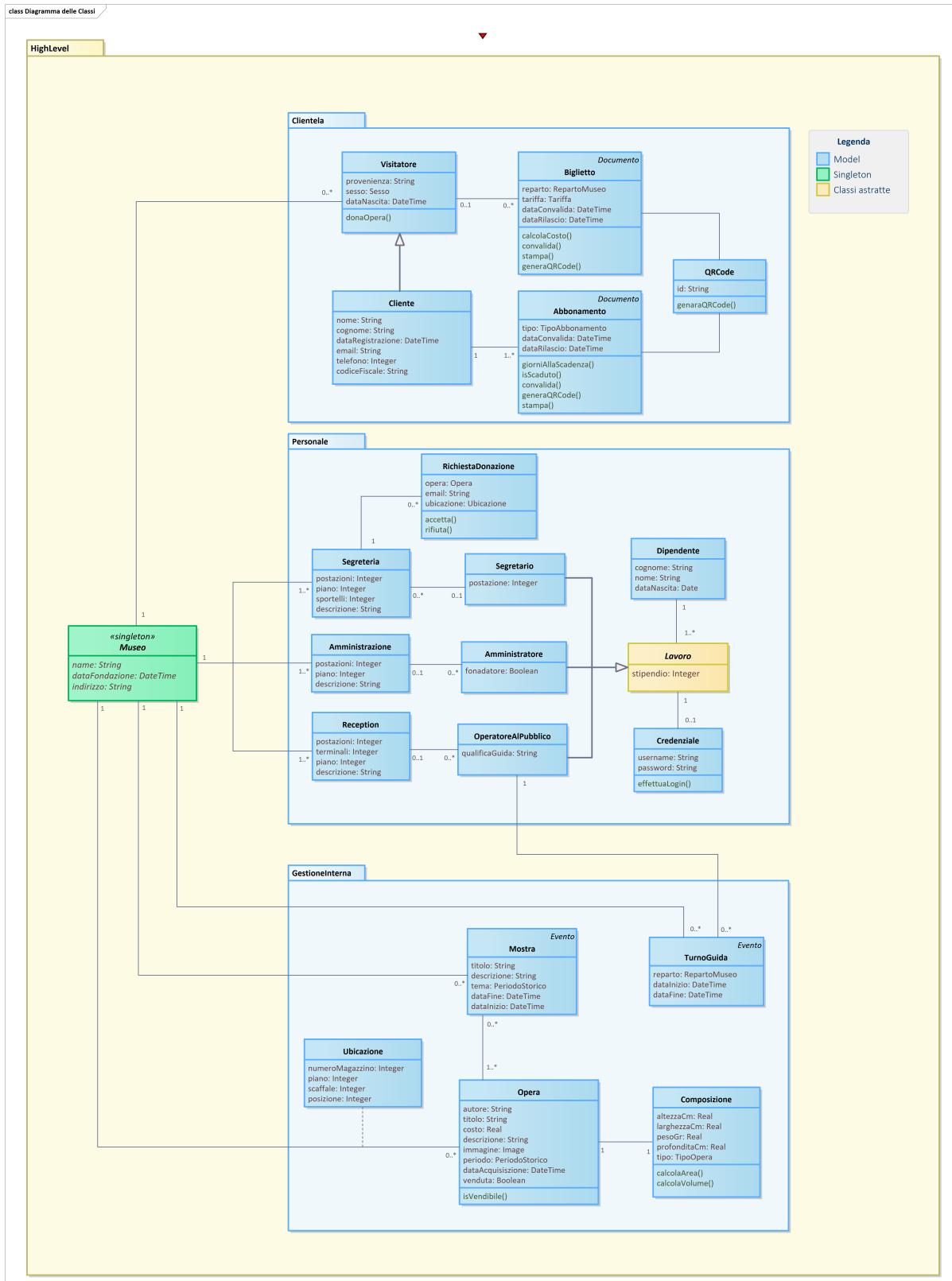


Figura 13: diagramma delle classi

6 Diagrammi di sequenza (analisi)

I **Diagrammi di Sequenza** espongono in maniera chiara come le diverse parti della struttura museale comunicano tra di loro e con il sistema scambiandosi messaggi con lo scopo di svolgere le loro funzioni.

In questi diagrammi di sequenza di analisi, Il concetto di **gestore** è stato usato con lo scopo di **interporre un'interfaccia**, nel senso lato del termine, tra gli attori e le classi di business. Gli attori si interfacciano dunque con questo oggetto che fornisce un **livello di astrazione maggiore**, rispetto alle classi del modello. Come si vedrà poi in fase di progettazione, gli autori hanno preso la scelta di **adottare il pattern Model View Controller** al fine di modellare l'architettura del sistema. Dunque il concetto di **gestore sarà concretamente raffinato in quello di viste** che offrono una rappresentazione grafica dei dati memorizzati dal model e permettono all'utente di comunicare con il sistema interagendo con gli elementi dell'interfaccia.

Vedremo ora nel dettaglio i seguenti diagrammi di sequenza, è stato riportato vicino ad ogni elemento della lista, l'identificativo del **caso d'uso** dal quale è stata estratta la sequenza.

- **Gestione Amministrazione**

- **CompraOpera** (CU 20)
- **GeneraReportIncassi** (CU 23)
- **GestisciDipendenti** (CU 36)

- **Gestione Cliente**

- **CompraBiglietto** (CU 01)
- **ConvalidaBiglietto** (CU 06)
- **DonaOpera** (CU 10)
- **RicercaBiglietto** (CU 07)
- **VendiOpera** (CU 08)
- **VerificaPrivilegio** (CU 04)

- **Gestione Segreteria**

- **AbbonaCliente** (CU 11)
- **CUDAbbonamento** (CU 12)
- **GestisciDonazione** (CU 15)

- **Gestione Sistema**

- **ControllaAbbonamento** (CU 24)
- **EffettuaBackup** (CU 27)
- **InviaAvviso** (CU 25)
- **VerificaAbbonamento** (CU 26)

- **Gestione Statistiche**

- **InserisciDatiClienti** (CU 28)
- **VisualizzaStatistiche** (CU 32)

6.1 Gestione Amministrazione

CompraOpera

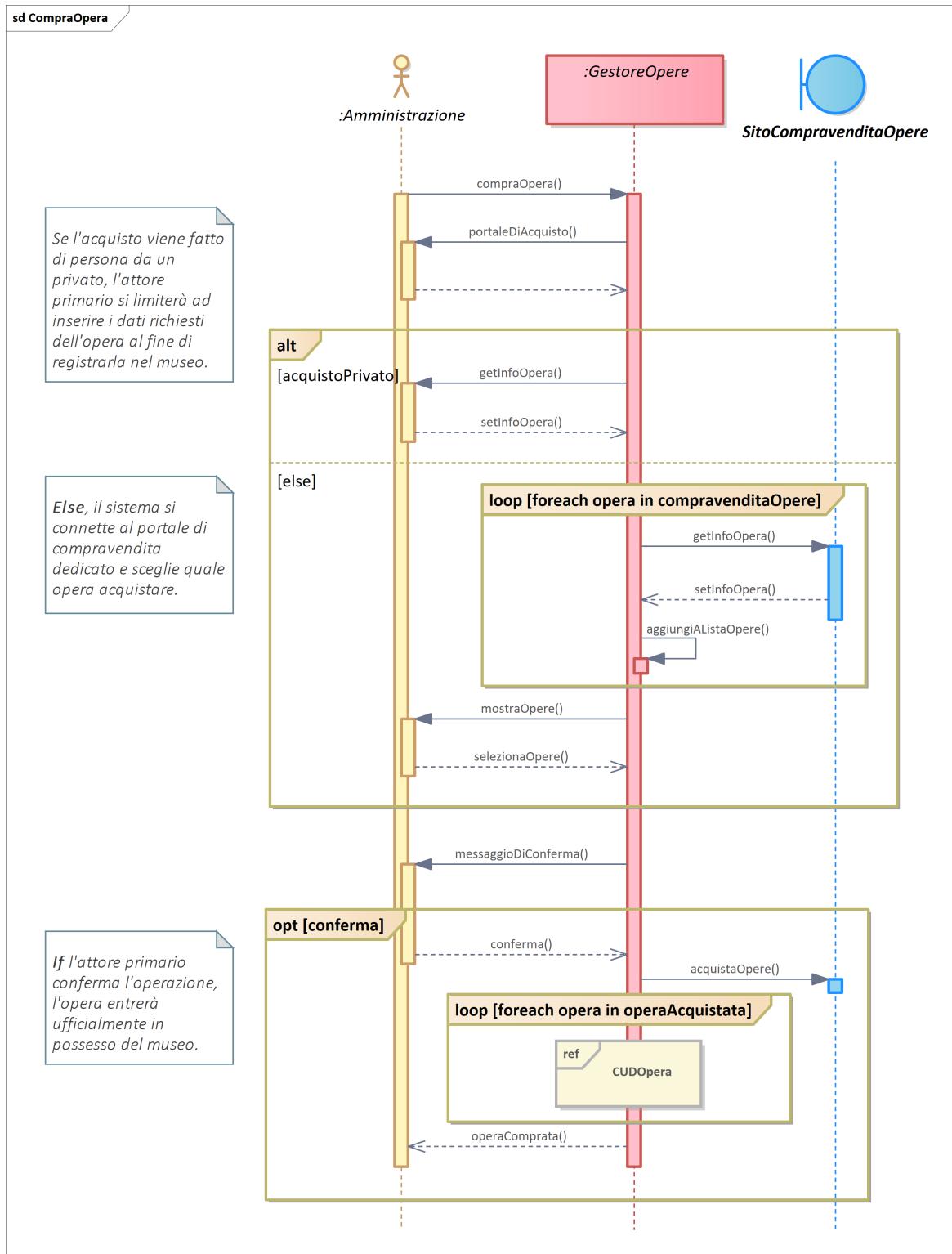


Figura 14: CompraOpera

GeneraReportIncassi

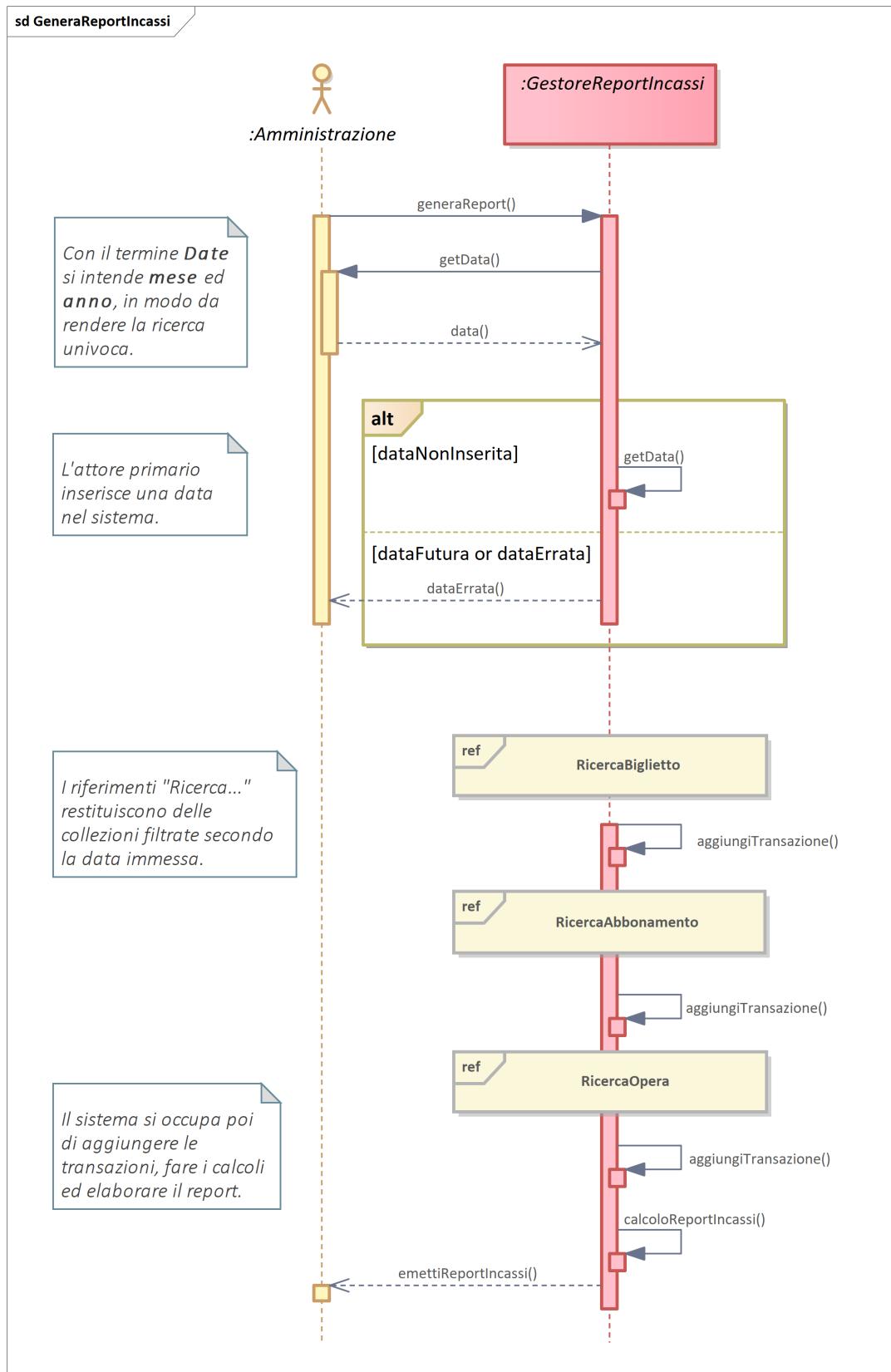


Figura 15: GeneraReportIncassi

GestisciDipendenti

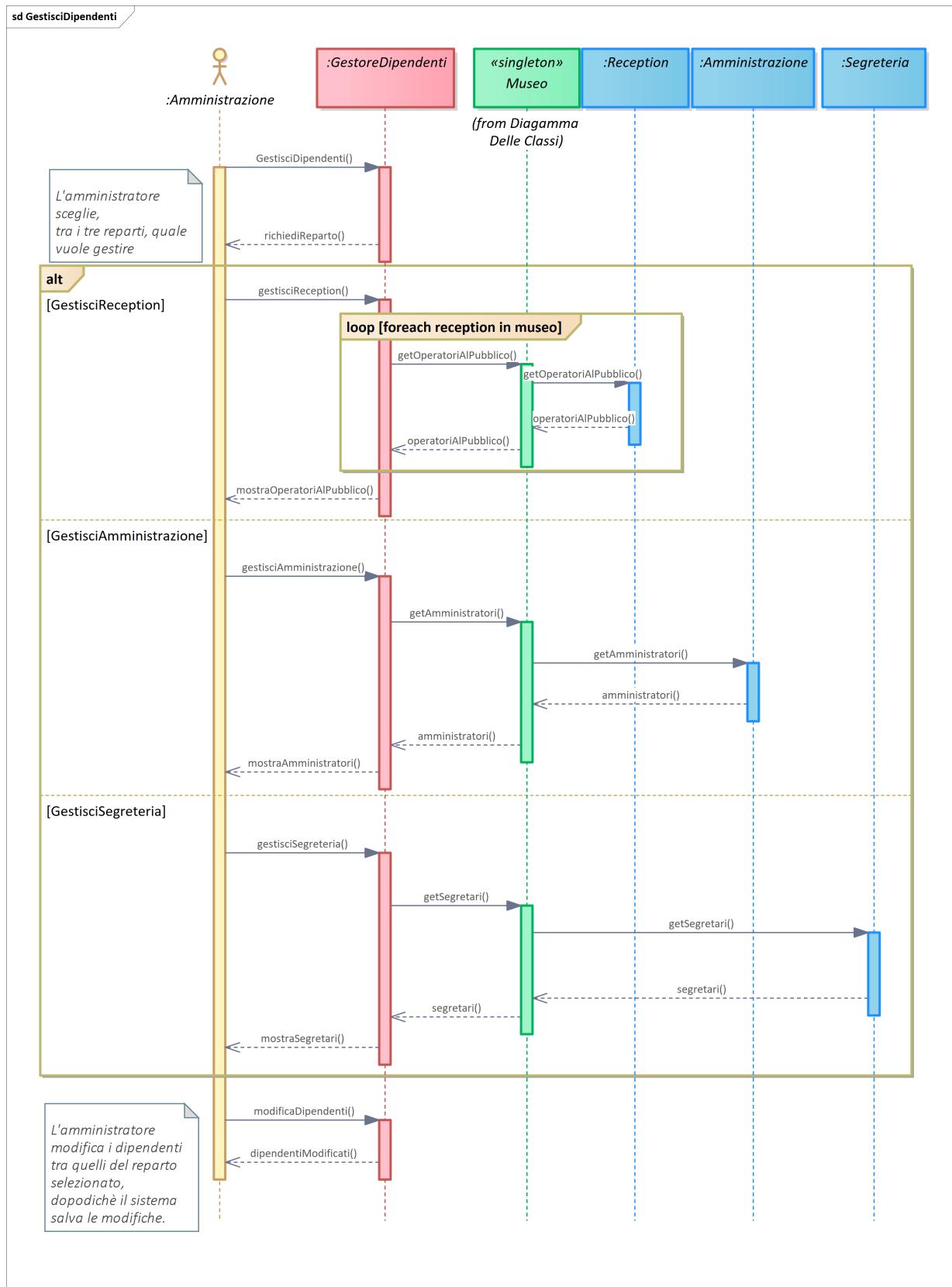


Figura 16: GestisciDipendenti

6.2 Gestione Cliente

CompraBiglietto 1/2

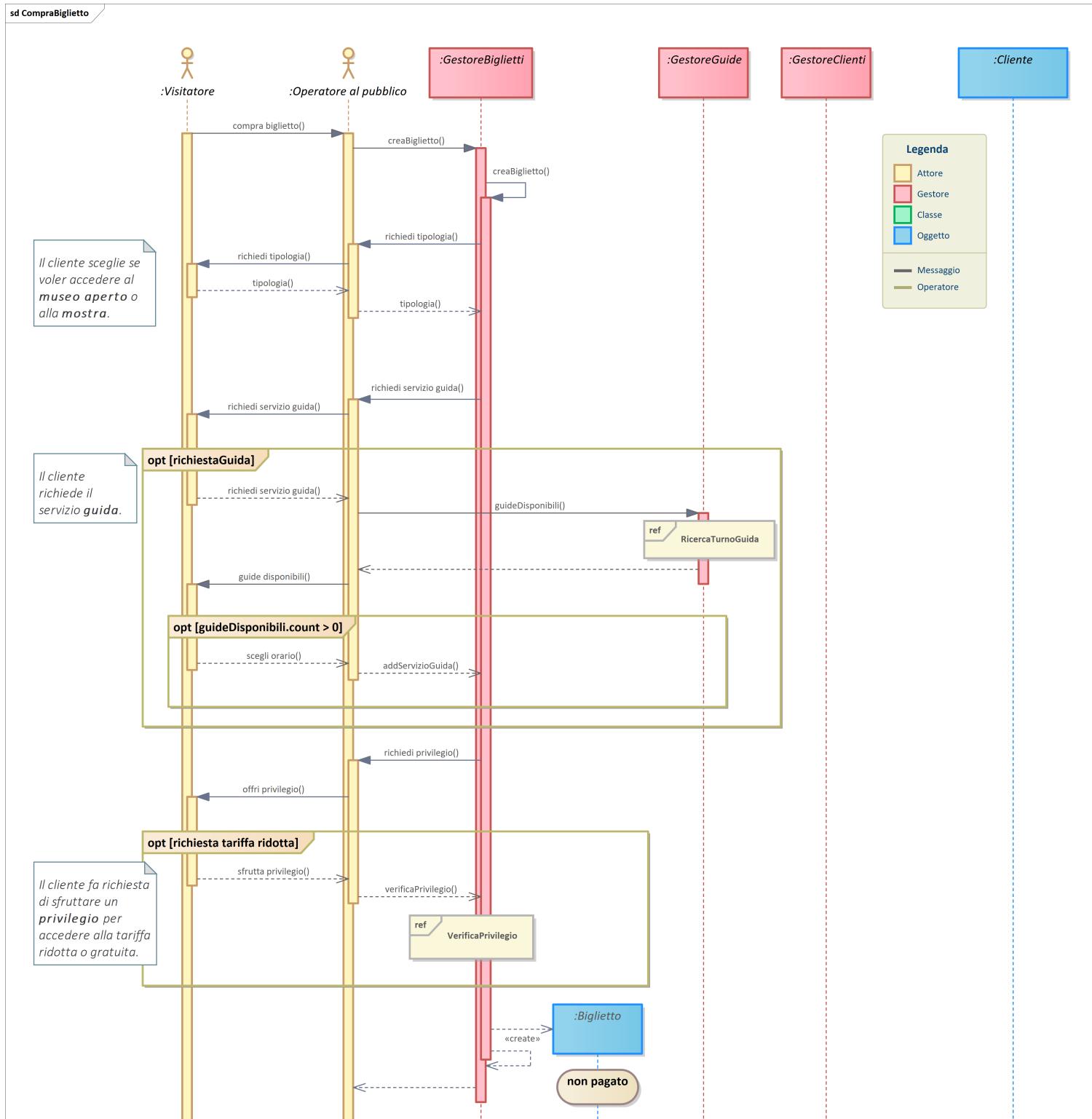


Figura 17: CompraBiglietto 1/2

CompraBiglietto 2/2

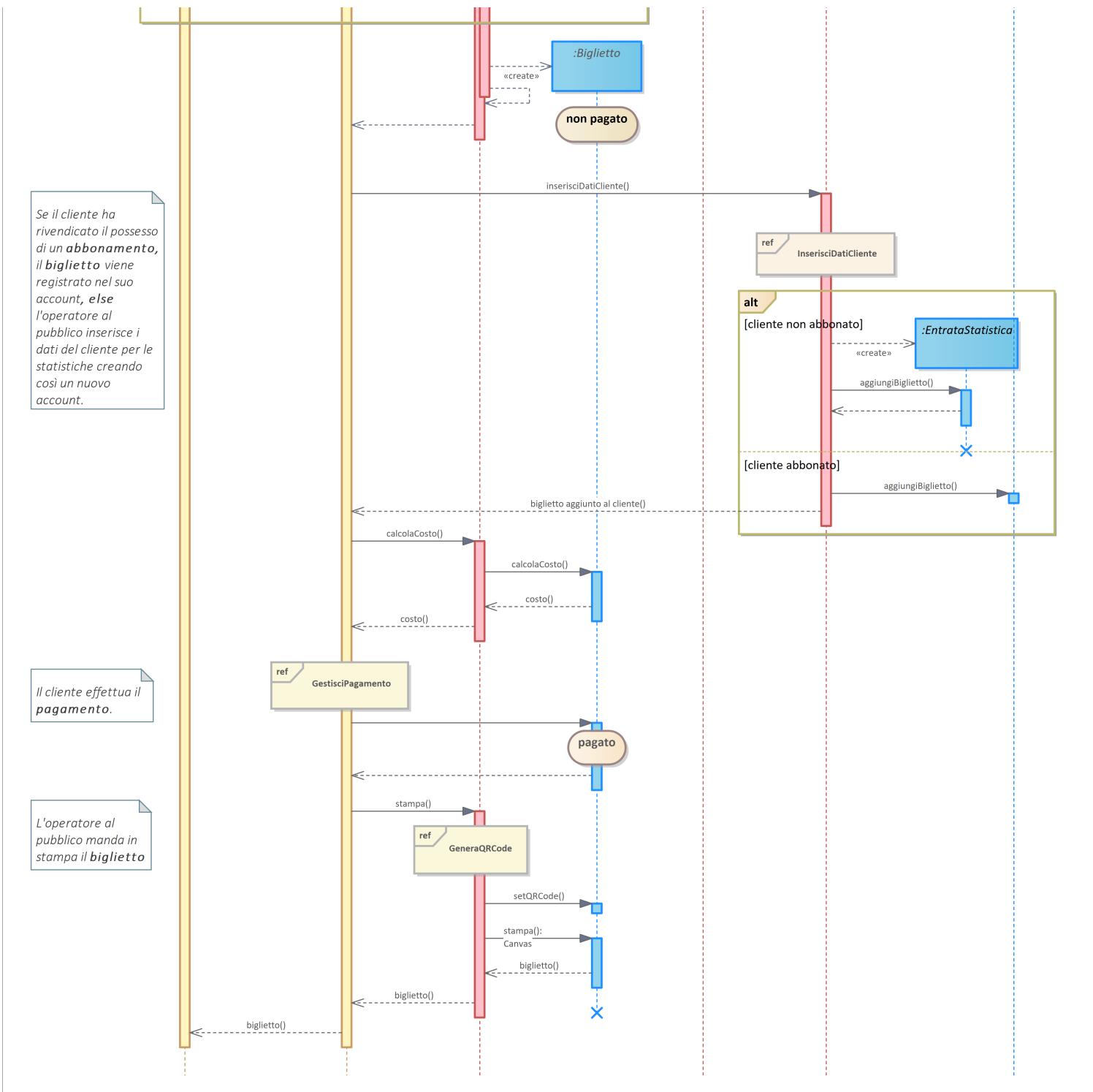


Figura 18: CompraBiglietto 2/2

ConvalidaBiglietto

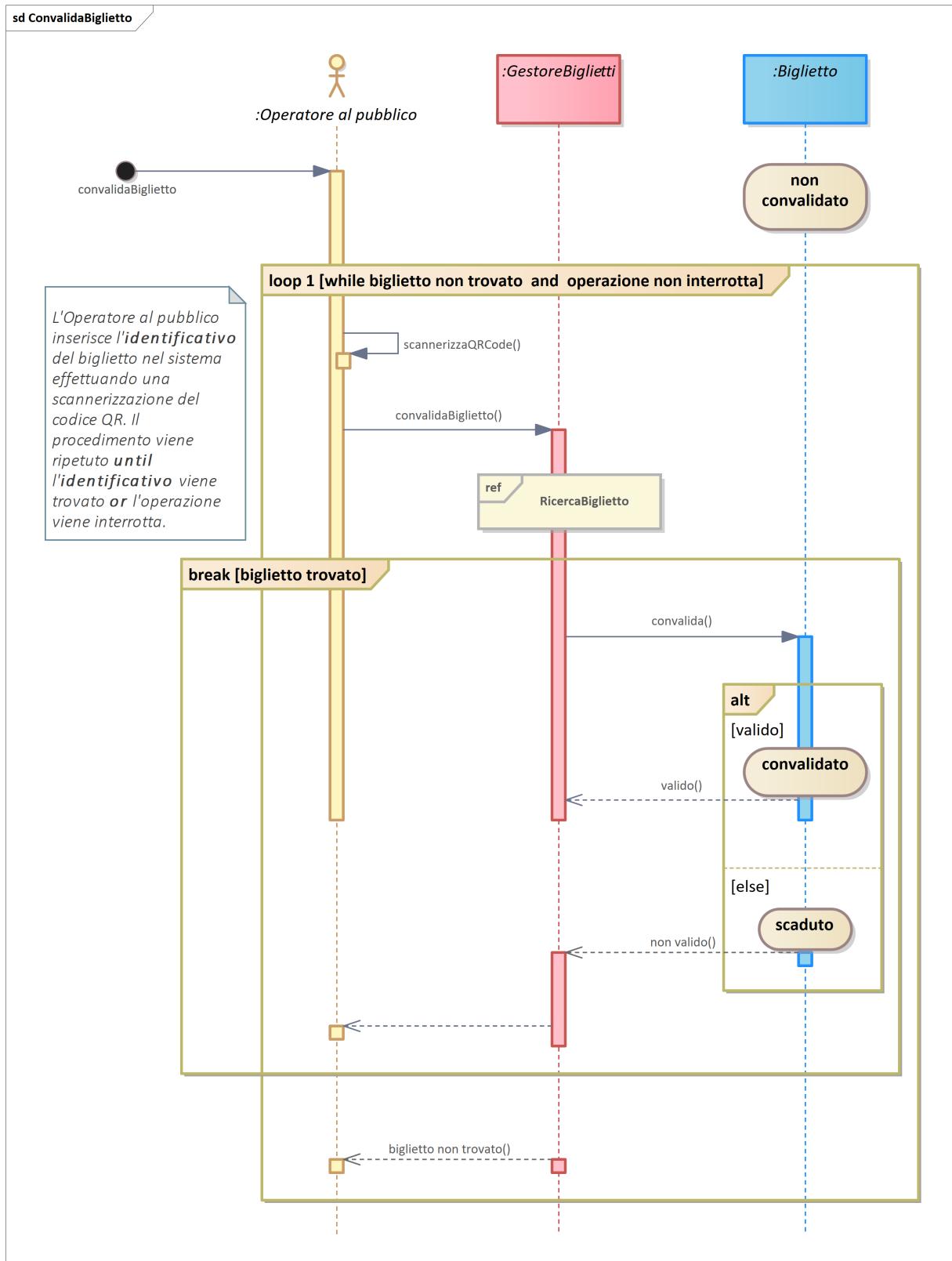


Figura 19: ConvalidaBiglietto

DonaOpera

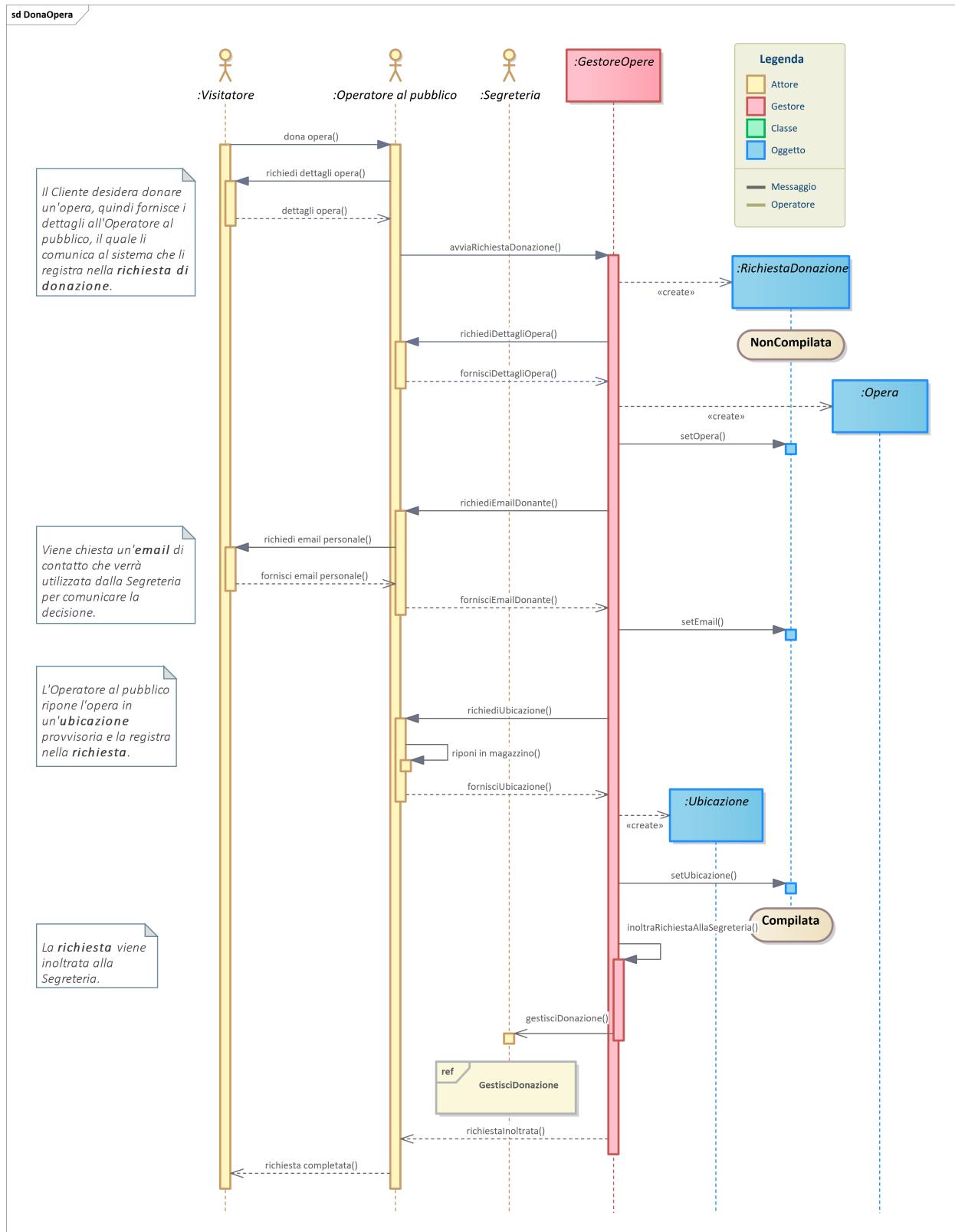


Figura 20: DonaOpera

RicercaBiglietto

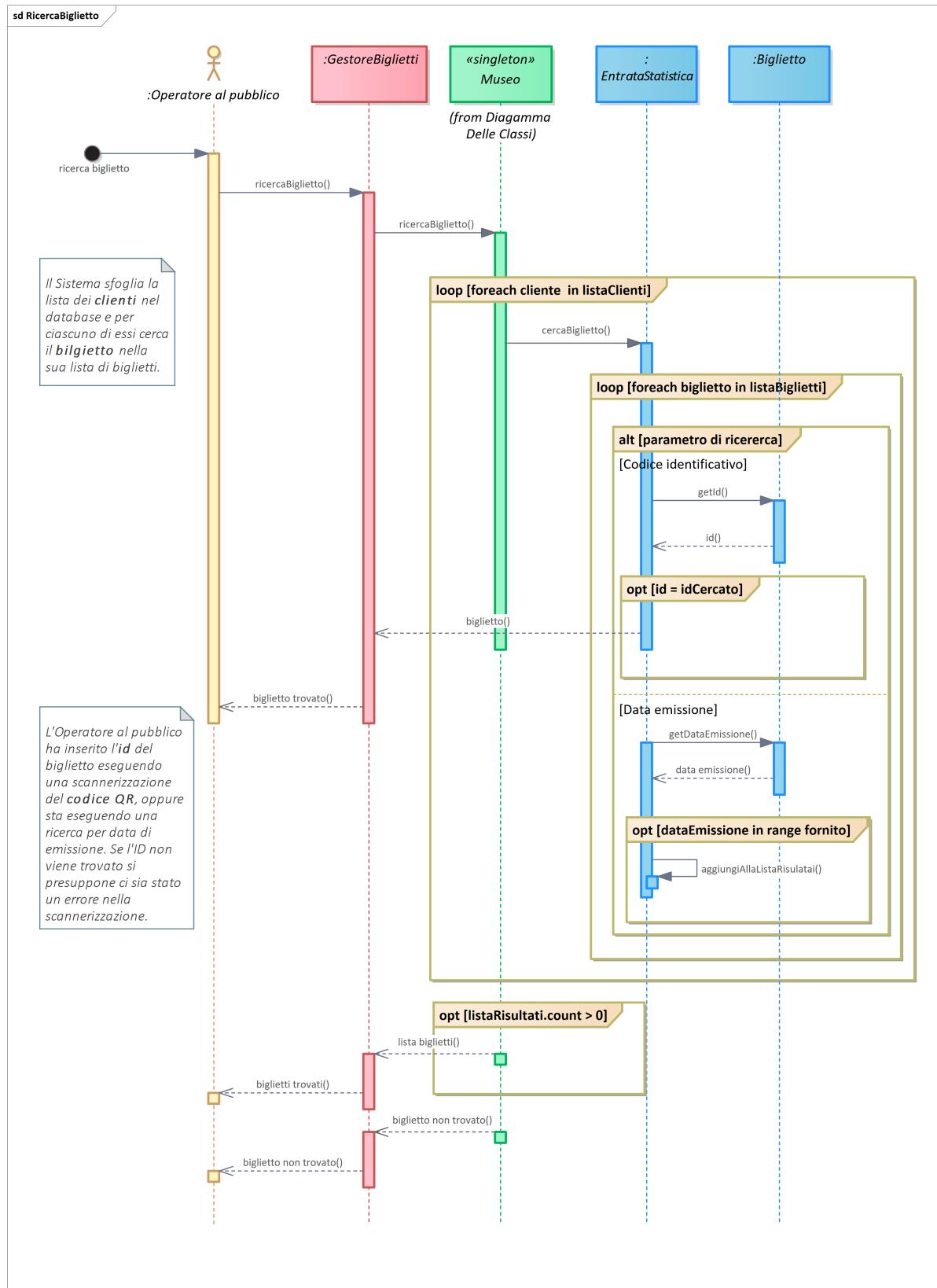


Figura 21: RicercaBiglietto

VendiOpera 1/2

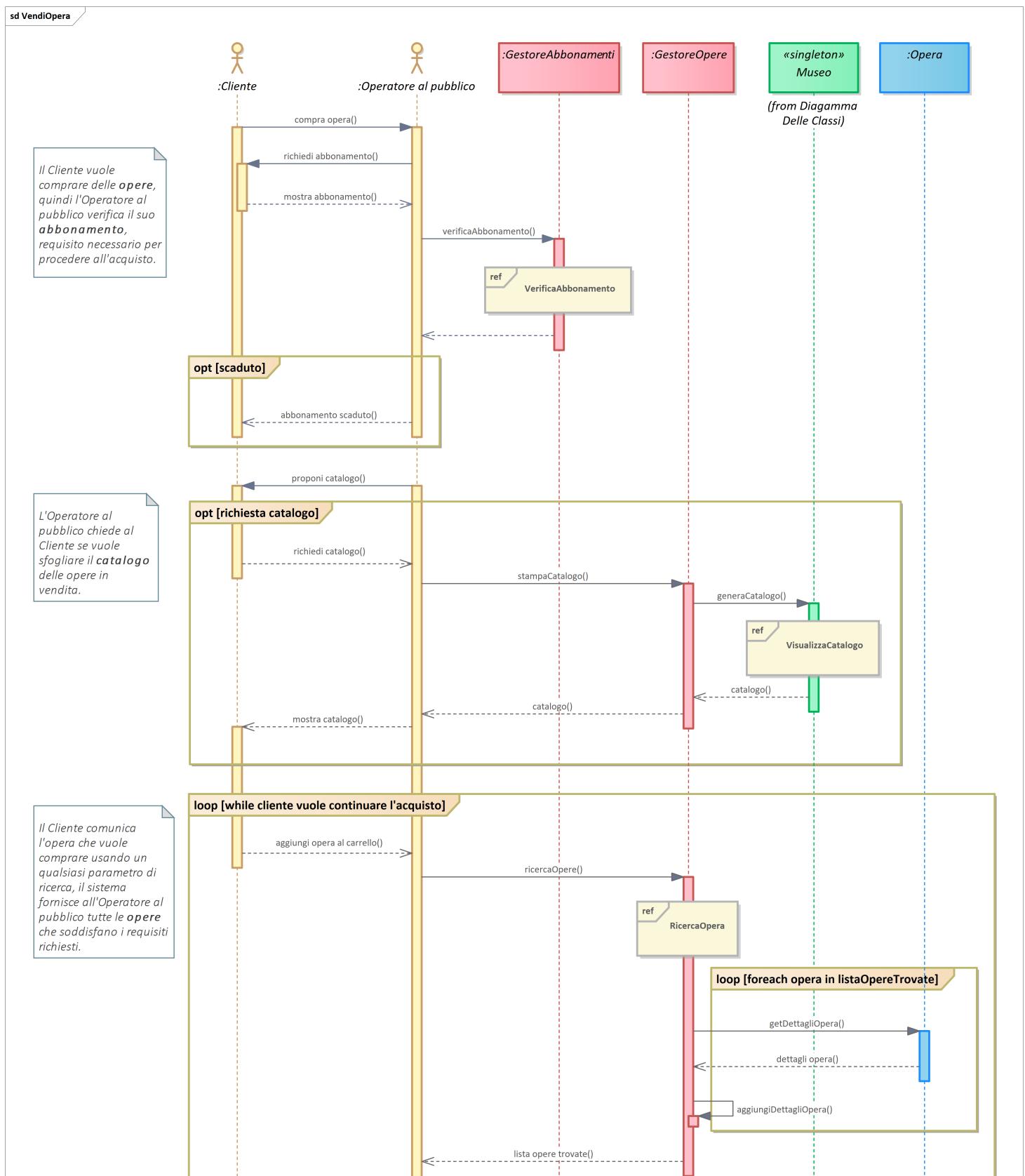


Figura 22: VendiOpera 1/2

VendiOpera 2/2

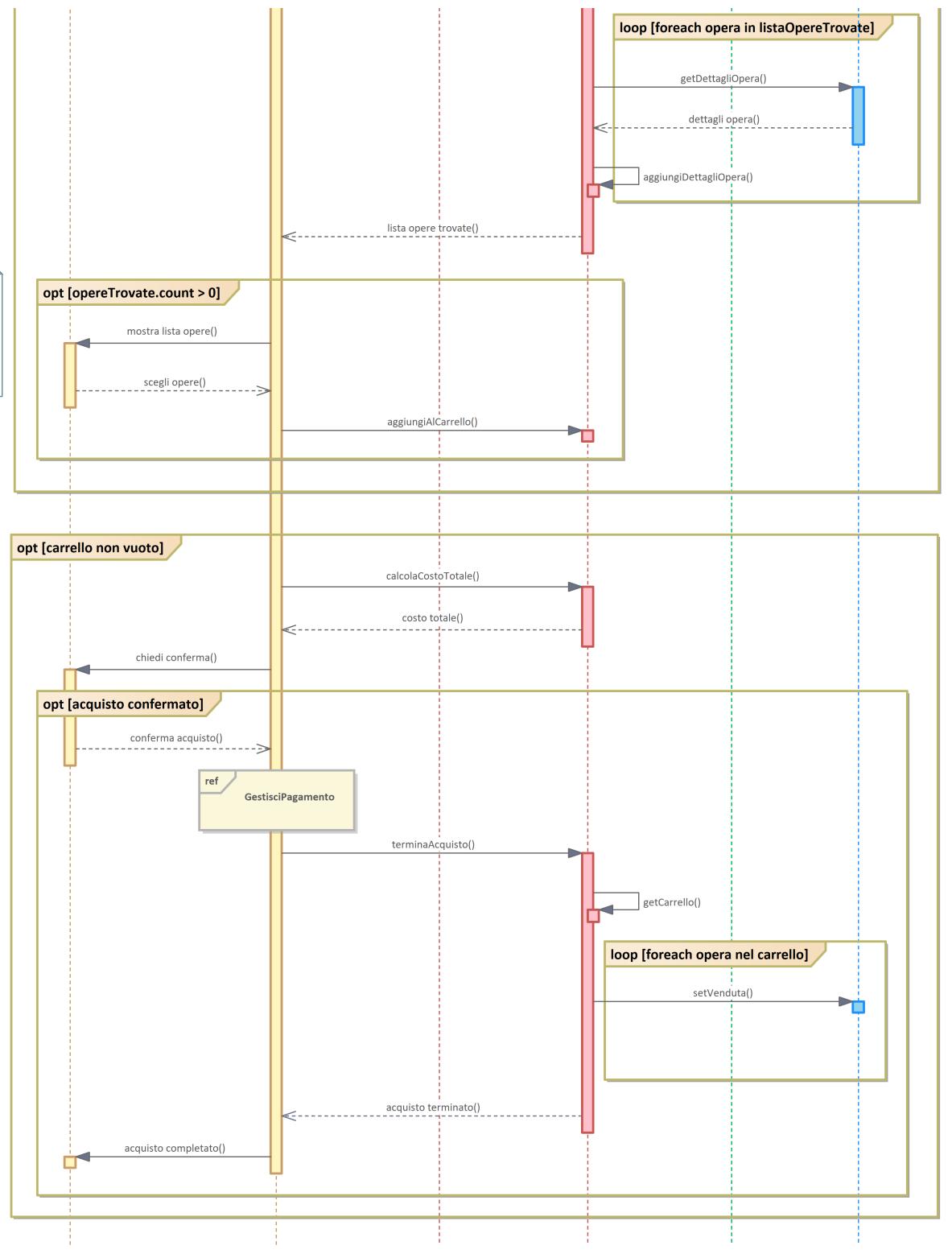


Figura 23: VendiOpera 2/2

VerificaPrivilegio 1/2

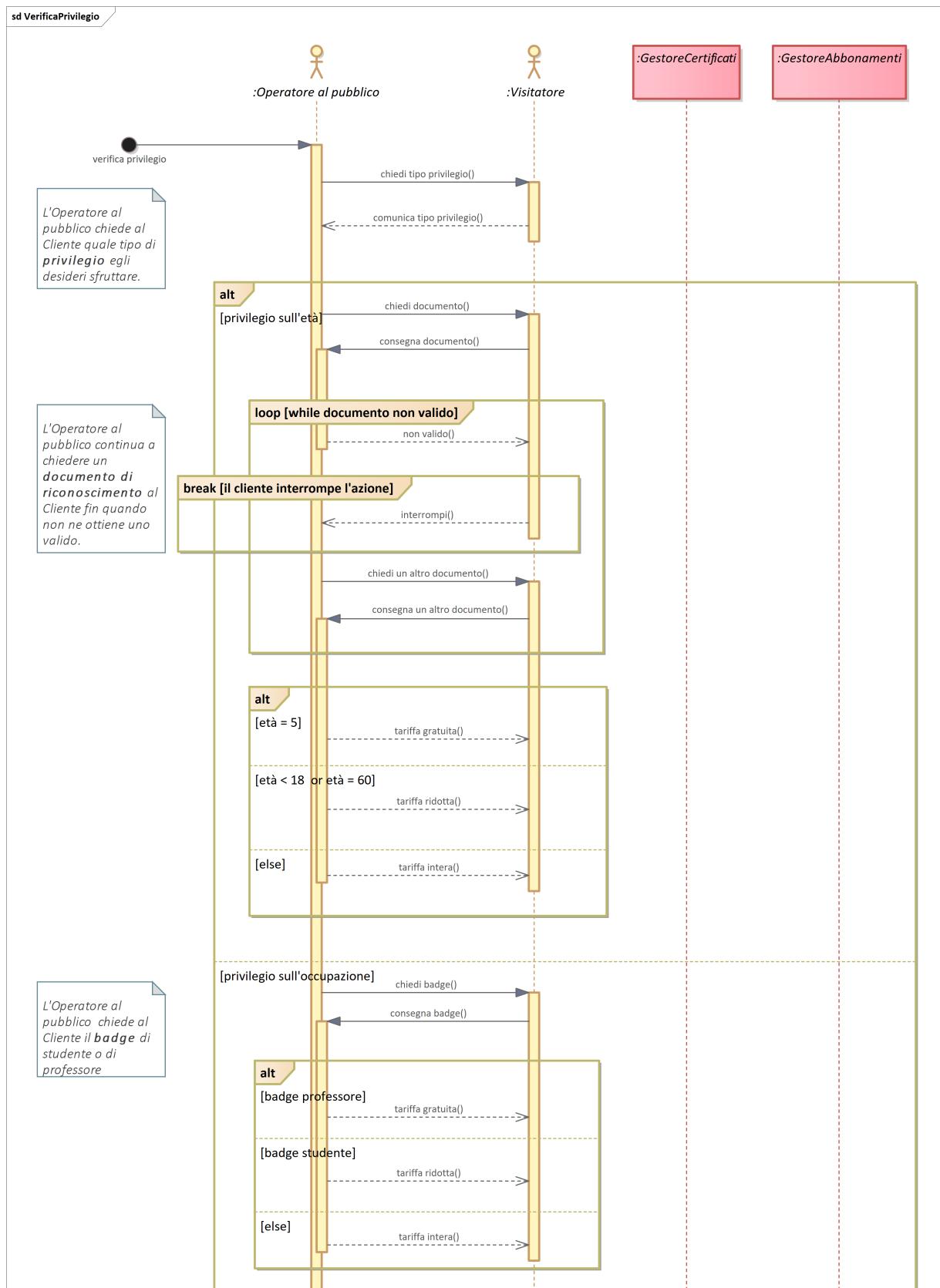


Figura 24: VerificaPrivilegio

VerificaPrivilegio 2/2

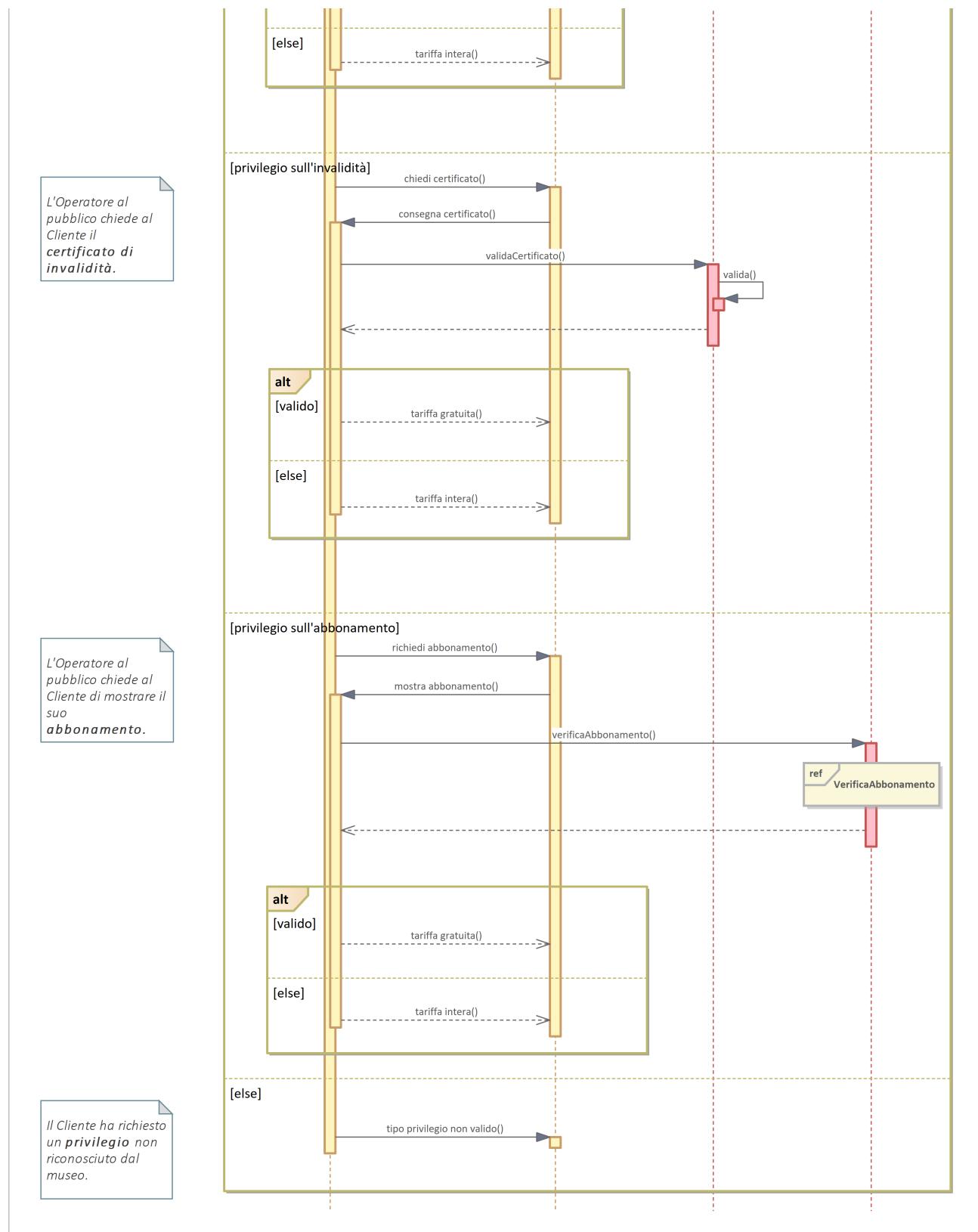


Figura 25: VerificaPrivilegio

6.3 Gestione Segreteria

AbbonaCliente 1/2

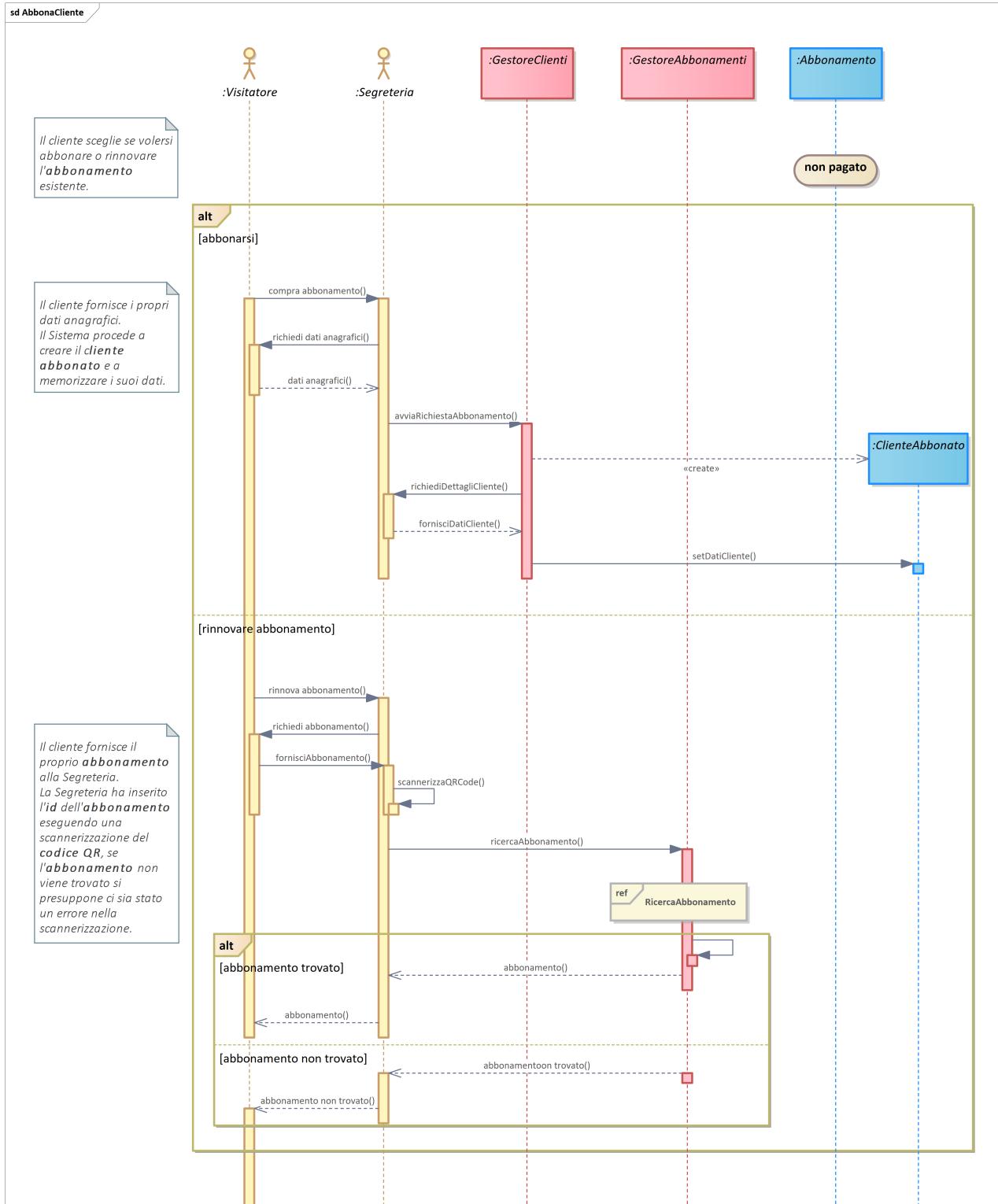


Figura 26: AbbonaCliente 1/2

AbbonaCliente 2/2

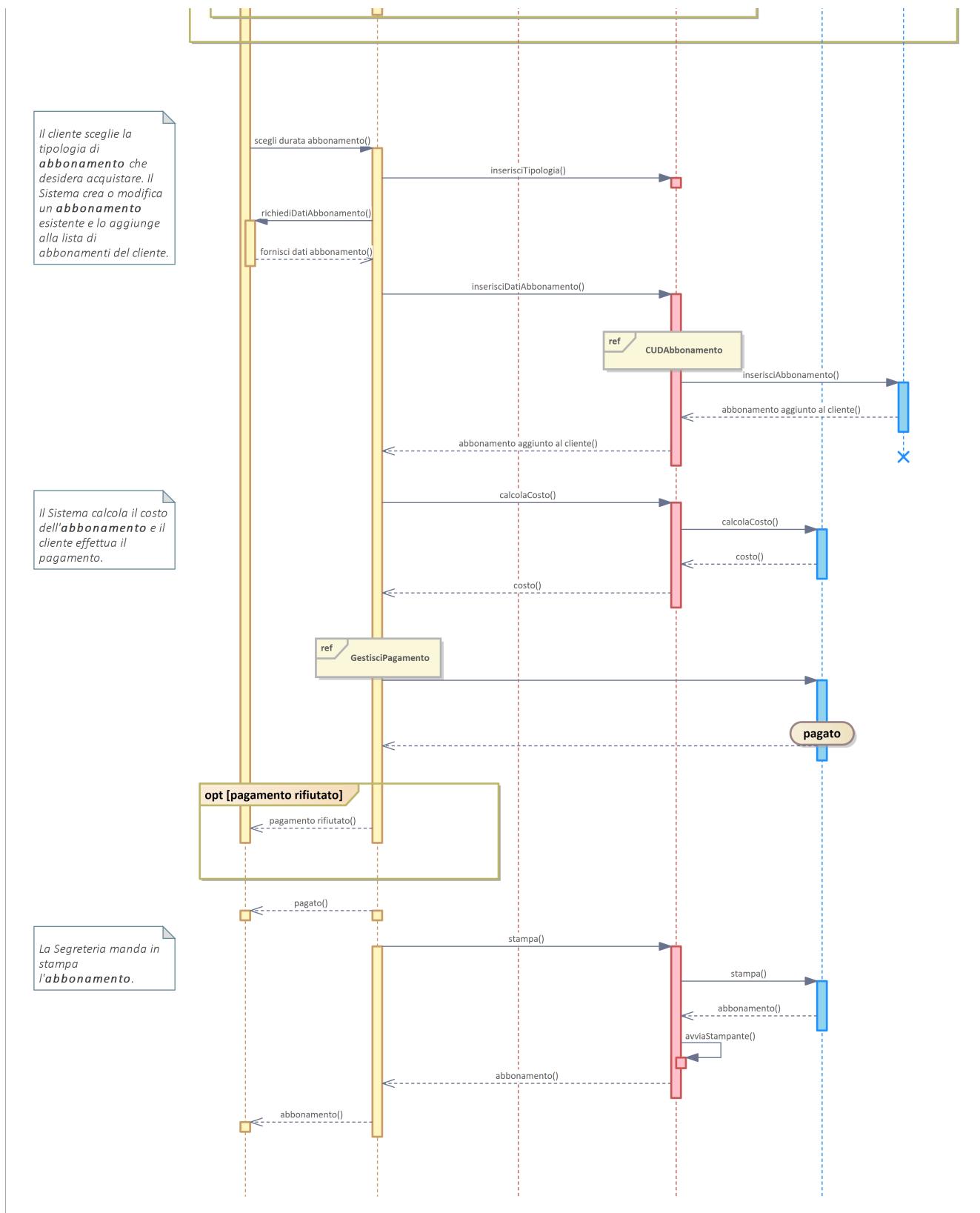


Figura 27: AbbonaCliente 2/2

CUDAbbonamento 1/2

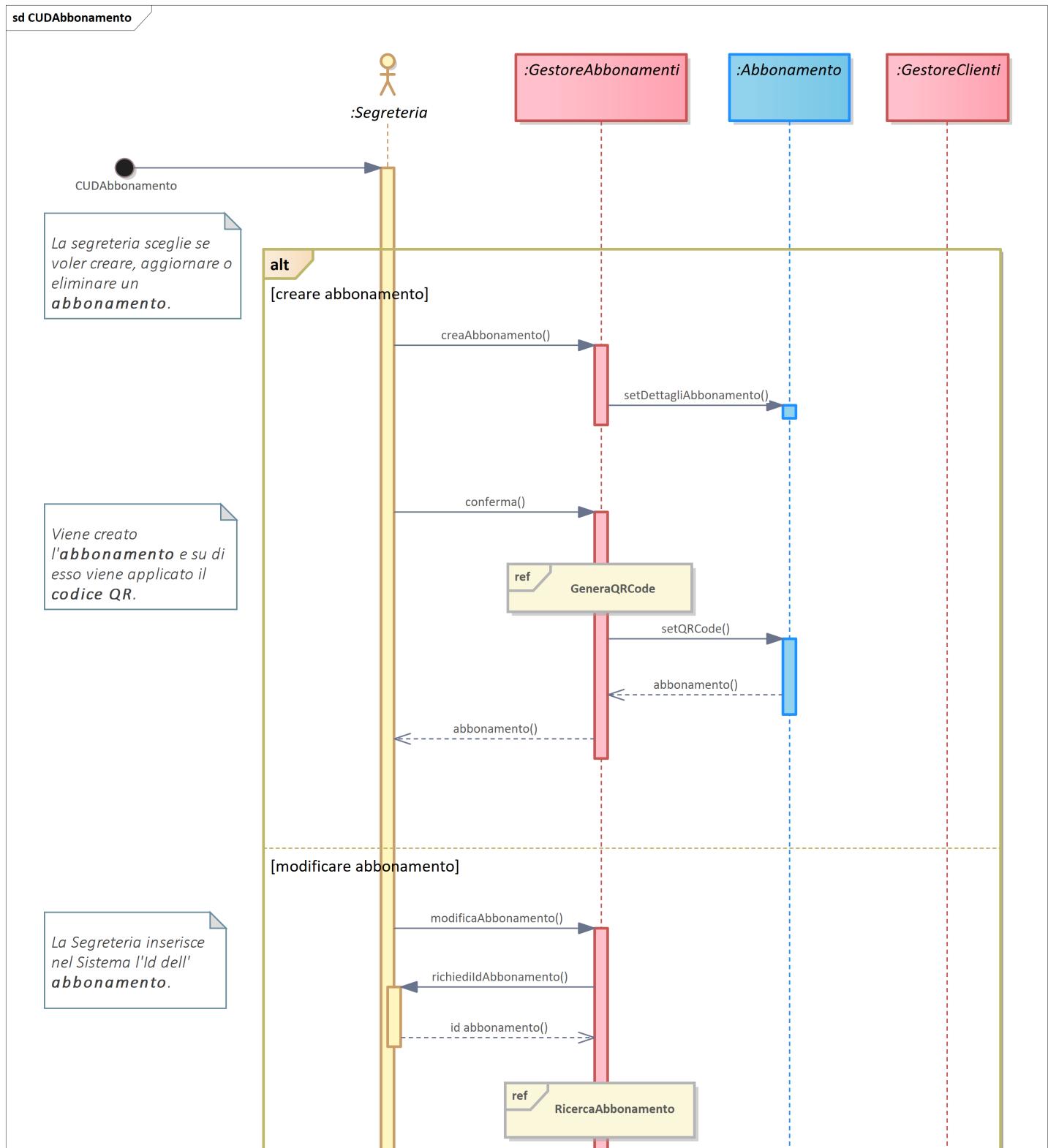


Figura 28: CUDAbbonamento 1/2

CUDAbbonamento 2/2

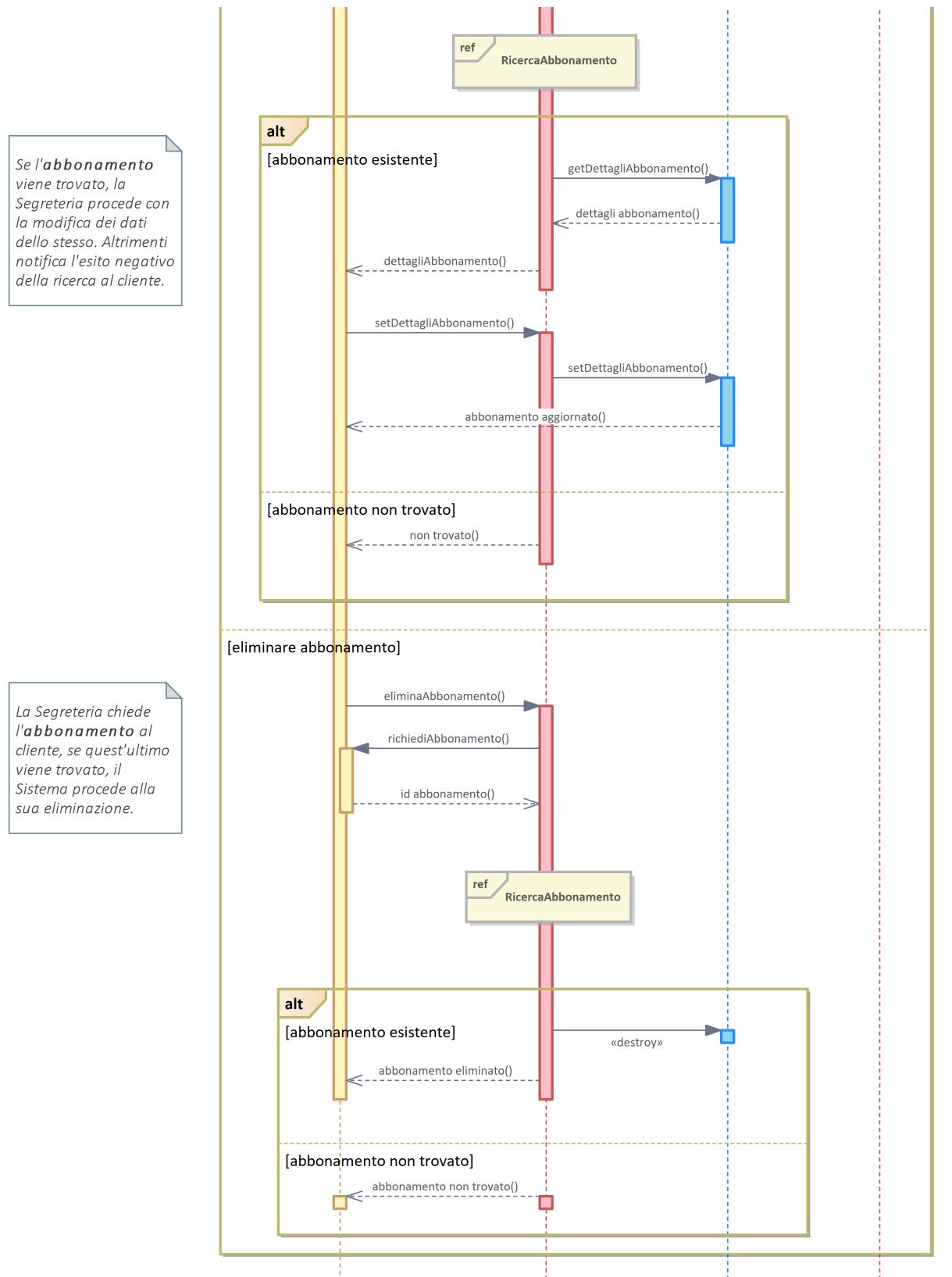


Figura 29: CUDAbbonamento 2/2

GestisciDonazione

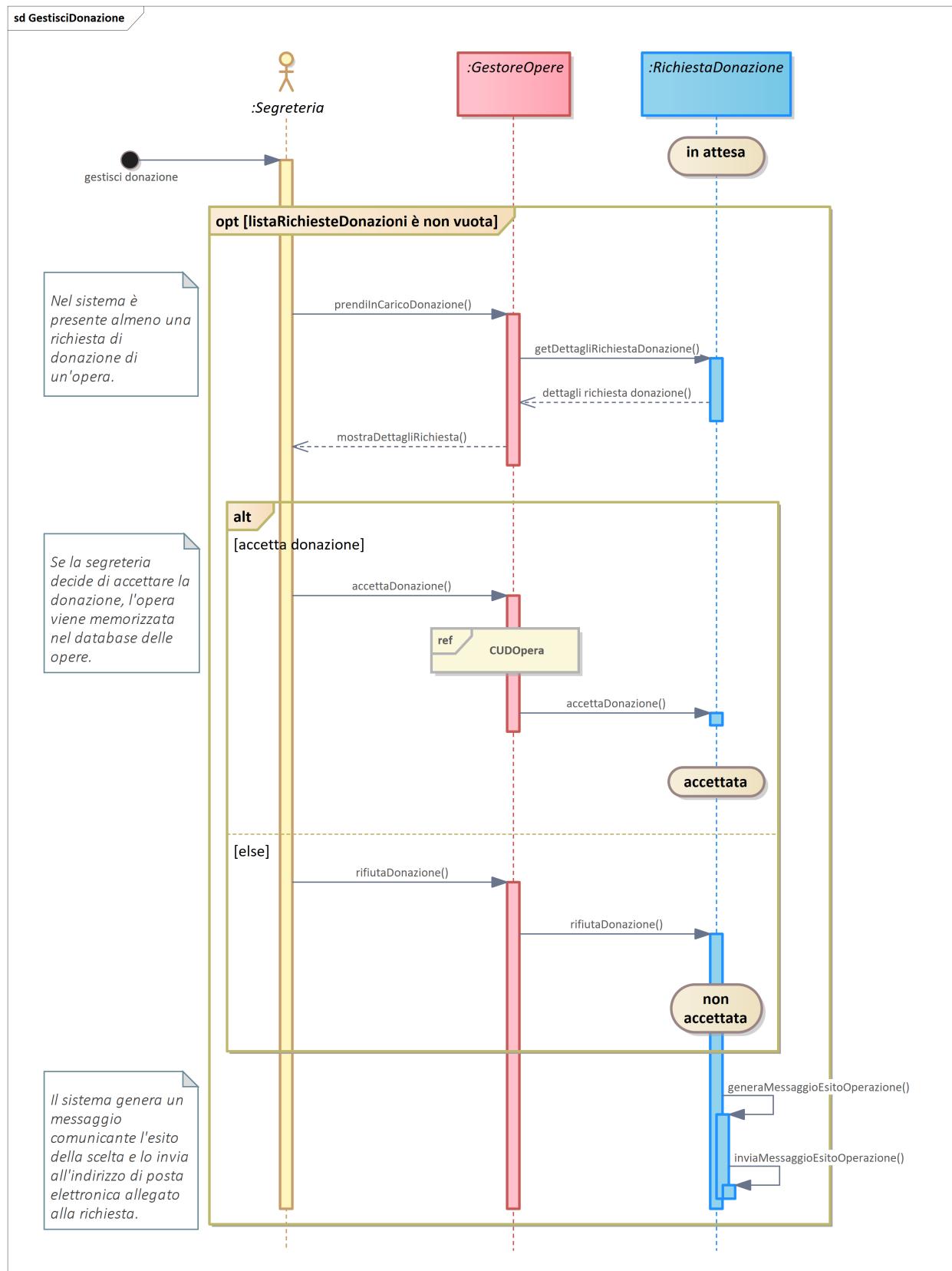


Figura 30: GestisciDonazione

6.4 Gestione Sistema

ControllaAbbonamento

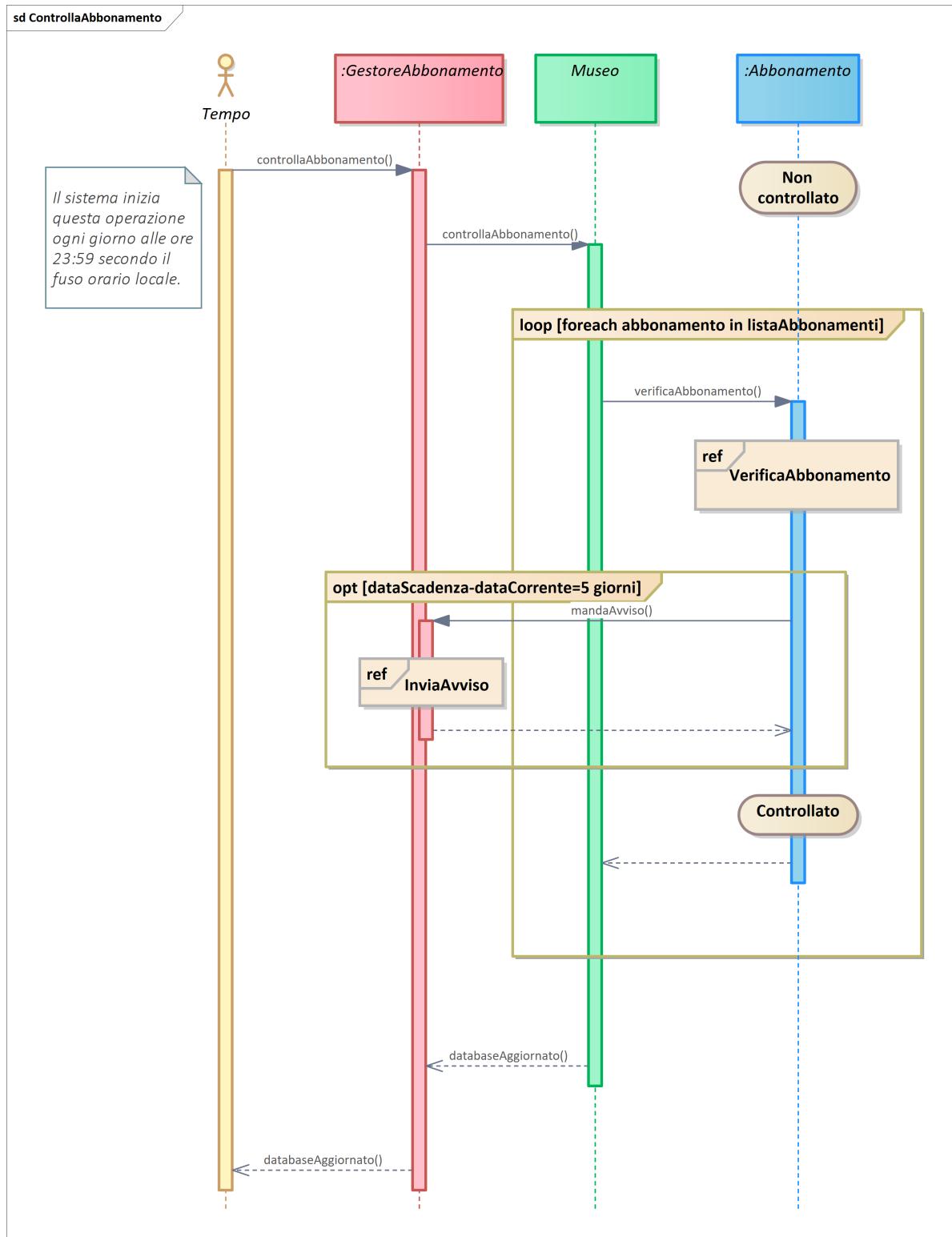


Figura 31: ControllaAbbonamento

EffettuaBackup

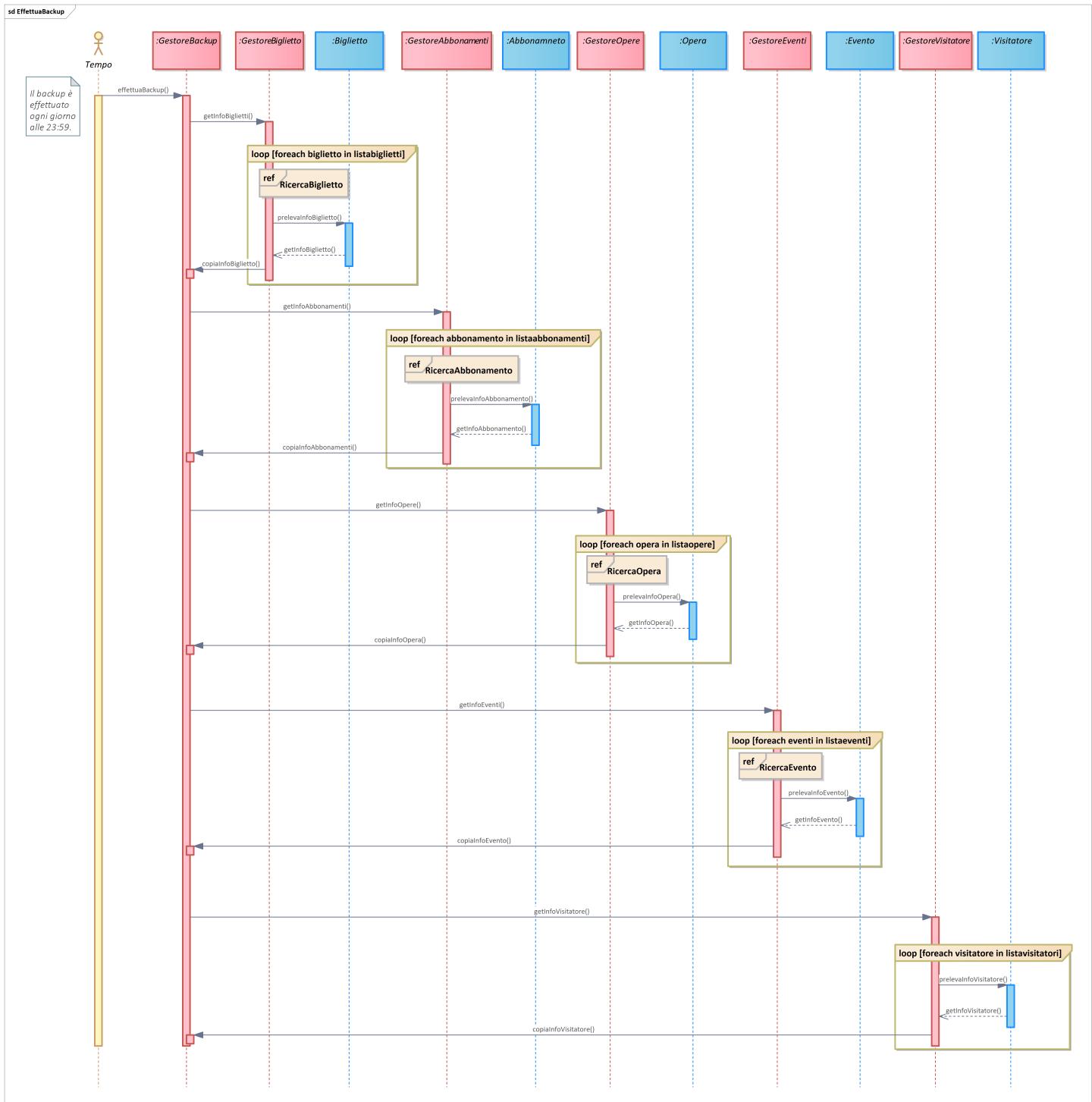


Figura 32: EffettuaBackup

InviaAvviso

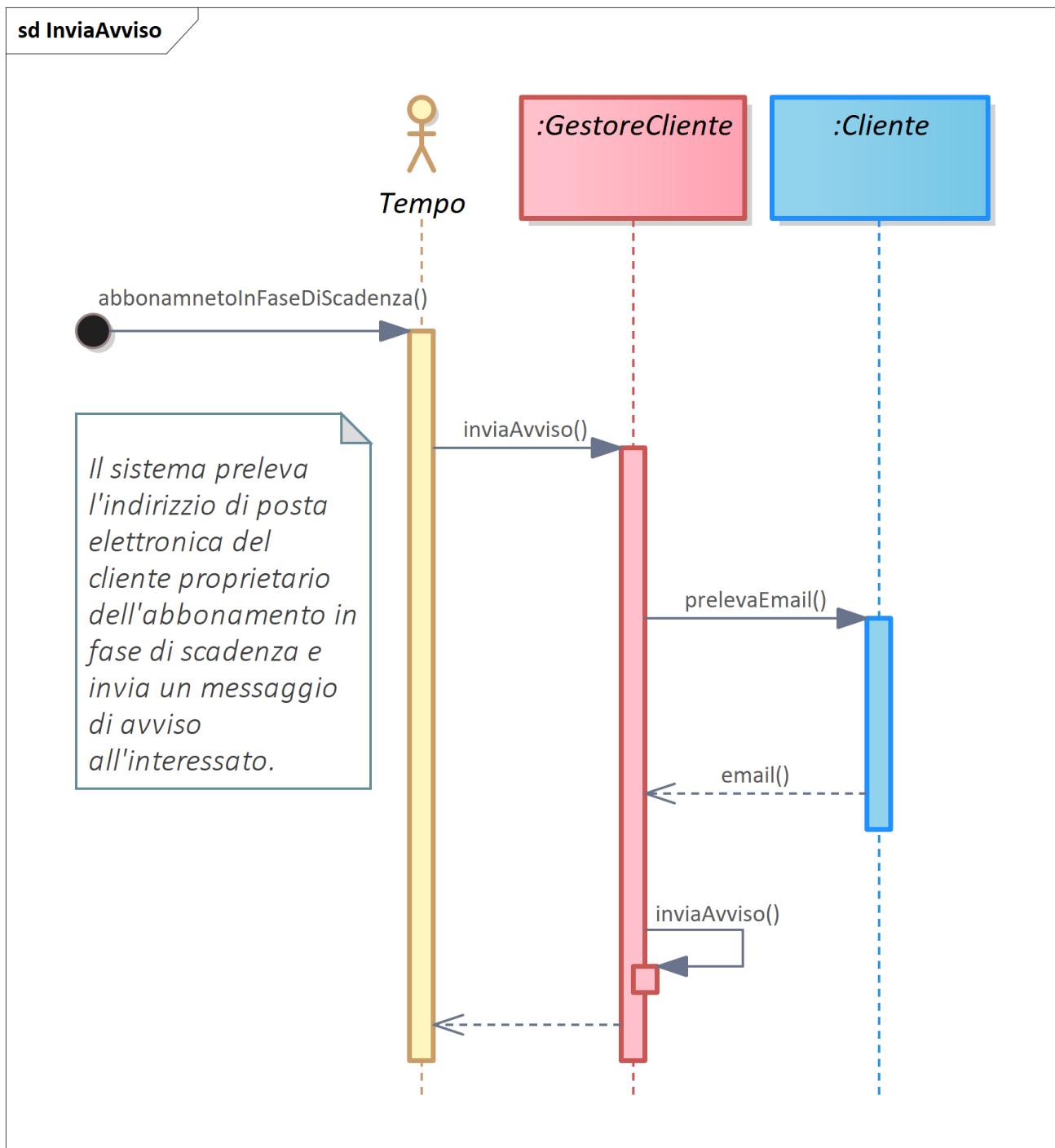


Figura 33: InviaAvviso

VerificaAbbonamento

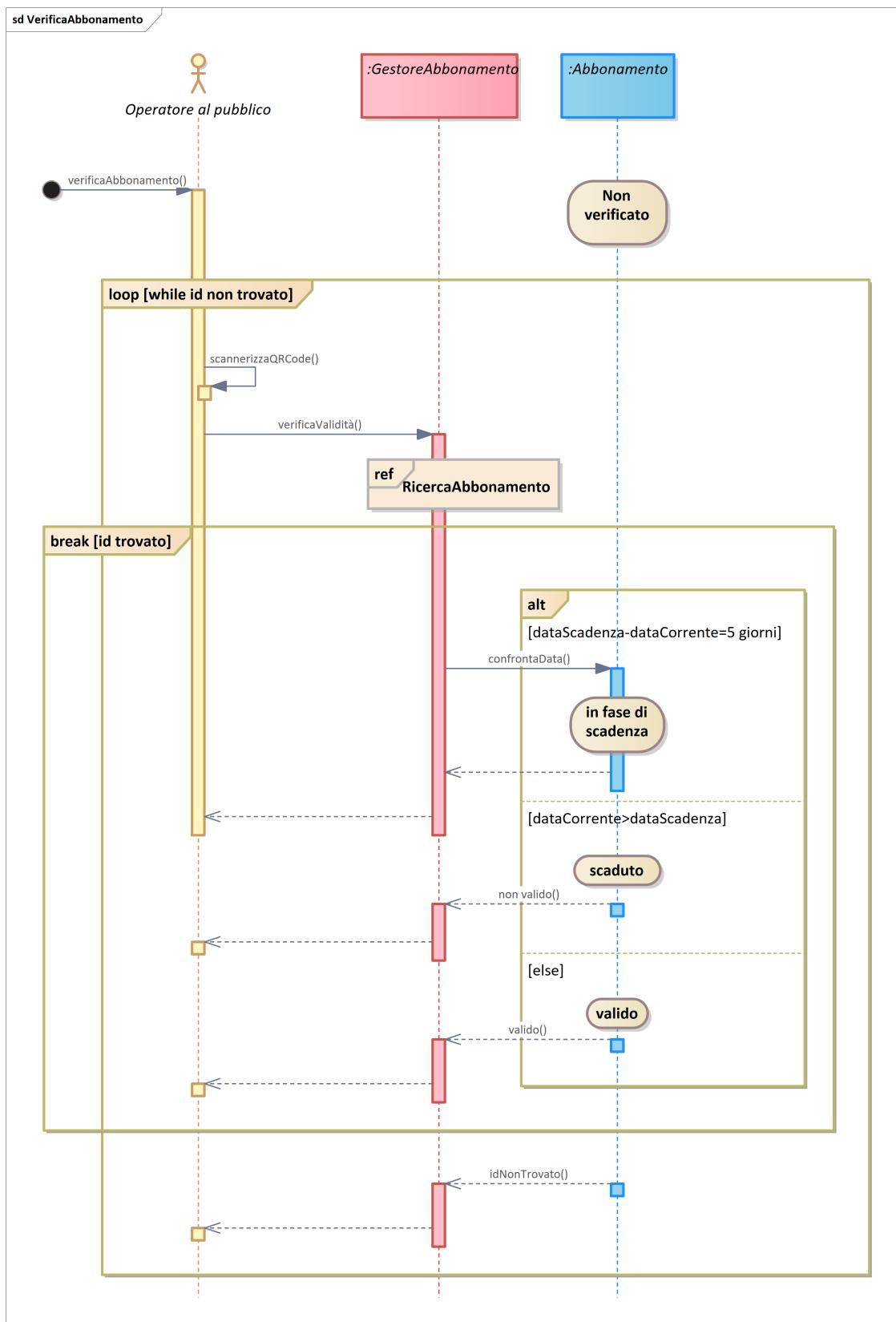


Figura 34: VerificaAbbonamento

6.5 Gestione Statistiche

InserisciDatiClienti

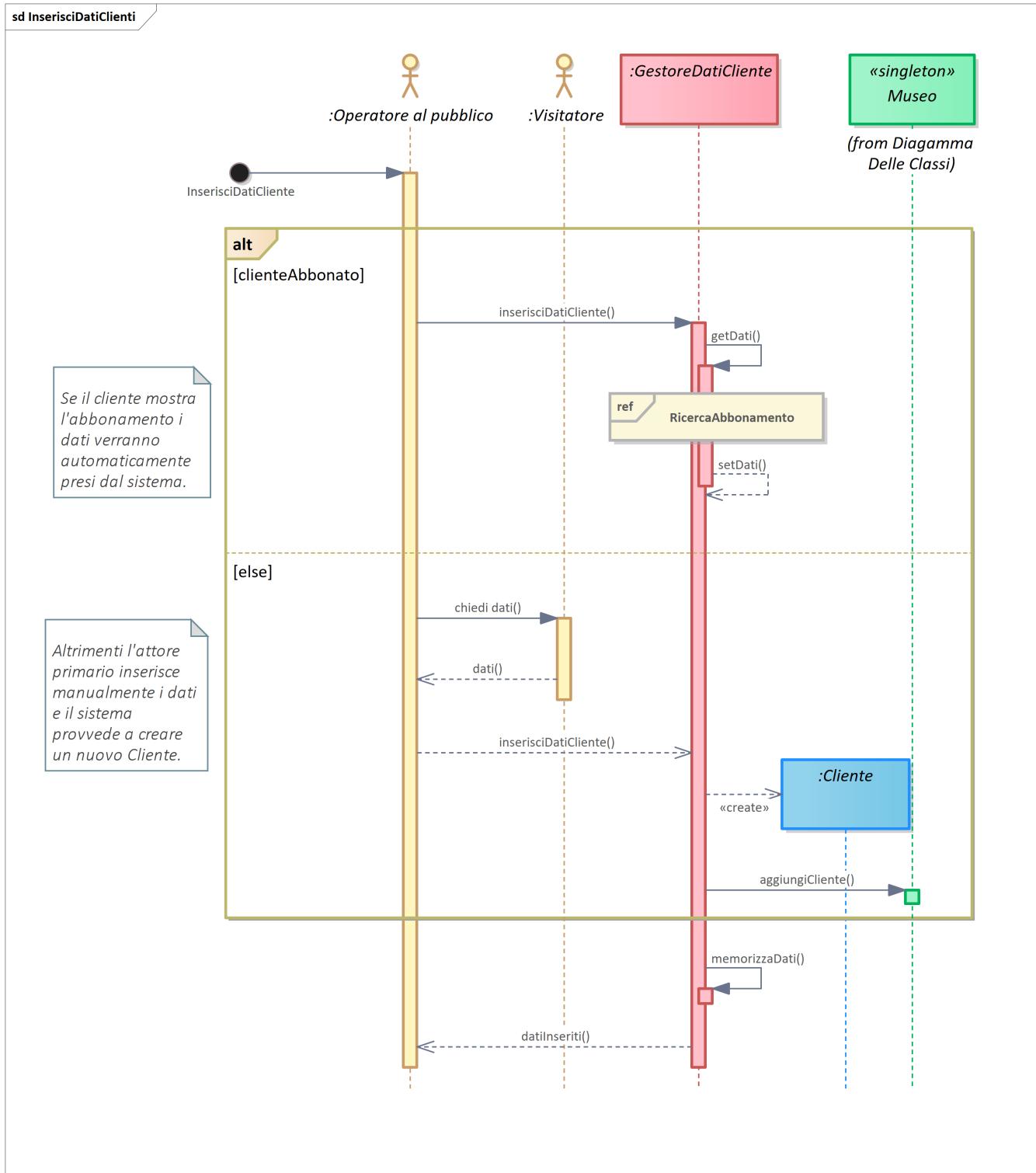


Figura 35: InserisciDatiClienti

VisualizzaStatistiche

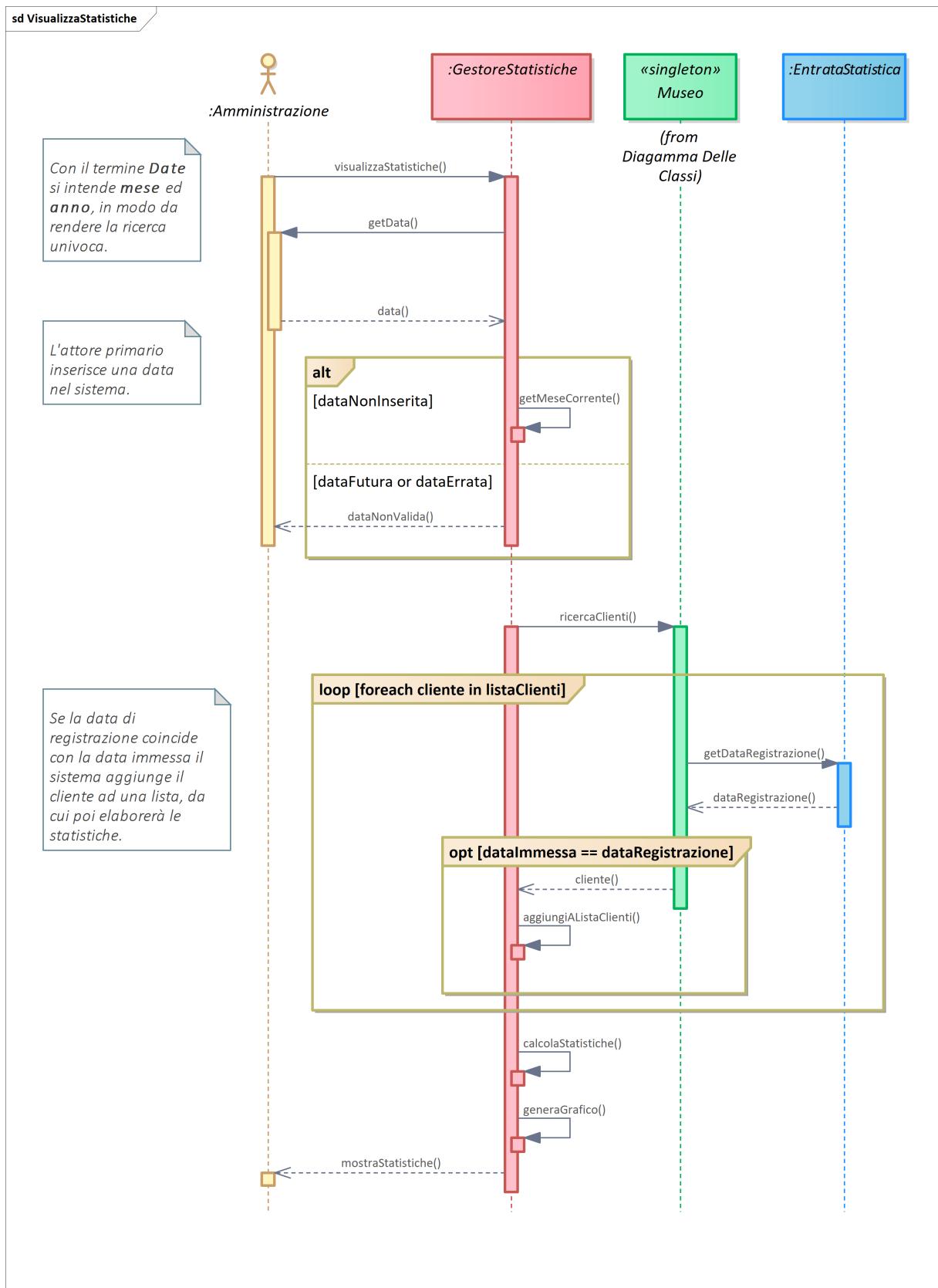


Figura 36: VisualizzaStatistiche

7 Diagrammi di attività (analisi)

I **diagrammi di attività** qui proposti ricalcano i comportamenti già descritti dai diagrammi di sequenza, ma questa volta usando un **approccio differente alla base**; non sono più i messaggi scambiati tra le parti i soggetti del diagramma, bensì le azioni "atomiche" nelle quali vengono scomposti gli algoritmi descritti.

Si noti come qui, come fatto nei diagrammi di sequenza, è stata scelta una legenda che contraddistingue ciascun elemento di modellazione del diagramma con un **colore**, in modo da rendere il più facile e piacevole possibile la lettura da parte del lettore.

In questa fase di analisi, è stato realizzato un diagramma di attività per ogni diagramma di sequenza già visto, **in modo da fornire due visioni dell'aspetto comportamentale di ciascun caso d'uso preso in esame**.

I diagrammi di attività che seguono sono riassumibili nei seguenti punti:

- **Gestione Amministrazione**

- `CompraOpera` (CU 20)
- `GeneraReportIncassi` (CU 23)
- `GestisciDipendenti` (CU 36)

- **Gestione Cliente**

- `CompraBiglietto` (CU 01)
- `ConvalidaBiglietto` (CU 06)
- `DonaOpera` (CU 10)
- `RicercaBiglietto` (CU 07)
- `VendiOpera` (CU 08)
- `VerificaPrivilegio` (CU 04)

- **Gestione Segreteria**

- `AbbonaCliente` (CU 11)
- `CUDAbbonamento` (CU 12)
- `GestisciDonazione` (CU 15)

- **Gestione Sistema**

- `ControllaAbbonamento` (CU 24)
- `EffettuaBackup` (CU 27)
- `InviaAvviso` (CU 25)
- `VerificaAbbonamento` (CU 26)

- **Gestione Statistiche**

- `InserisciDatiClienti` (CU 28)
- `VisualizzaStatistiche` (CU 32)

7.1 Gestione Amministrazione

CompraOpera

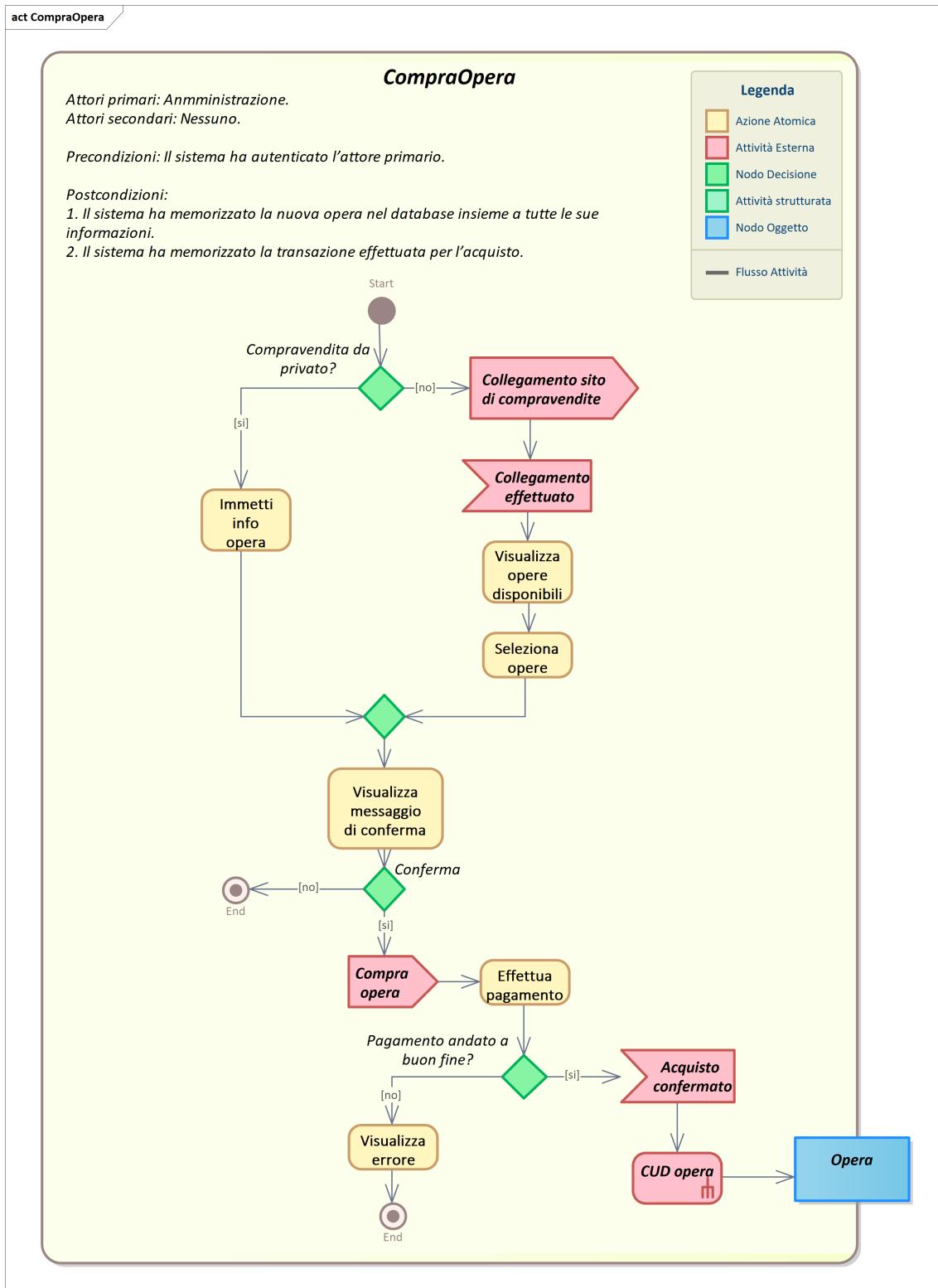


Figura 37: CompraOpera

GeneraReportIncassi

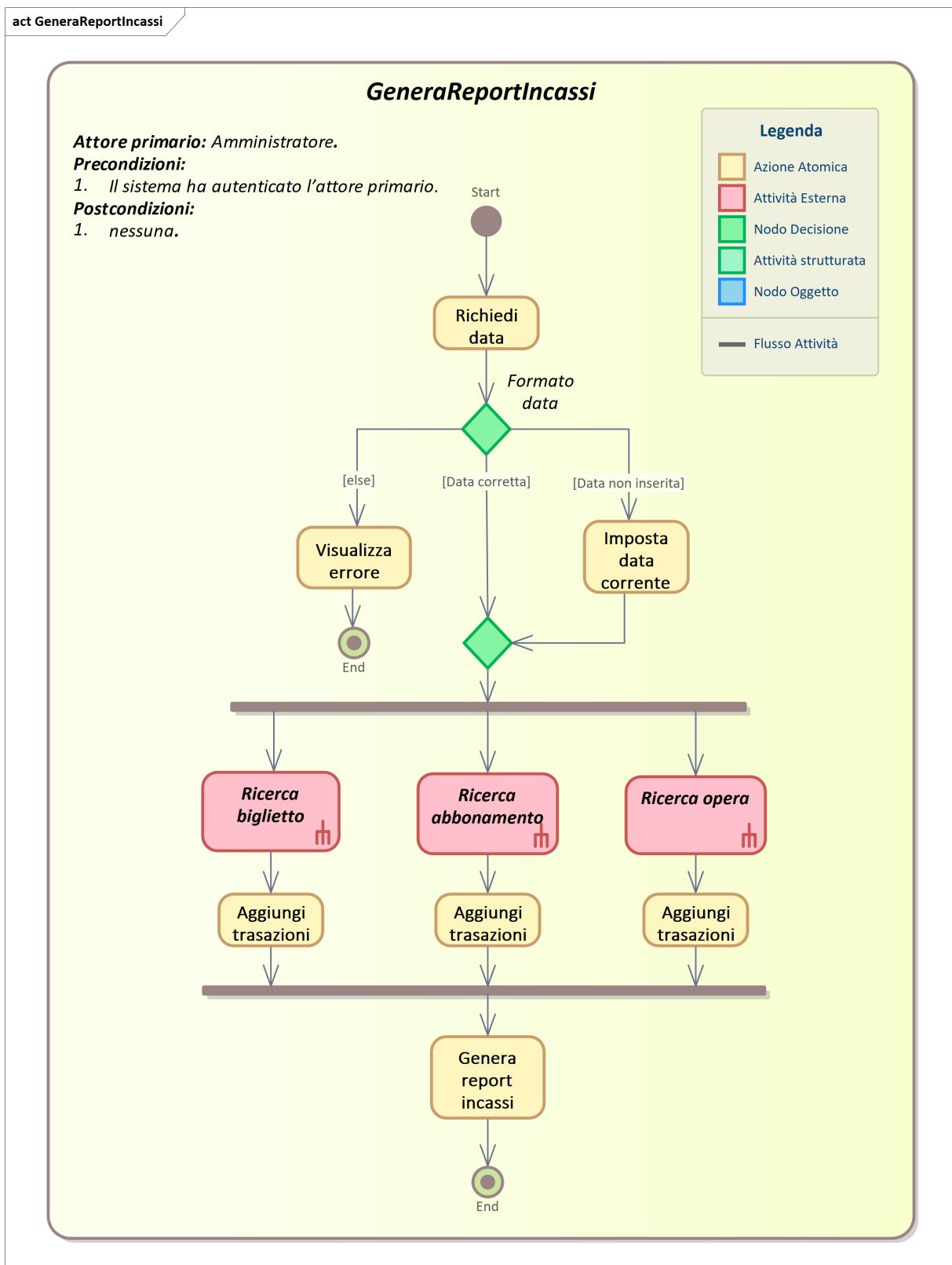


Figura 38: GeneraReportIncassi

GestisciDipendenti

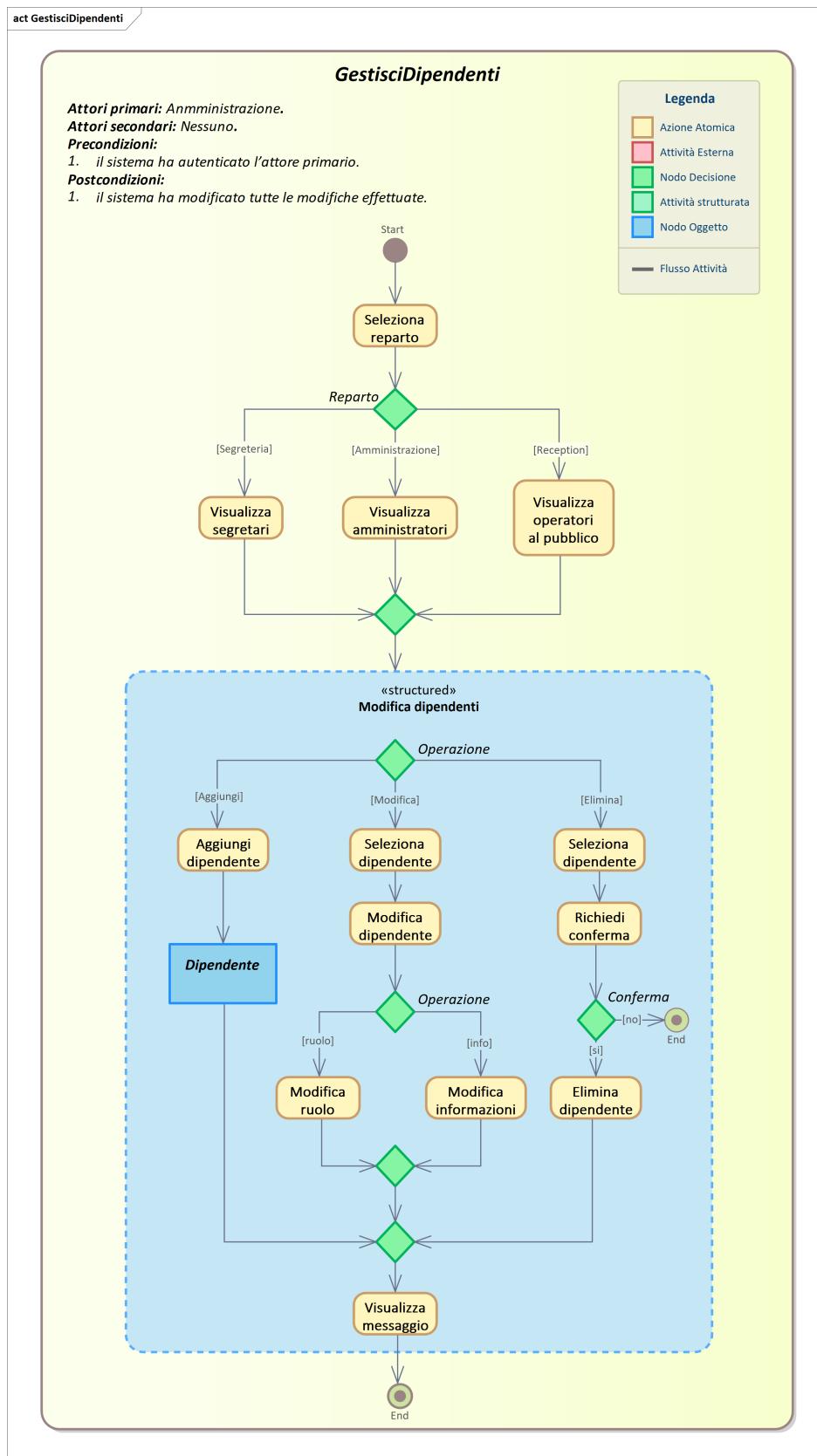


Figura 39: GestisciDipendenti

7.2 Gestione Cliente

CompraBiglietto 1/2

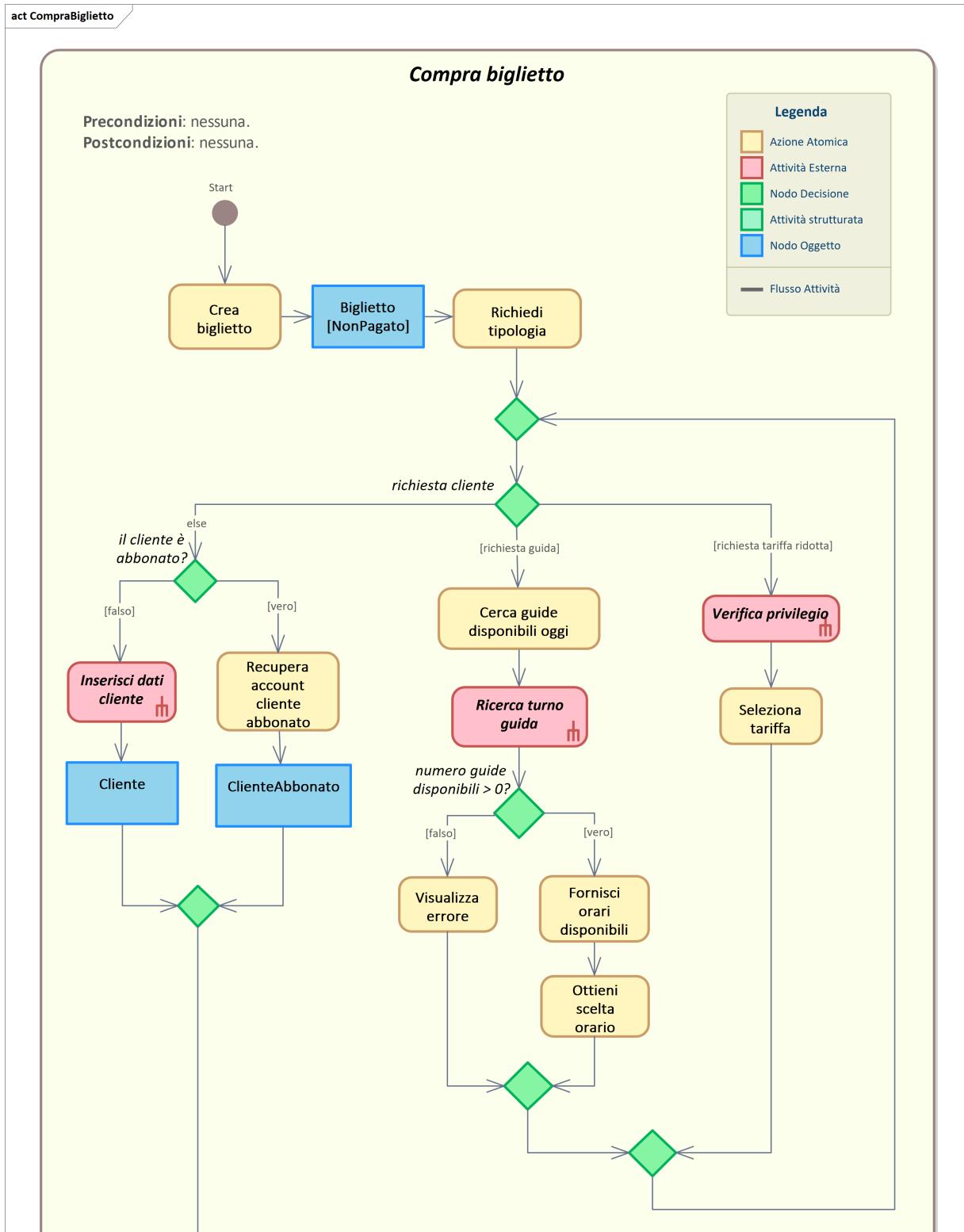


Figura 40: CompraBiglietto 1/2

CompraBiglietto 2/2

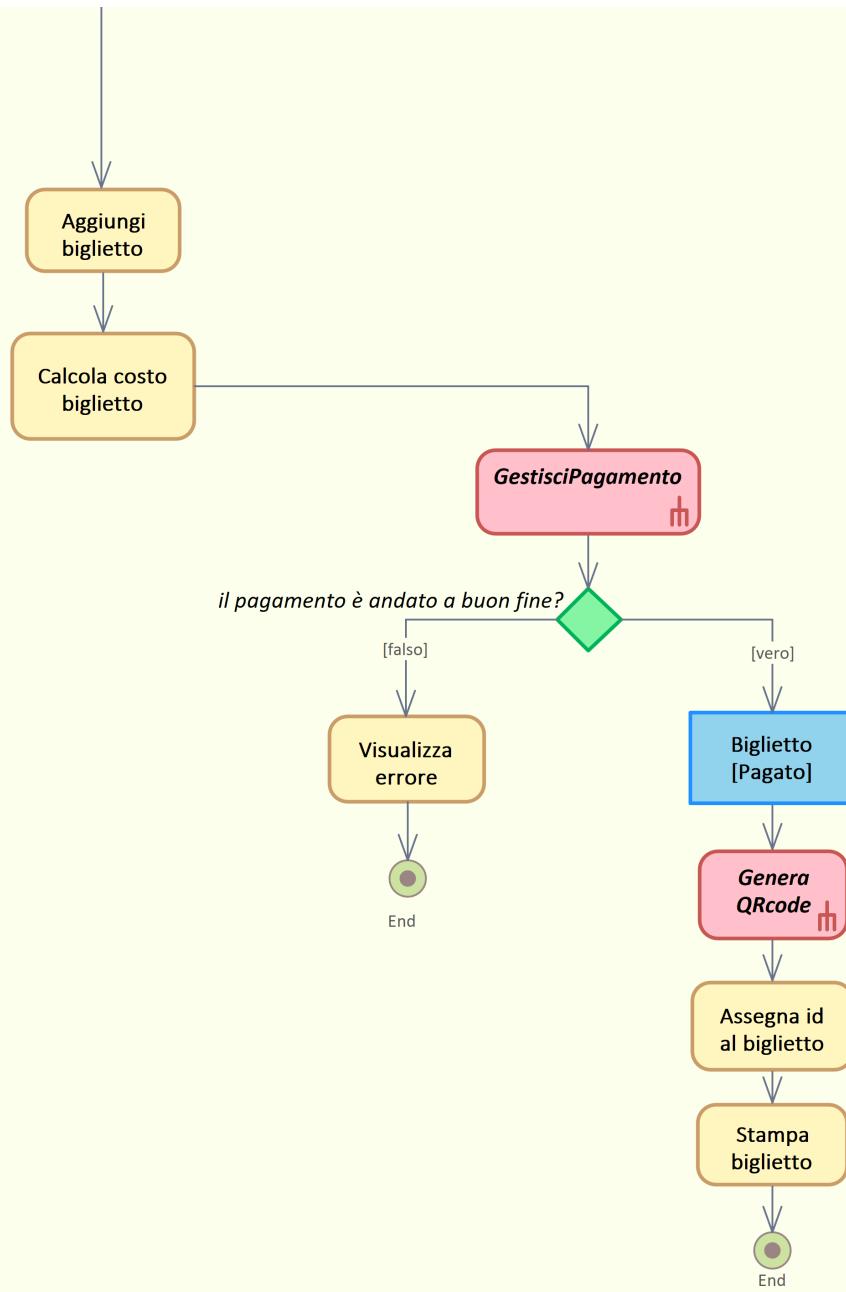


Figura 41: CompraBiglietto 2/2

ConvalidaBiglietto 1/2

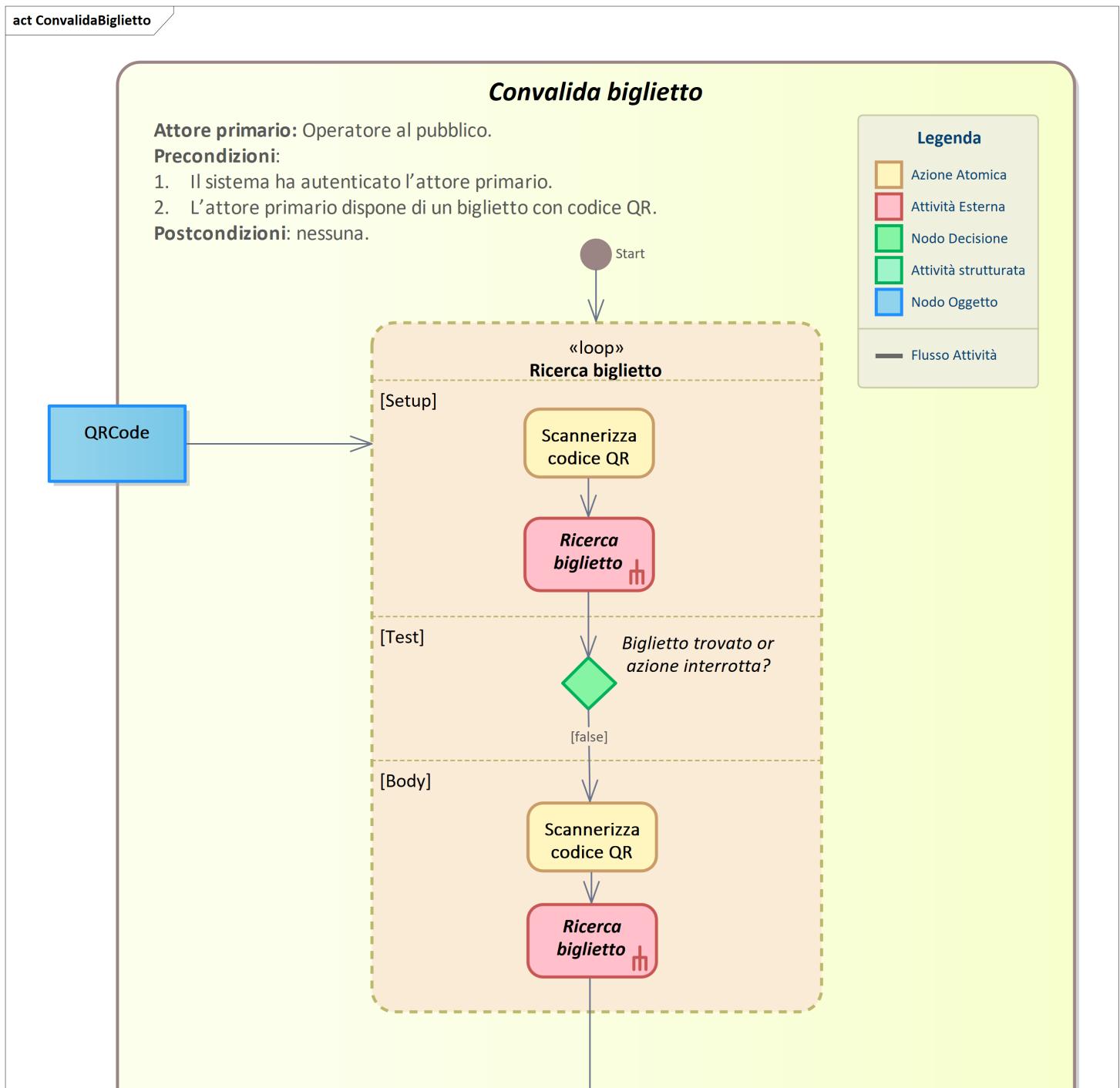


Figura 42: ConvalidaBiglietto 1/2

ConvalidaBiglietto 2/2

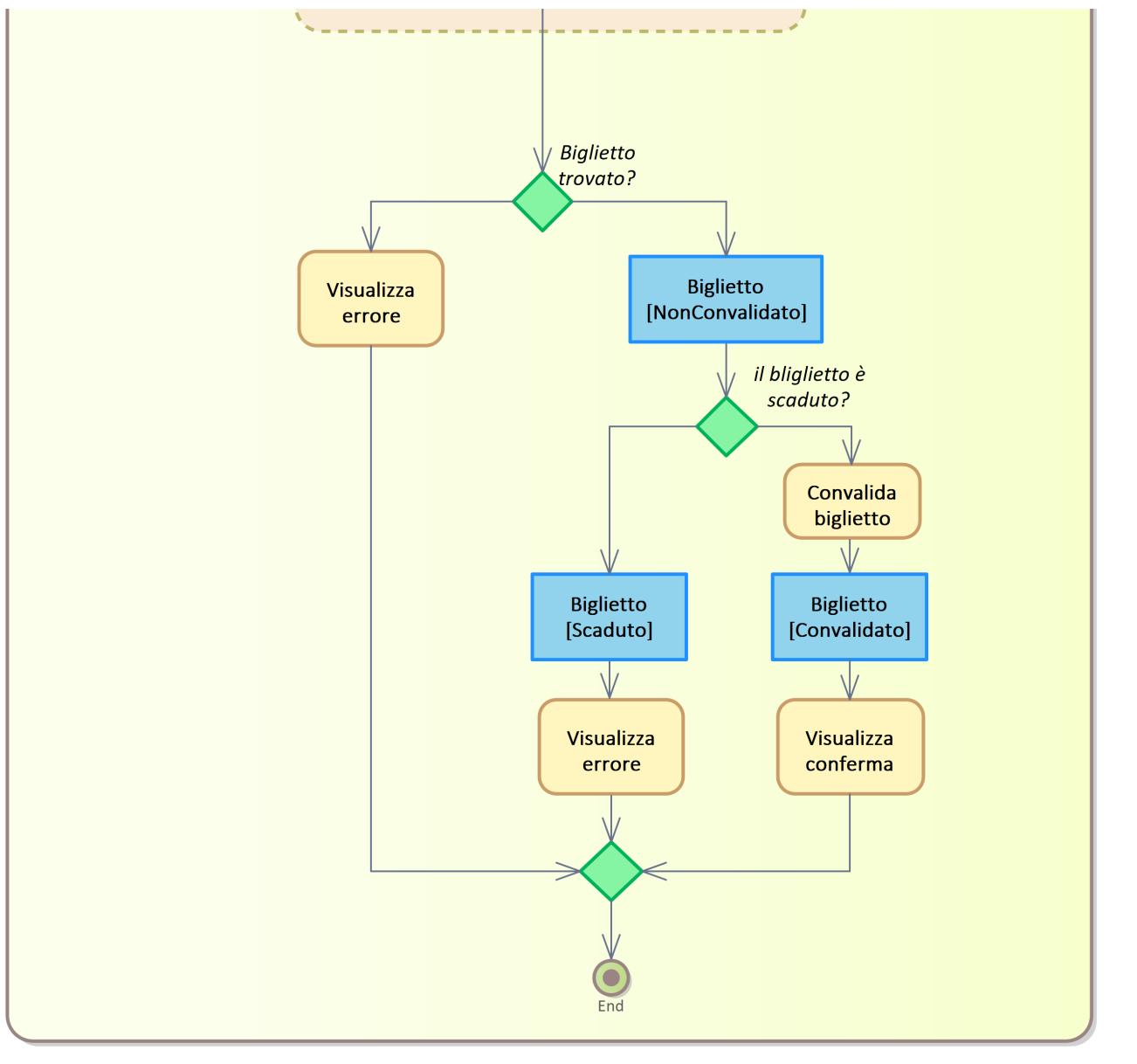


Figura 43: ConvalidaBiglietto 2/2

DonaOpera

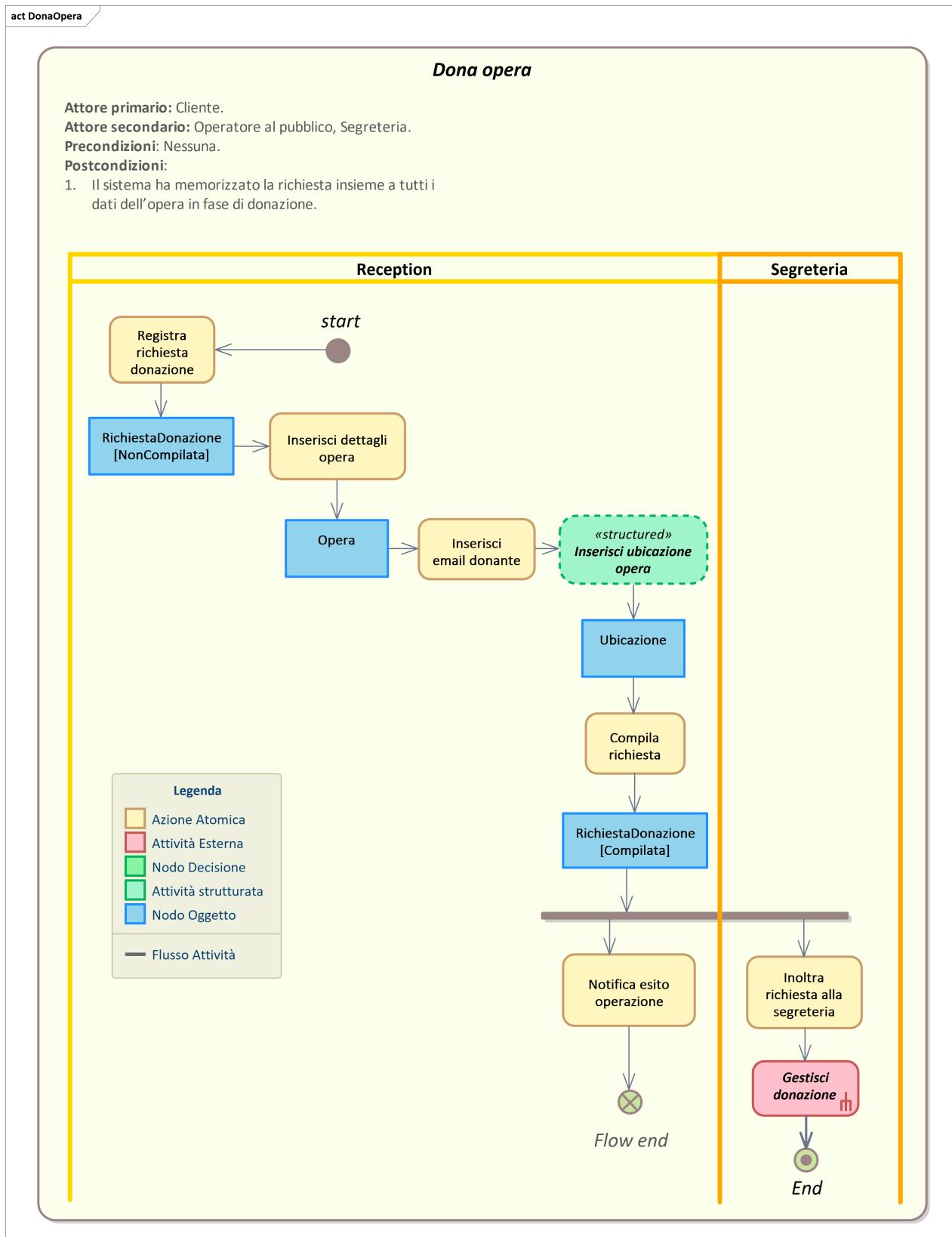


Figura 44: DonaOpera

RicercaBiglietto 1/2

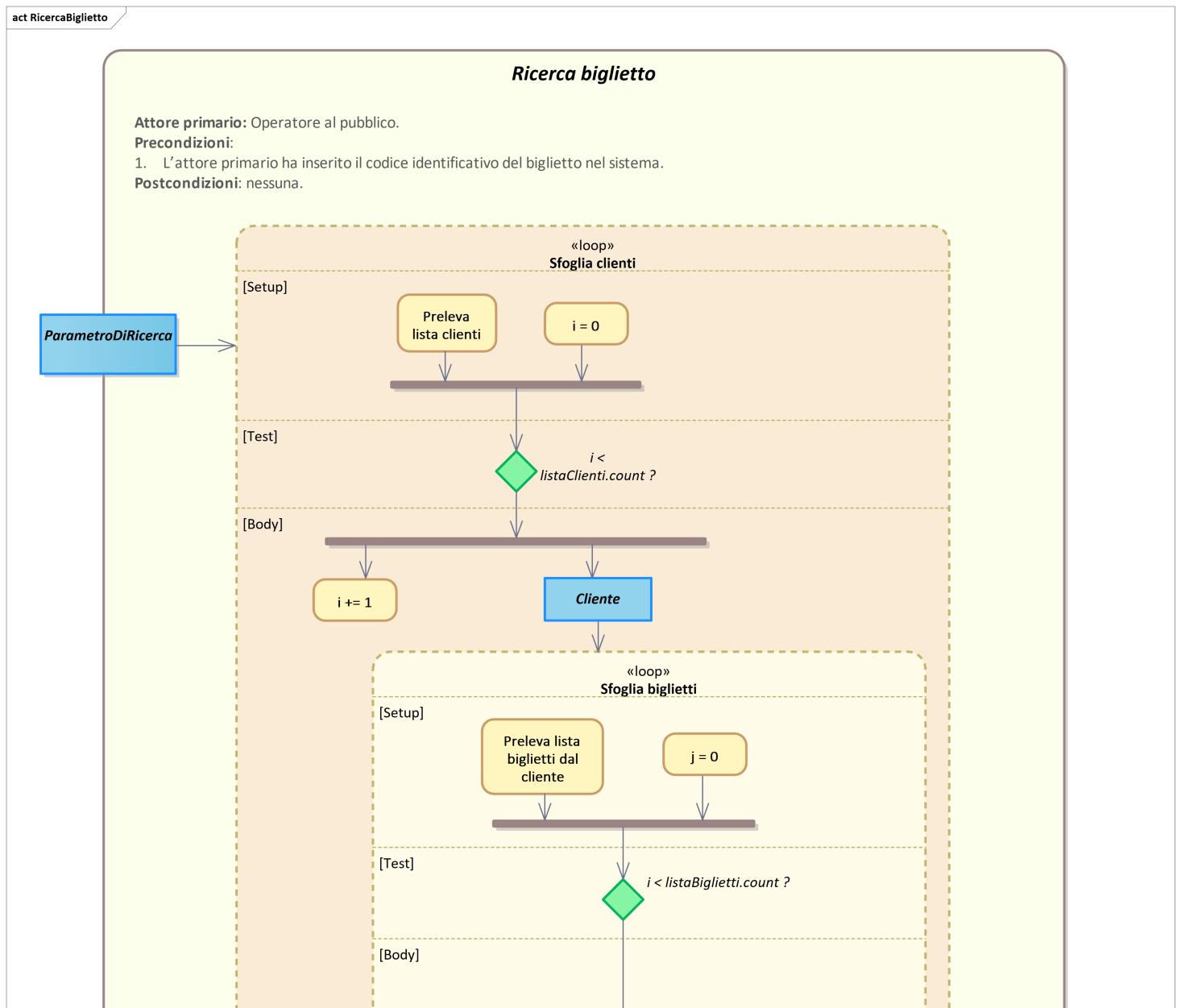


Figura 45: RicercaBiglietto 1/2

RicercaBiglietto 2/2

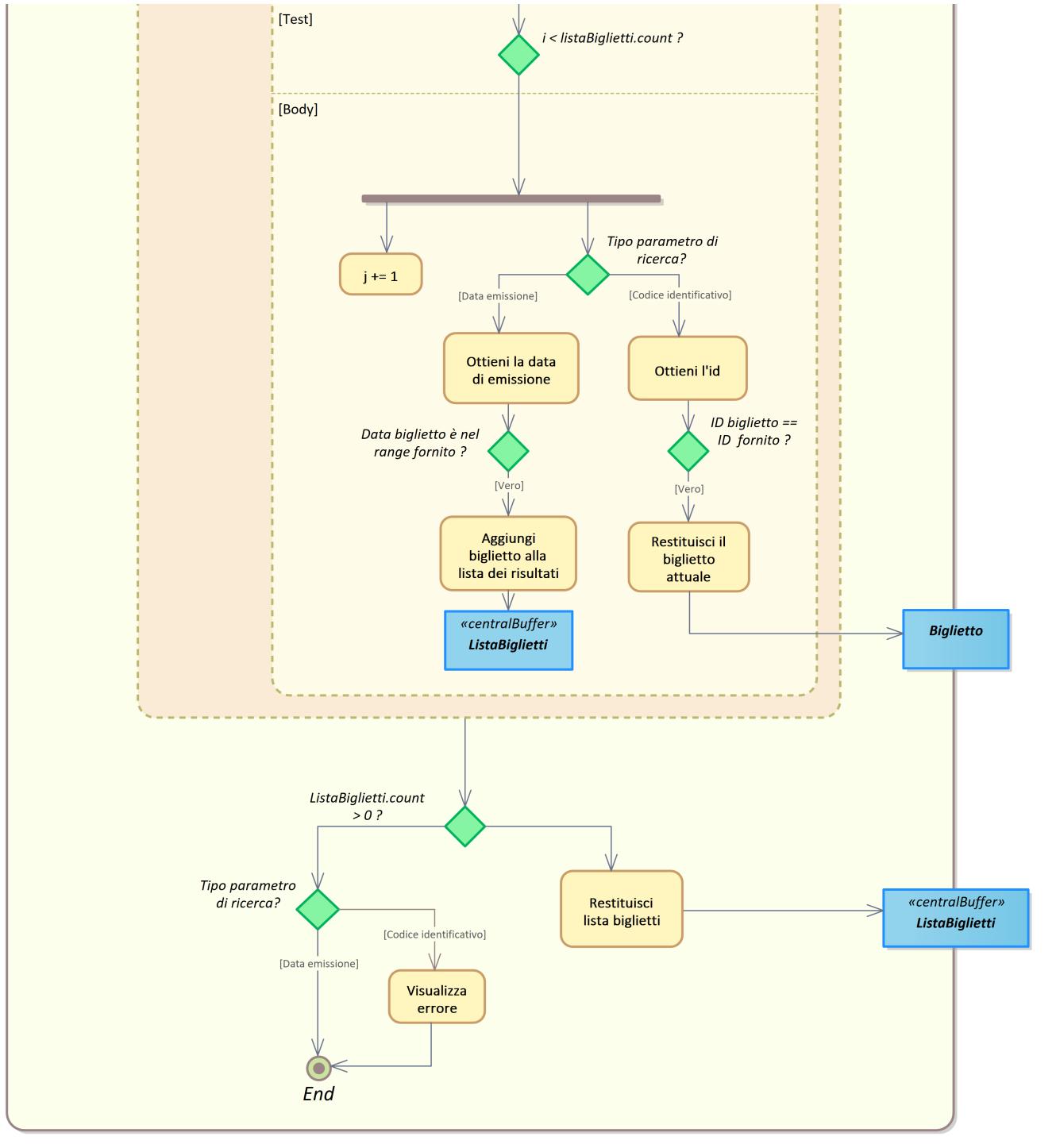
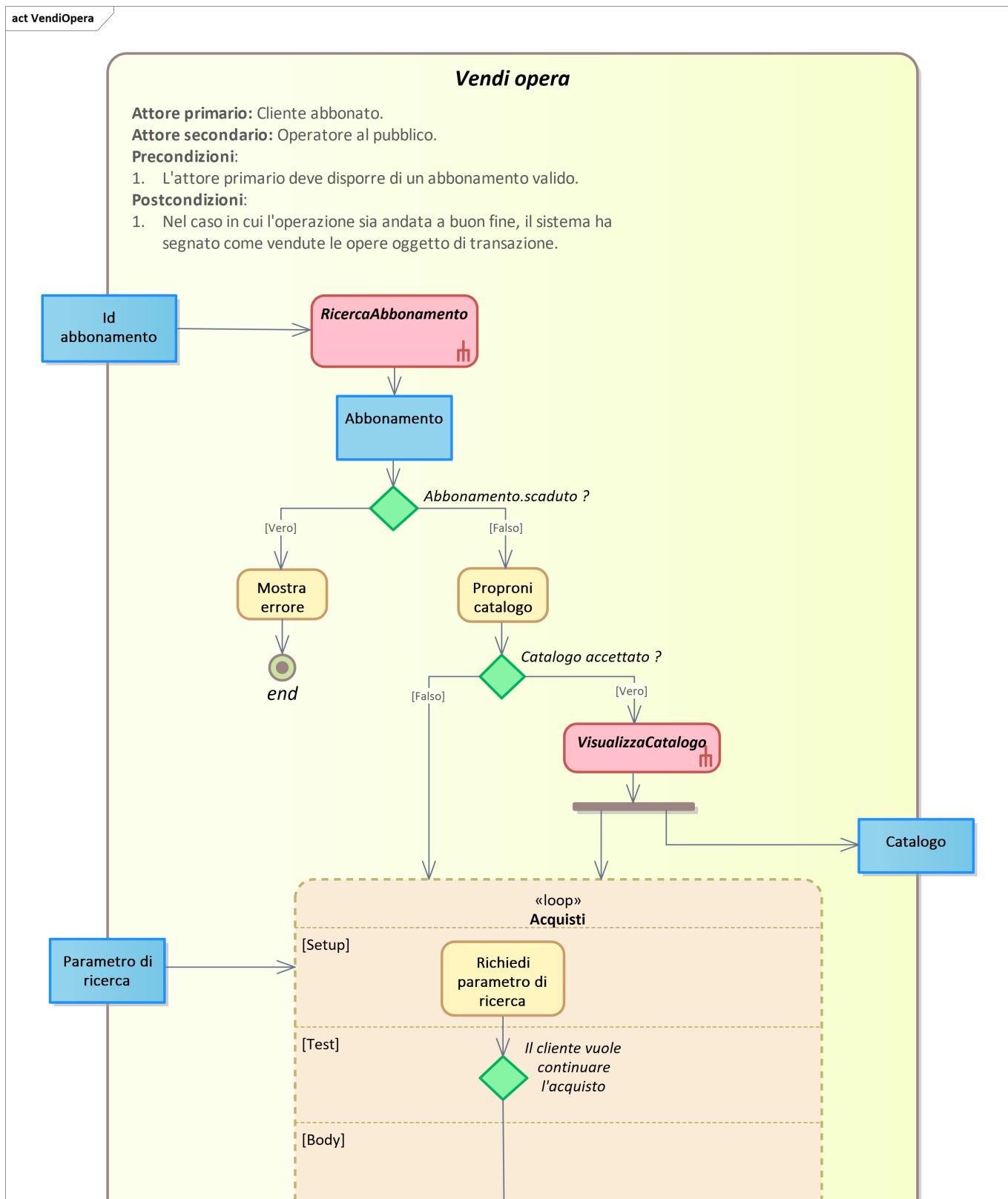


Figura 46: RicercaBiglietto 2/2

VendiOpera 1/2



VendiOpera 2/2

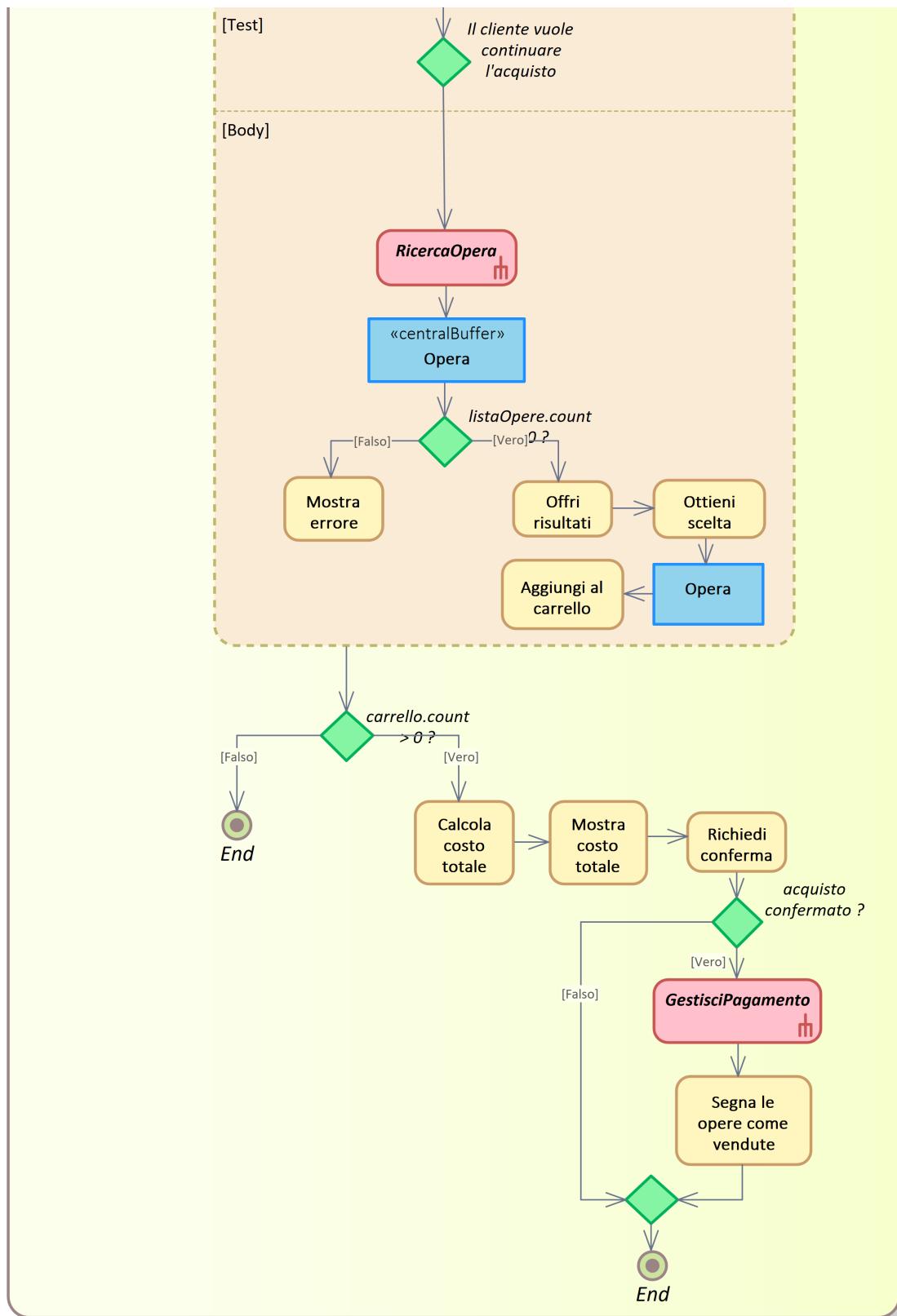


Figura 48: VendiOpera 2/2

VerificaPrivilegio

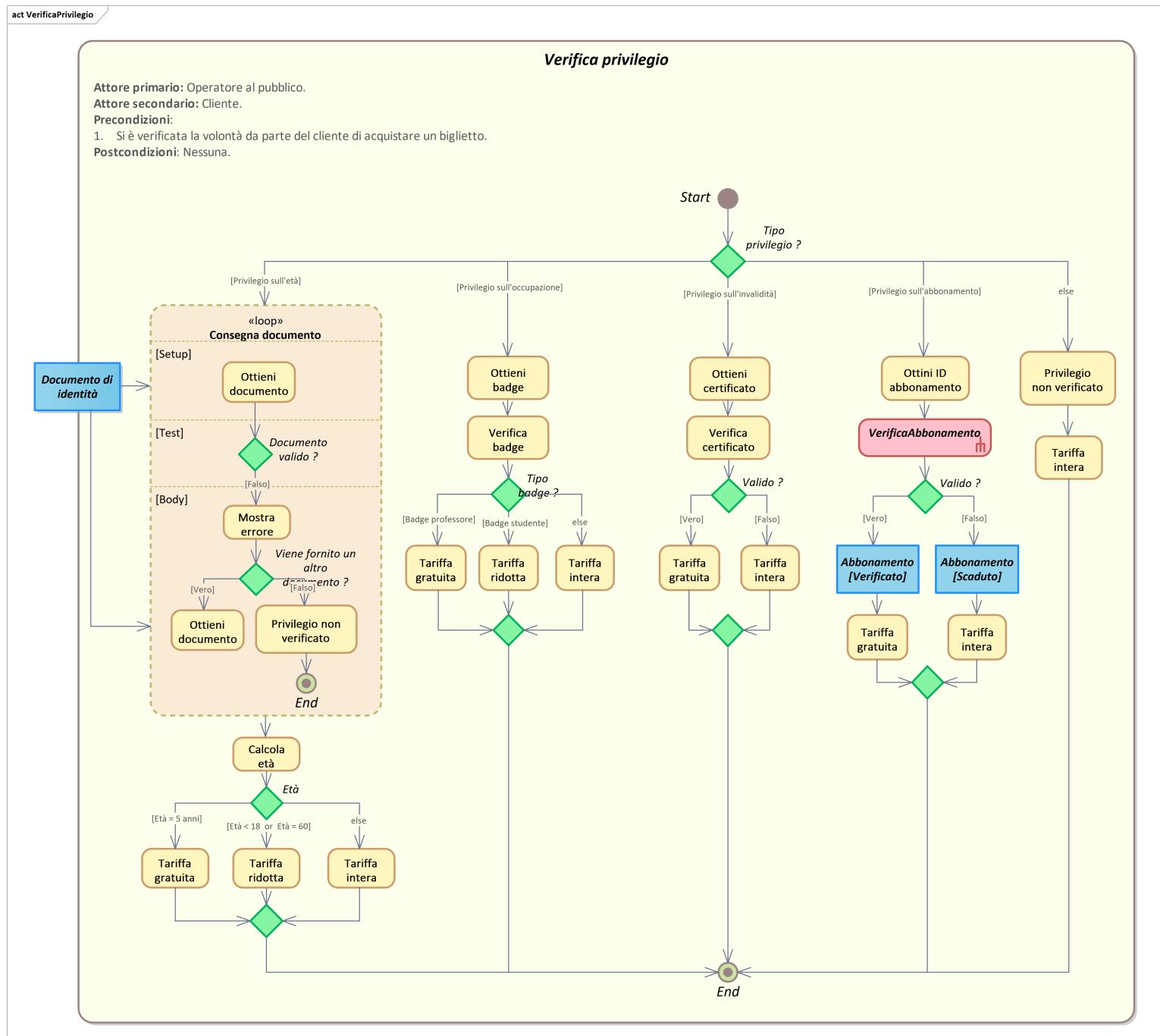


Figura 49: VerificaPrivilegio

7.3 Gestione Segreteria

AbbonaCliente 1/2

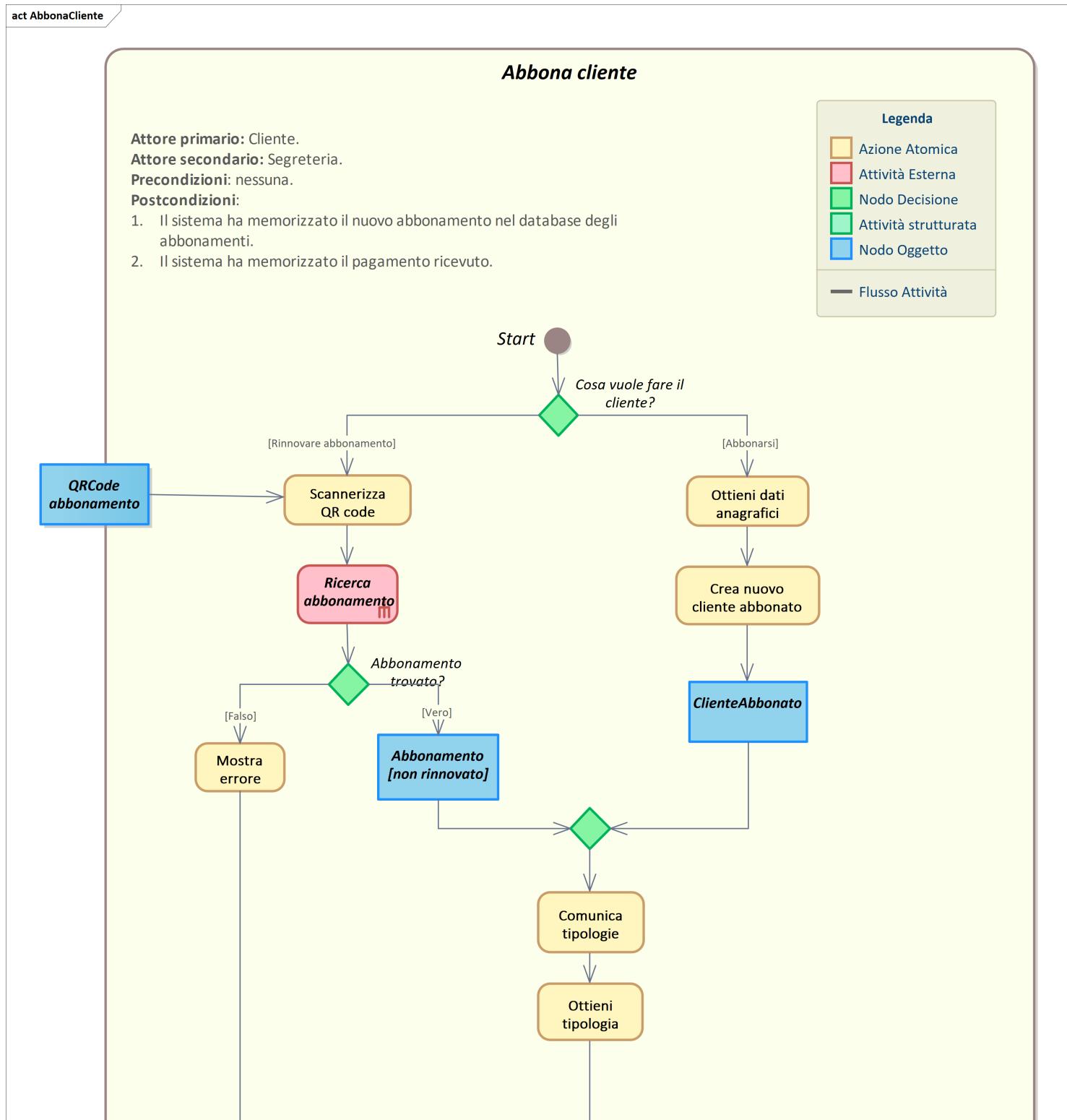


Figura 50: AbbonaCliente 1/2

AbbonaCliente 2/2

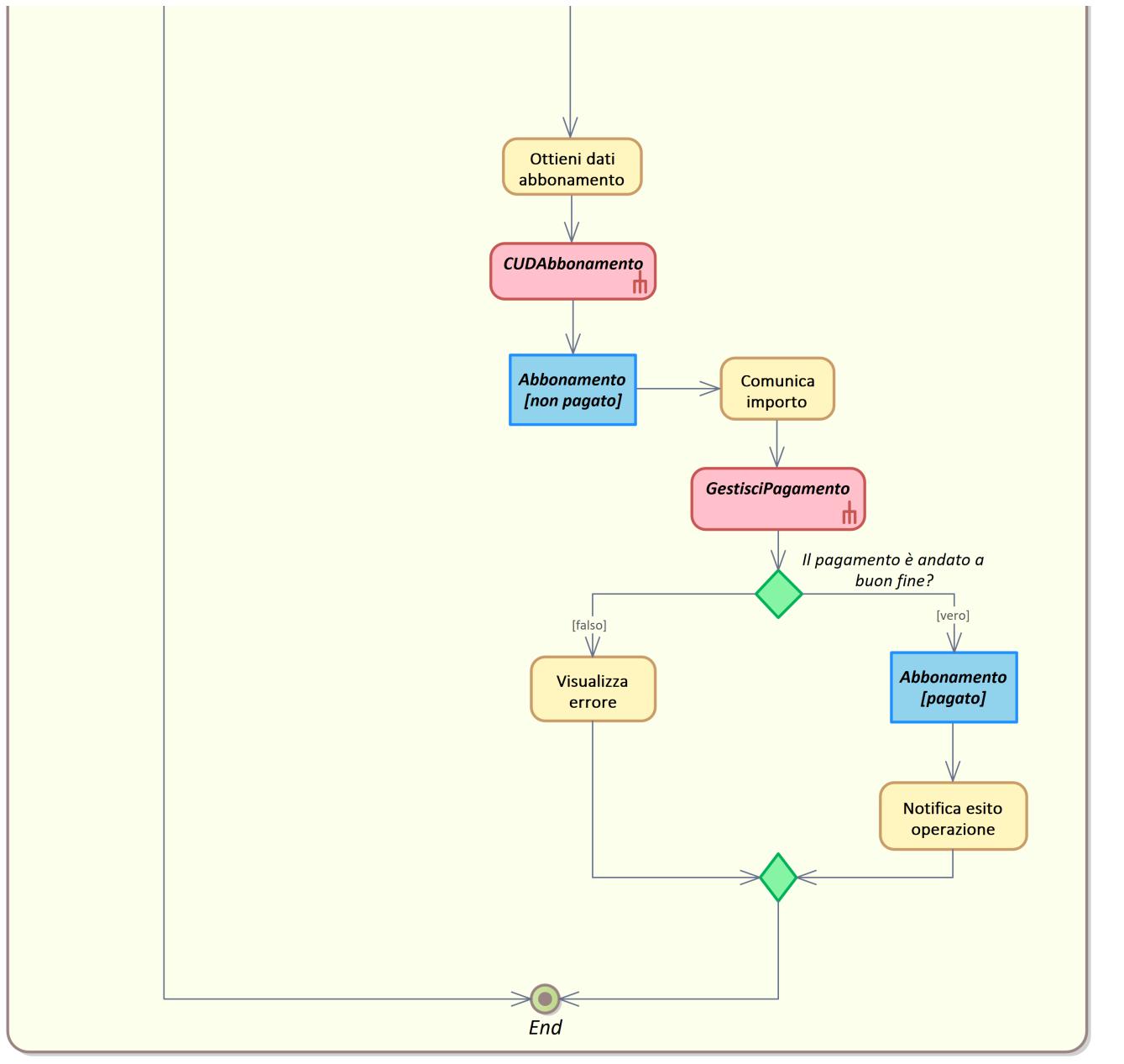


Figura 51: AbbonaCliente 2/2

CUDAbbonamento

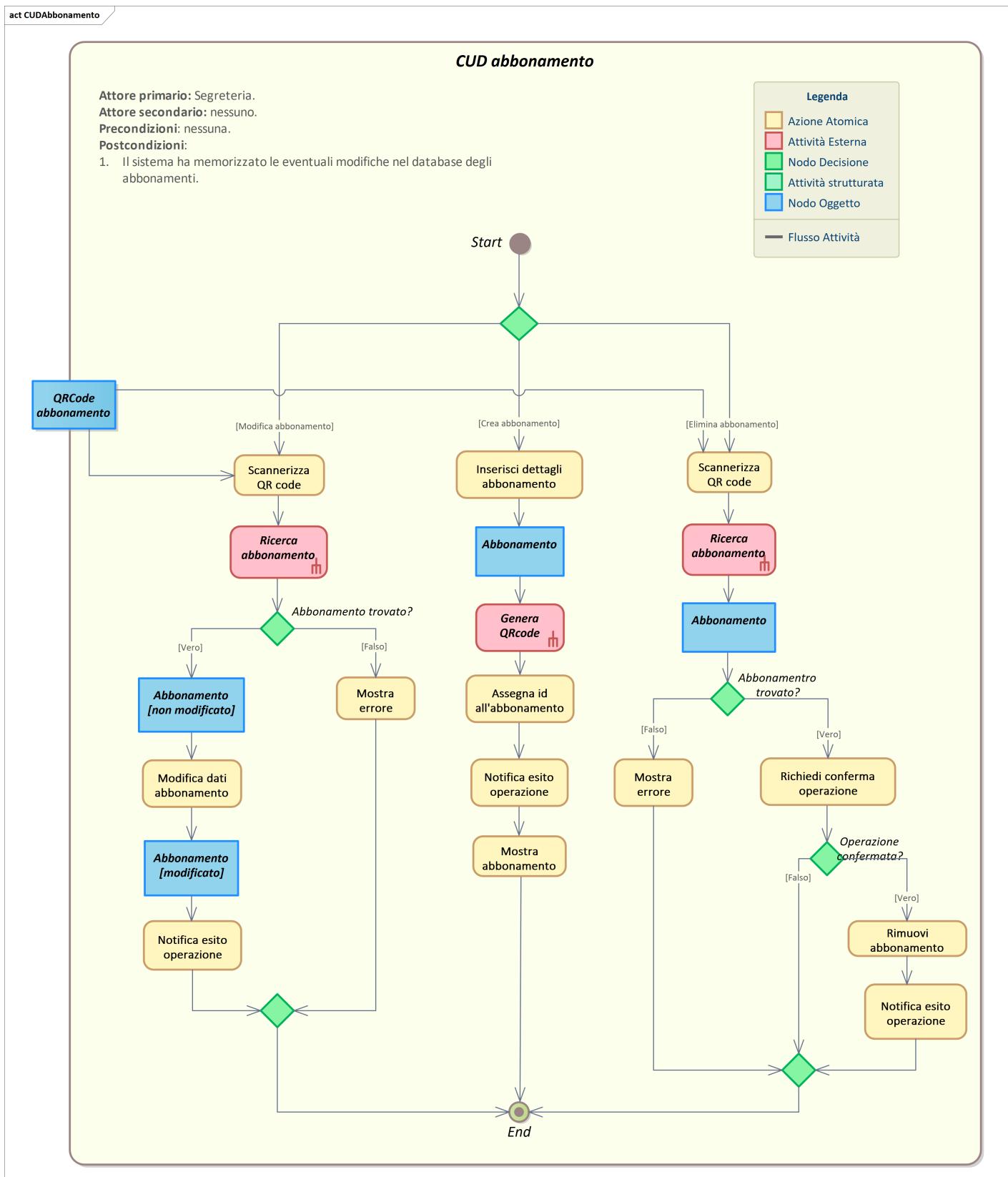


Figura 52: CUDAbbonamento

GestisciDonazione

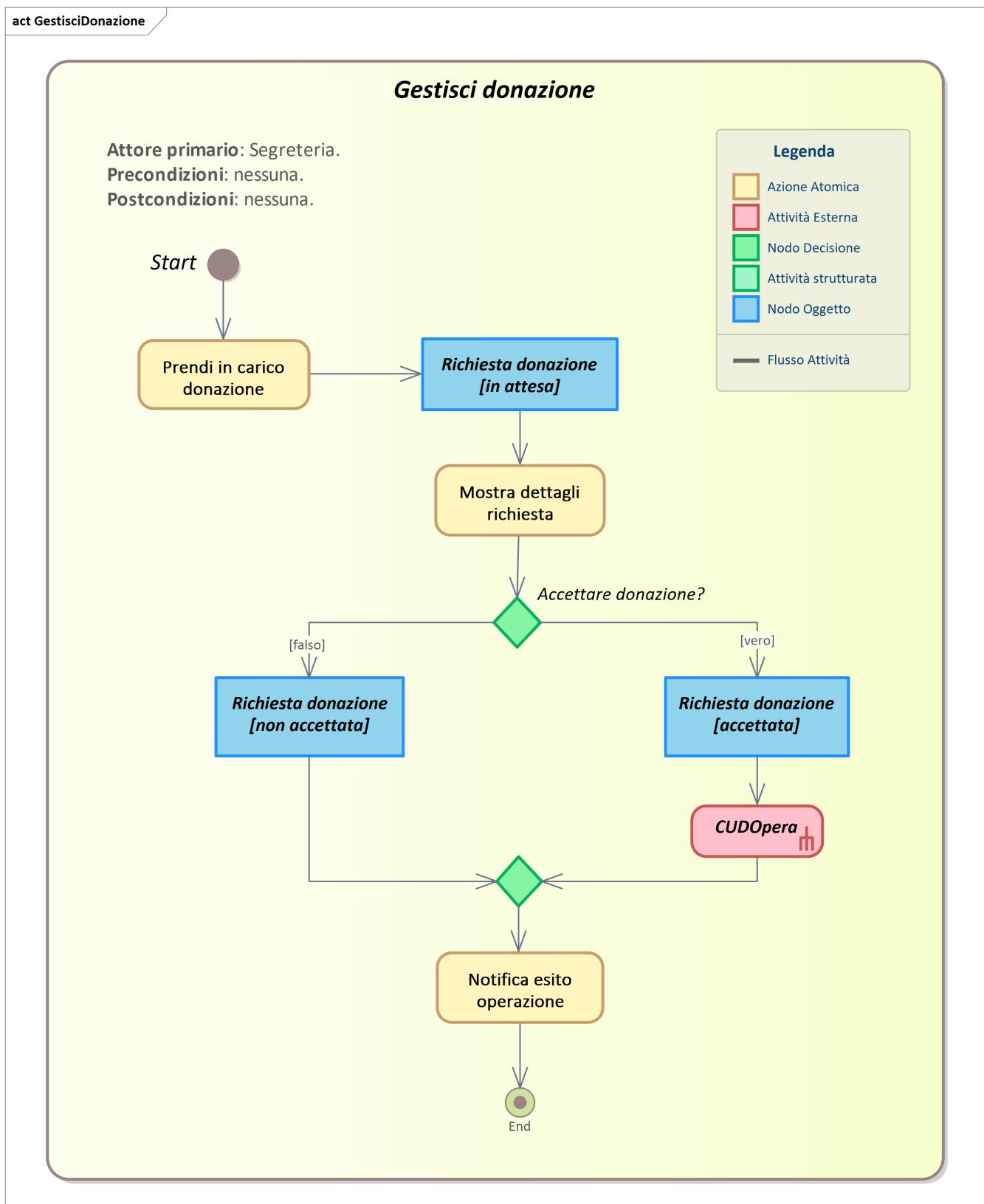


Figura 53: GestisciDonazione

7.4 Gestione Sistema

ControllaAbbonamento

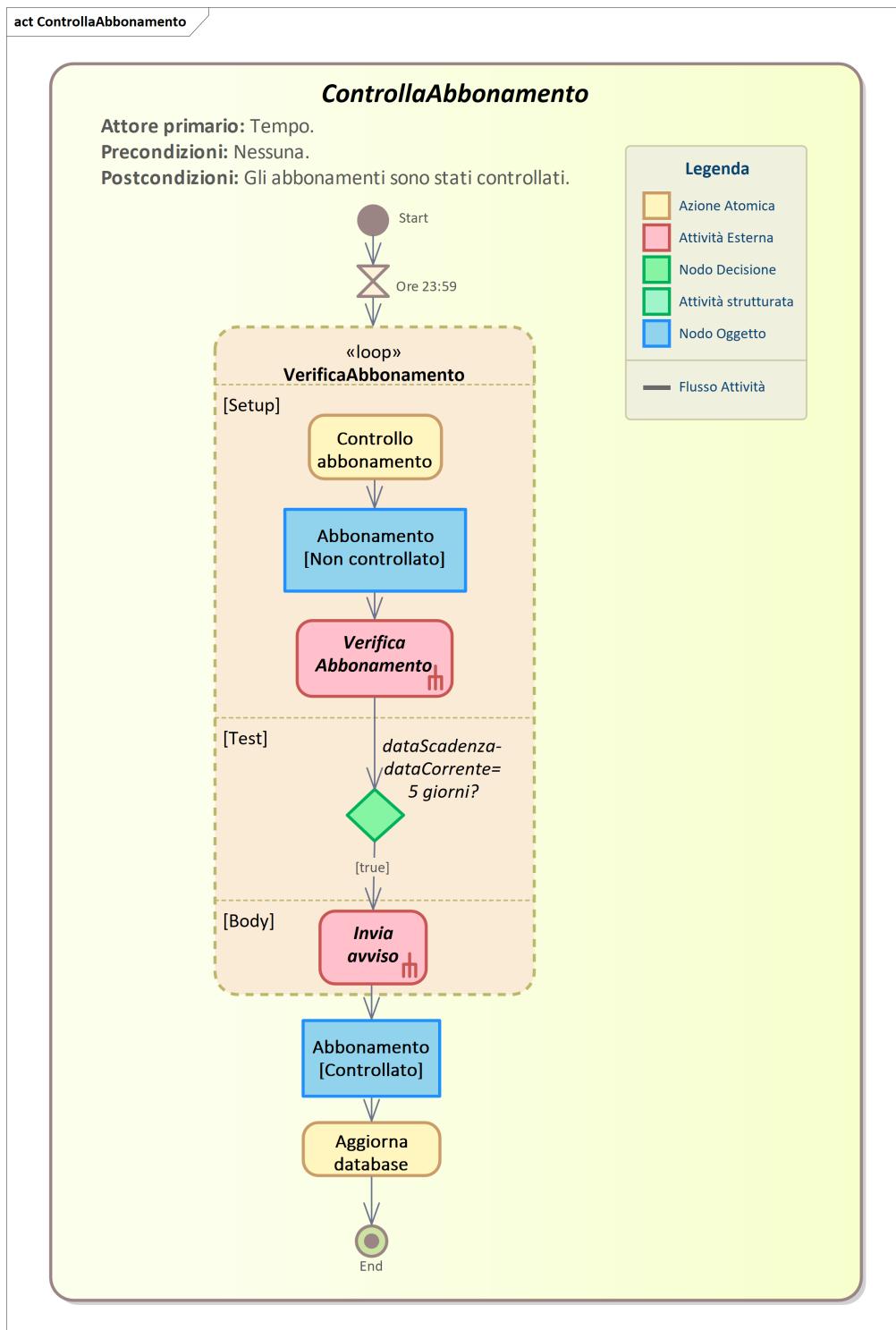


Figura 54: ControllaAbbonamento

EffettuaBackup

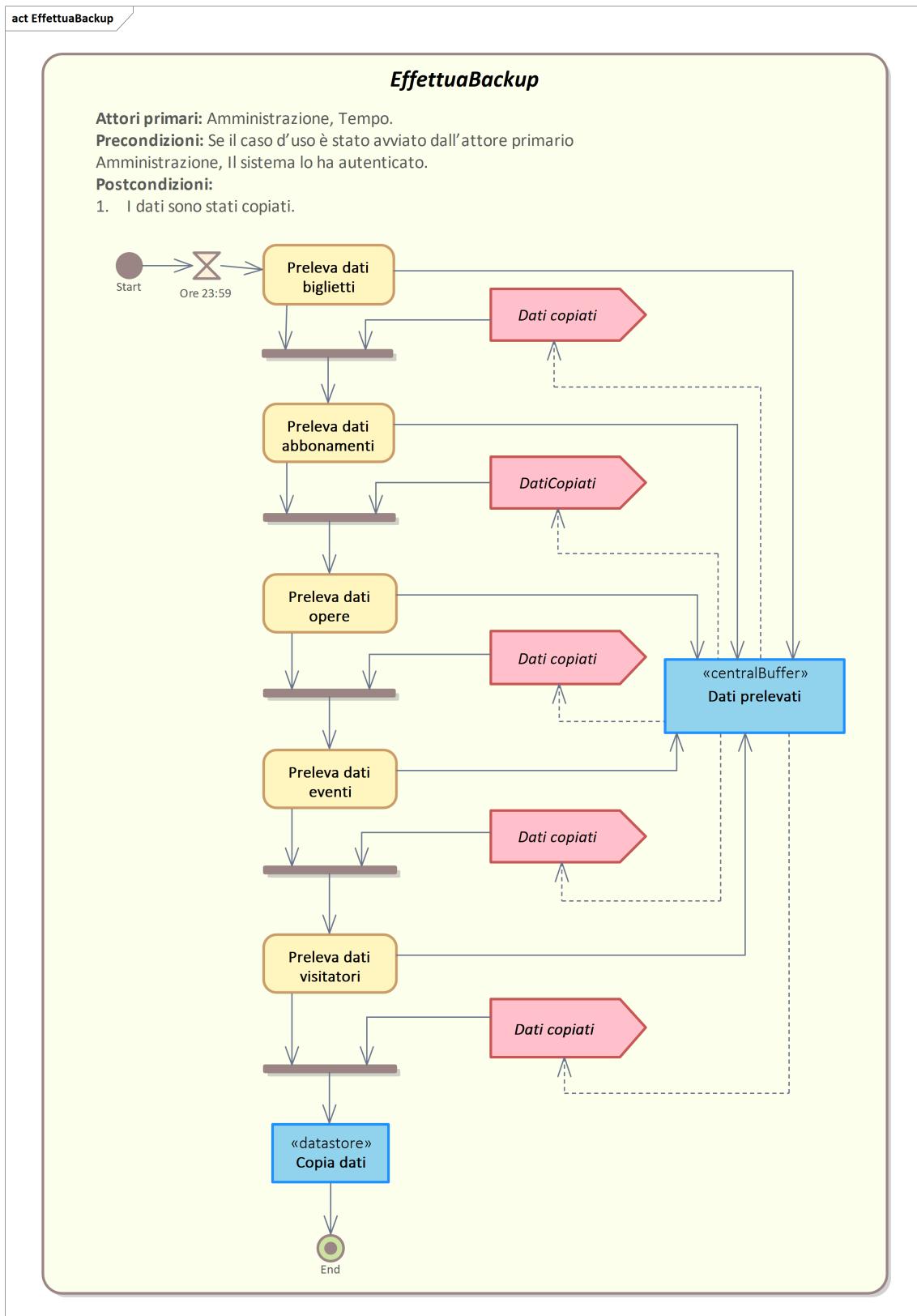


Figura 55: EffettuaBackup

InviaAvviso

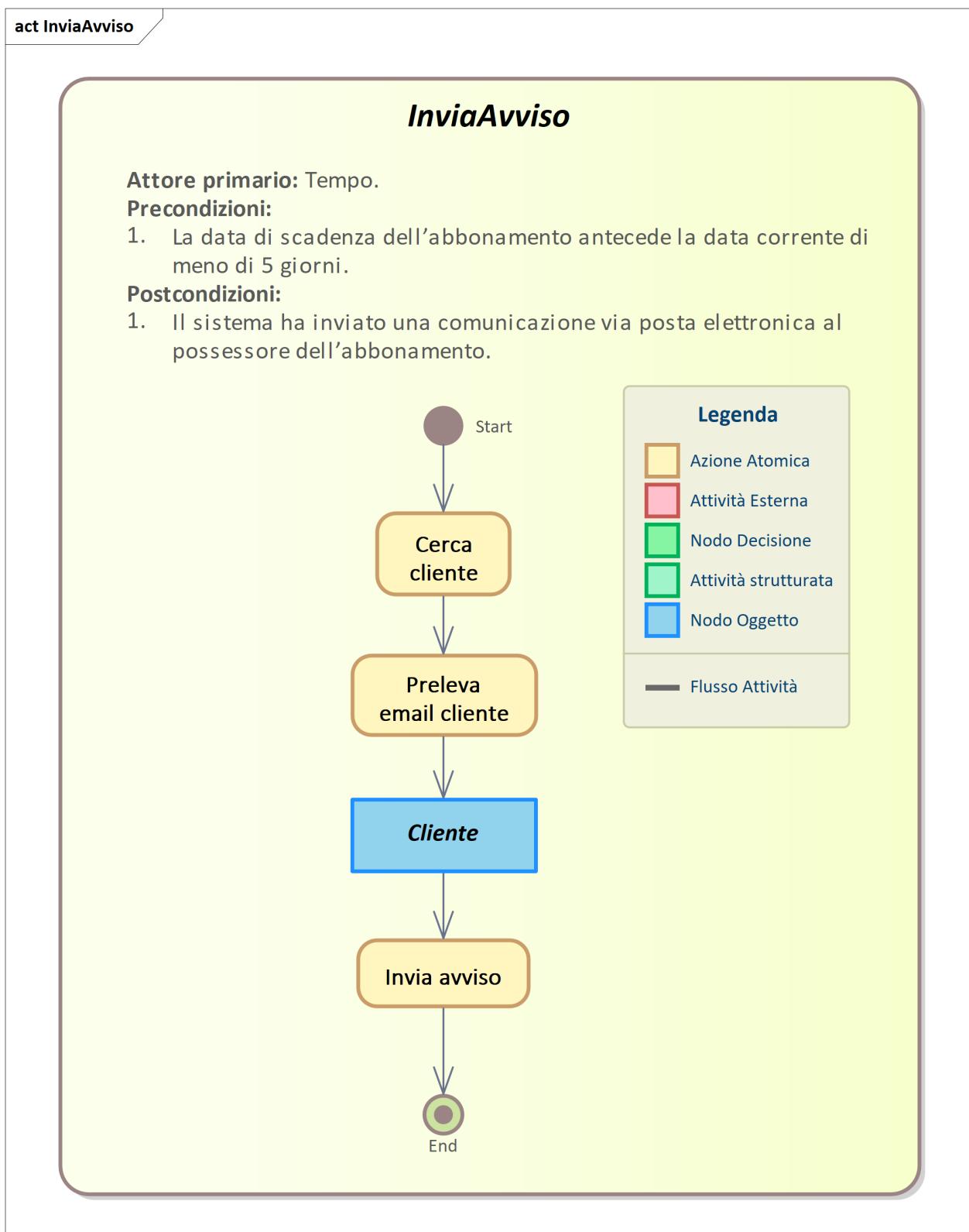


Figura 56: InviaAvviso

VerificaAbbonamento 1/2

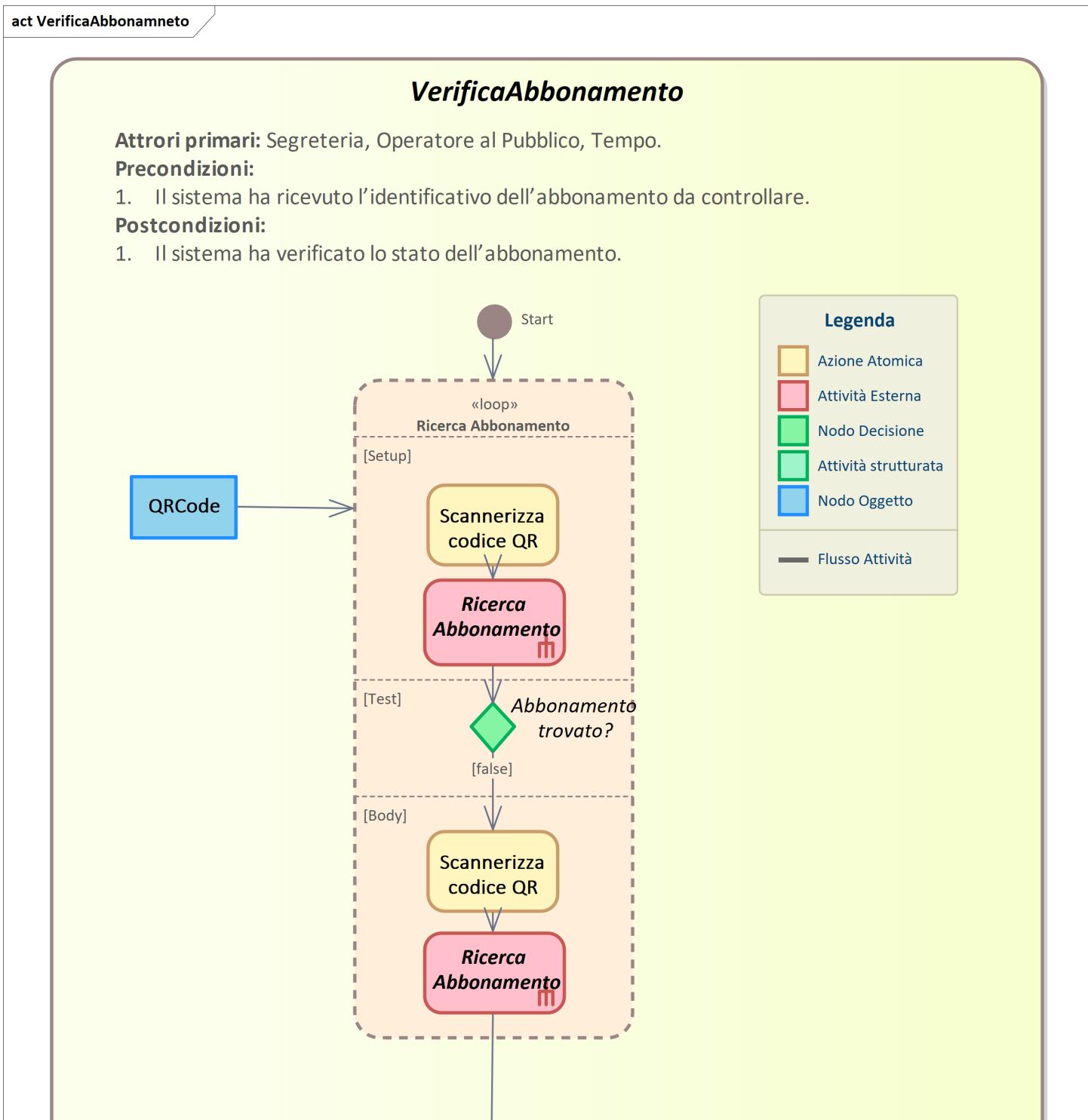


Figura 57: VerificaAbbonamento 1/2

VerificaAbbonamento 2/2

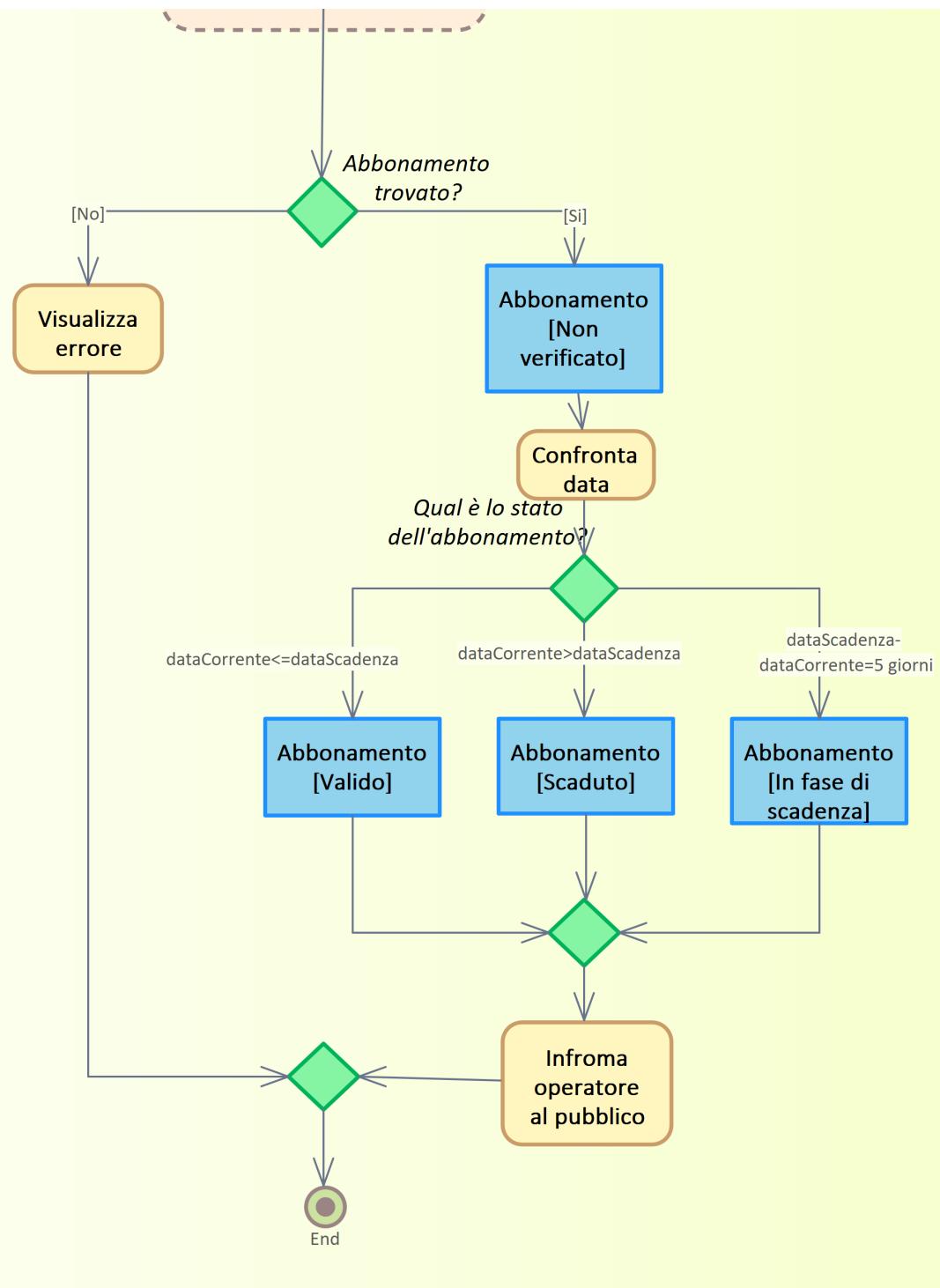


Figura 58: VerificaAbbonamento 2/2

7.5 Gestione Statistiche

InserisciDatiCliente

act InserisciDatiCliente

InserisciDatiCliente

Attori primari: Operatore al pubblico.

Attori secondari: Cliente.

Precondizioni:

1. Il sistema ha autenticato l'attore primario.
2. Nel caso in cui il cliente disponga di un abbonamento, l'attore primario ha inserito nel sistema l'identificativo dell'abbonamento eseguendo una scannerizzazione del codice QR.

Postcondizioni:

1. Il sistema ha memorizzato i dati inseriti in forma anonima.

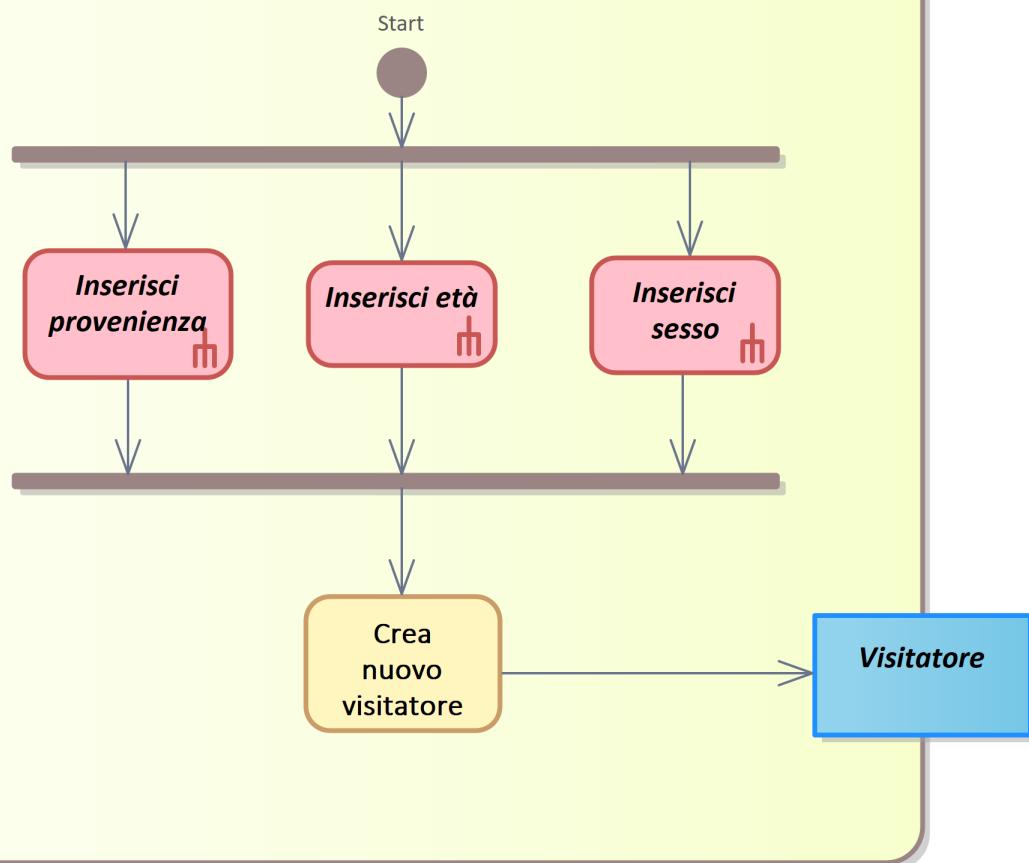


Figura 59: InserisciDatiCliente

VisualizzaStatistiche 1/2

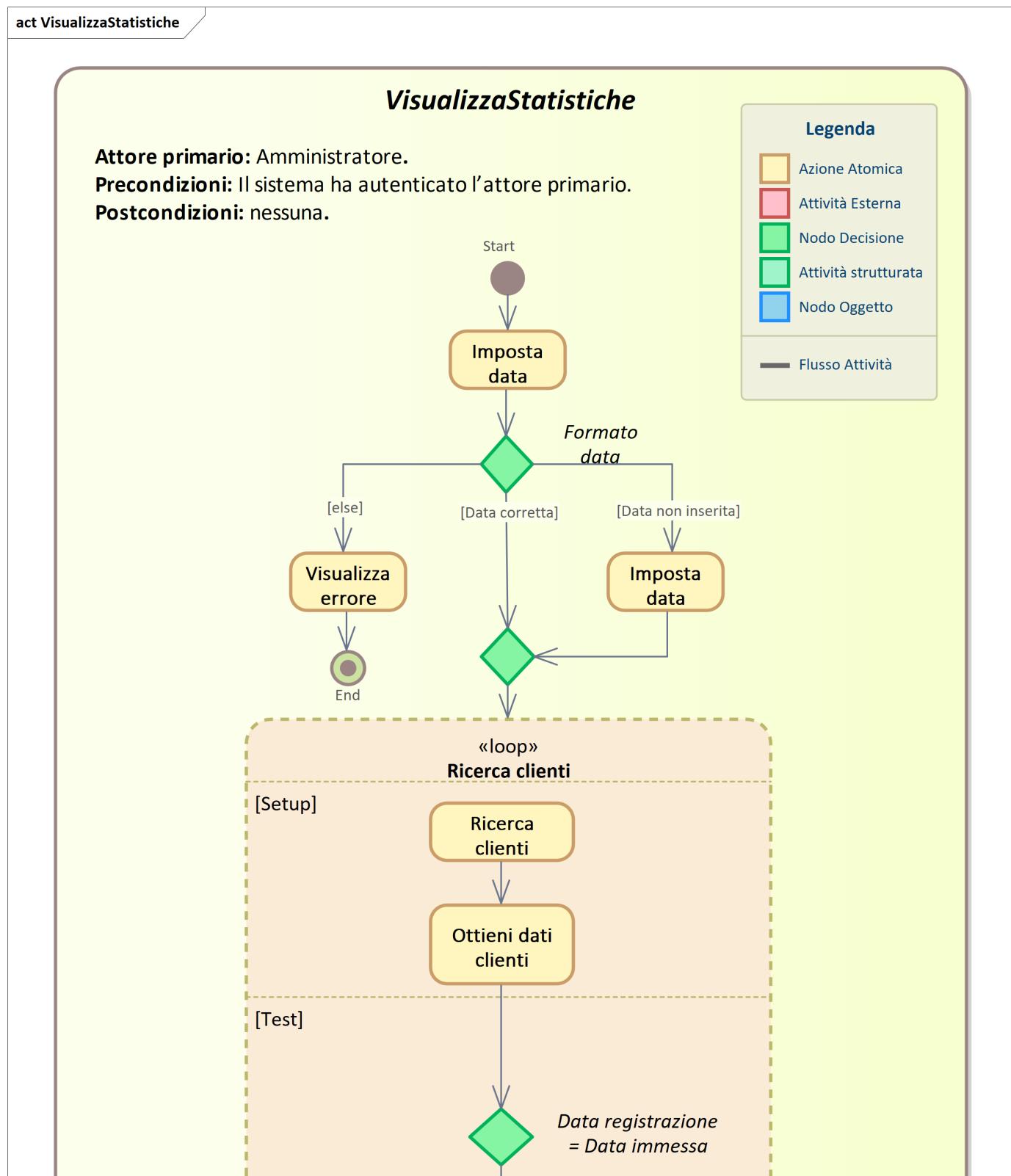


Figura 60: VisualizzaStatistiche 1/2

VisualizzaStatistiche 2/2

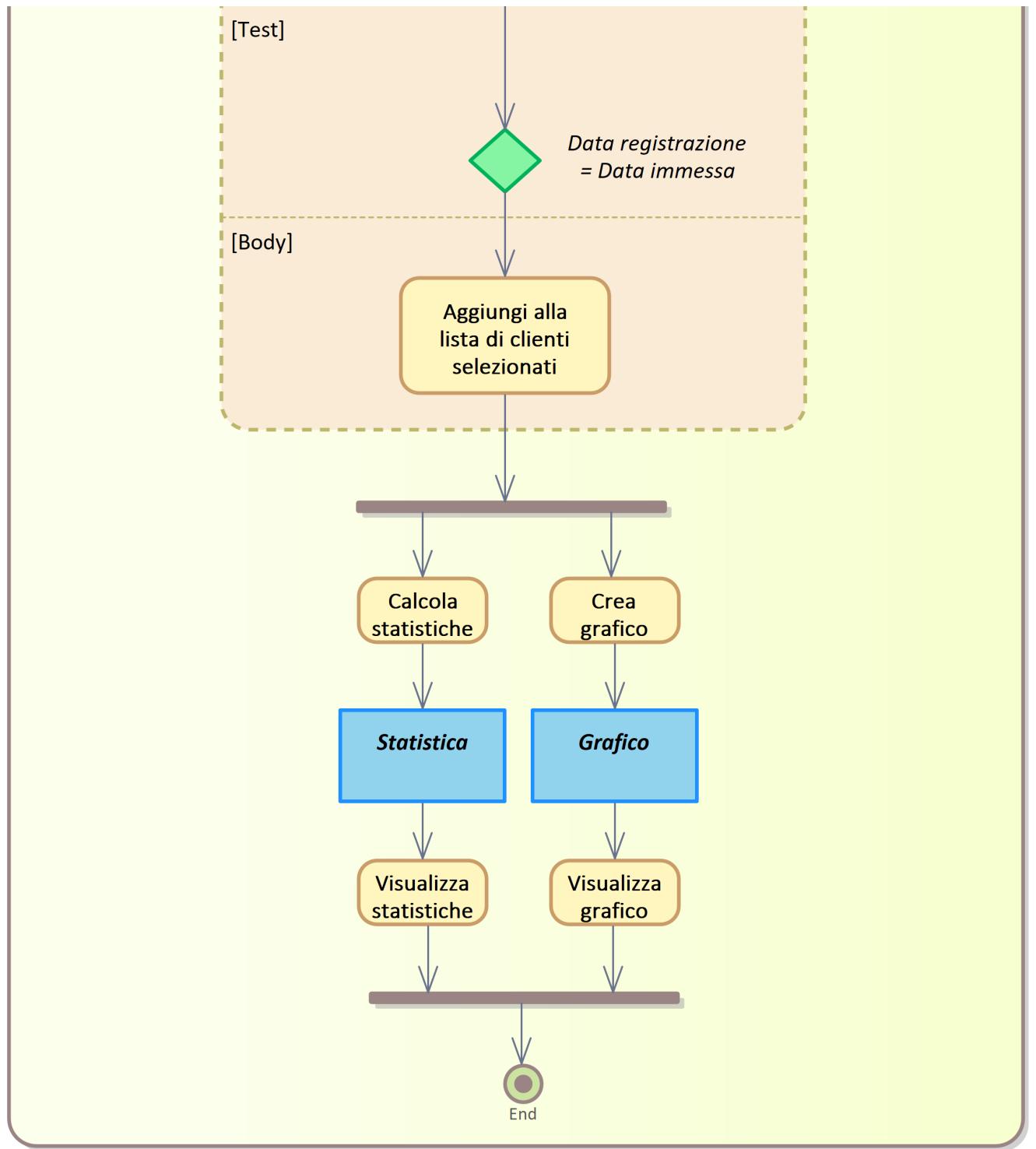


Figura 61: VisualizzaStatistiche 2/2

8 Diagrammi delle classi (progettazione)

Nella visione di questo capitolo, come del resto accadeva già nel diagramma delle classi di analisi, il lettore noterà certamente come nonostante sia noto dai requisiti non funzionali che il linguaggio di destinazione per l'implementazione è il Python, **i diagrammi mantengono una indipendenza dal linguaggio di destinazione**; lo si noterà certamente dai tipi primitivi scelti tra quelli forniti dall'UML e dall'OCL, dalla firma dei costruttori e dalla sintassi in generale diversa da quella del Python.

Mantenere un'indipendenza del progetto dal linguaggio di destinazione presenta molteplici vantaggi, primo fra tutti quello di **rendere intercambiabile la scelta della piattaforma di destinazione**.

In questa fase di progettazione sono stati utilizzati i seguenti pattern:

- **Model View Controller** (architetturale)
- **Strategy** (design)
- **Observer** (design)
- **Singleton** (design)
- **Decorator** (design)

Nel seguito tale scelte verranno giustificate.

8.1 Package delle classi

Essendo l'interfaccia grafica un punto fondamentale del presente software si è scelto di adottare il **pattern architettonico Model View Controller** che prevede la decomposizione in tre parti dell'architettura. In realtà, essendo per definizione le viste e i controller sempre in coppie, si può dire che le parti fondamentali in cui viene diviso il sistema sono due:

- **Backend**

- **Frontend.**

Mentre il primo contiene le classi del **Model**, ovvero quelle contenenti la logica di business e che giocano il ruolo di "nucleo" del software, il frontend è composto dalle viste e dai controller, ovvero gli elementi con i quali si interesseranno gli attori per inserire nuovi dati e visualizzare quelli già presenti nel model, renderizzati nelle opportune viste.

Il motivo per il quale si è scelto di utilizzare questo pattern risiede nella **flessibilità** che il sistema acquisisce; questo è reso possibile grazie alla **ripartizione delle responsabilità** resa mediante la sopravvissuta suddivisione. L'indipendenza tra il core di business e la vista dell'utente permette la **creazione di altre piattaforme** di interazioni con il sistema (che sia un applicativo web o un'app mobile...) **senza dover modificare in alcun modo quanto già realizzato**. Si noti infine come, grazie all'adozione di questo pattern, sia possibile con facilità **sostituire il backend in una zona ad esso dedicata**, come un **server privato** e permettere le comunicazioni con le istanze di frontend mediante un protocollo di rete come l'**http**.

Vediamo ora nel dettaglio le varie classi che compongono il diagramma.

8.1.1 Backend

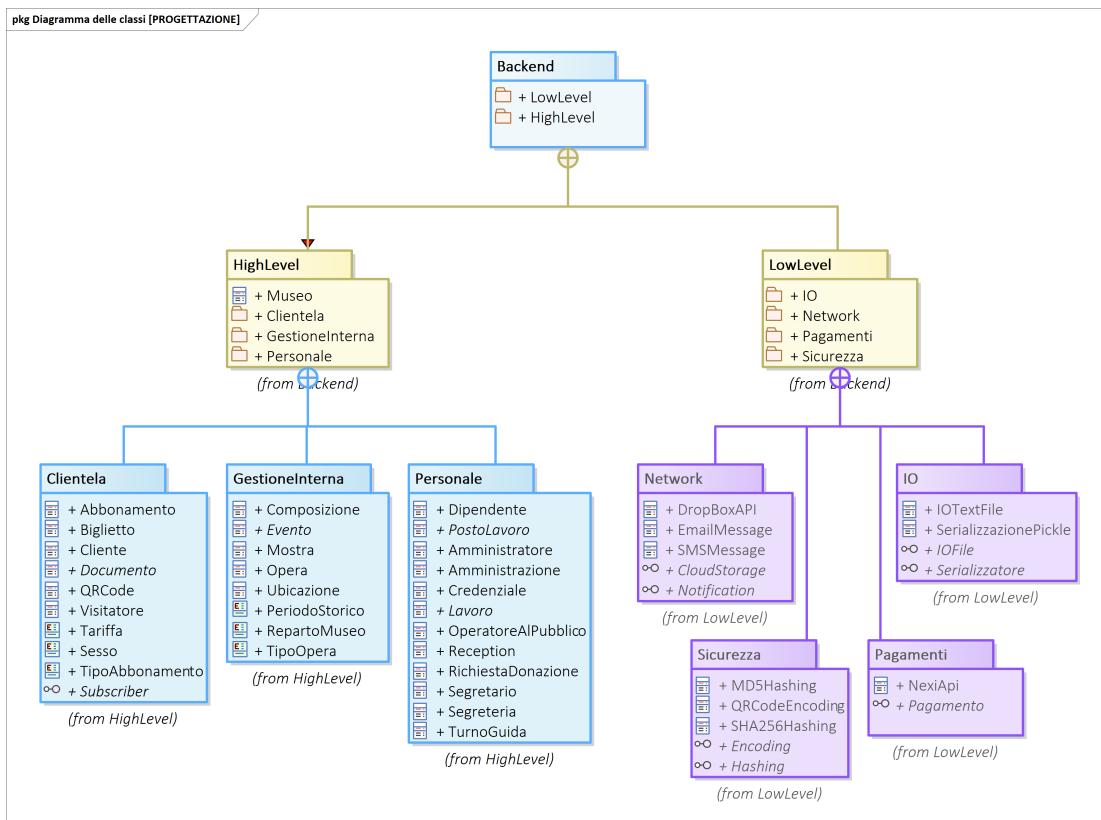


Figura 62: Package Backend

8.1.2 Frontend

Il frontend contiene le classi dei *Controllers* e delle *Views*. Come sarà poi ancora più evidente nel diagramma delle classi, questi due elementi **sono sempre presenti in coppia**; quindi per ogni vista c'è un *controller*.

Infatti, sempre rispettando la logica del pattern, la vista non è altro che un contenitore di elementi grafici visibili dall'utente, ma **non implementa alcun tipo di comportamento**; non può comunicare con il model e non gestisce gli eventi di interazione dell'utente. Queste responsabilità sono delegate al *controller* che la "controlla"; insomma **le viste sono degli involucri vuoti ma ben decorati esternamente**.

Per quanto detto dunque, i *controller* svolgono una funzione di collante tra il model e la singola vista; tra le altre cose si occupano di **prendere i dati memorizzati dal modello e di rappresentarli opportunatamente nella vista**.

In realtà ci sono due eccezioni a quanto detto:

- Ci possono essere più *controller* per una stessa *view* **qualora la stessa vista mostra un comportamento differente a seconda del contesto**, si adotta quindi il pattern *Strategy*;
- Il *model* può in alcuni casi comunicare direttamente con la *view* per **avvertirla del cambiamento del valore di un dato** si adotta quindi pattern *Observer*.

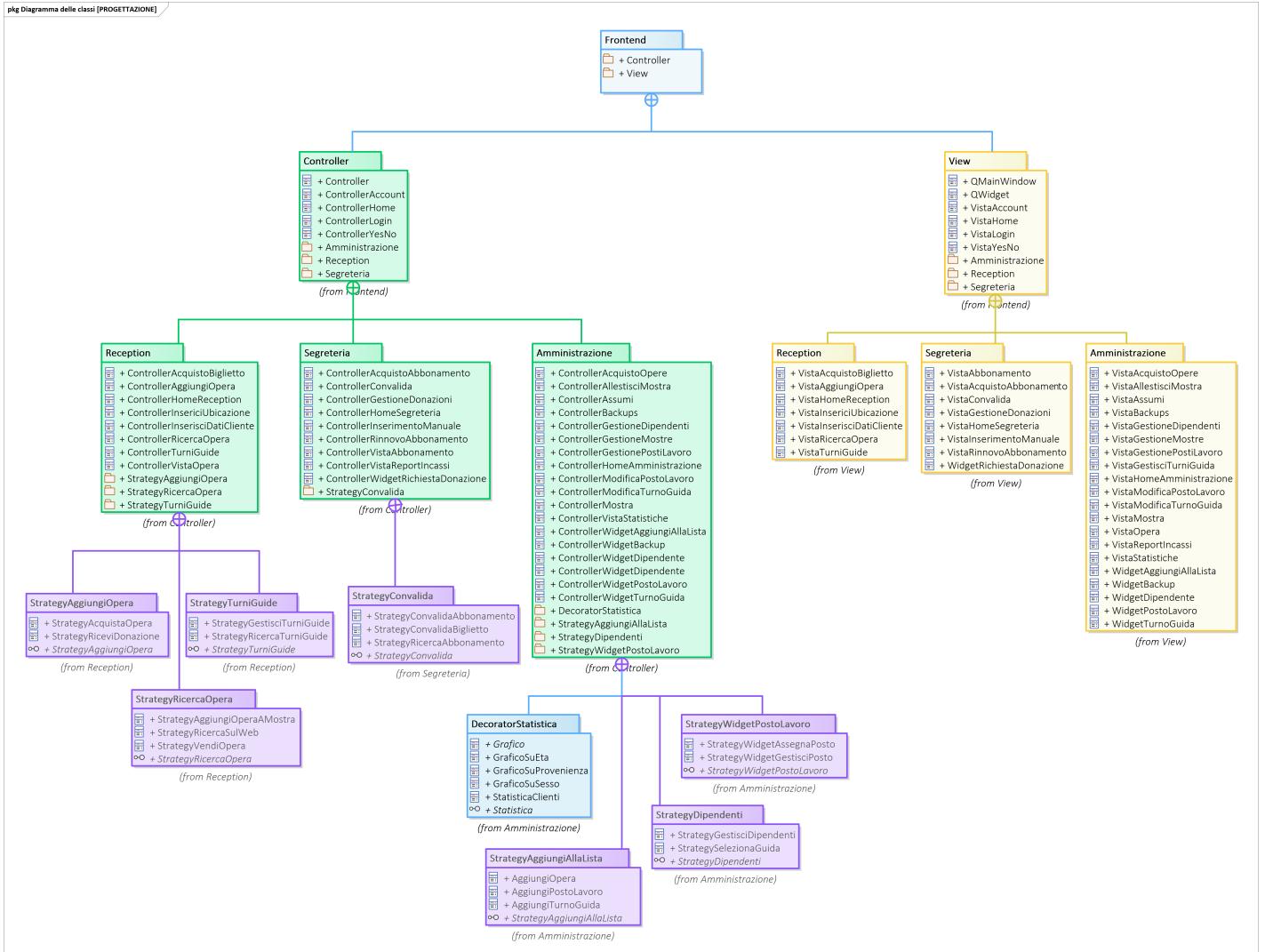


Figura 63: Package Frontend

8.2 Diagramma delle classi

8.2.1 Visione generale

Come si può vedere dalle immagini, i diagrammi sono stati **arricchiti di post-it bianchi** nelle zone riguardanti l'**adozione di un pattern** o in quelle dove sono state prese delle scelte rilevanti.

Sono inoltre presenti dei **post-it viola**, nei quali vengono espresse, mediante l'*Object Constraint Language*, i vincoli che i metodi di riferimento devono soddisfare o i valori che devono ritornare. Questi vincoli fanno a volte uso di valori di oggetti ottenuti tramite i relativi *getters*, proprio per questo motivo si è scelto di **evidenziare le proprietà *isQuery* dei getters**, in quanto sono quei metodi che non cambiando lo stato dell'oggetto, possono essere richiamati dalle espressioni OCL.

Poiché immagini come la seguente sono ricche di dettagli piccoli da leggere, si consiglia il lettore di utilizzare un lettore pdf che permette di visualizzare l'immagine al pieno della sua risoluzione (64MP), verrano comunque divise ed ingrandite nel seguito.



Figura 64: Diagramma delle classi completo

8.2.2 Backend-HighLevel

Le classi di backend (in altre parole il Model dell'MVC) sono state divise in:

- **High-level**
- **Low-level**

questa suddivisione permette di separare le classi dettate dalla **logica di business**, da quelle (presenti in generale in tutti i sistemi) che operano **operazioni tecniche** (a basso livello appunto) come la scrittura su file, la serializzazione di oggetti, l'*hashing* di password, le richieste alle API del *cloud* e così via.

Come spiegato nel relativo post-it, **l'interposizione delle interfacce tra le relazioni** dei due tipi, consente **l'inversione delle dipendenze** (quinto principio *SOLID*) e permette dunque un'**alta flessibilità** in caso di modifiche o ampliamenti alle operazioni di basso livello (basta in effetti aggiungere una classe che realizza quell'interfaccia, **senza operare modifiche alle classi "clienti" di alto livello**).

Così come se un giorno si decidesse di passare dal cloud *DropBox* al cloud fornito da amazon *WebService*, non sarebbe necessario apportare modifiche alla classe *Museo* che, fra le altre responsabilità, ha quella di caricare il backup sul cloud; infatti **lavorando la stessa con l'interfaccia *CloudStorage* e non direttamente con le sue slassi realizzatrici, non ha alcun interesse di come verrà implementato il metodo *upload()***, ma si limita a richiamarlo. Sarà la classe chiamante a passare un'istanza della nuova classe *AmazonAWSAPI* invece di una di *DropBoxAPI*, la quale è obbligata (da qui **l'inversione delle dipendenze**) dal fatto di implementare l'interfaccia *CloudStorage*, a fornire il comportamento che da essa ci si aspetta.

Al fine di dare maggiore spazio all'immagine che segue, il resto della pagina è stato lasciato volutamente bianco.

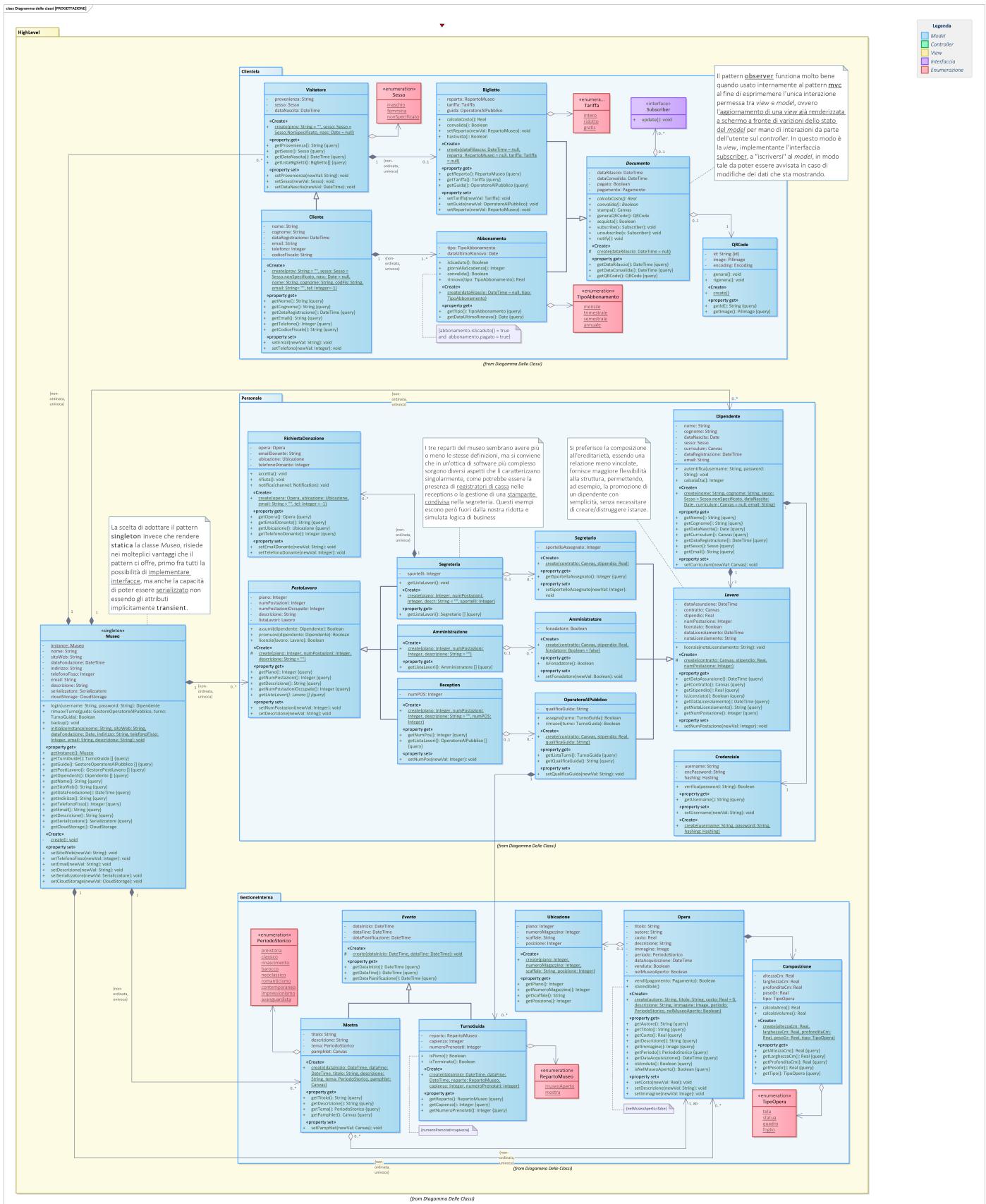


Figura 65: classi di backend di alto livello

8.2.3 Backend-LowLevel

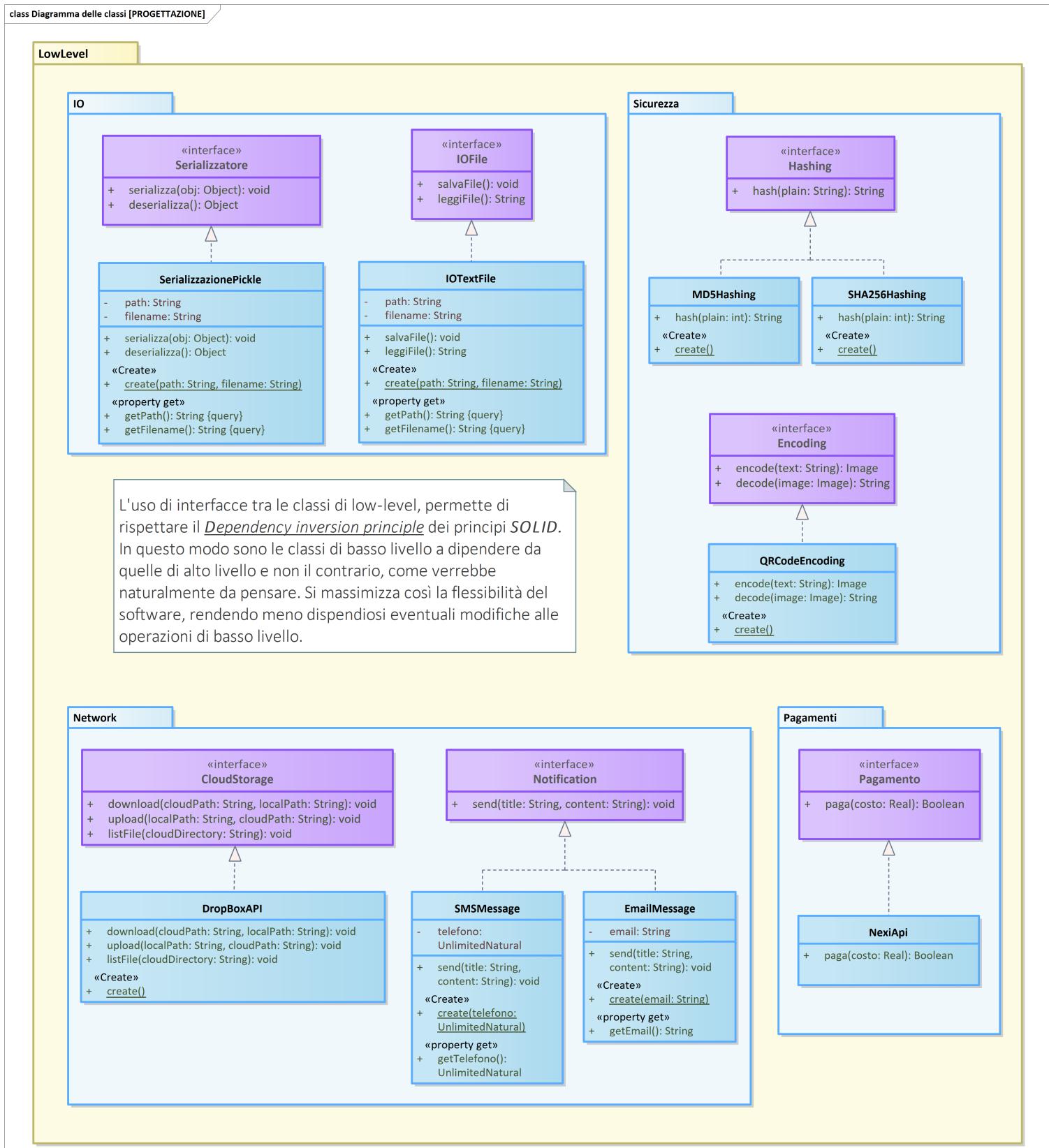


Figura 66: classi di backend di basso livello

8.2.4 Frontend

Come già anticipato nel capitolo 8.1.2, i pattern *Strategy* e *Observer* sono stati qui presenti per estendere comodamente il comportamento dell'MVC.

Le classi di controllo che sono **fornitori** dell’interfaccia *Subscriber*, **si sottoscrivono alla lista di subscribers della classe di model relativa**, in modo da poter essere notificati in caso di aggiornamento di dati, in questo modo, **si evita** di dover aspettare il riavvio della vista o di implementare un **periodico refresh** della vista direttamente dal frontend.

Per quanto riguarda invece i controller con sopra le interfacce *Strategy*, essi sono **polimorfi nel comportamento**, in quanto, a seconda del contesto, eseguiranno sulla vista di controllo, delle azioni differenti. Si evita quindi, grazie al pattern *Strategy*, **di dover creare tante viste per ogni comportamento**, ma si "ricicla" la stessa.

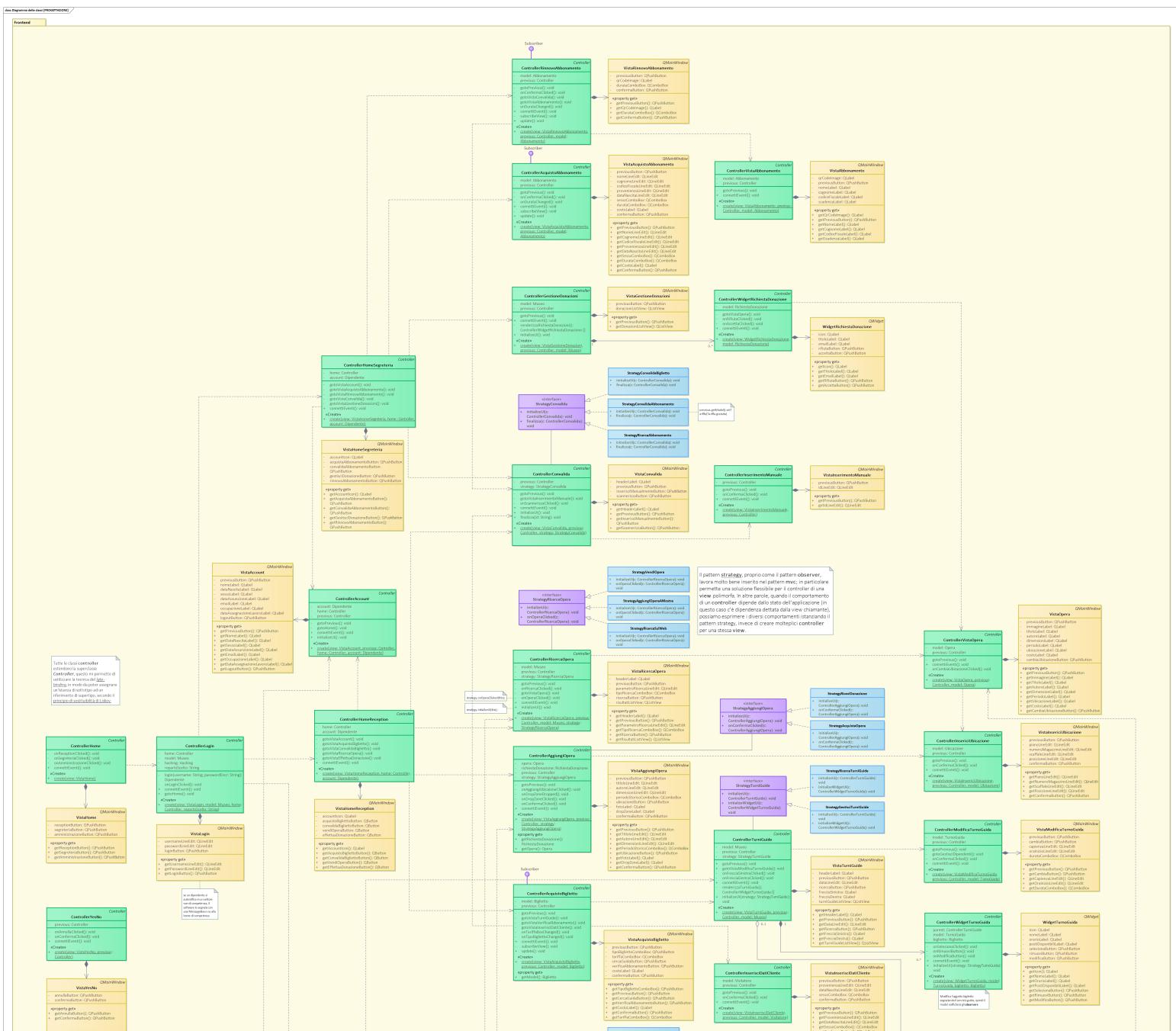


Figura 67: Package Frontend 1/2

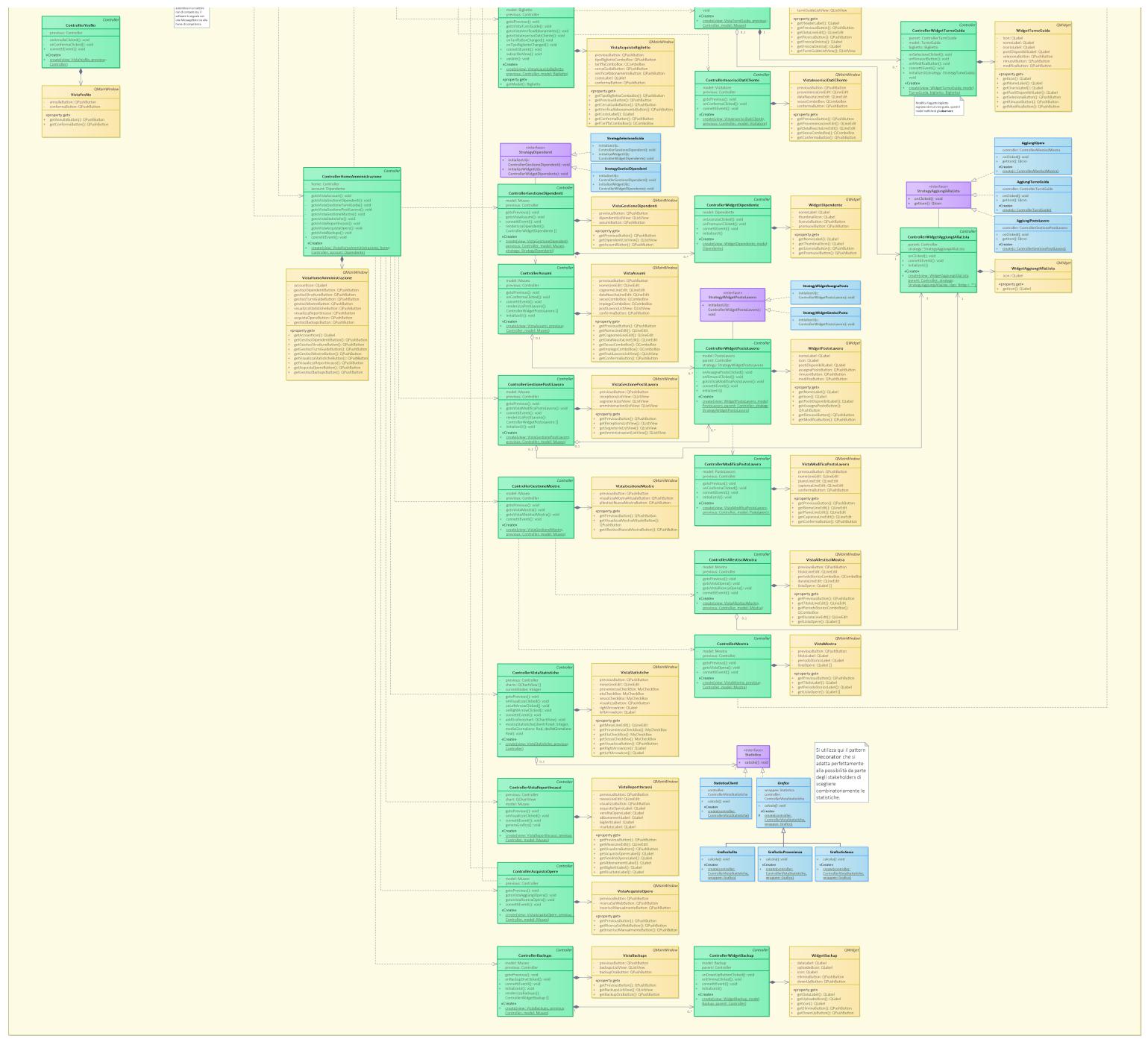


Figura 68: Package Frontend 2/2

9 Diagrammi di sequenza raffinati

Questo capitolo propone un **raffinamento** dei diagrammi di sequenza di analisi visti nel capitolo 6, alla luce delle diverse modifiche apportate al diagramma delle classi viste nel capitolo 8.

La modifica più importante, oltre al maggior dettaglio in generale, è , come già si preannunciava nell'introduzione del capitolo 6, la **sostituzione dei "Gestori" con le view e i controller** studiati poi in progettazione.

L'aver adottato il pattern Model View Control, porta in maniera molto naturale, grazie alle forti analogie vigenti, alla scelta di adottare **il pattern di architettura le Entity Control Boundary nella rappresentazione delle sequenze**, al fine di ottenere dei **diagrammi più robusti e comodi alla lettura**.

A rafforzamento di quanto detto, si è scelto di **raffinare solo quelle sequenze** che hanno in qualche modo a che fare con le viste.

Vedremo ora nel dettaglio i seguenti diagrammi di sequenza, è stato riportato vicino ad ogni elemento della lista, l'identificativo del **caso d'uso** dal quale è stata estratta la sequenza.

- **CompraBiglietto Raffinato**
 - CompraBiglietto (CU 01)
 - RicercaTurnoGuida (CU 17)
 - InserisciDatiClienti (CU 28)
- **CompraOpera Raffinato**
 - CompraOpera (CU 20)
- **VerificaAbbonamento Raffinato**
 - VerificaAbbonamento (CU 26)
- **AbbonaCliente Raffinato**
 - AbbonaCliente (CU 11)

9.1 CompraBiglietto Raffinato

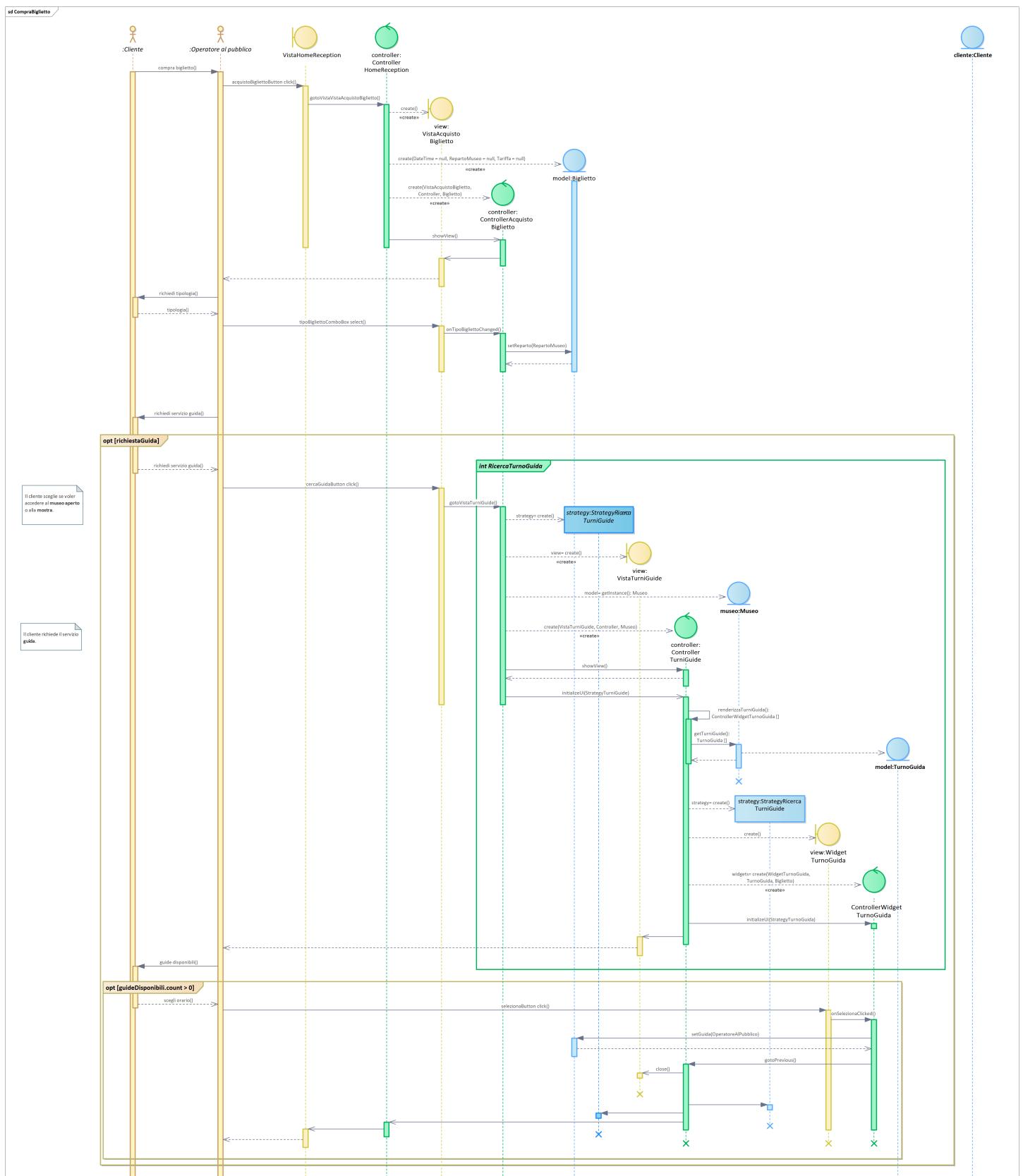


Figura 69: CompraBiglietto Raffinato 1/2

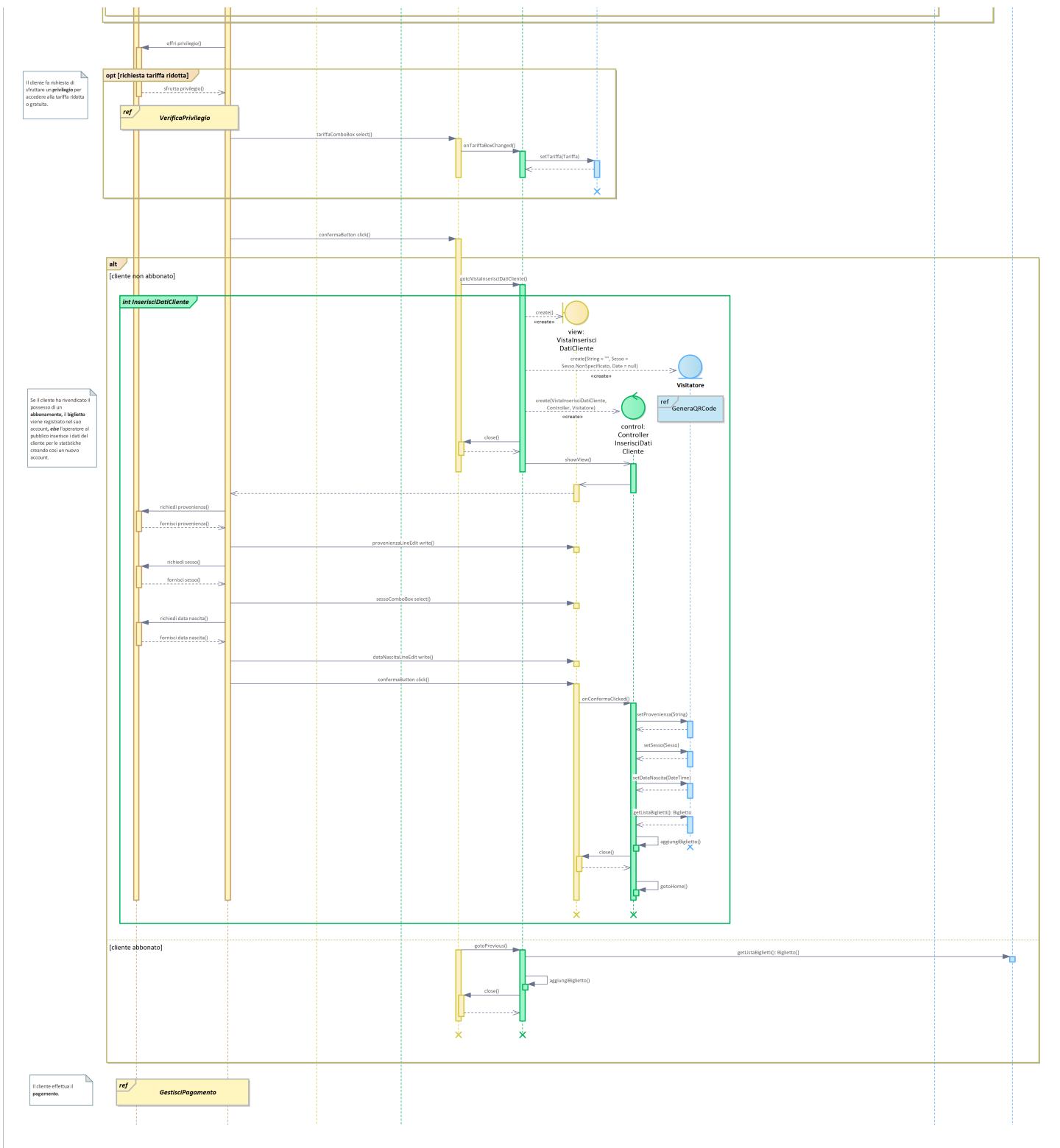


Figura 70: CompraBiglietto Raffinato 2/2

9.2 CompraOpera Raffinato

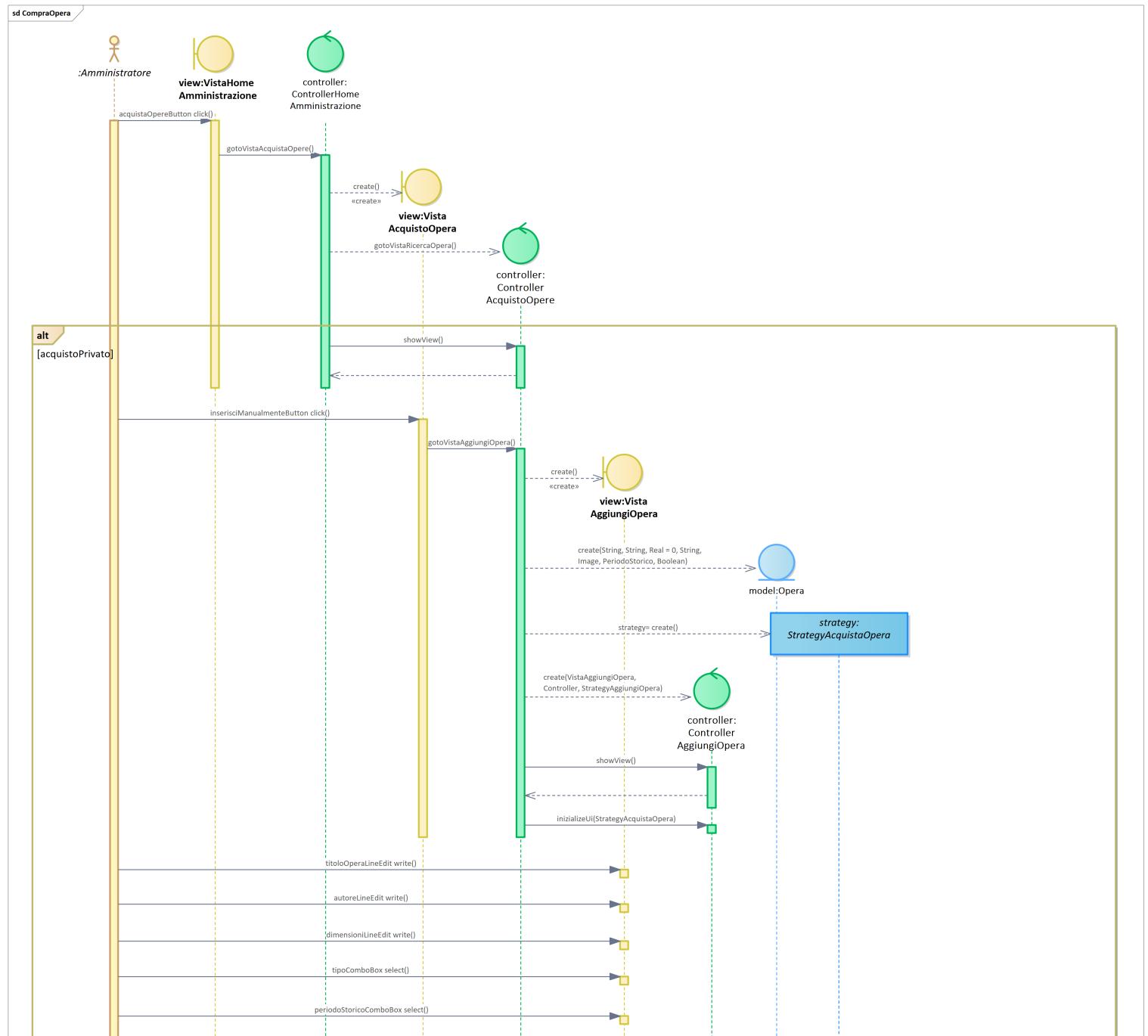


Figura 71: CompraOpera Raffinato 1/3

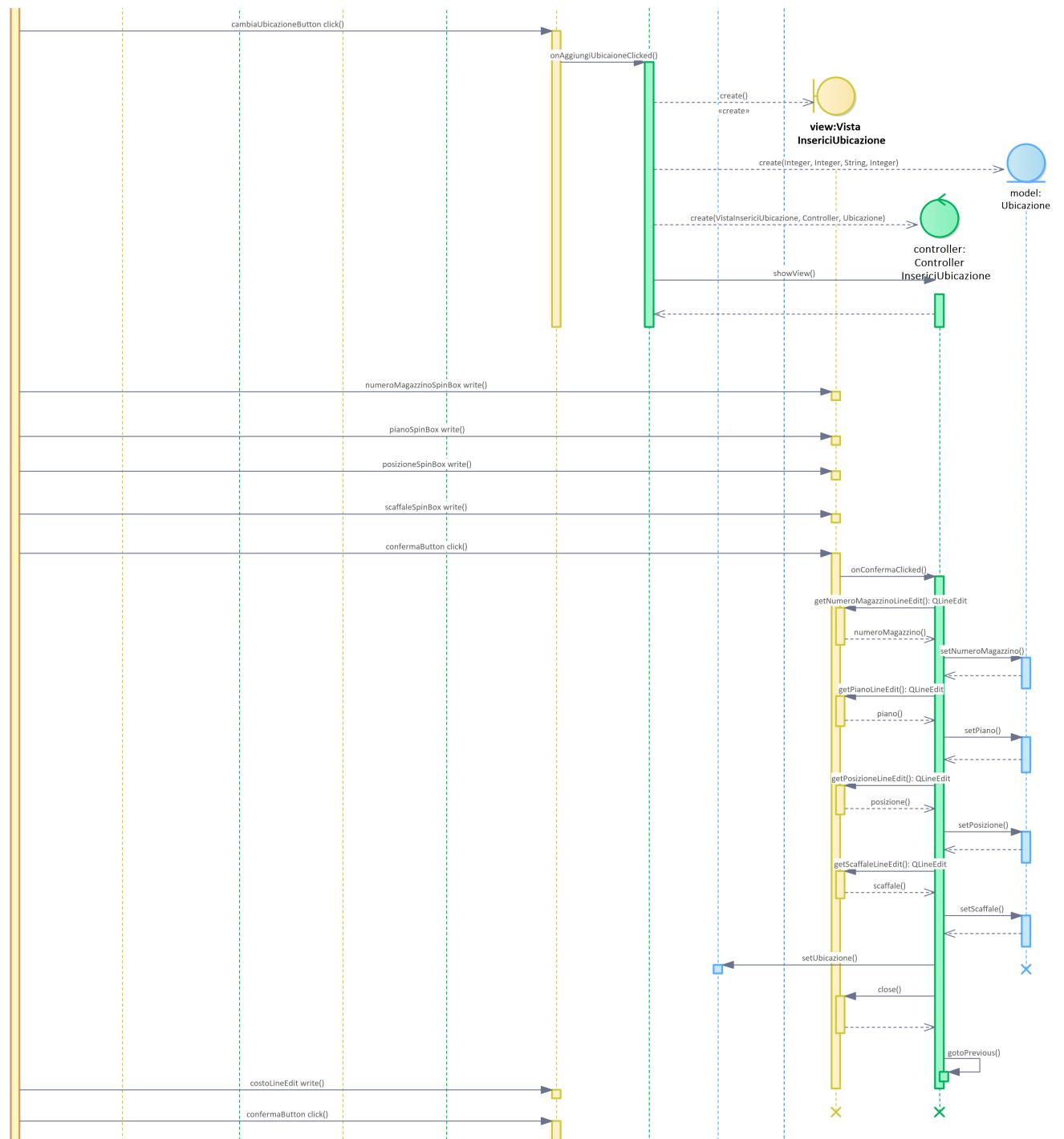


Figura 72: CompraOpera Raffinato 2/3

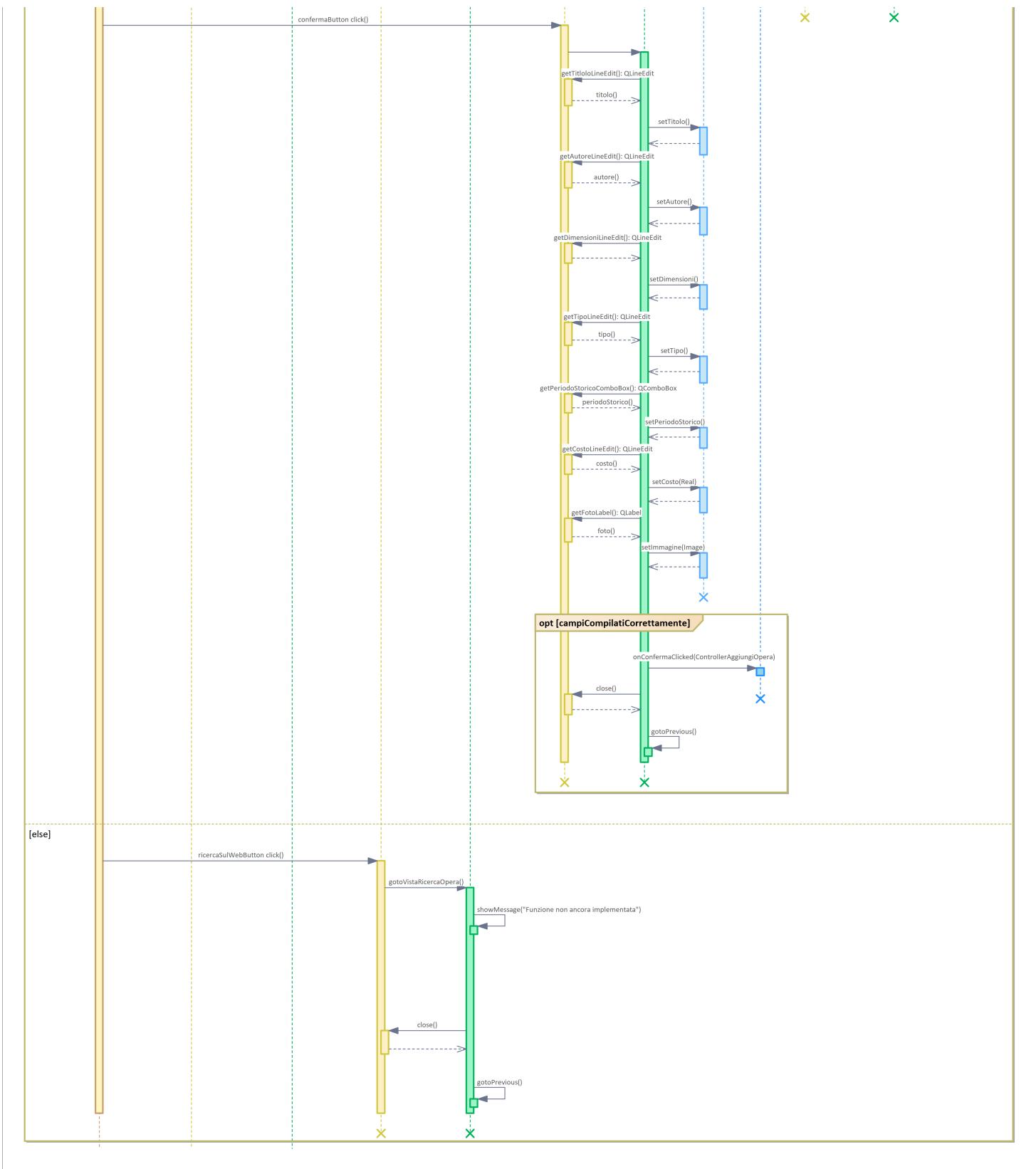
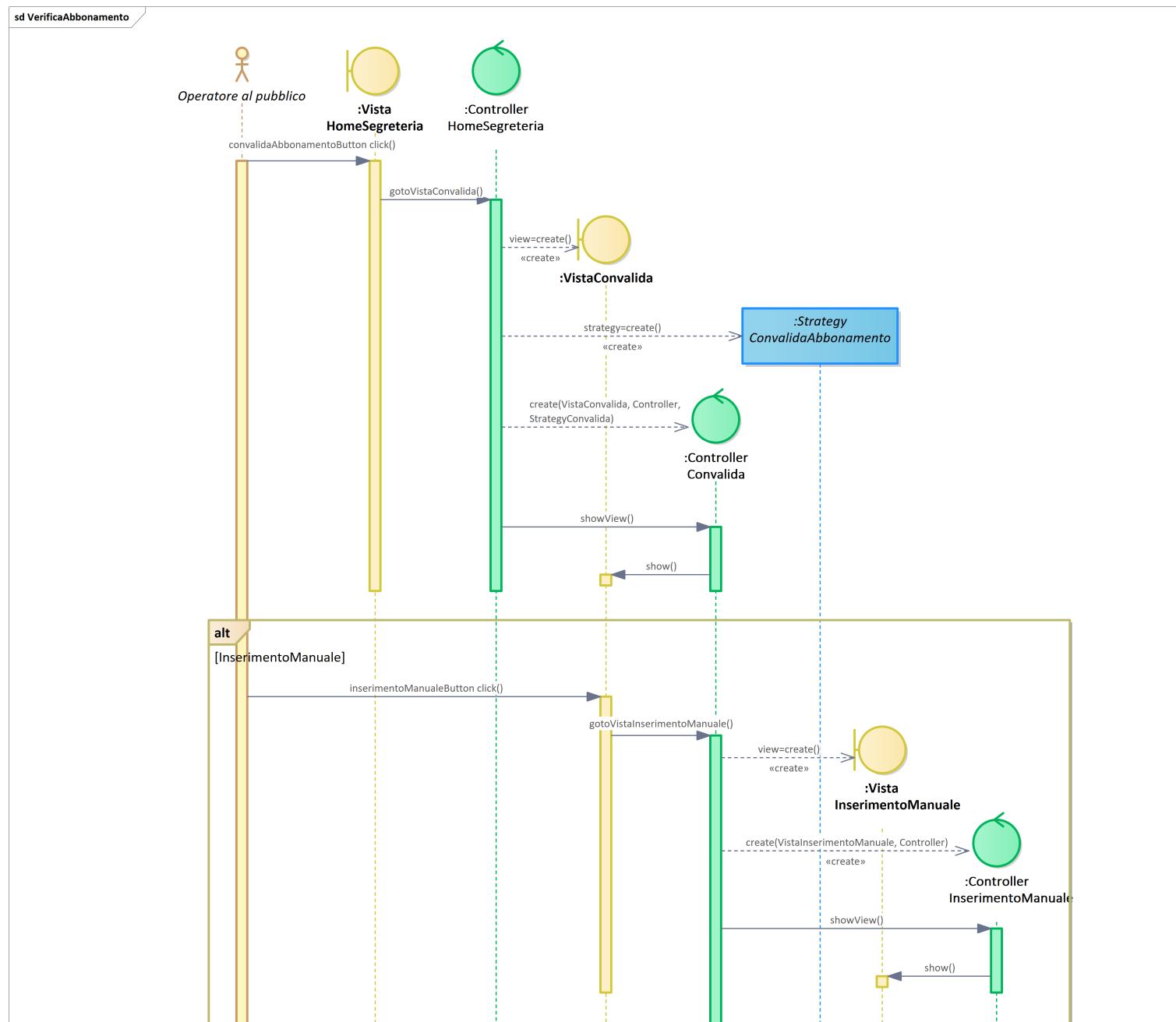


Figura 73: CompraOpera Raffinato 3/3

9.3 VerificaAbbonamento Raffinato



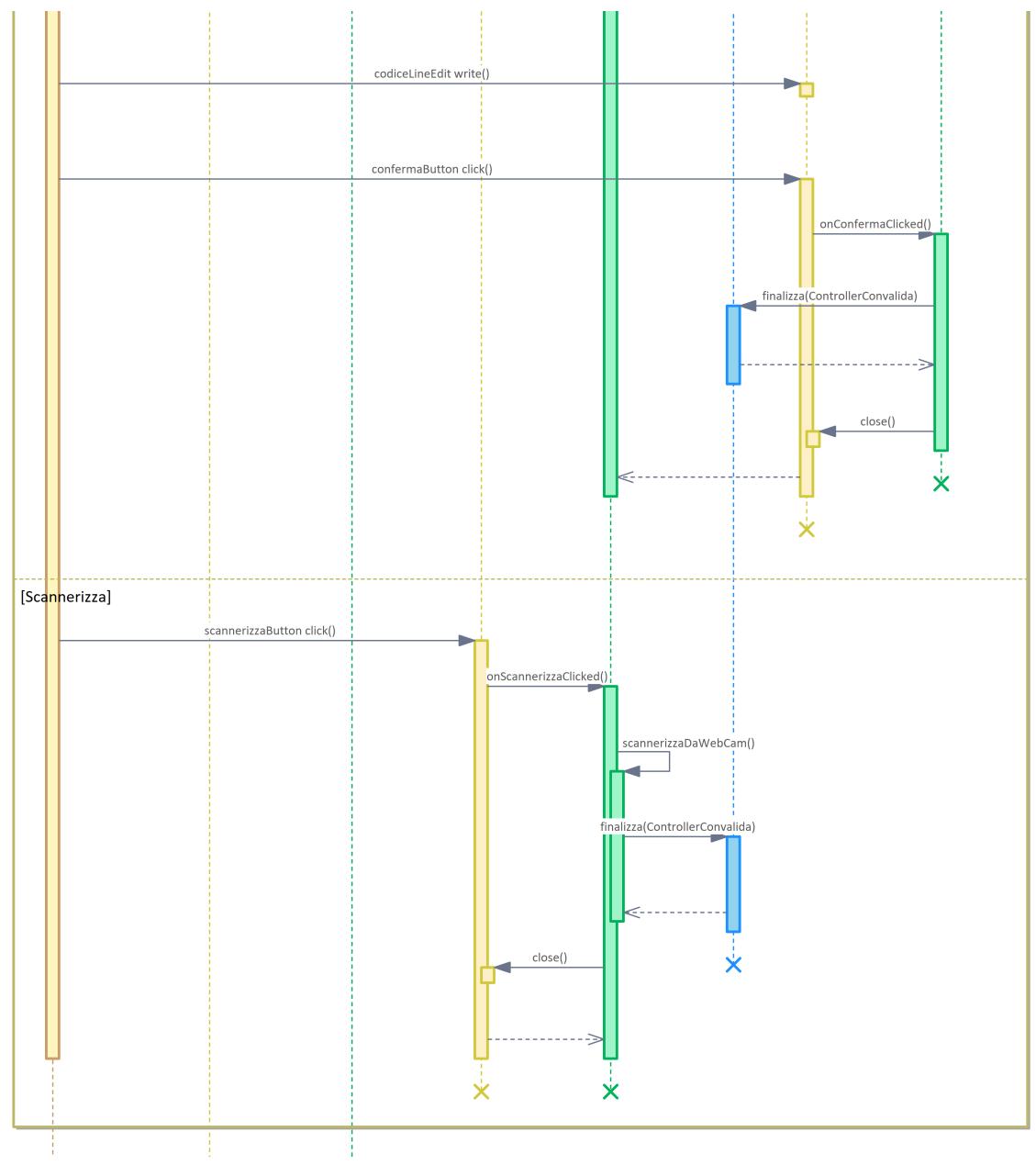


Figura 74: VerificaAbbonamento Raffinato 2/2

9.4 AbbonaCliente Raffinato

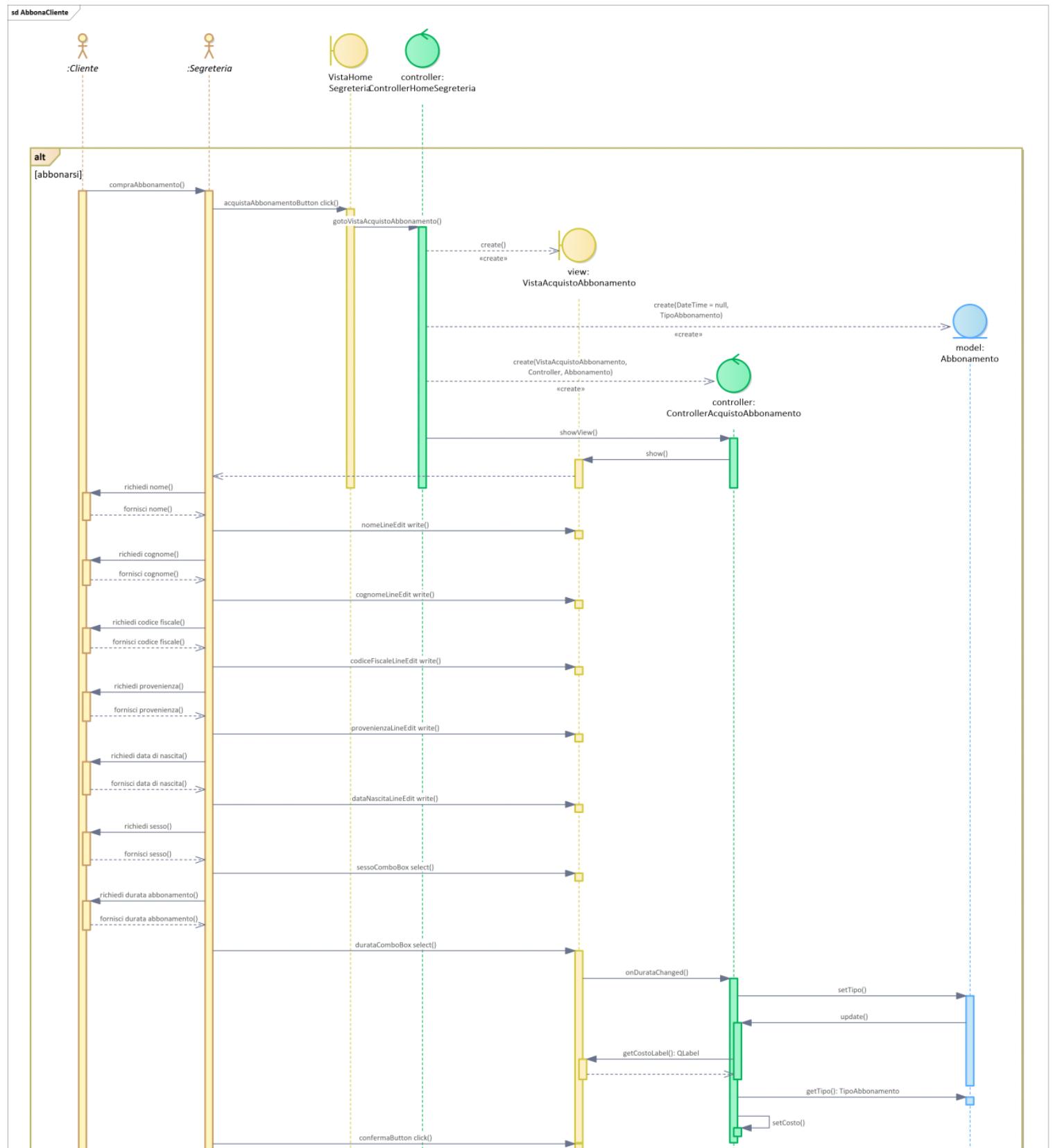


Figura 75: AbbonaCliente Raffinato 1/3

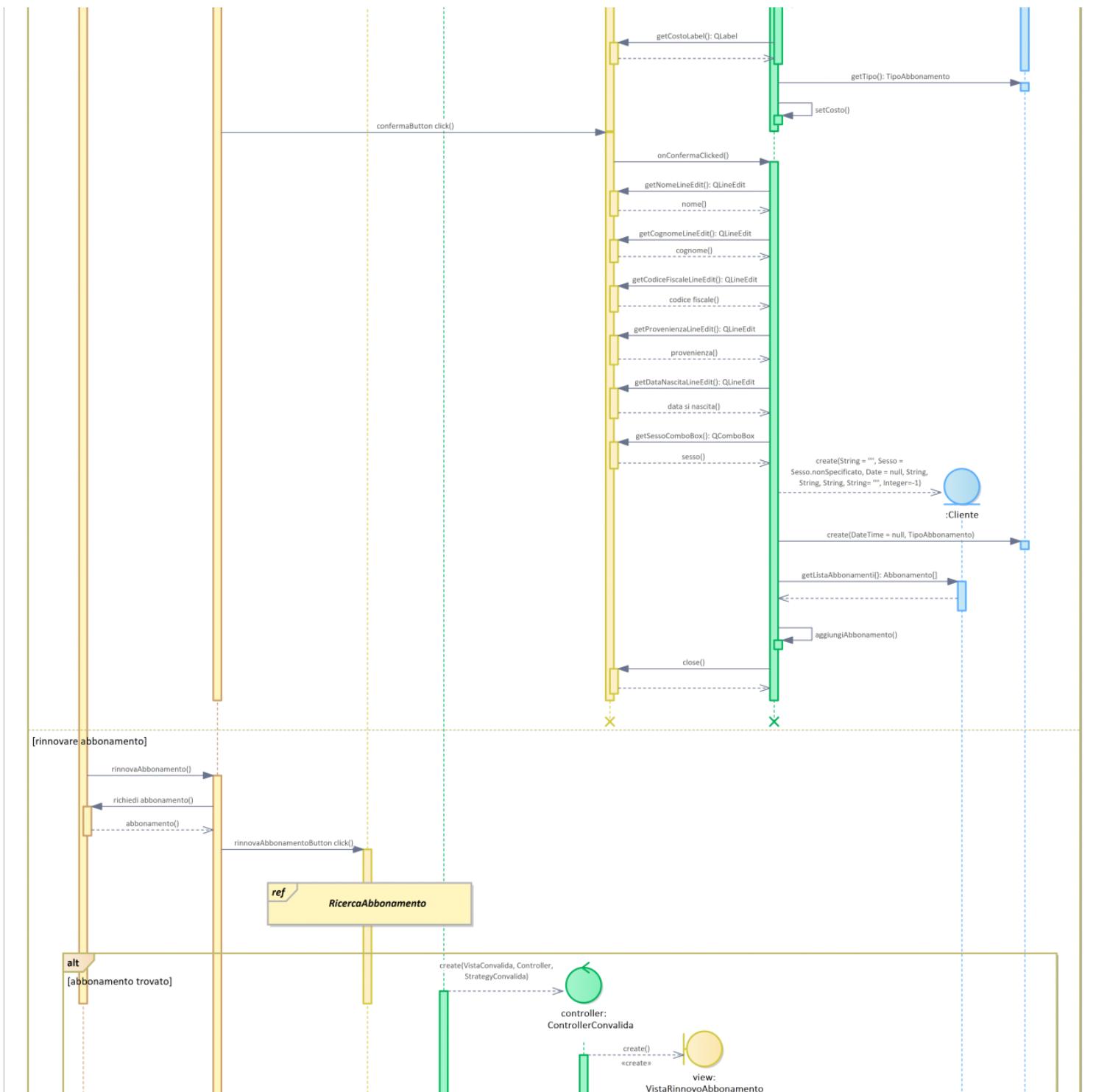


Figura 76: AbbonaCliente Raffinato 2/3

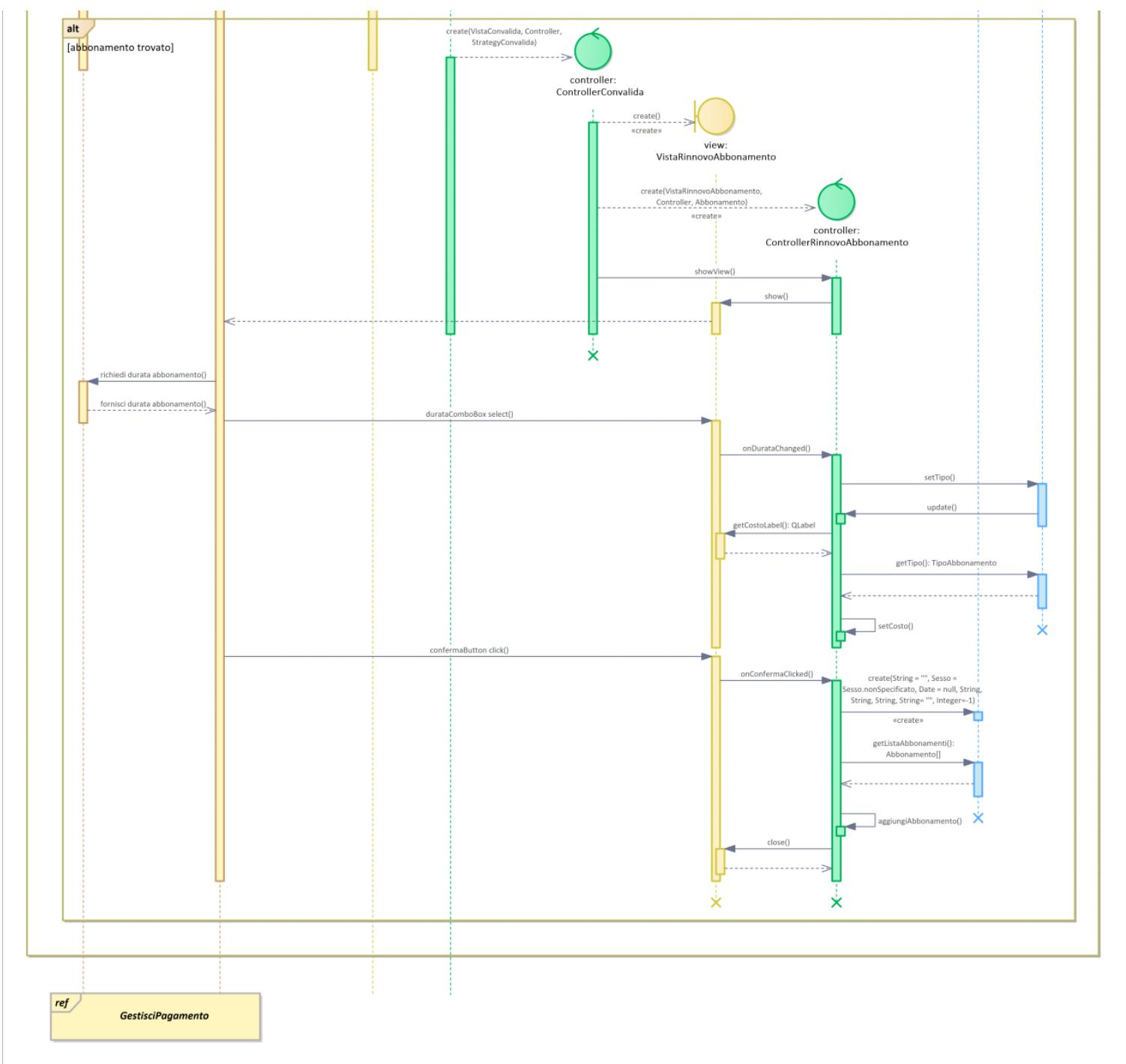


Figura 77: AbbonaCliente Raffinato 3/3

10 Diag. delle macchine a stati

Nel seguente capitolo vedremo modellati tramite i **Diagrammi delle macchine a stati** i cicli di vita di entità reattive ovvero oggetti, casi d'uso, classi. I diagrammi rappresentano il comportamento dinamico delle classi prese in considerazione. Vengono rappresentati gli stati attraverso i quali esse transitano e anche gli eventi che causano le transizioni stesse.

Vedremo ora nel dettaglio i seguenti diagrammi delle macchine a stati, sono state scelte le seguenti classi, le uniche con uno stato interessante da descrivere:

- Museo
- Biglietto
- Abbonamento
- RichiestaDonazione
- Dipendente (Login)
- Dipendente (Lavoro)

10.1 Museo

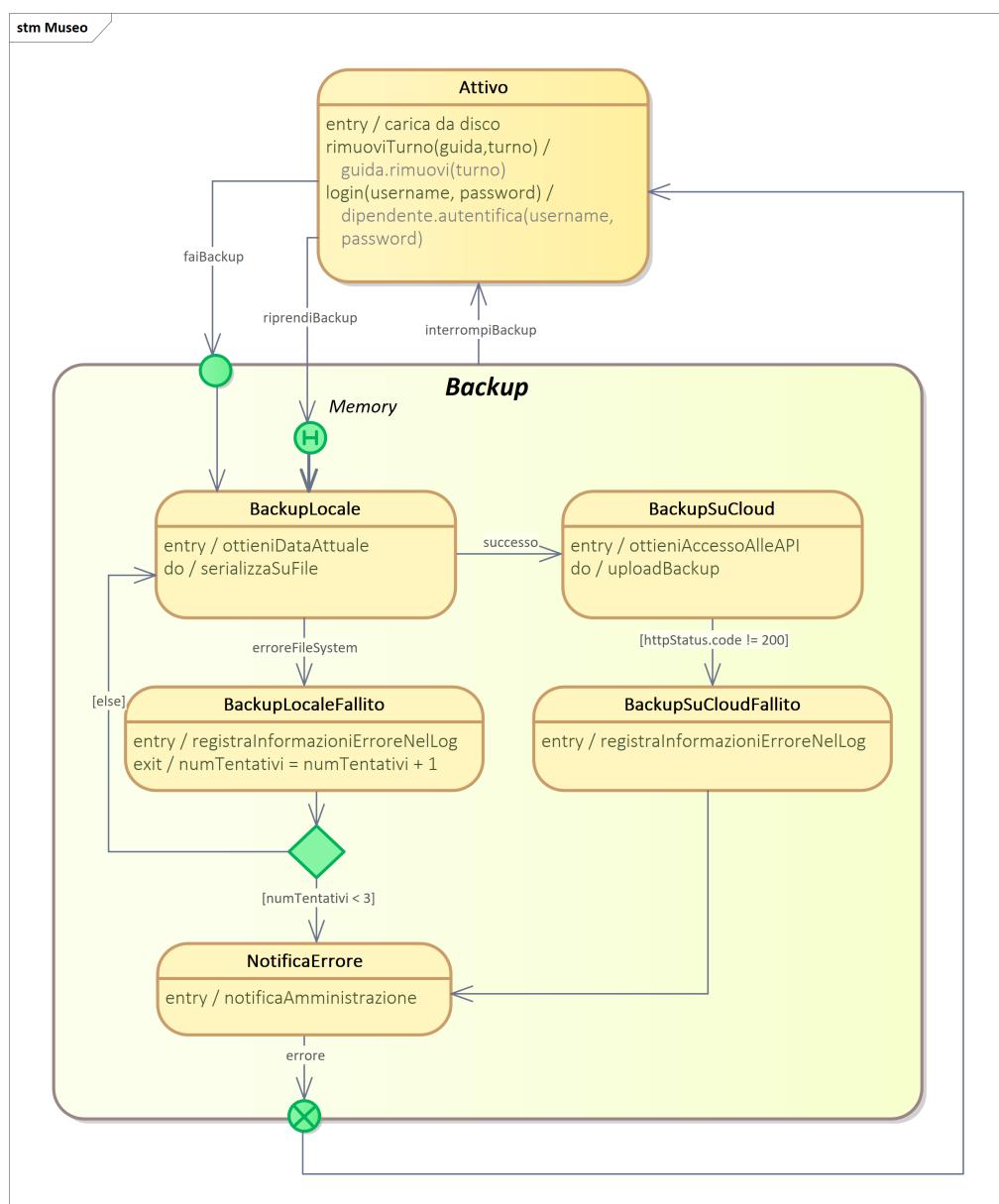


Figura 78: Museo

10.2 Biglietto

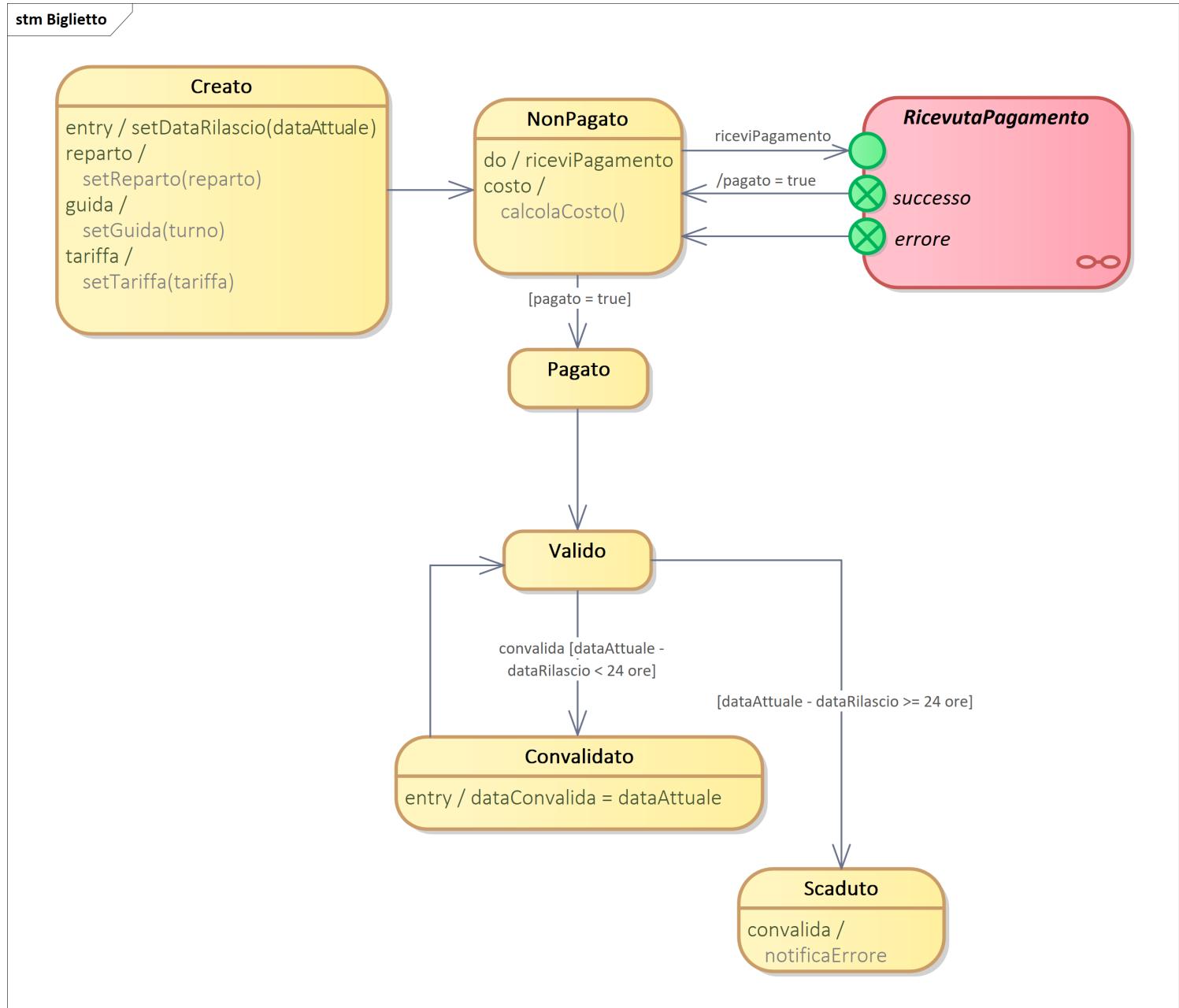


Figura 79: Biglietto

10.3 Abbonamento

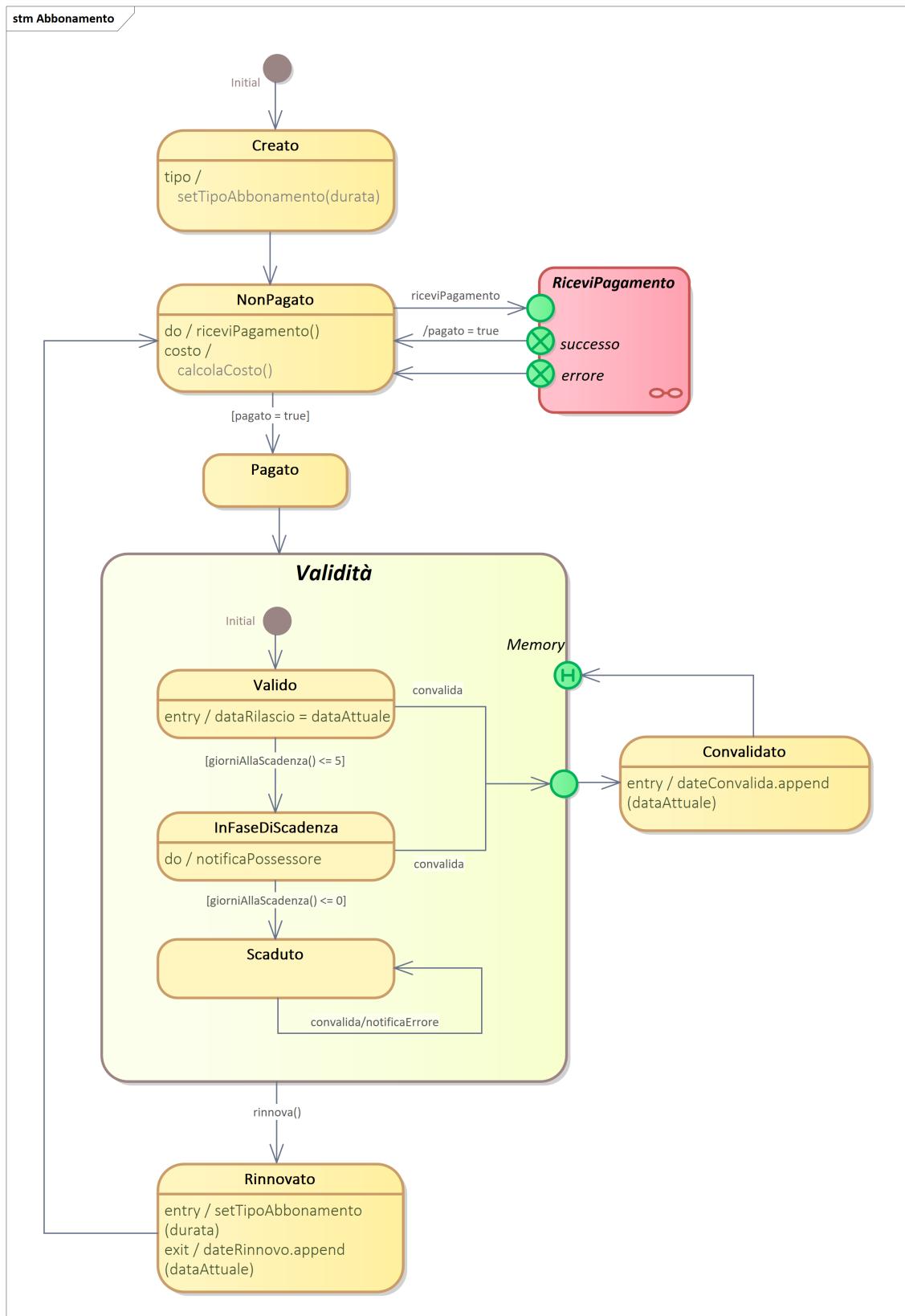


Figura 80: Abbonamento

10.4 RichiestaDonazione

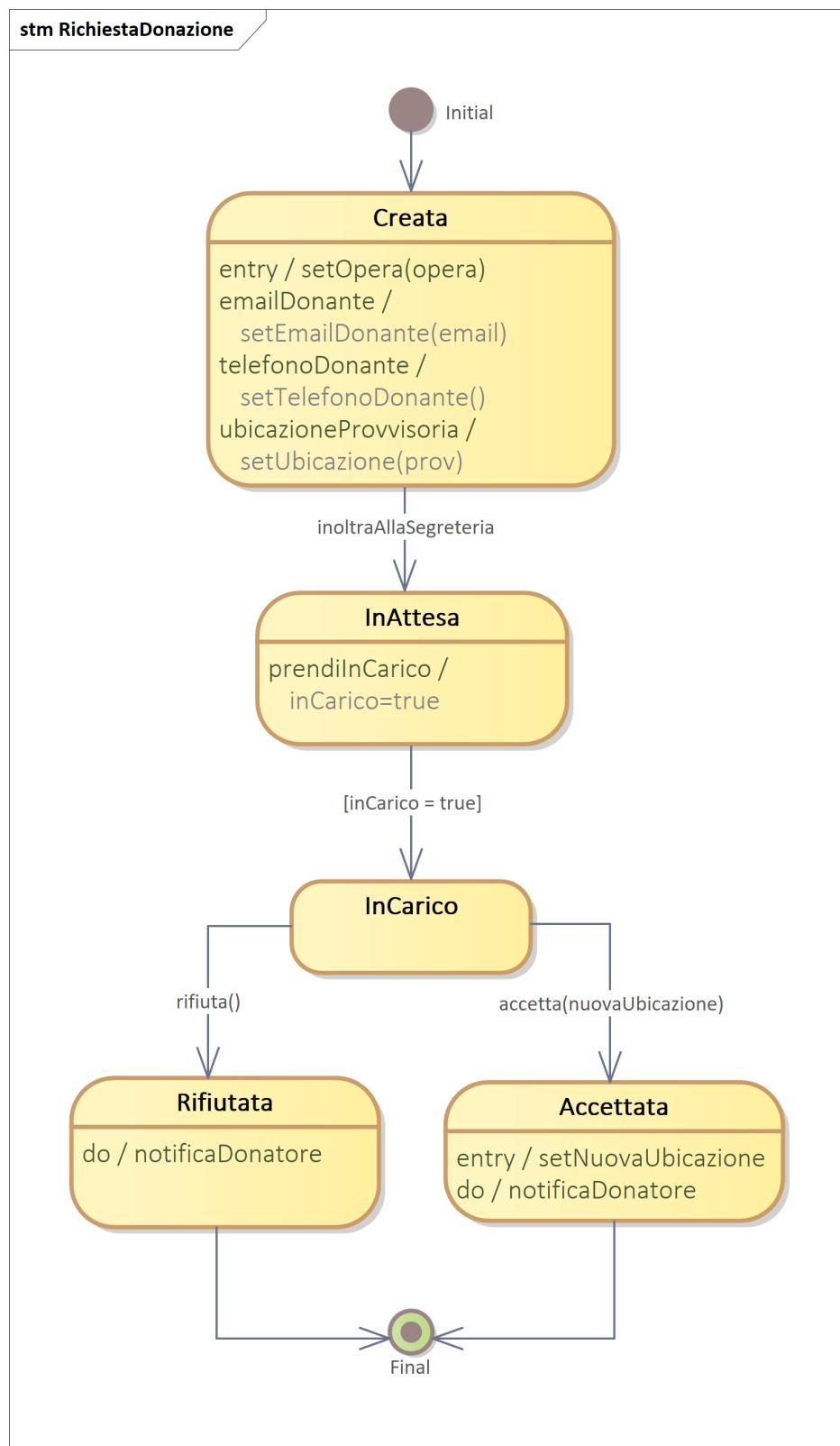


Figura 81: RichiestaDonazione

10.5 Dipendente (Login)

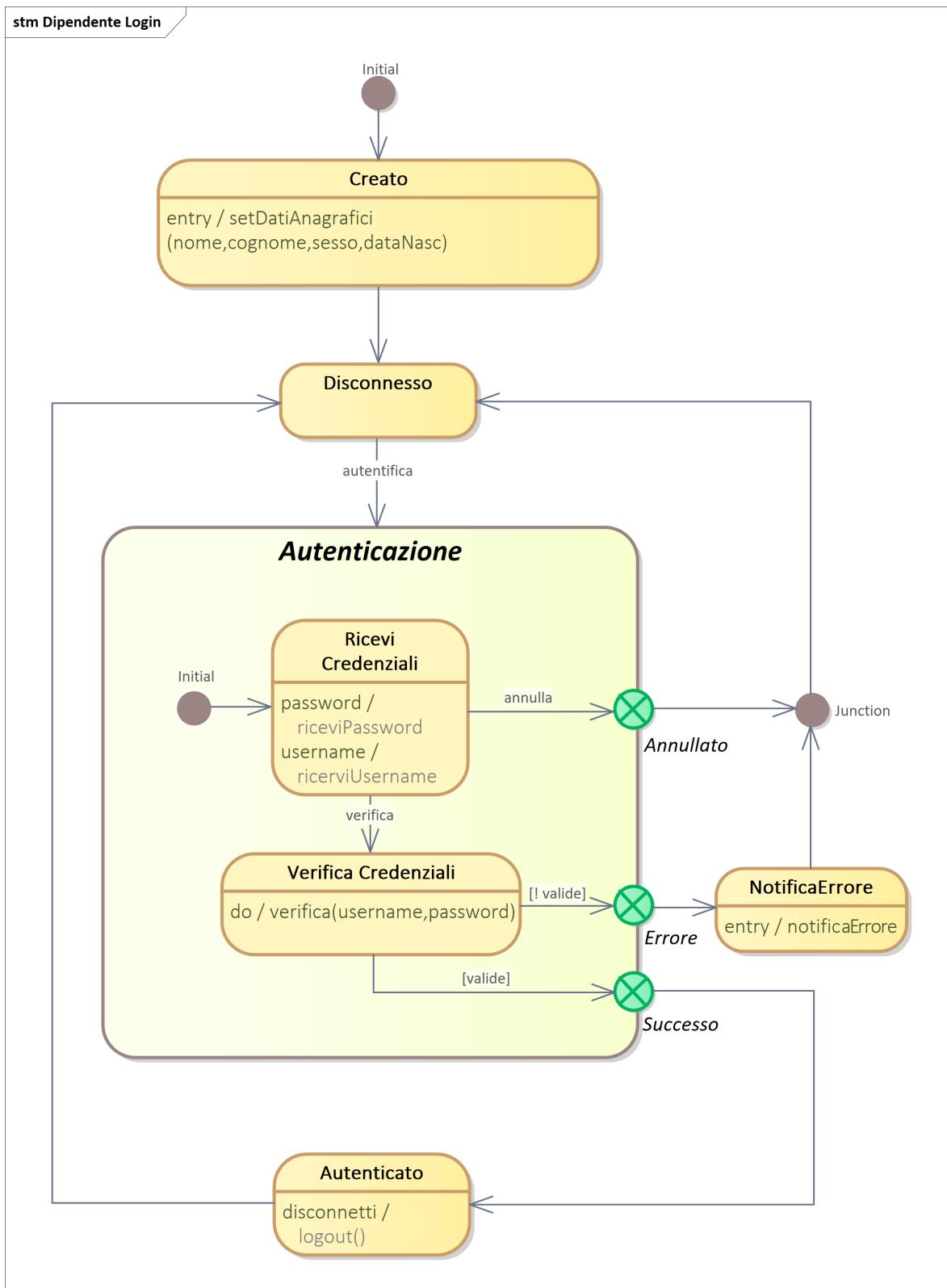


Figura 82: Dipendente (Login)

10.6 Dipendente (Lavoro)

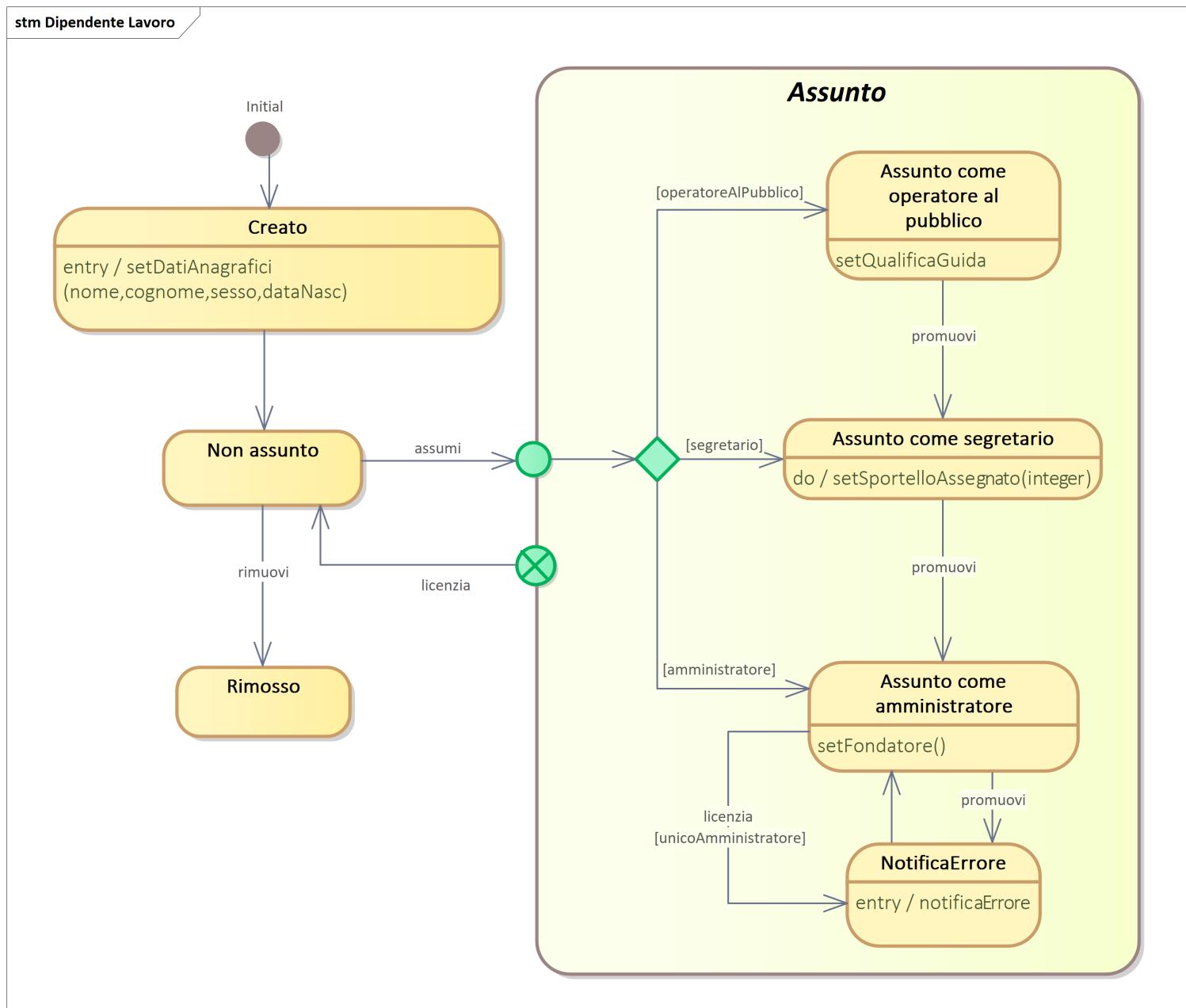


Figura 83: Dipendente (Lavoro)

11 Diagramma dei componenti

In questo capitolo verrà preso in considerazione il **diagramma dei componenti**, creato a partire dal diagramma delle classi di progettazione visto nel capitolo 8.

11.1 Diagramma dei componenti completo

Un componente, secondo **UML**, è un modulo che viene descritto interamente dalle *interfacce* che questo fornisce e richiede. Tenendo presente la definizione si è scelto di **estrarre i componenti dai package di progettazione**, e di segnare dunque tutte le classi interne e le interazioni tra diversi package mediati dalle interfacce.

Si noti come, rispettando quanto detto finora nel capitolo 8 delle classi di progettazione, le dipendenze tra le classi di alto livello e quelle di basso livello sono mediate attraverso le interfacce, questo crea una **struttura così definita loosely coupled**, ovvero i vari componenti che costituiscono il package di basso livello, offrono dei comportamenti fini a se stessi, sostituibili in qualsiasi momento da altri alternativi o migliori da qui il *Component Based Development Paradigm*.

Per quanto riguarda invece l'unica interazione tra il componente di *frontend* e quello di *backend*, è quella descritta dal pattern **observer**, la classe *Documento* del model, accetta classi realizzatrici dell'interfaccia *Subscriber*, e le notifica in caso di cambiamenti allo stato dell'oggetto stesso.

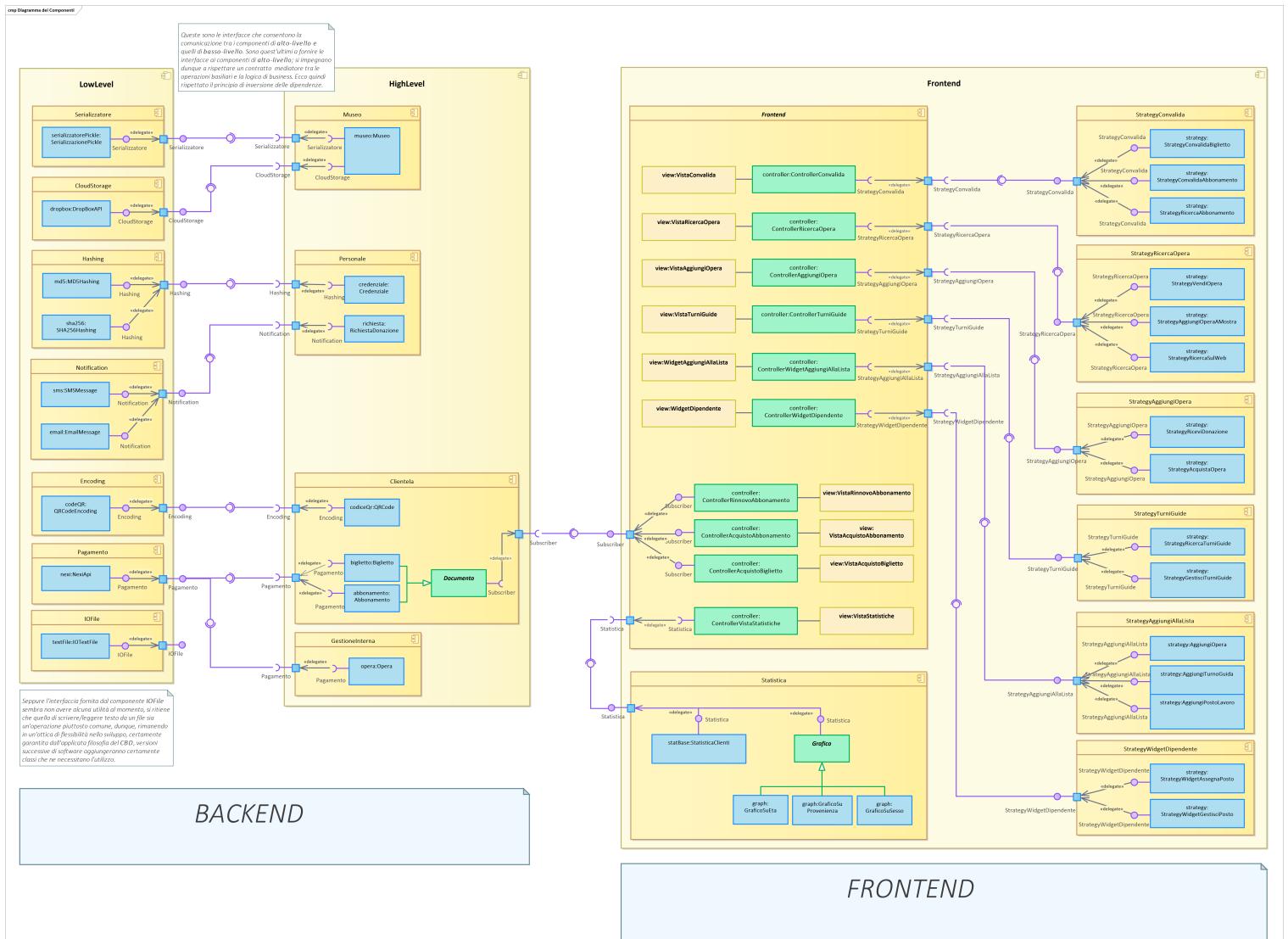


Figura 84: Diagramma dei Componenti completo

11.2 Diagramma dei componenti backend

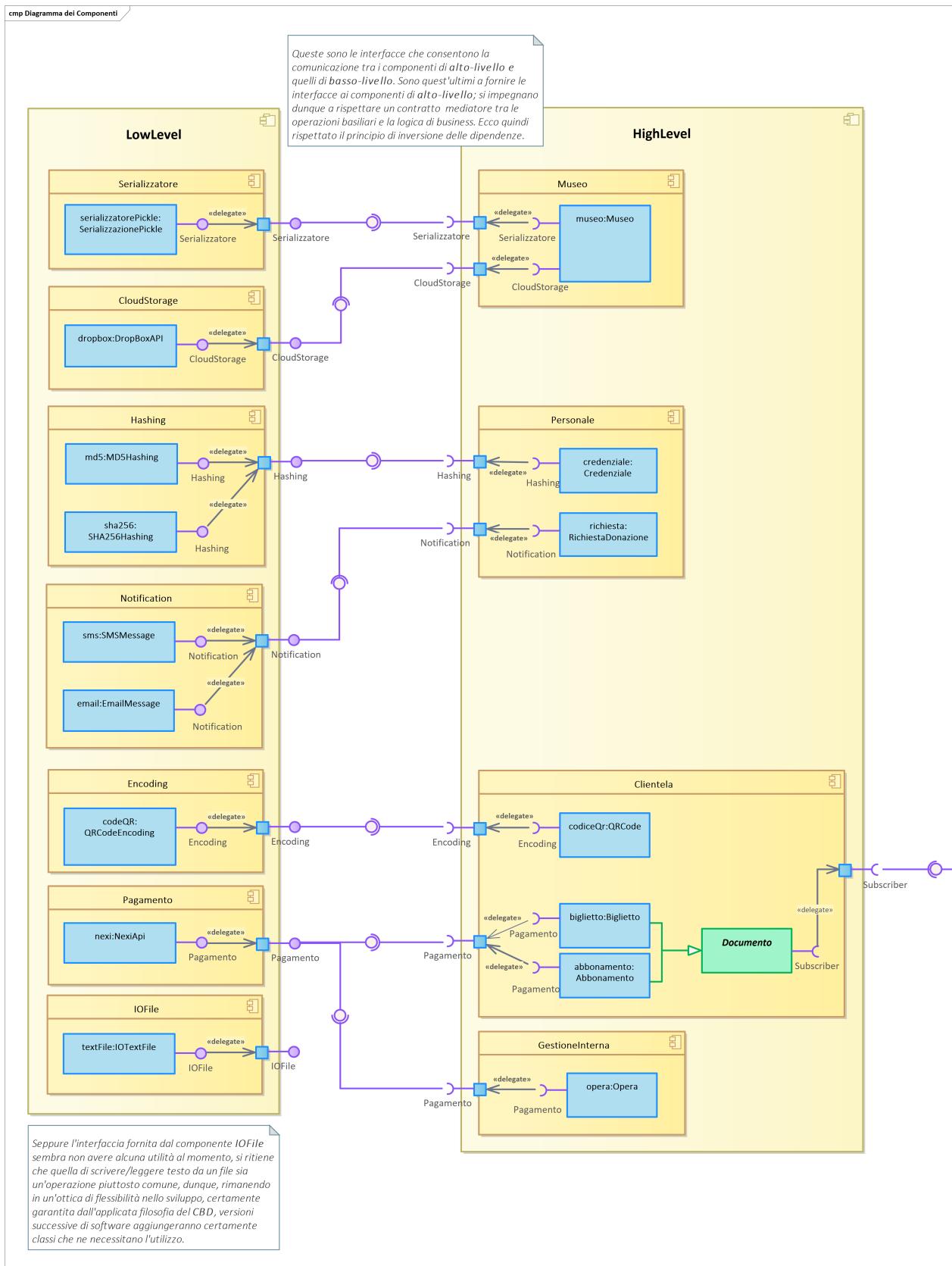


Figura 85: Diagramma dei Componenti backend

11.3 Diagramma dei componenti frontend

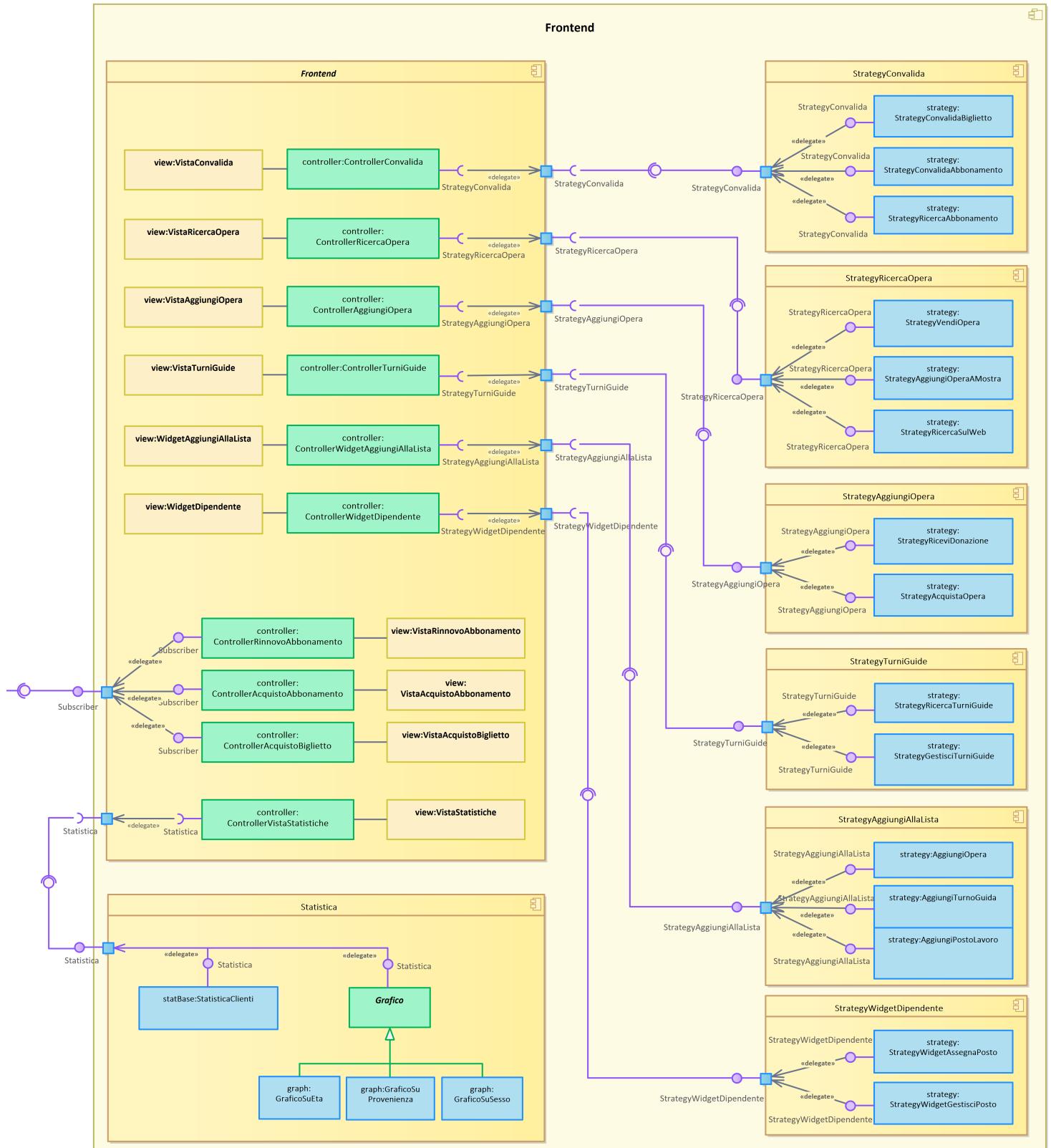


Figura 86: Diagramma dei Componenti frontend

12 Mockup

Concludiamo il flusso di lavoro della progettazione con la realizzazione dei **prototipi delle interfacce grafiche**. A tal fine è stato utilizzato il software *Wondershare Mockitt* (abbonamento mensile).

Vedremo ora nel dettaglio, in questo capitolo, i **mockup** relativi a tutte le interfacce del software, divisi in base alla consequenzialità della loro visualizzazione.

Una parte rilevante del flusso di lavoro è stata impiegata nell'ideare interfacce che fossero *intuitive* e di facile utilizzo e nella scelta di una paletta di colori che risultasse uniforme in tutte le viste.

L'utilizzo dei mockup come modelli è risultato molto utile in fase di implementazione nella creazione delle **GUI** (*graphical user interfaces*).

12.1 Mockups delle Home e dell'account



Figura 87: VistaHome

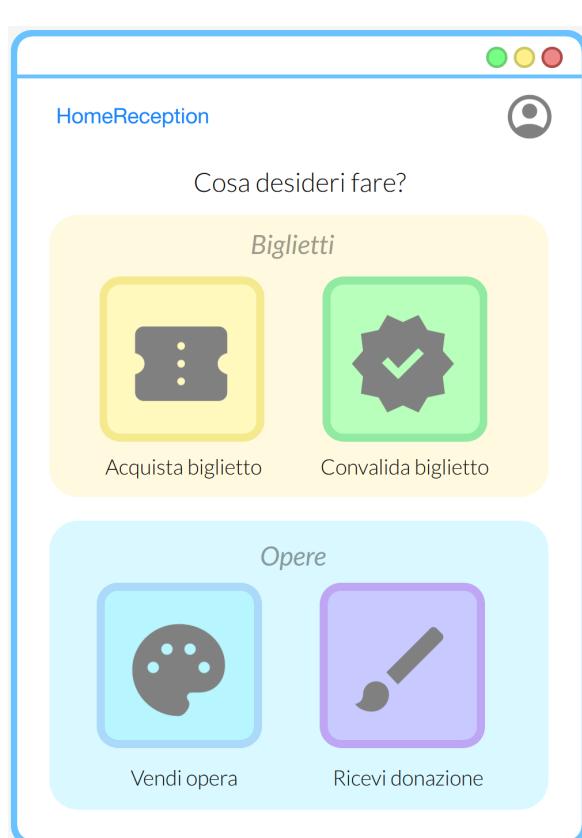


Figura 88: VistaHomeReception



Figura 89: VistaHomeSegreteria

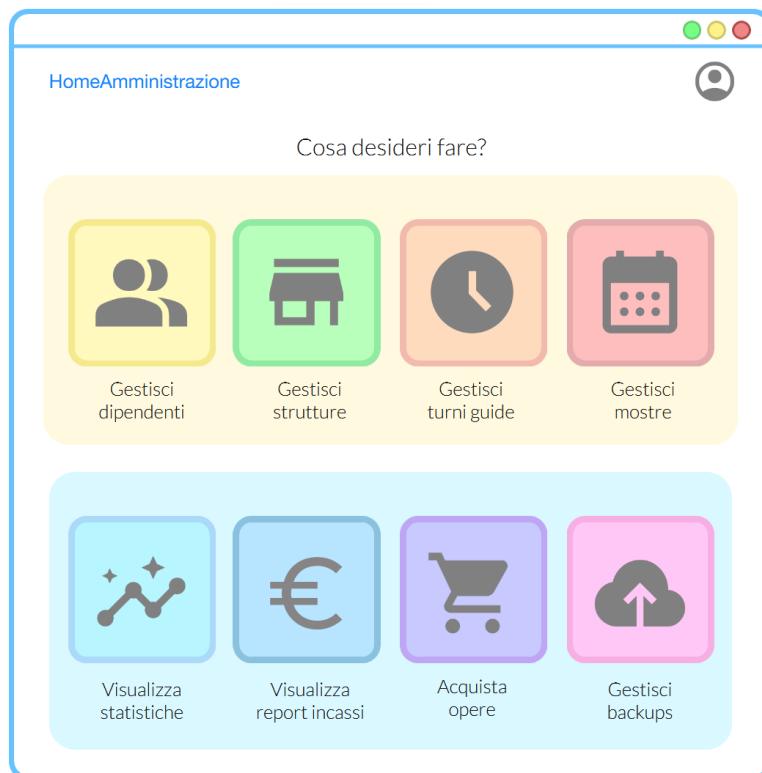


Figura 90: VistaHomeAmministrazione

The mockup shows the "HomeReception → Account" screen. It features a back arrow at the top right. The main content includes a "Dettagli account" section with five entries: "Mario Rossi" (profile icon), "01/01/2000" (date of birth icon), "Maschio" (gender icon), "01/01/2020" (date icon), and "mariorossi@gmail.com" (email icon). Below this is an "Occupazione" section with two entries: "Operatore al pubblico" (operator icon) and "01/01/2020" (date icon). At the bottom is a blue "Logout" button.

Figura 91: VistaAccount

The mockup shows the "Home → Login" screen. It features a back arrow at the top right. The main content includes a "Username:" field with a user icon and a placeholder "Username", and a "Password:" field with a key icon and a redacted placeholder. At the bottom is a large blue "Login" button.

Figura 92: VistaLogin

12.2 Mockups da HomeReception

HomeReception → CompraBiglietto

Tipo biglietto*:	Museo aperto
Tariffa*:	Intera
Supplemento guida:	Cerca guida
Abbonamento:	Verifica abbonamento

Costo totale: **€ 10.00**

Conferma

Figura 93: VistaAcquistaBiglietto



Figura 94: VistaConvalida

HomeReception → VendiOpera

Parametro di ricerca:

Picasso	Autore
---------	--------

Ricerca

Tieni premuto su un'immagine per visualizzare i dettagli

Figura 95: VistaVendiOpera

HomeReception → RiceviDonazione

Titolo opera:	Nascita di Venere
Autore:	Sandro Botticelli
Dimensioni:	172,5 cm × 278,50 cm
Tipo:	Tela
Periodo storico:	Rinascimento
Ubicazione:	Aggiungi

Aggiungi immagine:

Conferma

Figura 96: VistaRiceviDonazione

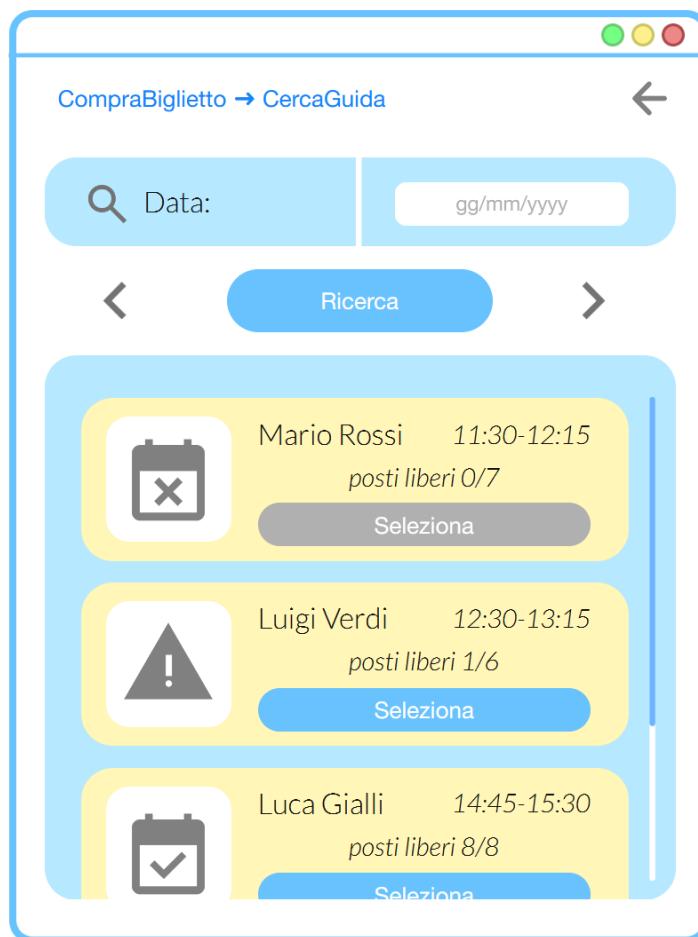


Figura 97: VistaCercaGuida

The screen title is 'CompraBiglietto → DatiCliente'. It contains three input fields in a grid:

	Provenienza:	Provenienza
	Data di nascita:	gg/mm/yyyy
	Sesso:	Gender ▾

At the bottom is a blue 'Conferma' (Confirm) button.

Figura 98: VistaInserisciDatiCliente

The screen title is 'Convalida → InserisciManualmente'. It has a yellow background with the instruction 'Inserisci qui l'identificativo' and a QR code. Below the QR code is the identifier 'C1098293'. At the bottom is a blue 'Conferma' (Confirm) button.

Figura 99: VistaInserisciManualmente

12.3 Mockups da HomeSegreteria

HomeSegreteria → CompraAbbonamento

Dati anagrafici

	Nome*:	Mario
	Cognome*:	Rossi
	Codice fiscale:	AAAAAA00A00A000A

Dati per statistiche

	Provenienza:	Provenienza
	Data di nascita:	gg/mm/yyyy
	Sesso:	Gender ▾

Dettagli abbonamento

	Durata*:	Durata ▾
--	----------	----------

Costo totale: **€ 9,99**

Conferma

Figura 100: VistaCompraAbbonamento

RinnovaAbbonamento → Abbonamento

Dati anagrafici

	Mario
	Rossi
	AAAAAA00A00A000A

Scadenza abbonamento:

	01/01/2000 (99 giorni fa)
--	---------------------------

Figura 101: VistaAbbonamento

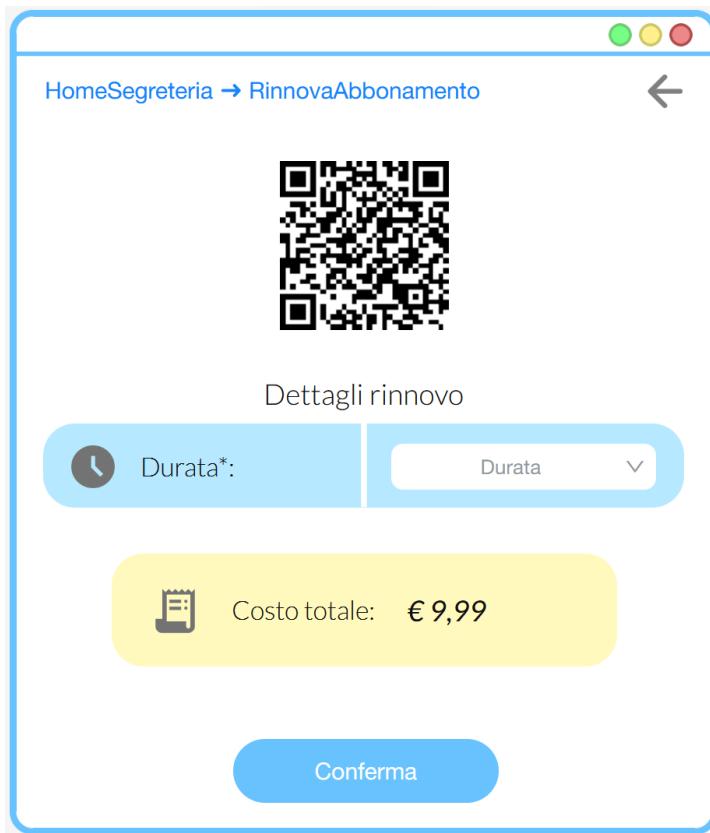


Figura 102: VistaRinnovaAbbonamento

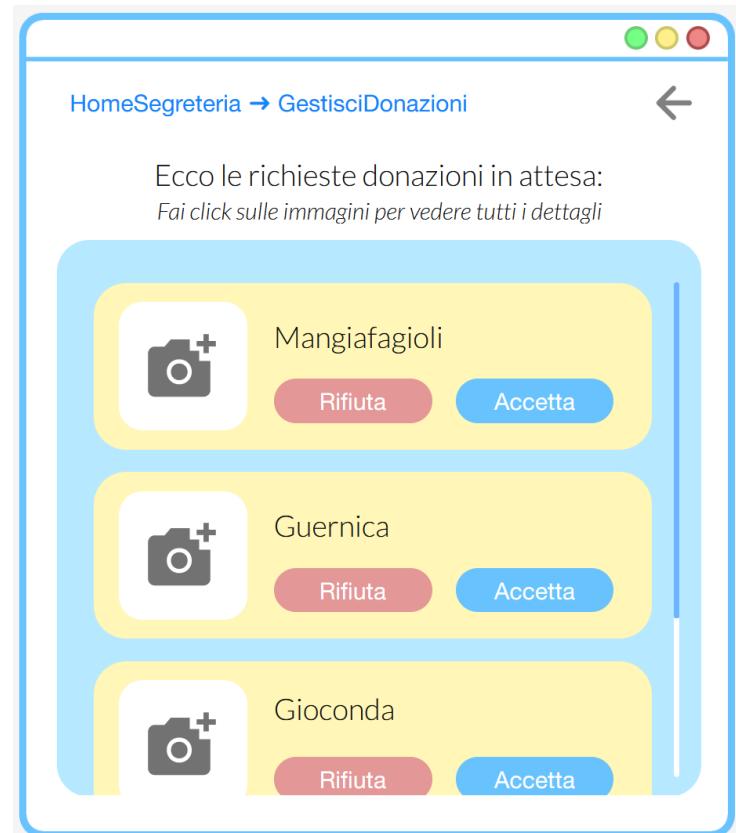


Figura 103: VistaGestisciDonazioni

12.4 Mockups da HomeAmministrazione

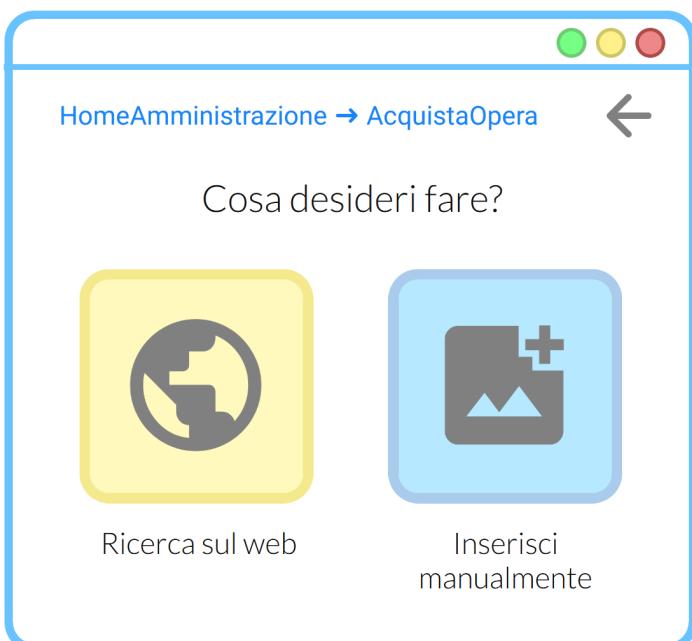


Figura 104: VistaAcquistaOpera

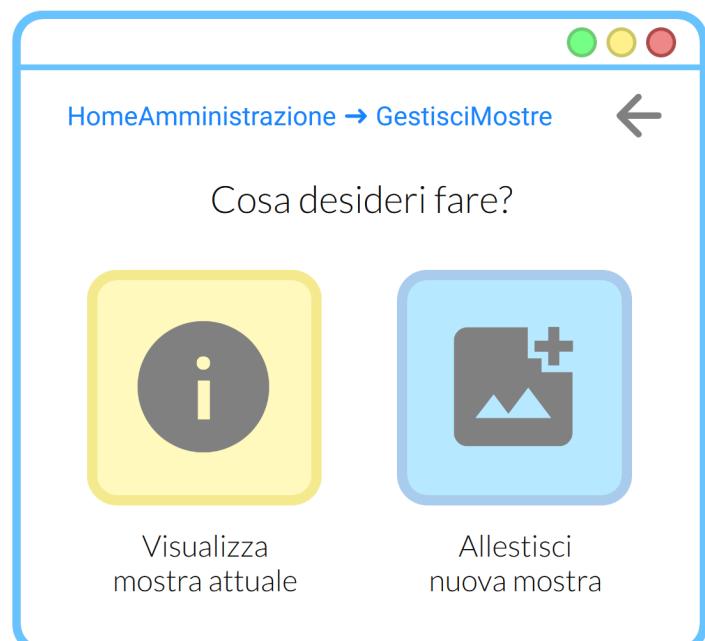


Figura 105: VistaGestisciMostre

HomeAmministrazione → GestisciDipendenti

Ecco la lista dei dipendenti del museo:
Fai click sulle immagini per vedere tutti i dettagli

	Mario Rossi	Licenzia	Promuovi
	Luigi Verdi	Licenzia	Promuovi
	Luca Gialli	Licenzia	Promuovi

Assumi



Figura 106: VistaGestisciDipendenti

GestisciDipendenti → Assumi

Dati anagrafici

	Nome*:	Mario
	Cognome*:	Rossi
	Data di nascita*:	gg/mm/aaaa
	Sesso*:	Gender

Dettagli lavoro

	Impiego*:	Segretario
---	-----------	------------

Conferma



Figura 107: VistaAssumi

GestisciPostiLavoro → ModificaPostoLavoro

Dati generali:

	Nome*:	Segreteria A
	Piano*:	- 1 +
	Capienza*:	- 7 +

Conferma



Figura 108: VistaGestisciDipendenti

CercaGuida → ModificaTurnoGuida

Turno guida:

	Guida:	Cambia
	Capienza:	- 10 +
	Inizio:	12 : 30
	Durata (min):	75

Conferma



Figura 109: VistaAssumi



Figura 110: VistaGestisciPostiLavoro



Figura 111: VistaReportIncassi



Figura 112: VistaSelezioneDipendente

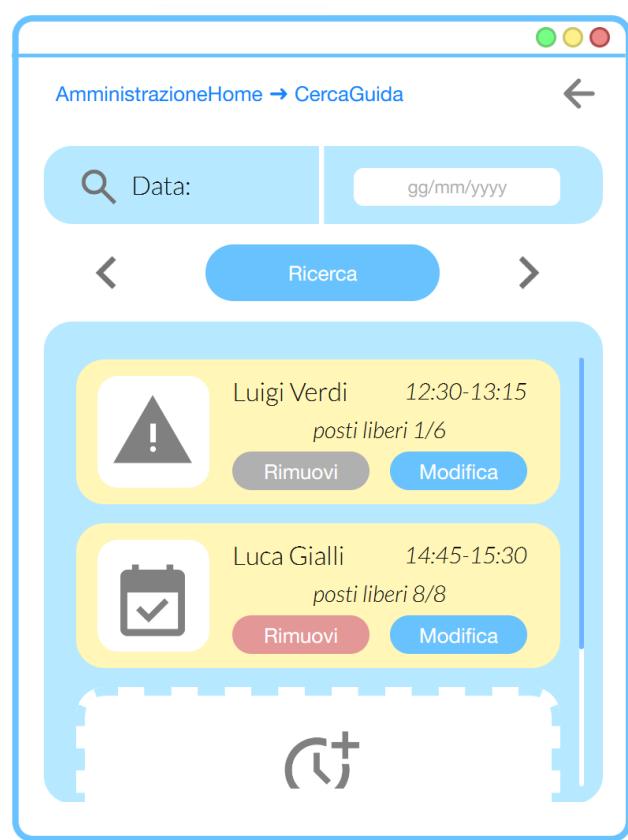


Figura 113: VistaGestisciTurniGuide

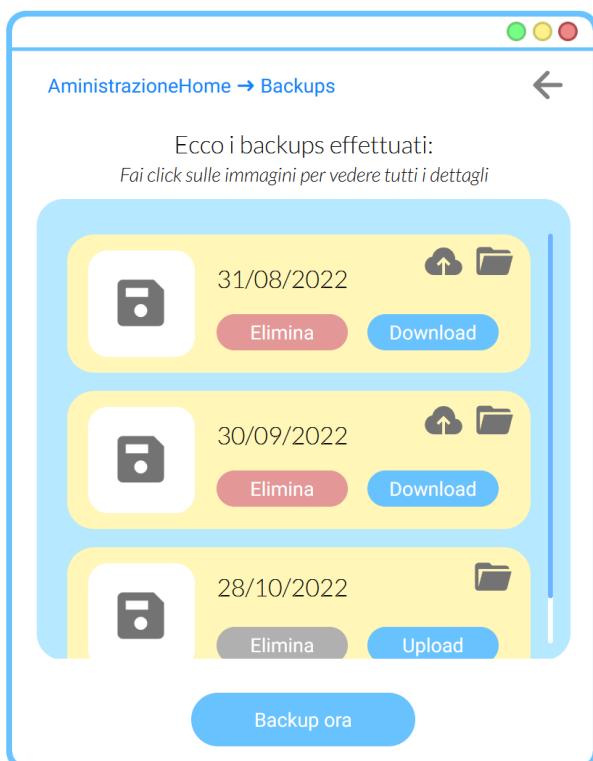


Figura 114: VistaBackup



Figura 115: VistaOpera

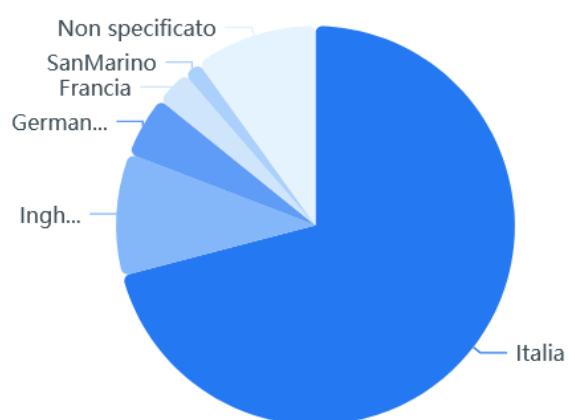
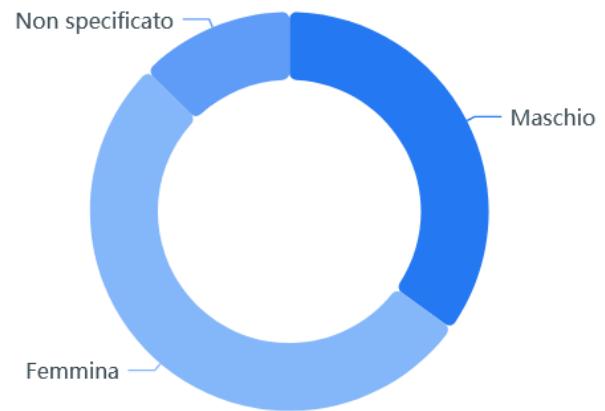


Figura 116: VistaStatistiche



Figura 117: VistaAllestisciMostra

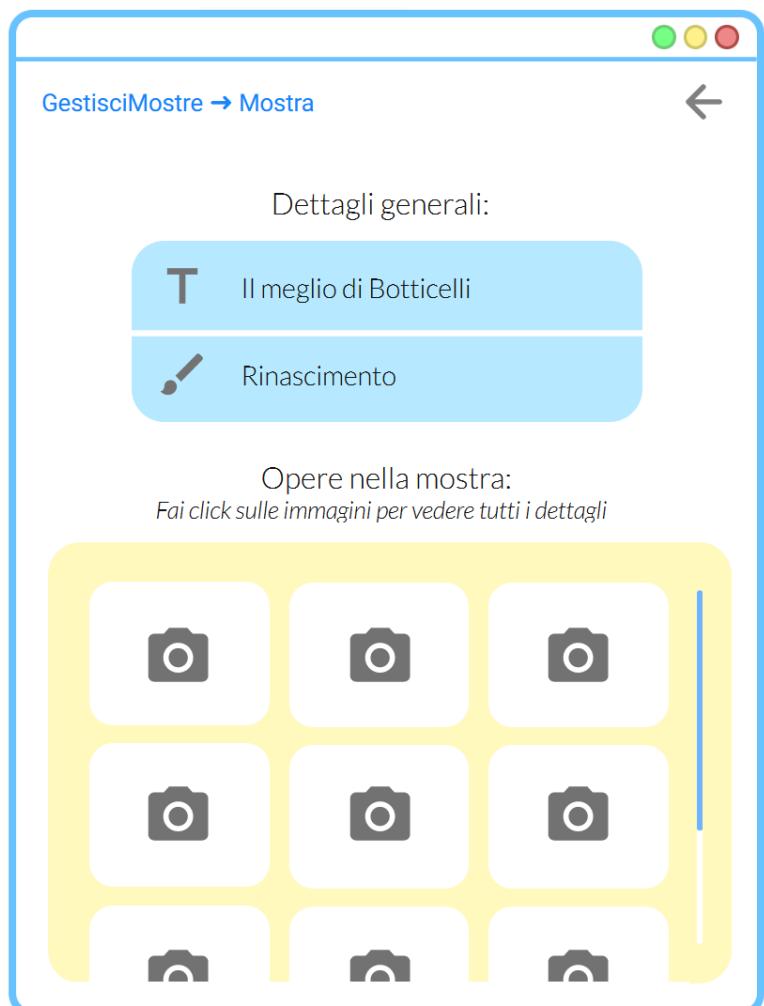


Figura 118: VistaMostra

12.5 Altri mockups

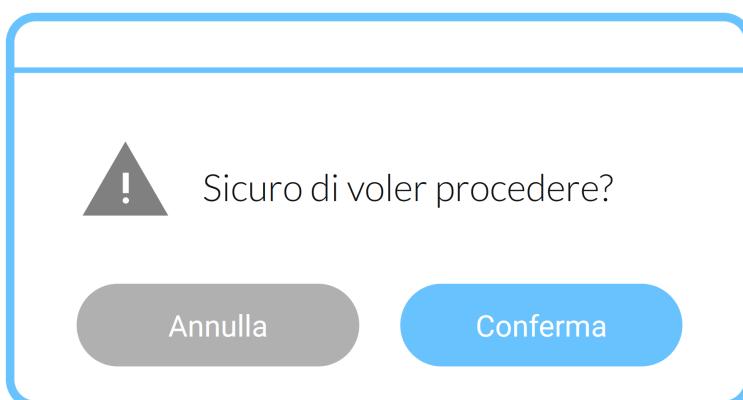


Figura 119: VistaYesNoMessageBox

13 Diagramma di deployment

Iniziamo ora la fase di implementazione con la realizzazione del diagramma di deployment. Esso rappresenta come il software viene distribuito sulle risorse hardware. Si tenga presente che il presente sistema viene implementato in **Python 3** e **non è multipiattaforma**, ma disponibile solo per il sistema operativo **Windows**.

13.1 Diagramma di Deployment

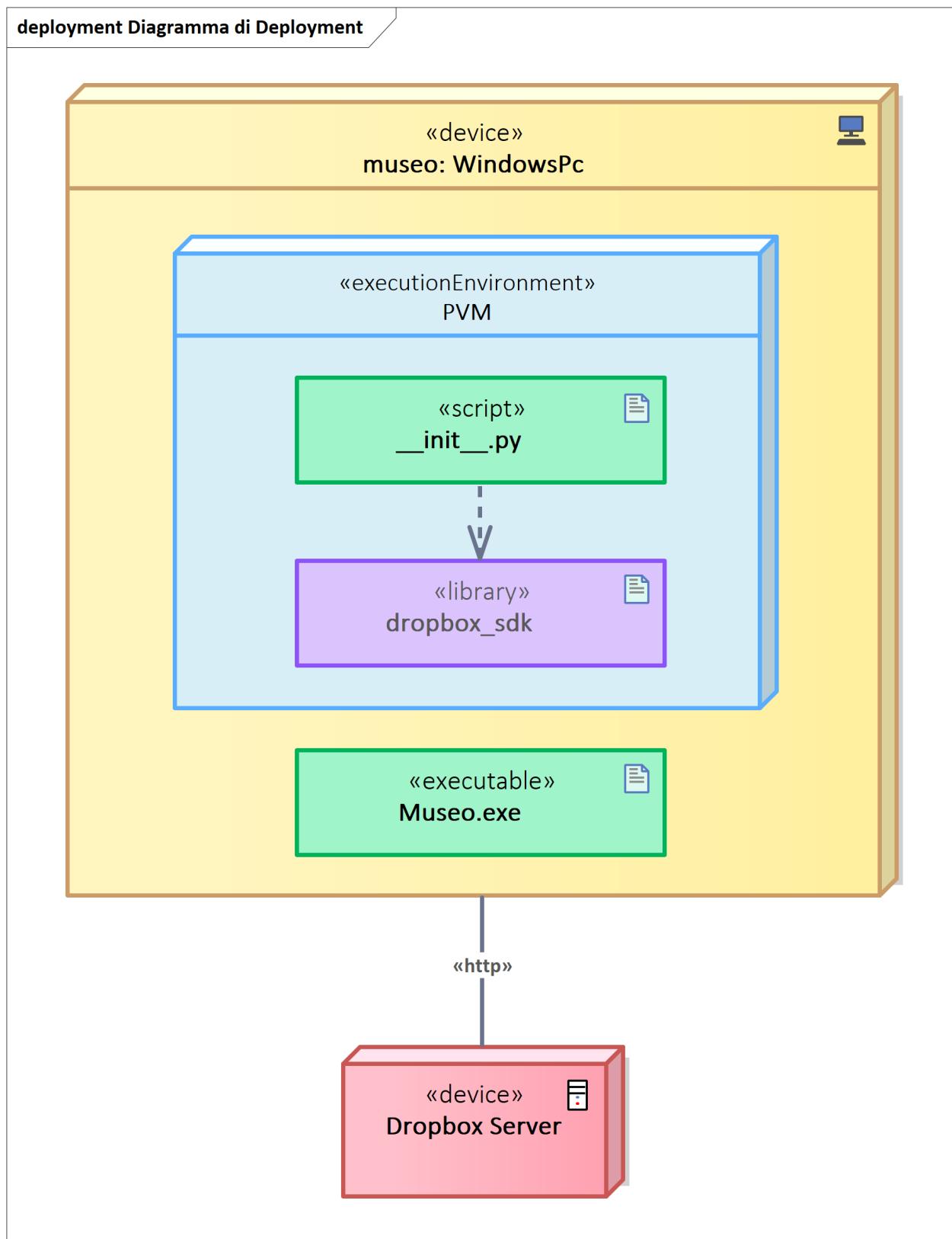


Figura 120: Diagramma di Deployment

14 Implementazione

Nella fase di implementazione è stato usato *git* per *versioning* del *software*, il lettore può trovare il *software* alla seguente *repository GitHub*:

- <https://github.com/MrPio/MuseoOmero.git>.
-

14.1 Flusso di lavoro

Nella prima parte della fase di implementazione, il flusso di lavoro è stato diviso in 2 attività eseguite in parallelo:

- **Creazione delle viste** con il *framework Qt* e l’ausilio del software *Qt Designer* e di *Wondershare Mockitt* (il software utilizzato per la realizzazione dei mockitt) per l’**esportazione dei codici css**.

Il vantaggio nell’utilizzo di *Qt Designer* è stato quello di poter costruire **interattivamente** (e non direttamente tramite codice Python) le finestre e i loro elementi; a quest’ultimo, nel caso in cui non fossero elementi statici, è stato assegnato un opportuno *object name*, attraverso il quale è stato poi possibile **referenziarli da codice** e dunque **modificarli dinamicamente runtime**.

Occorre evidenziare il fatto che, poiché quello di realizzare una **grafica piacevole** è uno dei requisiti non funzionali, è stato fondamentale in primis realizzare i *templates* degli elementi grafici ricorrenti nelle viste, come ad esempio la finestra stessa, il **button**, la **checkbox**, la **lineEdit**, e la **label**, aggiungendo del **css** agli elementi di default offerti dal *Qt*.

Tutte queste viste sono state salvate in un’apposita cartella insieme al file “*.qrc*” contenente le icone utilizzate in esse prese dal *Google material*.

Ciascun file, come si vedrà in sequito, è stato poi caricato dinamicamente dalla libreria ***Pyuic*** di *Qt*.

- **Implementazione dei metodi delle classi del Model** e **verifica** mediante test (facilitati dall’utilizzo del **framework PyUnit**) degli stessi.

Lo scheletro del progetto è stato **interamente esportato dal software di modellazione UML** utilizzato per il progetto (*Enterprise Architect* v15.2) che, a partire dal diagramma delle **classi** di progettazione, una volta **specificato Python come linguaggio** di programmazione per l’implementazione, **ha generato**, in appositi files “*.py*”, tutte le **classi**, gli **attributi** con le relative **visibilità**, e le **firme dei metodi**.

Il **body** di quest’ultimi è stato poi **implementato manualmente** per far sì che il software rispettasse i diagrammi visti nei precedenti capitoli.

Una volta concluse queste due attività, il team di implementazione si è sincronizzato per procedere alla terza ed ultima fase:

- **Implementazione delle coppie View-Controller**

A questo punto, disponendo di tutti i files “*.ui*” delle viste, ci siamo limitati a definire, per ogni vista, i **getters per gli elementi grafici dinamici** (qui l’utilizzo del loro *object name*) e il **controller** con i suoi metodi.

Terminano questa terza attività i **numerosi test sulle viste svolti in parallelo** dalle varie parti del team, testando la robustezza dei **vincoli di integrità dei dati** e la corretta interazione del *frontend* con il *Model*.

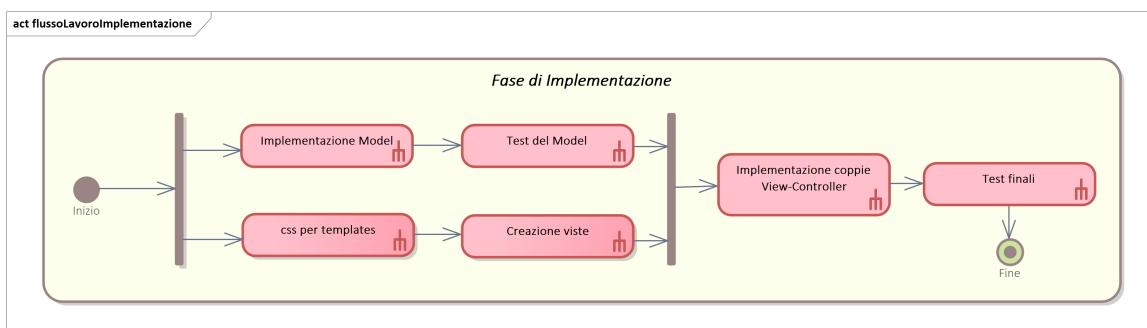


Figura 121: Flusso di lavoro dell’implementazione

14.2 Motivazioni

La motivazione di questa suddivisione del flusso di lavoro risiede in parte dietro un approssimato **studio preliminare** riguardo lo **sforzo necessario nel coordinamento** tra le varie parti nel lavoro parallelo. Utilizzando l'**equazione risorse-sforzo**, e facendo una stima dei 4 parametri che la compongono, si ottiene il seguente risultato non molto distante dal caso reale (considerando che il flusso dell'implementazione è durato in totale circa **21 giorni**, **7 giorni** per le **prime due** attività parallele e **14 giorni** per la **terza** attività):

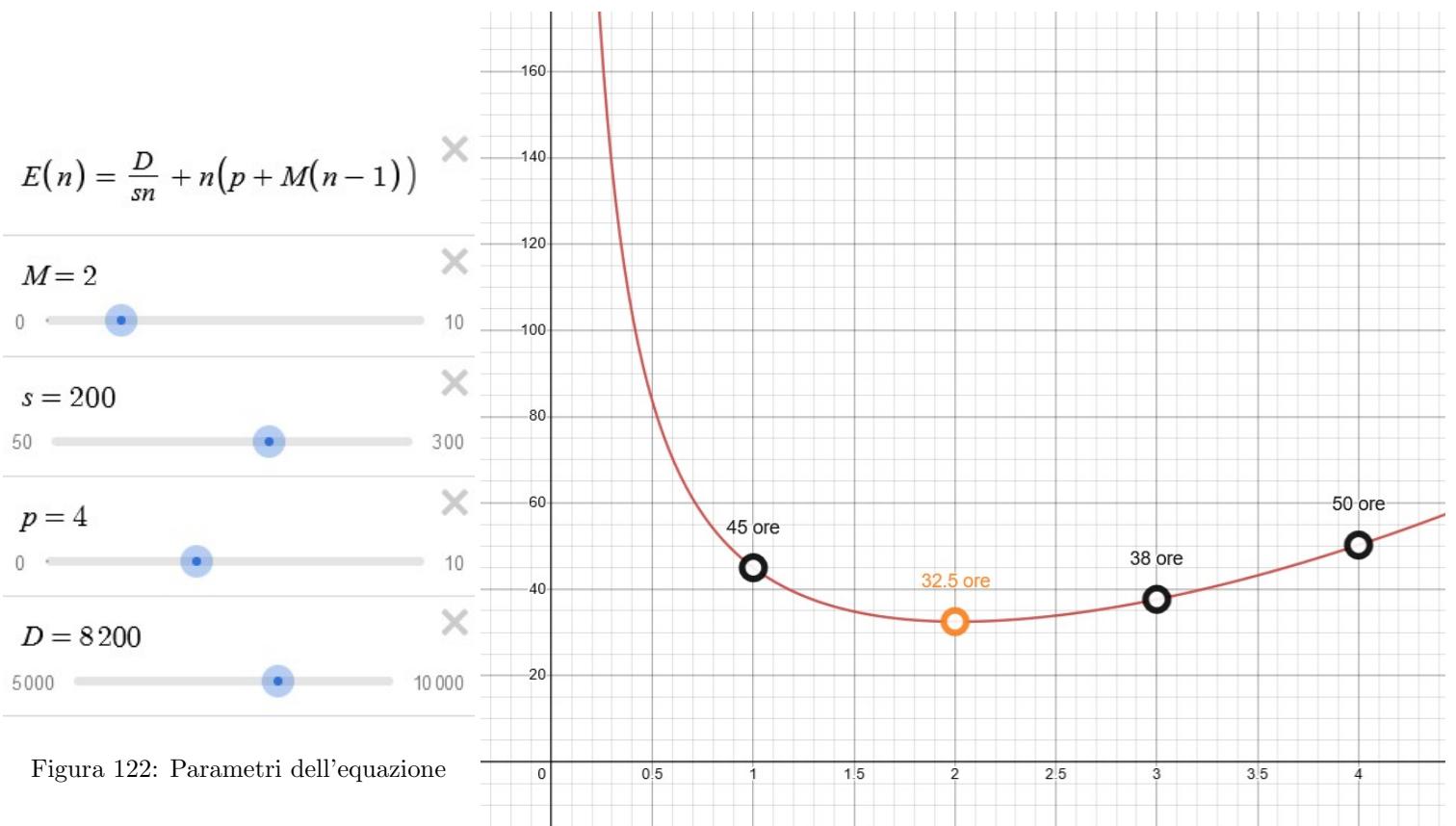


Figura 122: Parametri dell'equazione

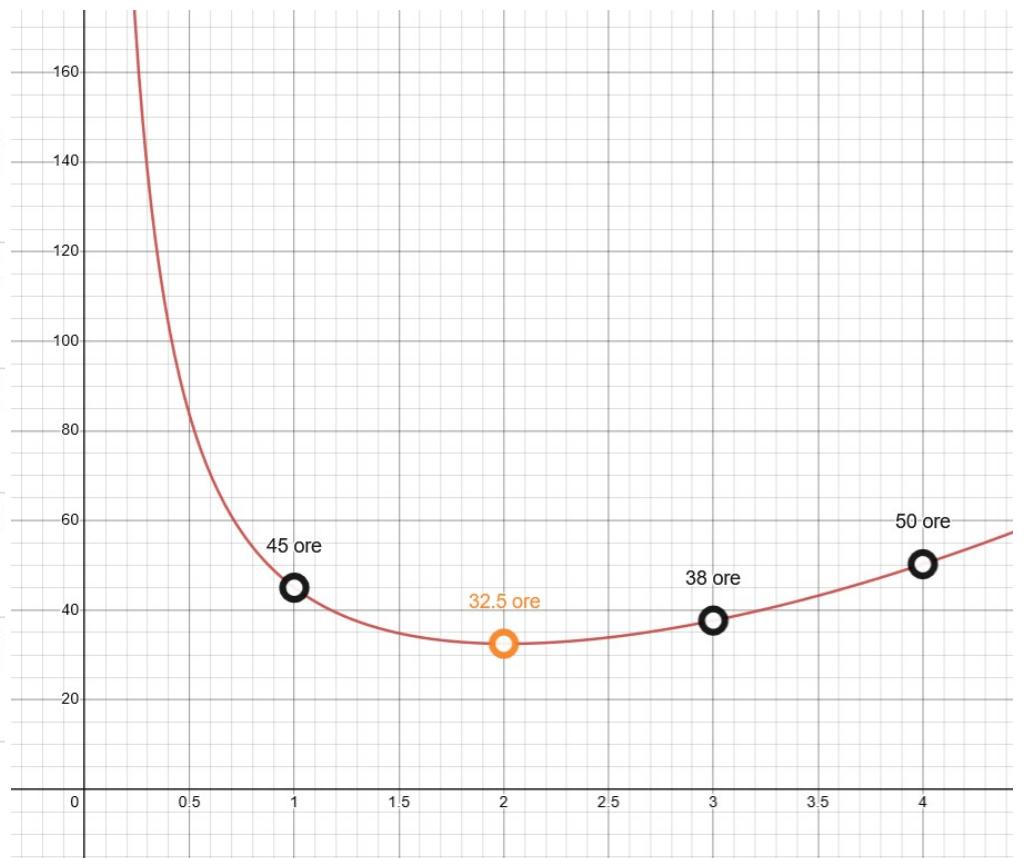


Figura 123: Grafico risorse-sforzo

Si deduce dal grafico che **la scelta ottimale** risiede nel minimo discreto della funzione, ovvero:

$$n : -D/(sN^2) + p - M + 2Mm = 0$$

Quindi la scelta ottimale è l'**impiego di 2 componenti** del gruppo per l'implementazione del codice. In realtà questo studio è stato rispettato fino alla **conclusione delle prime due attività parallele** dove il team si è diviso in due parti, **diminuendo così i tempi di coordinamento**. Nella terza attività, seppur rallentati dai tempi di sincronizzazione giornalieri, abbiamo deciso di **riunirci** per permettere a tutti i componenti di "mettere mano" al codice.

14.3 Directories di implementazione

Ad implementazione completata, **rispettando i package già visti in progettazione**, queste sono le *directory* con i moduli e files *Python* creati:

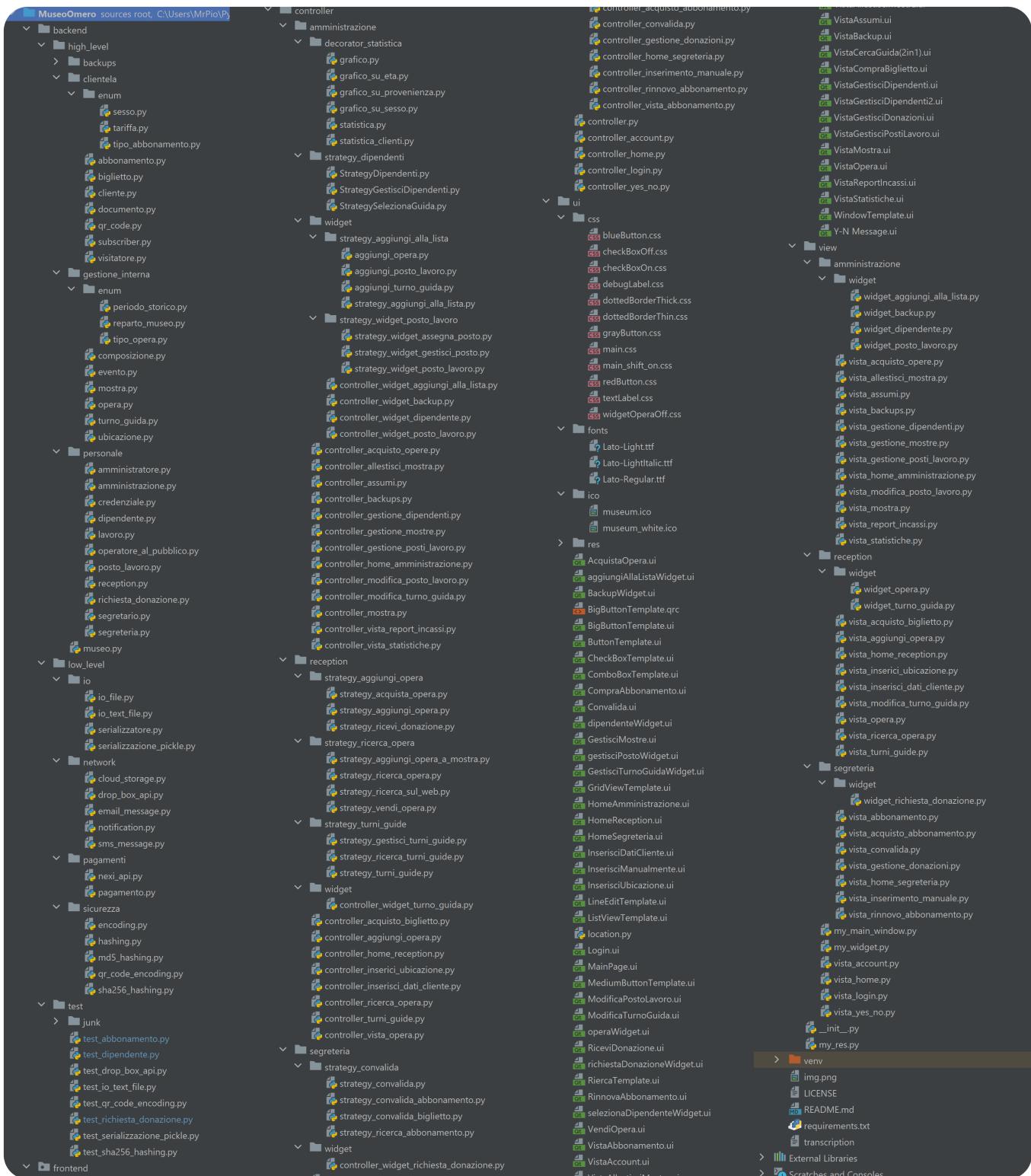


Figura 124: Directory di implementazione

14.4 Requisiti

Sono stati utilizzati all'interno del progetto i seguenti pacchetti, installabili con `pip`, una volta posizionatisi nella *root* del progetto, tramite il comando

```
pip install -r requirements.txt
```

proprio come si legge nel *README.md* della repository.

```
dropbox==11.34.0
fontTools==4.38.0
opencv_python==4.6.0.66
Pillow==9.3.0
PyQt5==5.15.7
python_dateutil==2.8.2
qrcode==7.3.1
tkinterDnD==0.0.0
winotify==1.1.0
```

Figura 125: Requirements.txt

15 Test

I test realizzati coprono i **comportamenti più salienti delle classi di *backend***; viene testato innanzitutto il funzionamento delle operazioni più comuni delle **classi di *business***, poi quello delle classi di **basso livello**.

15.1 Test delle classi di alto livello

Testiamo innanzitutto il funzionamento degli aspetti più cruciali delle classi di alto livello, testeremo poi nel capitolo 15.2 le classi di basso livello **le quali dipendono dalle prime grazie all' inversione delle dipendenze.**

Le classi testate in questo capitolo sono le seguenti:

- Abbonamento
- Dipendente
- RichiestaDonazione

15.1.1 TestAbbonamento

Dopo aver creato un abbonamento **valido**, uno **scaduto** e uno **non pagato**, proviamo a convalidarli tutti, e testiamo voi valore booleano di ritorno; **ci aspettiamo che solo l'abbonamento valido comporti un successo dell'operazione.**



```

1 class TestAbbonamento(TestCase):
2
3     def setUp(self) -> None:
4         self.abbonamento_scaduto = Abbonamento(
5             dataRilascio=datetime(2022, 1, 1),
6             tipo=TipoAbbonamento.MENSILE
7         )
8         self.abbonamento_scaduto.pagato = True
9         self.abbonamento_valido = Abbonamento(
10            dataRilascio=datetime(2022, 10, 16),
11            tipo=TipoAbbonamento.ANNUALE
12        )
13         self.abbonamento_valido.pagato = True
14         self.abbonamento_non_pagato = Abbonamento(
15            dataRilascio=datetime(2022, 10, 16),
16            tipo=TipoAbbonamento.ANNUALE
17        )
18         self.abbonamento_non_pagato.pagato = False
19
20     def test_convalida(self):
21         self.assertEqual(
22             first=self.abbonamento_scaduto.convalida(),
23             second= False
24         )
25         self.assertEqual(
26             first=self.abbonamento_valido.convalida(),
27             second= True
28         )
29         self.assertEqual(
30             first=self.abbonamento_non_pagato.convalida(),
31             second= False
32         )

```

Figura 126: TestAbbonamento

15.1.2 TestDipendente

Creiamo un'istanza di **dipendente**, gli assegniamo delle **credenziali** e creiamo anche un posto **lavoro in amministrazione**, infine creiamo due lavori uno di tipo **amministratore** l'altro tipo **segretario**.

Lo scopo del test è quello di verificare il corretto funzionamento della gestione dei lavori dei dipendenti; non deve essere possibile quindi **assegnare un segretario ad un posto di lavoro in amministrazione**, così come non deve essere possibile **assumere un dipendente che ha già un lavoro un licenziarne un altro che non ne ha uno**.

```

1 class TestDipendete(TestCase):
2
3     def setUp(self) -> None:
4         self.username = 'Mario145'
5         self.password = 'superPassword'
6         self.hashing = SHA256Hashing()
7
8         self.dipendente = Dipendente(
9             nome="Mario",
10            cognome="Rossi",
11            dataNascita=datetime(2000, 1, 1),
12            credenziale=Credenziale(
13                username=self.username,
14                password=self.password,
15                hashing=self.hashing,
16            )
17        )
18        self.amministrazione=Amministrazione(
19            nome='amm1',
20            piano=2,
21            numPostazioni=5,
22            descrizione='test',
23        )
24        self.amministratore=Amministratore(
25            stipendio=2200,
26            numPostazione=0,
27        )
28        self.segretario=Segretario(
29            stipendio=1800,
30            numPostazione=5,
31            sportelloAssegnato=0,
32    ) def

```

Figura 127: TestDipendete 1/2

```
1  def test_eta(self):
2      self.assertEqual(
3          first=self.dipendente.calcolaEta(),
4          second=22
5      )
6
7  def test_credenziale(self):
8      self.assertEqual(
9          first=self.dipendente.credenziale.username,
10         second=self.username
11     )
12
13     self.assertEqual(
14         first=self.dipendente.credenziale.enc_password,
15         second=self.hashing.hash(self.password)
16     )
17
18  def test_lavoro(self):
19      with self.assertRaises(Exception) as context:
20          self.dipendente.assumi(
21              self.segretario, self.amministrazione)
22          self.assertTrue('Non posso assumere un Segretario
23 in una Amministrazione!' in context.exception.args)
24
25      with self.assertRaises(Exception) as context:
26          self.dipendente.licenzia(
27              'licenziamento per giusta causa'
28          )
29          self.assertTrue('Non posso licenziare un
30 dipendente che non ha un lavoro,
31 prima devi assumerlo!' in context.exception.args)
32
33      self.assertTrue(self.dipendente.assumi(
34          self.amministratore, self.amministrazione))
```

Figura 128: TestDipendete 2/2

15.1.3 TestRichiestaDonazione

Creiamo una richiesta di donazione e **verifichiamo il suo ciclo di vita**, sia nel caso in cui essa venga **rifiutata**, sia nel caso in cui essa venga **accettata** dalla segreteria.



```
1 class TestRichiestaDonazione(TestCase):
2
3     def setUp(self) -> None:
4         self.ubicazione = Ubicazione()
5         self.opera = Opera()
6         self.richiesta = RichiestaDonazione(
7             opera=self.opera,
8             ubicazioneProvvisoria=self.ubicazione,
9             notification=EmailMessage(''),
10        )
11
12     def test_gestione_richiesta(self):
13         self.assertEqual(
14             first=self.richiesta.accettata,
15             second=False
16         )
17
18         self.richiesta.accetta(self.ubicazione)
19         self.assertEqual(
20             first=self.richiesta.accettata,
21             second=True
22         )
23
24         self.richiesta.rifiuta()
25         self.assertEqual(
26             first=self.richiesta.accettata,
27             second=False
28         )
```

Figura 129: TestRichiestaDonazione

15.2 Test delle classi di basso livello

Testiamo qui il funzionamento delle classi di basso livello, per ciascuna interfaccia fornita verifichiamo che le classi che la realizzano rispettino il contratto richiesto dall'interfaccia stessa in maniera corretta, facendo esclusivamente e correttamente ciò che il cliente si aspetta da esse.

Le classi testate in questo capitolo sono le seguenti:

- IOTextFile
- SerializzazionePickle
- DropBoxAPI
- SHA256Hashing
- QRCodeEncoding

15.2.1 TestIOTextFile

Definiamo un *filename*, un **testo di prova** e una **directory** dove creare il file, quindi lo **scriviamo** e lo **leggiamo** per verificare il corretto funzionamento della scrittura su disco.



```

1 class TestIOTextFile(TestCase):
2
3     def setUp(self) -> None:
4         self.io_text_file = IOTextFile()
5         self.content = 'file di test'
6         self.path = 'junk/TestIOTextFile/'
7         self.filename = 'test.pickle'
8
9     def test_salva_file(self):
10        self.assertTrue(
11            expr=self.io_text_file.salvaFile(
12                content=self.content,
13                path=self.path,
14                filename=self.filename,
15            )
16        )
17
18    def test_leggi_file(self):
19        self.assertEqual(
20            first=self.io_text_file.leggiFile(
21                path=self.path + self.filename,
22            ),
23            second=self.content,
24        )

```

Figura 130: TestIOTextFile

15.2.2 TestSerializzazionePickle

Creiamo un oggetto di tipo biglietto e lo **serializziamo** su disco, poi l'ho **deserializzato** per verificare che i **due oggetti corrispondano nei loro valori**.

```
1 class TestSerializzazionePickle(TestCase):
2
3     def setUp(self) -> None:
4         self.serializzatore_pickle = SerializzazionePickle()
5         self.biglietto_test = Biglietto(
6             dataRilascio=datetime.datetime.now(),
7             reparto=RepartoMuseo.MOSTRA,
8             tariffa=Tariffa.RIDOTTO,
9         )
10        self.path = 'junk/TestSerializzazionePickle/'
11        self.filename = 'biglietto123.pickle'
12
13    def test_serializza(self):
14        self.assertTrue(
15            expr=self.serializzatore_pickle.serializza(
16                obj=self.biglietto_test,
17                path=self.path,
18                filename=self.filename,
19            ),
20        )
21
22    def test_deserializza(self):
23        self.assertEqual(
24            first=self.biglietto_test.tariffa,
25            second=self.serializzatore_pickle.deserializza(
26                path=self.path + self.filename,
27                ).tariffa,
28        )
```

Figura 131: TestSerializzazionePickle

15.2.3 TestDropBoxAPI

Testiamo le operazioni *CRUD* sul cloud di Dropbox; a tal fine facciamo l'*upload* di un file, verifichiamo la sua presenza sul cloud, proviamo a fare il *download*, e infine lo eliminiamo dal cloud.

```
● ● ●
```

```
1 class TestDropBoxAPI(TestCase):
2
3     def setUp(self) -> None:
4         self.dropbox_api = DropBoxAPI()
5         self.path = 'junk/TestDropBoxAPI/'
6         self.filename = 'OneOfMyTurns-PinkFloyd.flac'
7         self.filename_downloaded = 'OneOfMyTurns'
8             '-PinkFloyd [downloaded].flac'
9
10    def test_download(self):
11        if os.path.exists(self.path + self.filename_downloaded):
12            os.remove(self.path + self.filename_downloaded)
13        self.assertTrue(
14            expr=self.dropbox_api.download(
15                cloudPath=DropBoxAPI.cloud_root_dir + self.filename,
16                localPath=self.path + self.filename_downloaded
17            ),
18        )
19        self.assertTrue(
20            os.path.exists(self.path + self.filename_downloaded))
21
```

Figura 132: TestDropBoxAPI 1/2

```
1  def test_upload(self):
2      self.assertTrue(
3          expr=self.dropbox_api.upload(
4              path=self.path,
5              filename=self.filename,
6          ),
7      )
8
9  def test_list_file(self):
10     print(
11         'files list->',
12         self.dropbox_api.listFiles(
13             cloudDirectory=DropBoxAPI.cloud_root_dir,
14         ),
15     )
16
17 def test_list_file2(self):
18     print(self.dropbox_api.listFiles(
19         DropBoxAPI.cloud_root_dir + 'backups/'))
20
21 def test_delete_file(self):
22     self.assertTrue(
23         self.dropbox_api.deleteFile(
24             DropBoxAPI.cloud_root_dir + 'backups/test',
25         )
26     )
```

Figura 133: TestDropBoxAPI 2/2

15.2.4 TestSHA256Hashing

Facciamo l'*hashing* di una password scritta in *plain text* e verifichiamo che essa corrisponde alla password originaria criptata utilizzando l'algoritmo *SHA256*.

```
1 class TestSHA256Hashing(TestCase):
2
3     def setUp(self) -> None:
4         self.hashing: Hashing = SHA256Hashing()
5         self.plain = 'password_super_secreta'
6
7     def test_hash(self):
8         self.assertEqual(
9             first=self.hashing.hash(self.plain),
10            second='4c2b929779ccab43c5b52fb6aaa14258e05
11                           ca3dd7efaaf6113244c6fd79f7161',
12         )
```

Figura 134: TestSHA256Hashing

15.2.5 TestQRCodeEncoding

Proviamo a **creare un codice QR** a partire da un testo di prova, poi lo decodifichiamo per verificare il messaggio corrisponde a quello originario.

```
1 class TestQRCodeEncoding(TestCase):
2
3     def test_encode(self):
4         img = qrcode.make('Ciao, questo è un QrCode di prova')
5         img.save("junk\\test_QrCode_image\\QrCode.png")
6
7     def test_decode(self):
8         image = cv2.imread(
9             "junk\\test_QrCode_image\\QrCode.png")
10        print(cv2.QRCodeDetector().detectAndDecode(image)[0])
```

Figura 135: TestQRCodeEncoding

16 Conclusioni

Concludiamo la tesina con un **elenco dei software utilizzati e link utili** per realizzare il presente progetto e rimandiamo il lettore alla repository contenente il file del progetto creato con il software *Enterprise Architect*.

- <https://github.com/SbattellaMattia/SoftwareEngineering>
-

16.1 Link Utili

Per realizzare questo progetto, sono stati utilizzati i seguenti software:

- **Overleaf**

Per la scrittura della tesi in [LATEX](#)

- **EnterpriseArchitect (v.15.2)**

Per la realizzazione del progetto utilizzando gli strumenti dell'UML 2.0. Tutti i diagrammi visti fino ad ora sono stati creati con questo software.

- **Git & GitHub**

Per il versioning, non solo del codice in fase di implementazione, ma anche della repository del progetto, ripetiamo qui i link alle due repositories:

- <https://github.com/SbattellaMattia/SoftwareEngineering>
- <https://github.com/MrPio/MuseoOmero>

- **Dive into Design Patterns**

Da questo libro sono state prese quasi tutte le decisioni e strategie in progettazione, dai principi di programmazione all'utilizzo dei pattern.

- **Wondershare Mockitt**

Per la realizzazione dei prototipi delle interfacce grafiche, implementati successivamente con [QtDesigner](#)

- **QtDesigner**

Per la realizzazione delle viste.

- **Python v.3.10.6**

il linguaggio utilizzato.

- **Jetbrains Pycharm & CodeWithMe**

Per l'implementazione, compilazione e test del progetto

- **Codeimg.io**

Per il rendering delle schermate di codice per i test.

- **Desmos**

Per la realizzazione del grafico risorse-sforzo.

16.2 Dimostrazioni di funzionamento del software

Si invita il lettore interessato alla visione di alcuni **video di demo di esecuzione del software** nella sua ultima versione, consultabili nei *links* seguenti:

- [Demo generale](#),
- [Convalida di un abbonamento](#),
- [Statistiche e report incassi](#),
- [Ripristino backup da cloud](#):

*“I bravi programmatori sanno scrivere.
I migliori sanno riutilizzare.”*

- gli Autori