POLITECNICO DI MILANO
SCUOLA DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Progetto di Ingegneria del Software 2
Travel Dream
DD
(Design Document)

Responsabile:
Prof. Luca Mottola

Progetto di:
Giorgio Conte Matricola n. 815641
Lorenzo Di Tucci Matricola n. 814275
Leonardo Cavagnis Matricola n.816646

ANNO ACCADEMICO 2013-2014

1 Introduzione

In questo documento forniamo una descrizione della struttura concettuale del sistema che implementeremo a supporto delle operazioni di vendita dell'agenzia Travel Dream. Questo documento si basa sulle specifiche descritte nel precedente elaborato di analisi dei requisiti (RASD).

A chi è rivolto. Questo documento è rivolto primariamente al gruppo di lavoro incaricato all'implementazione e alla manutenzione del software.

1.1 Acronimi e abbreviazioni

RASD	Requirementss Analysis and Specification Document
DD	Design Document
PB	Prodotto Base
GL	Gift List
DBMS	DataBase Management System
ER	Entità Relazione

1.2 Panoramica

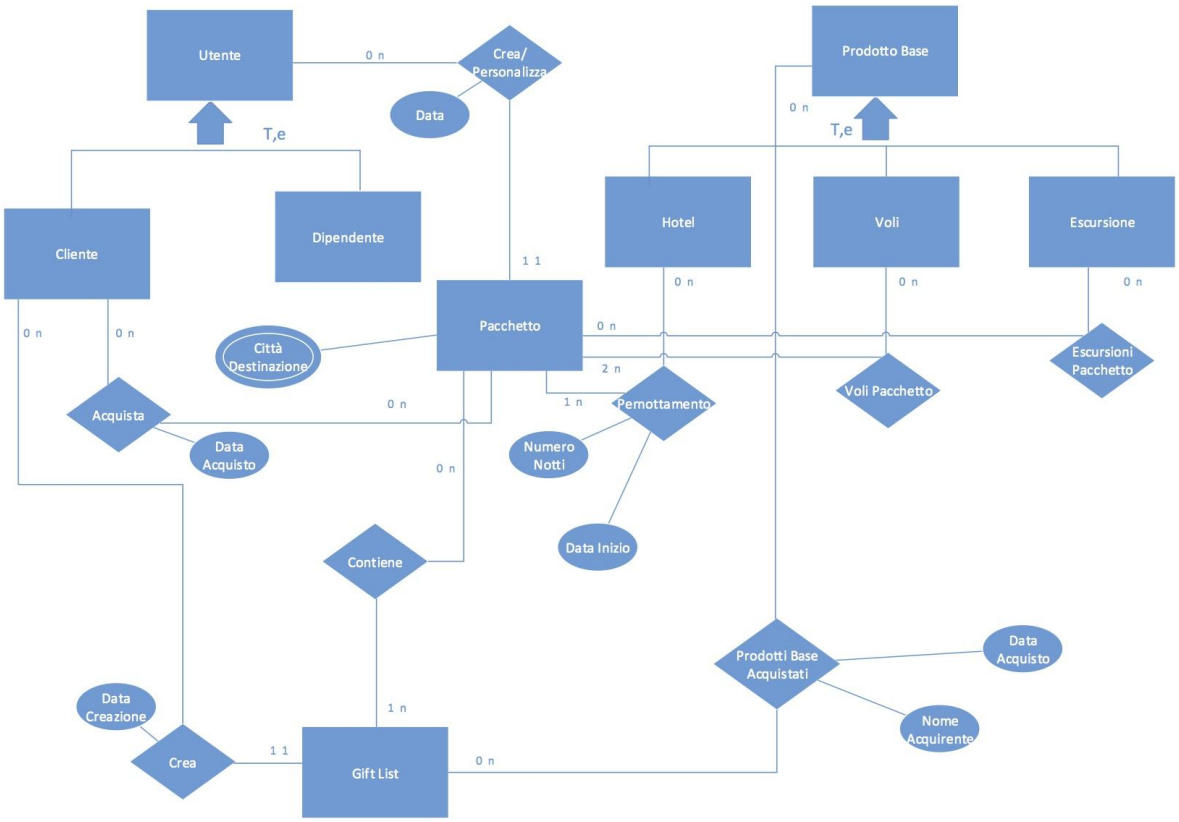
In questo elaborato viene presentata l'architettura e il modello dei dati a supporto dell'applicazione. Nel capitolo seguente è descritto il modello dei dati mediante il modello Entità - Relazione e la relativa traduzione logica.

Nel terzo capitolo, invece, sono presentati l'architettura del sistema, i diagrammi di navigazione e, infine, alcuni diagrammi di sequenza che descrivono l'interazione tra le diverse entità del sistema per fornire alcune delle funzionalità dell'applicazione.

2 Design dei Dati

2.1 Modello concettuale Entità Relazione

In figura presentiamo il modello ER elaborato per memorizzare i dati utili all'applicazione web da sviluppare. Per una maggiore comprensione, gli attributi delle entità sono stati omessi nel modello generico e ogni entità, con i relativi attributi, sarà presentata singolarmente nella sezione seguente.



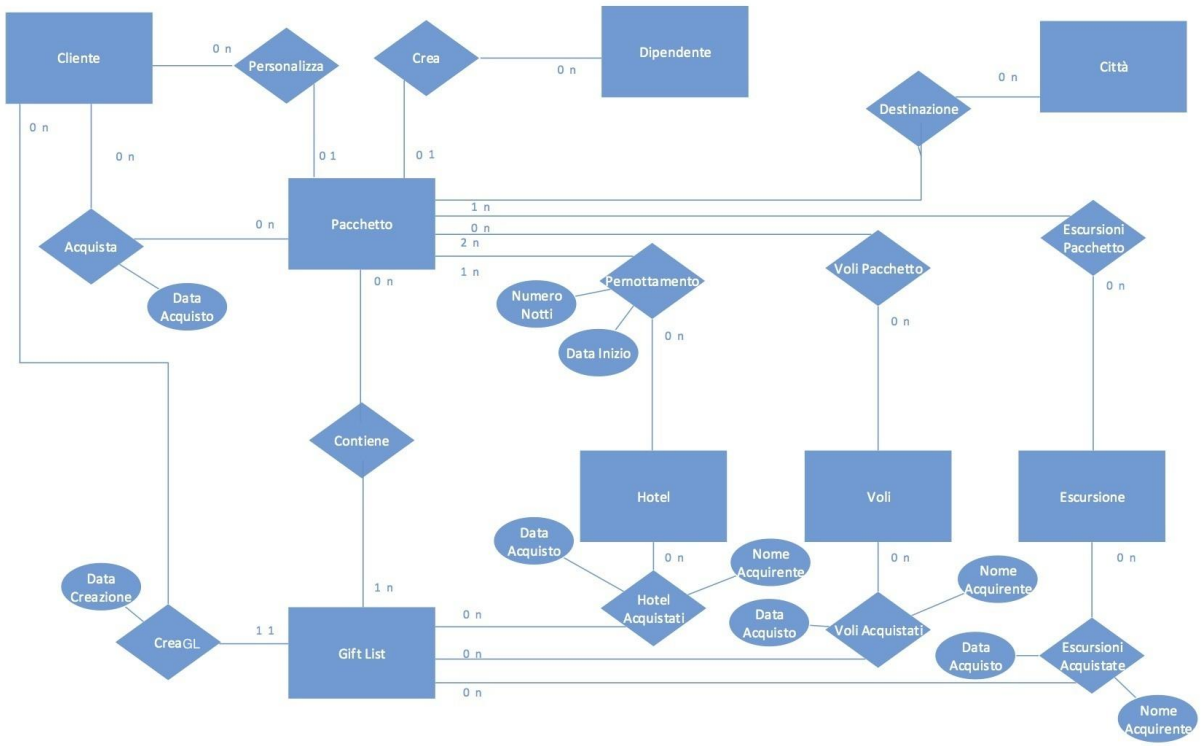
Modello ER

Le principali entità all'interno del sistema sono utente, prodotto base, gift list e pacchetto. Per quanto riguarda gli utenti, il sistema può memorizzare solo le informazioni relative agli utenti registrati (i clienti) e ai dipendenti.
L'entità prodotto base generalizza tutti i prodotti semplici che sono gestiti dall'applicazione web, in particolare poi si specializza in hotel, volo ed escursione. Questa soluzione permette facilmente di estendere il sistema con altri tipi di prodotti base quali ad esempio nuovi mezzi di trasporto.
"Pacchetto" è l'entità centrale di tutto lo schema in quanto rappresenta l'elemento più importante di tutta l'applicazione. Questa è collegata attraverso relazioni molti a molti con le entità "Hotel", "Volo" ed "Escursione" per rappresentarne il contenuto vero e proprio. La relazione "acquista" presente tra "cliente" e "pacchetto" serve per memorizzare tutti i pacchetti acquistati da ogni singolo cliente.
La relazione "Prodotti base acquistati" è necessaria per tenere traccia di tutti gli acquisti fatti sulla gift list creata dal cliente e permette a qualsiasi utente di comprare solo prodotti base, così come specificato dal documento di analisi dei requisiti. Lo schema ER, tuttavia, non evidenzia che si possono acquistare solo prodotti compresi nei pacchetti presenti nella gift list. Per garantire questa proprietà, è possibile definire alcune regole attive (trigger) sulla base di dati in modo che non permettano l'aggiunta di tuple non consistenti.

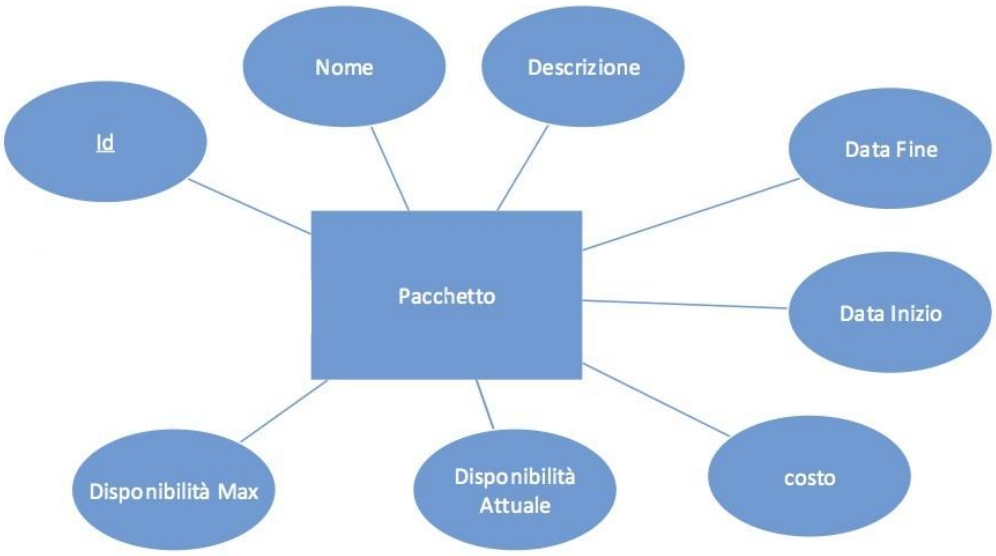
Per tenere traccia di tutte le operazioni effettuate sulla base di dati viene creato un file di log in cui registrare tutte le transazioni effettuate sul sistema. Il file di log è necessario per salvare tutto lo storico delle transazioni sulla base di dati e rintracciare facilmente operazioni errate.

2.2 Ristrutturazione del modello ER

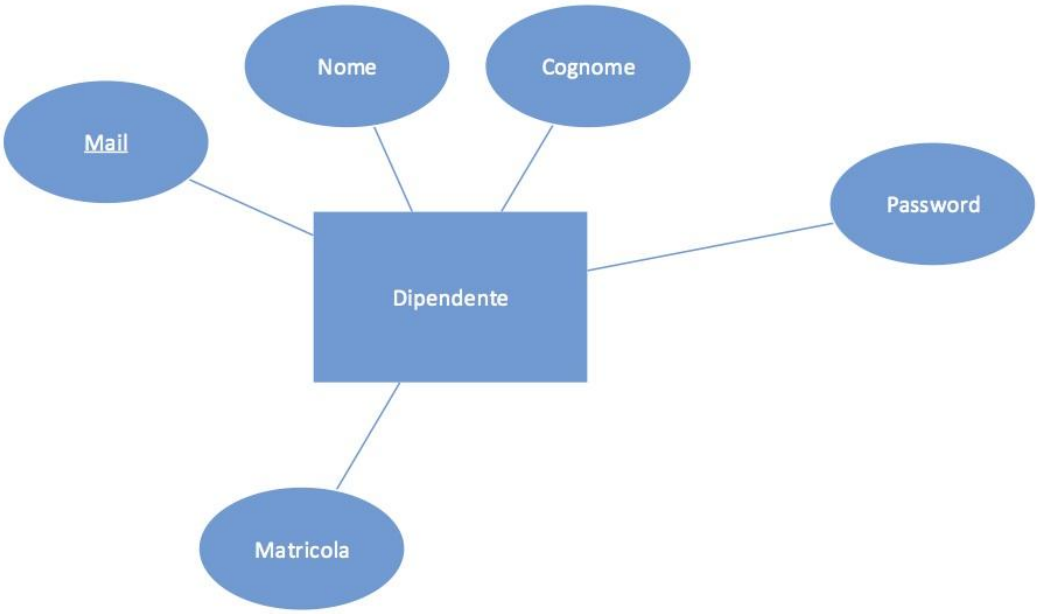
Prima di effettuare la traduzione al modello logico vero e proprio abbiamo semplificato le generalizzazioni presenti nella prima versione del modello ER e l'attributo multiplo "città destinazione" presente nell'entità "pacchetto". La generalizzazione "utente", "cliente" e "dipendente" è ristrutturata con un approccio "top down", gli attributi presenti sull'entità "utente" sono stati aggiunti alle due entità figlie e la relazione "crea/personalizza" è sdoppiata in due relazioni distinte: la relazione "personalizza" tra "cliente" e "pacchetto" e "crea" tra "dipendente" e "pacchetto". Le due relazioni create hanno cardinalità "0,1", tuttavia almeno una delle due relazioni deve esistere. Un pacchetto personalizzato viene salvato nella tabella relativa al pacchetto solo se su di esso viene effettuata un'operazione di acquisto o di condivisione.
Abbiamo usato lo stesso approccio di traduzione anche per la seconda generalizzazione presente nel modello. Anche in questo caso gli attributi dell'entità "prodotto base" sono aggiunti alle entità figlie e la relazione " prodotti base acquistati" è replicata sulle tre entità figlie.
Sebbene questa soluzione implichi un aumento delle relazioni presenti nel modello e, più concretamente, un aumento delle tabelle nella base di dati, ci pare più pulita e semplice da gestire.
Un approccio "bottom up" avrebbe creato una schema meno chiaro poichè le entità figlie sono indipendenti e hanno un significato distinto. Avremmo inoltre inserito numerosi campi con valore "null" per la maggior parte delle tuple, creando quindi numerose ambiguità nel sistema.
Per quanto riguarda l'attributo multiplo "città destinazione" è stata creata una relazione molti a molti e un'altra entità "città" in quanto una stessa città può essere raggiunta da più pacchetti e un pacchetto può raggiungere città diverse.
Di seguito presentiamo il modello ristrutturato e le singole entità.



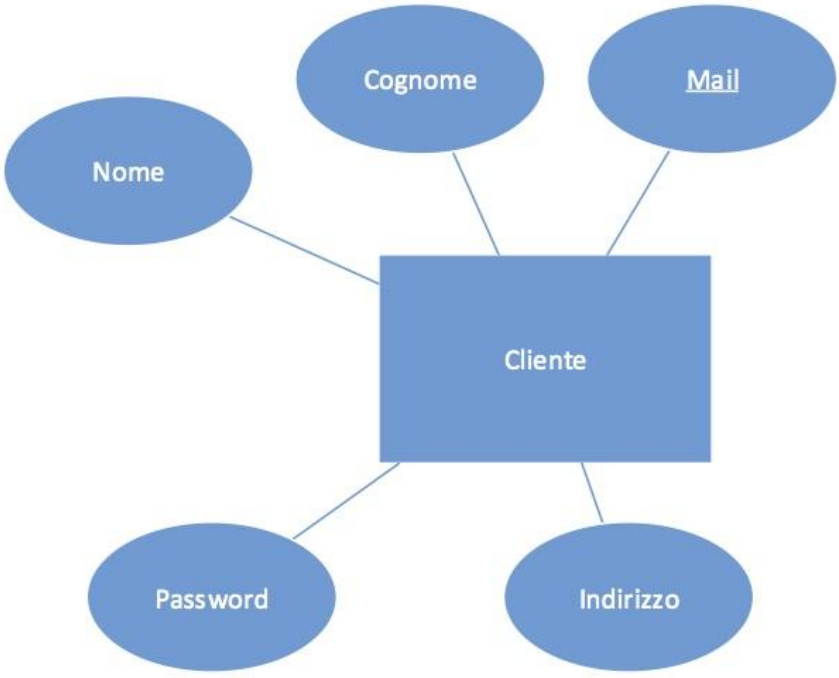
Modello ER ristrutturato.



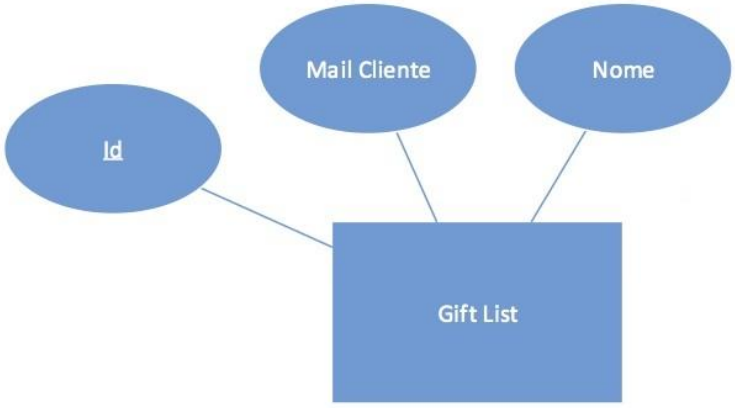
Entità Pacchetto



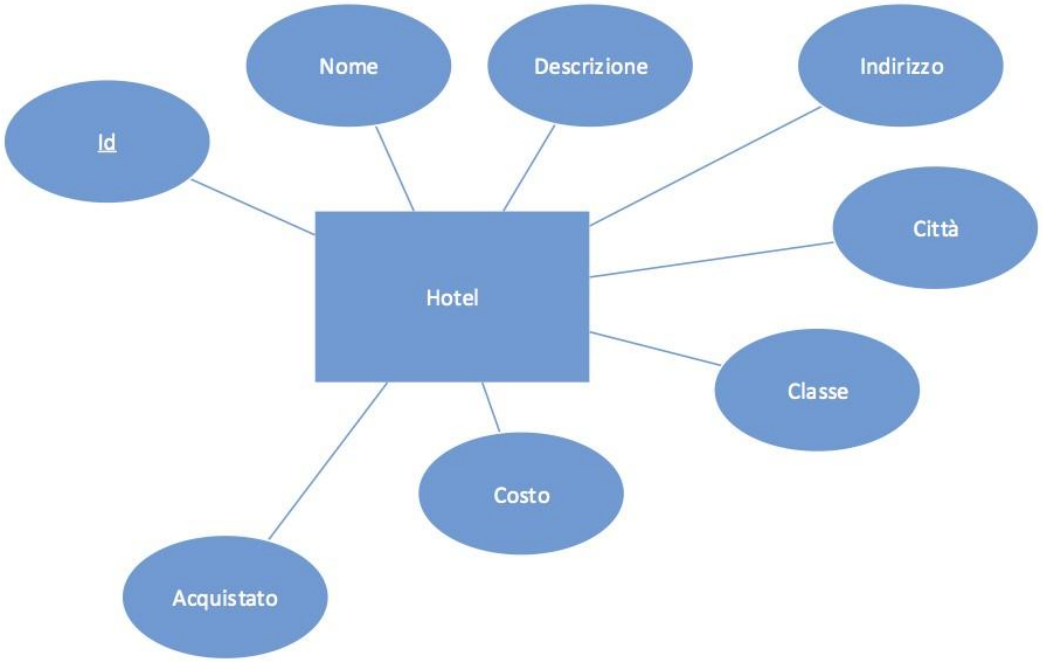
Entità Dipendente



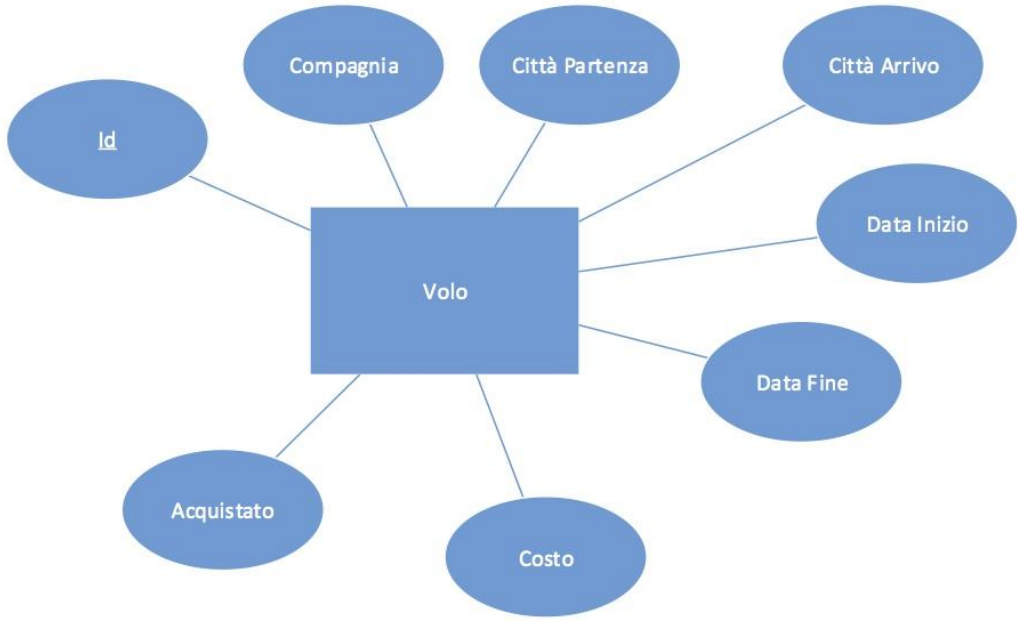
Entità Cliente



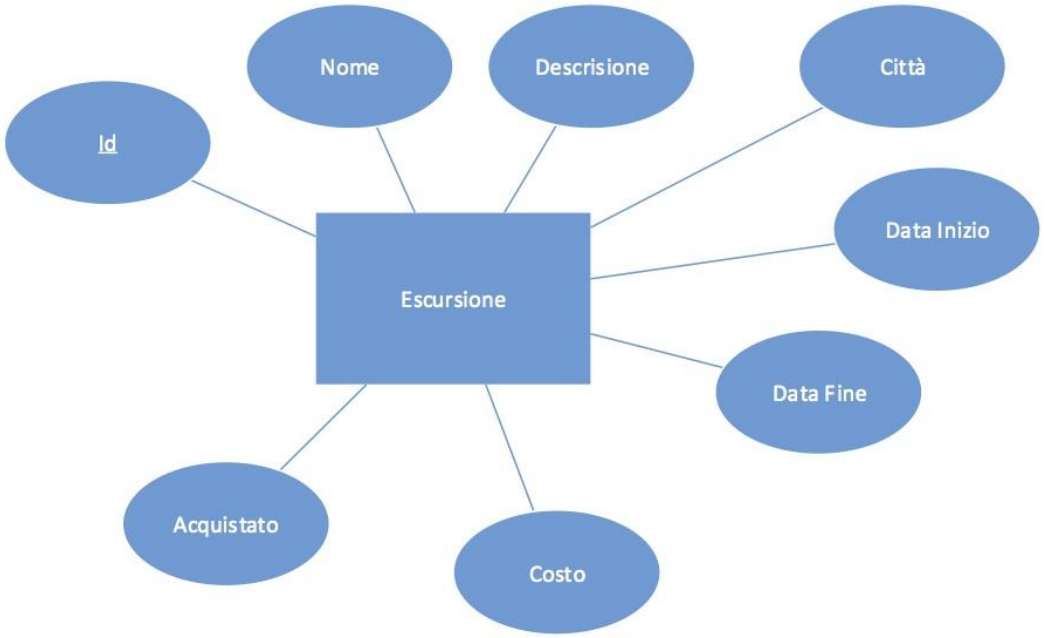
Entità Gift List



Entità Hotel



Entità Volo



Entità Escursione

2.3 Traduzione verso il modello logico

Effettuate tutte le operazioni descritte nel paragrafo precedente abbiamo tradotto il modello concettuale nel modello logico ottenendo le seguenti tabelle:

Traduzione delle entità.
Pacchetto (id-pacchetto, nome, descrizione, disponibilità-max, disponibilità-attuale, data-inizio, data-fine, costo, mail-cliente, mail-dipendente)
Hotel(id-hotel, nome, descrizione, indirizzo, città, classe, costo, acquistato)
Volo(id-volo, compagnia, città-partenza, città-arrivo, data-inizio, data-fine, costo, acquistato)
Cliente(mail-cliente, nome, cognome, password, indirizzo)
Escursione (id-escursione, nome, descrizione, data-inizio, data-fine, città, costo, acquistato)
Dipendente(mail-dipendente, nome, cognome, matricola,password)
GiftList (id-giftList, mail-cliente, nome, data-creazione)
Città(id-città, Nome)

Tabelle ponte (relazioni molti a molti)
Pernottamento(id-pacchetto, id-hotel, data-inizio, numero-notti)
VoliPacchetto(id-pacchetto, id-volo)
EscursioniPacchetto(id-pacchetto, id-escursione)
Acquista(mail-cliente,id-pacchetto, data-acquisto)
Contiene (id-giftList, id-pacchetto)
Destinazione(id-pacchetto, id-città)
HotelAcquistati(id-giftList,id-hotel,data-acquisto,nome-acquirente)
VoliAcquistati(id-giftList,id-volo,data-acquisto,nome-acquirente)
EscursioniAcquistate(id-giftList,id-escursione,data-acquisto,nome-acquirente)

3 Design dell'applicazione

3.1 Descrizione dell'architettura

Il modello architetturale scelto per implementare il prodotto software è quello multi-tier a tre layer. In particolare abbiamo individuato quattro tier: client, web, business e data. I quattro livelli logici sono allocati nel modo seguente:

- Terminale utente: client tier
- Application server: business tier, web tier
- DBMS: data tier

Questa impostazione è l'approccio classico nel design delle applicazioni Enterprise presente in letteratura. Ogni tier può comunicare solo con il livello immediatamente precedente o successivo, rendendo così l'applicazione modulare.
Di seguito presentiamo una descrizione dettagliata di ogni singolo livello logico dell'applicazione.

3.1.1 Client tier
In questo livello gli utenti accedono all'applicazione tramite un browser web generico. Il client tier comunica con il web tier attraverso il protocollo HTTP, inviando al server le richieste dell'utente e visualizzando le pagine HTML ricevute.

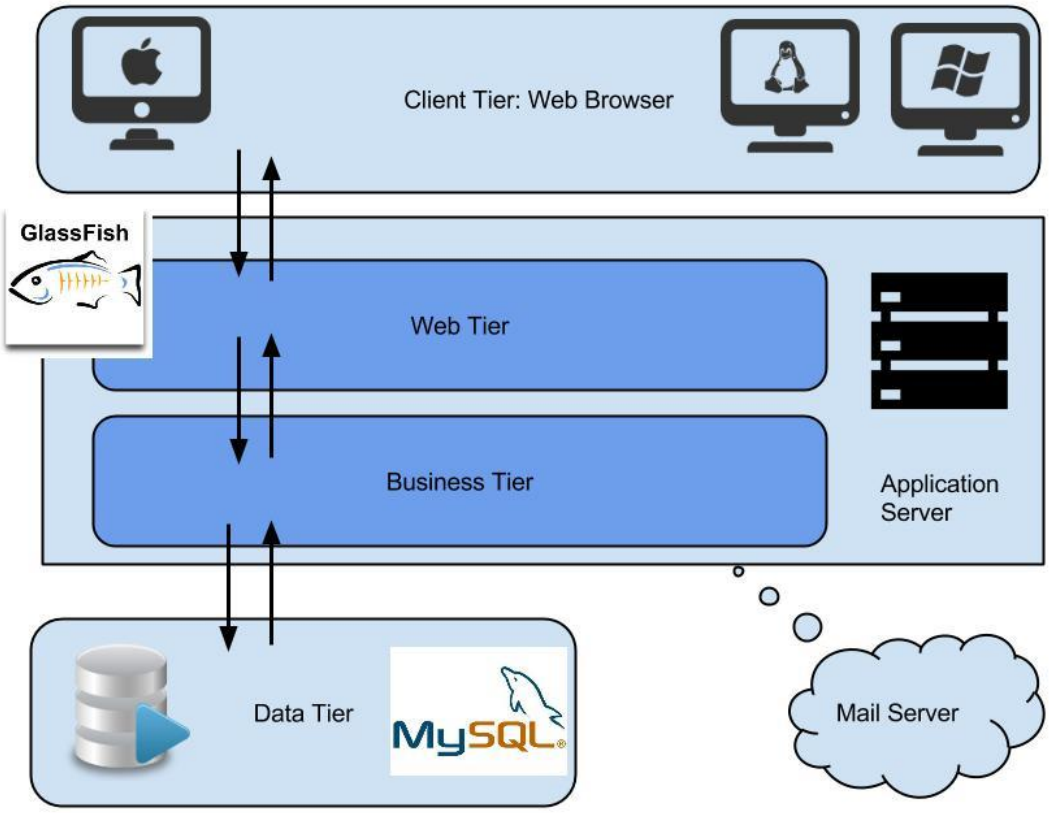
3.1.2 Web tier
Questo livello logico riceve le richieste HTTP dell'utente inviate dal client tier e risponde inviando pagine HTML. Le pagine sono generate dall'interazione con il business tier considerando le richieste dell'utente. Il web tier sarà implementato all'interno di un Application Server compatibile.

3.1.3 Business tier
La logica applicativa dell'applicazione, che rende operativo il nucleo di elaborazione, è contenuta in questo tier. Essa gestisce lo scambio di informazioni tra la base di dati e il web tier. Per accedere al database viene astratto il modello di tipo relazionale, implementato nella base di dati, in un modello di dati a oggetti con cui l'applicazione interagisce.

3.1.4 Data tier
Il data tier è costituito dal DBMS che garantisce la persistenza dei dati. Le comunicazioni tra il data tier e e il business tier avvengono in rete, quindi la base di dati e il business tier possono risiedere anche su macchine fisiche diverse.

E' necessario, inoltre, che il business tier interagisca con un server di posta elettronica per poter implementare le comunicazioni tra azienda e clienti e per condividere pacchetti tra clienti.

L'immagine seguente rappresenta graficamente l'architettura descritta.



3.2 Modelli di Navigazione

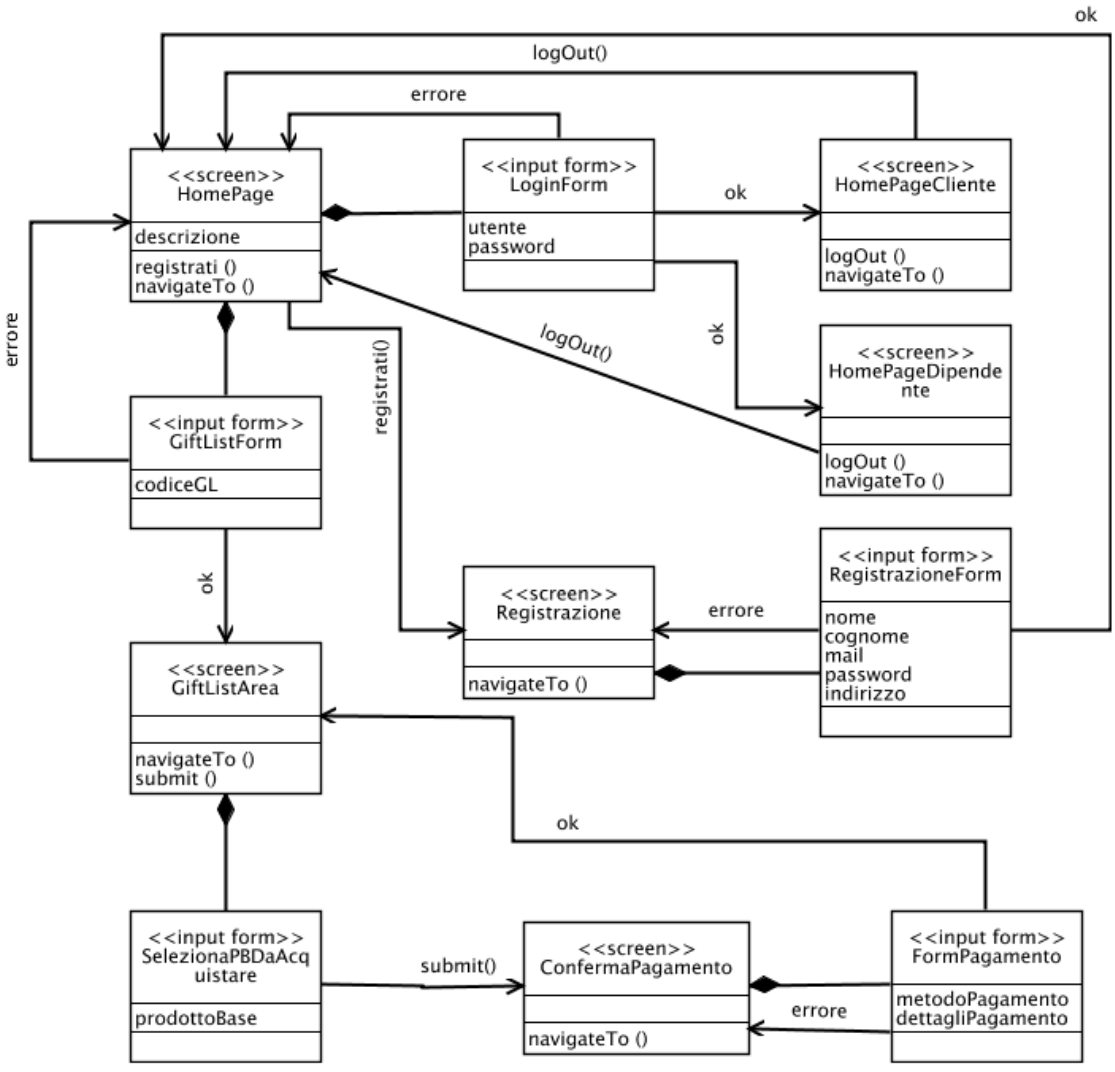
Per la progettazione dell'interfaccia utente, o meglio dei percorsi di navigazione relativi all'applicazione web che verrà implementata come client, abbiamo utilizzato una notazione basata su diagrammi delle classi UML: il modello UX (User eXperience).
Per non appesantire la notazione, i diagrammi che presentiamo di seguito sono separati per tipologia di utente e per funzionalità e non rappresentano in dettaglio alcune delle schermate dell'applicazione di importanza secondaria.
Parte delle schermate più importanti sono state già introdotte nella sezione "Interfaccia Utente" del RASD.

3.2.1 Utente Non Registrato

Il primo diagramma che presentiamo è relativo alla porzione dell'applicazione web accessibile agli utenti che non hanno effettuato il login. Un utente si collega al sito a partire dalla HomePage e può effettuare il login, la registrazione oppure accedere all'area gift list.
In particolare:

- se il login ha successo, l'utente viene reindirizzato alla schermata HomePageCliente se si tratta di un Cliente, altrimenti all'HomePageDipendente se si tratta di un Dipendente, queste schermate verrano presentate in dettaglio nei diagrammi successivi.

- se l'utente decide di effettuare la registrazione, e il processo va a buon fine, si viene reindirizzati sull'HomePage, altrimenti si ritorna alla pagina di Registrazione.
- se l'utente decide di effettuare l'accesso all'area Gift List, inserisce il codice della Gift List nel form e, se il codice è corretto, viene reindirizzato alla pagina contenente i dettagli della gift list.



3.2.2 Cliente

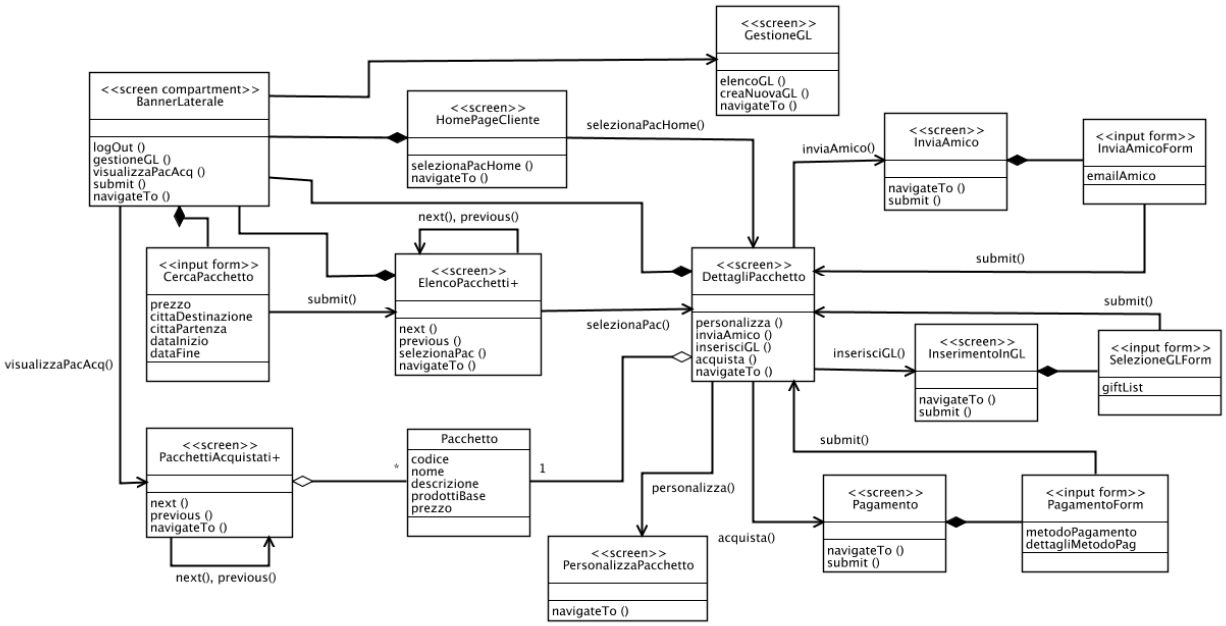
Il secondo insieme di diagrammi che presentiamo è relativo alla porzione dell'applicazione web accessibile agli utenti che hanno effettuato il login come Clienti.

3.2.2.1 Diagramma 1

Il primo diagramma racchiude una parte delle funzionalità offerte al Cliente tra cui:

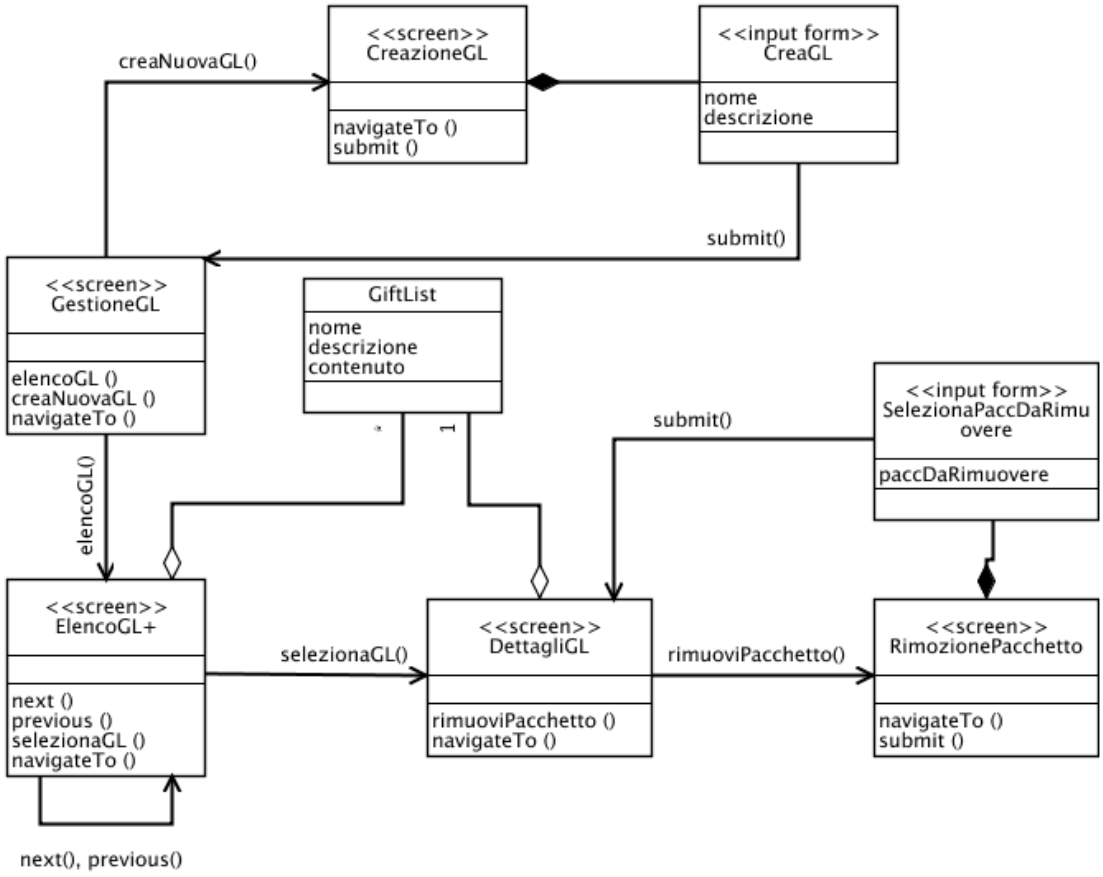
- ricerca di un pacchetto
- selezione di un pacchetto
- acquisto del pacchetto
- invio del pacchetto ad un amico
- inserimento pacchetto in GL
- visualizzazione dei pacchetti acquistati

Le funzionalità di Gestione delle GL create e di personalizzazione del pacchetto verrano mostrate in dettaglio nei diagrammi successivi.
La <<screen compartment>>BannerLaterale rappresenta la barra laterale fissa di ogni pagina web. Ogni <<screen>> della porzione di applicazione web dedicata al cliente possiede una relazione di “composizione” con la <<screen compartment>> ma, buona parte di queste relazioni, verrano omesse nei successivi diagrammi, questo per non appesantire la notazione e per rendere più leggibile il diagramma.



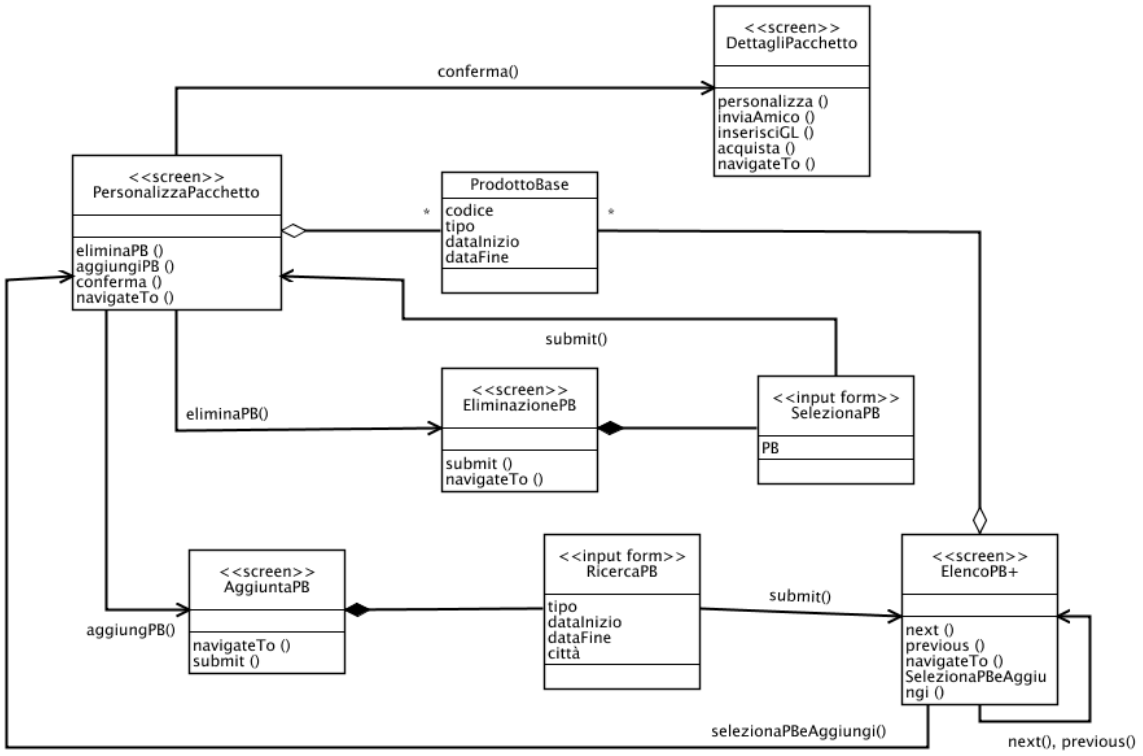
3.2.2.2 Diagramma 2

Il secondo diagramma presenta nel dettaglio la parte dell’applicazione web dedicata alla Gestione delle GL create, mostrando le pagine che offrono le funzionalità di creazione, visualizzazione elenco e modifica delle GL.



3.2.2.3 Diagramma 3

Il terzo e ultimo diagramma presenta la porzione dell'applicazione web dedicata alla Personalizzazione del pacchetto da parte del Cliente.
Questo diagramma mostra nel dettaglio i percorsi di navigazione che permettono di aggiungere o/e rimuovere prodotti base da un pacchetto esistente.



3.2.3 Dipendente

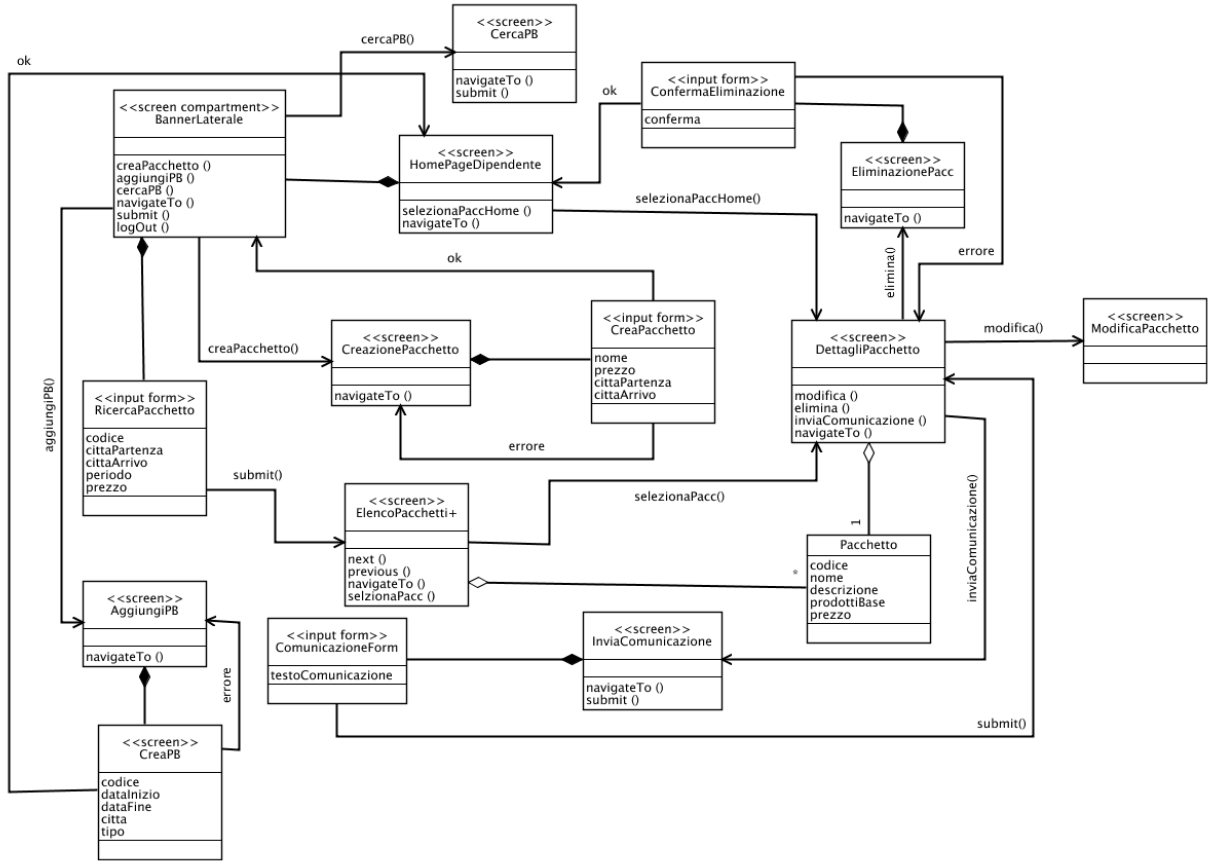
Il terzo insieme di diagrammi che presentiamo è relativo alla porzione dell'applicazione web accessibile agli utenti che hanno effettuato il login come Dipendenti.

3.2.3.1 Diagramma 1

Il primo diagramma racchiude una parte delle funzionalità a disposizione del Dipendente tra cui:

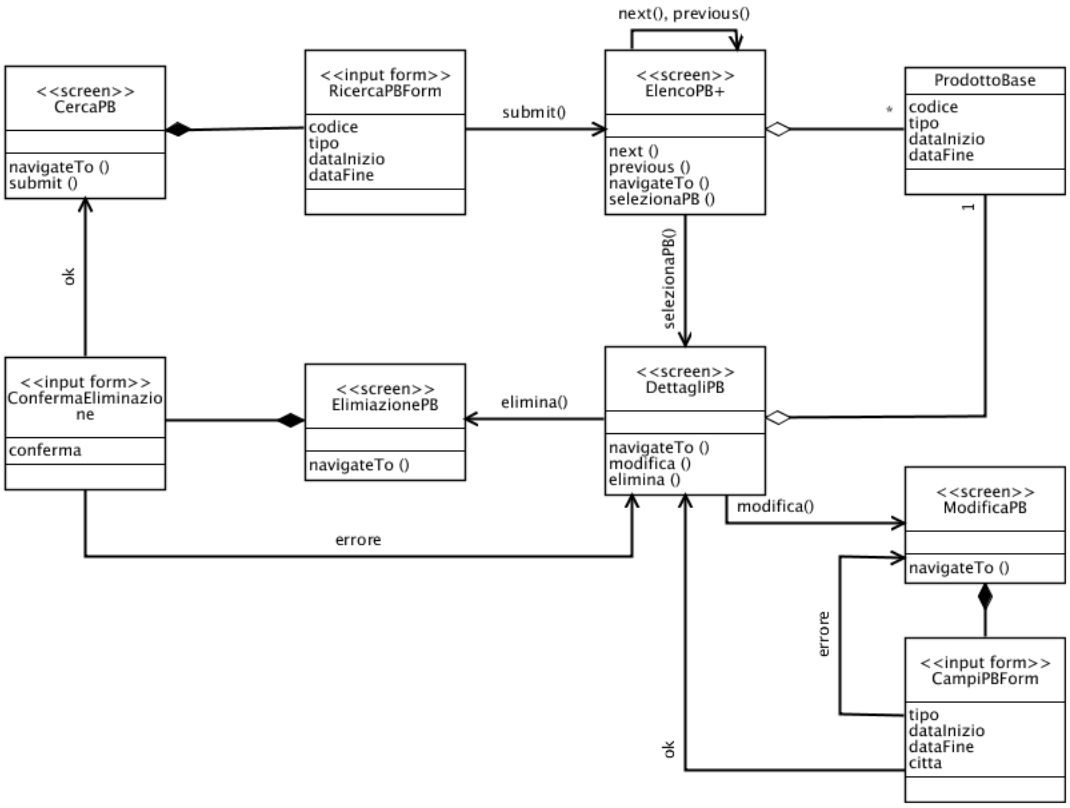
- Ricerca di un pacchetto
- Creazione di un pacchetto
- Eliminazione di un pacchetto
- Invio comunicazione ai clienti acquirenti di un pacchetto
- Creazione di un PB

Le funzionalità di Gestione dei PB e di modifica di un pacchetto verrano mostrate in dettaglio nei diagrammi successivi.
Per la <<screen compartment>> BannerLaterale vale la stessa osservazione fatta per i diagrammi relativi ai Clienti. Verrano omesse alcune relazioni di "composizione" verso essa considerando ogni <<screen>> in relazione con la <<screen comparment>>.
La <<screen compartment>>BannerLaterale del Dipendente offre, chiaramente, differenti funzionalità rispetto a quella del Cliente.



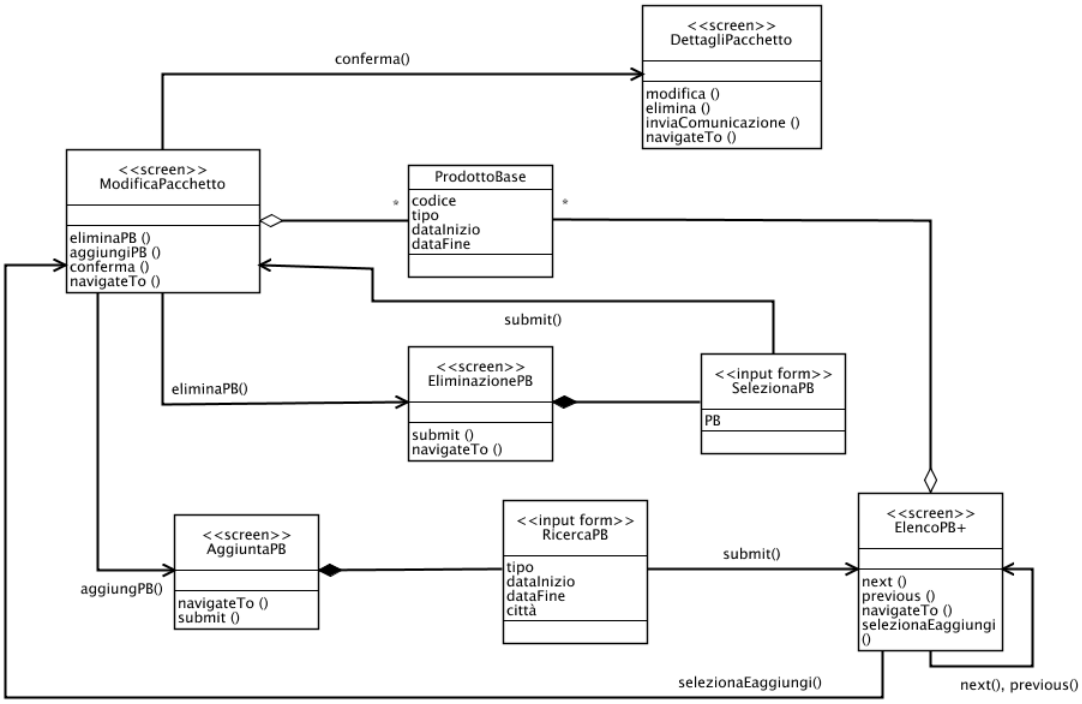
3.2.3.2 Diagramma 2

Il secondo diagramma presenta nel dettaglio la porzione dell'applicazione web dedicata alla Gestione dei PB, racchiudendo le funzionalità di ricerca, modifica ed eliminazione dei PB.



3.2.3.3 Diagramma 3

Il terzo e ultimo diagramma presenta la porzione dell'applicazione web dedicata alla Modifica del pacchetto da parte del Dipendente.
Questo diagramma mostra nel dettaglio i percorsi di navigazione che permettono di aggiungere o/e rimuovere prodotti base da un pacchetto selezionato.

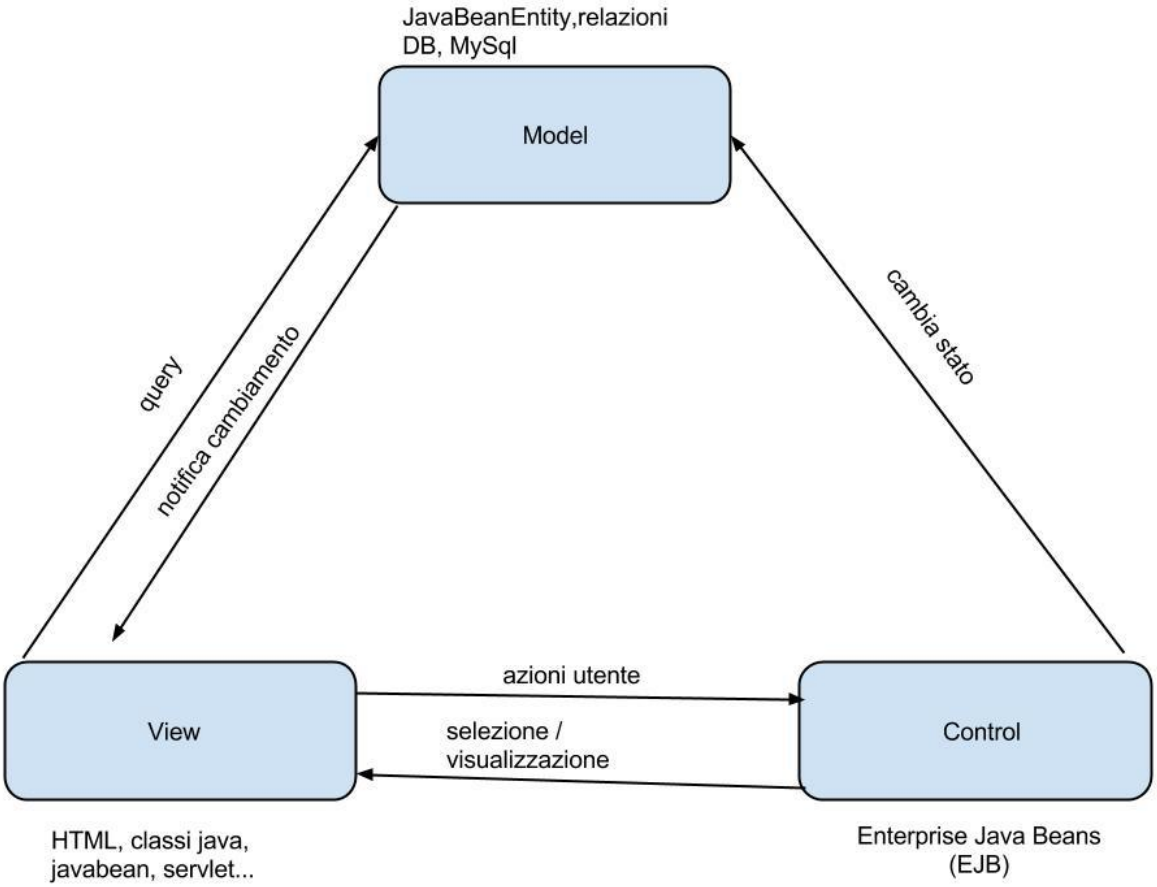


3.3 Componenti

Per la progettazione dei componenti del sistema abbiamo utilizzato una notazione basata su diagrammi delle classi UML: il modello BCE (Boundary, Control, Entity). In questo modo abbiamo potuto dividere l'architettura del software secondo il pattern architetturale Model-View-Controller (MVC). Possiamo, infatti, stabilire le seguenti corrispondenze:

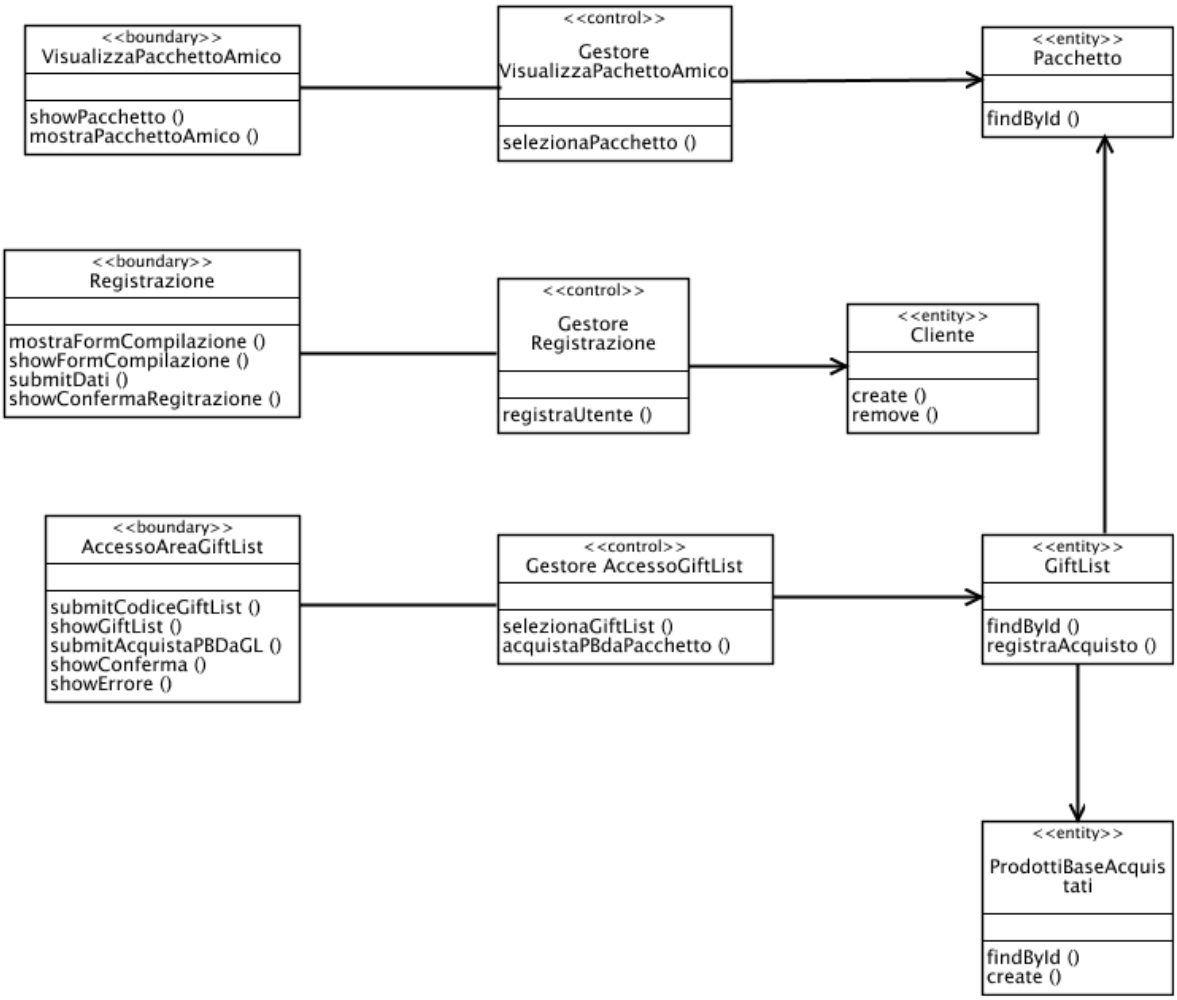
- **Control.** Le classi <<control>> inserite nei diagrammi di analisi corrispondono all'elemento "control" del pattern MVC e possono essere implementate come Enterpise Java Beans. Queste classi sono i mediatori tra l'interfaccia utente e i dati.
- **Model.** Le classi che sono utilizzate in questa parte del pattern MVC sono rappresentate dalle classi <<entity>> del diagramma BCE. Esse si interfacciano direttamente con la base di dati descritta in precedenza e gestita, nel nostro caso, da MySql.
- **View.** Le classi <<boundary>> del modello BCE corrispondono all'interfaccia esterna del sistema verso l'utente e rappresentano la componente "view" nel pattern MVC. Le classi <<boundary>> racchiudono le funzionalità offerte dalle varie pagine web.

Per semplicità e per una migliore comprensione dei diagrammi abbiamo omesso tutti gli attributi delle classi <<entity>>. Gli attributi omessi corrispondono a quelli descritti precedentemente nella parte di design dei dati. Allo stesso modo sono stati in parte omessi i metodi di base come getter, setter, create e remove che saranno sicuramente presenti nell'implementazione.
In figura è rappresentata la corispondenza tra il pattern MVC e la piattaforma Java2 Enterprise Edition (J2EE).

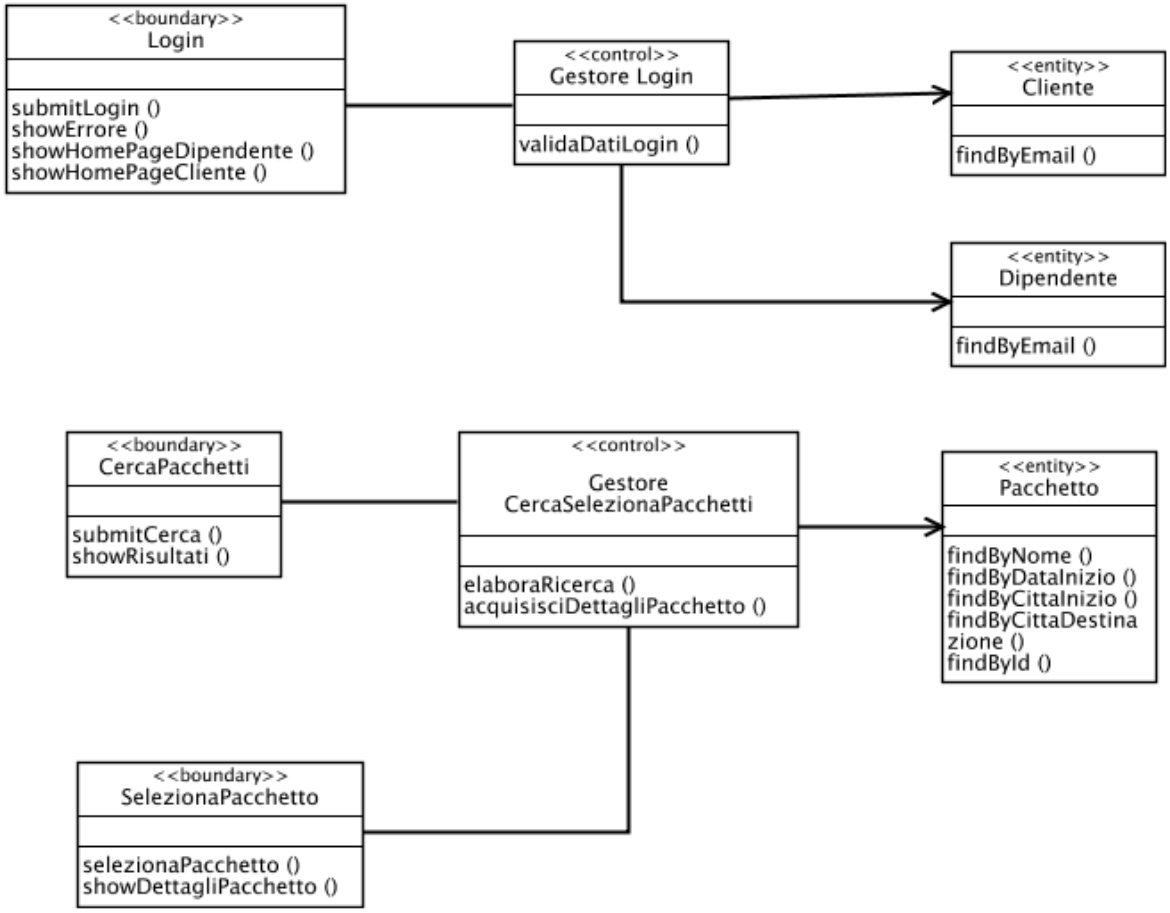


Nelle figure seguenti sono rappresentati i diagrammi BCE divisi per attore.

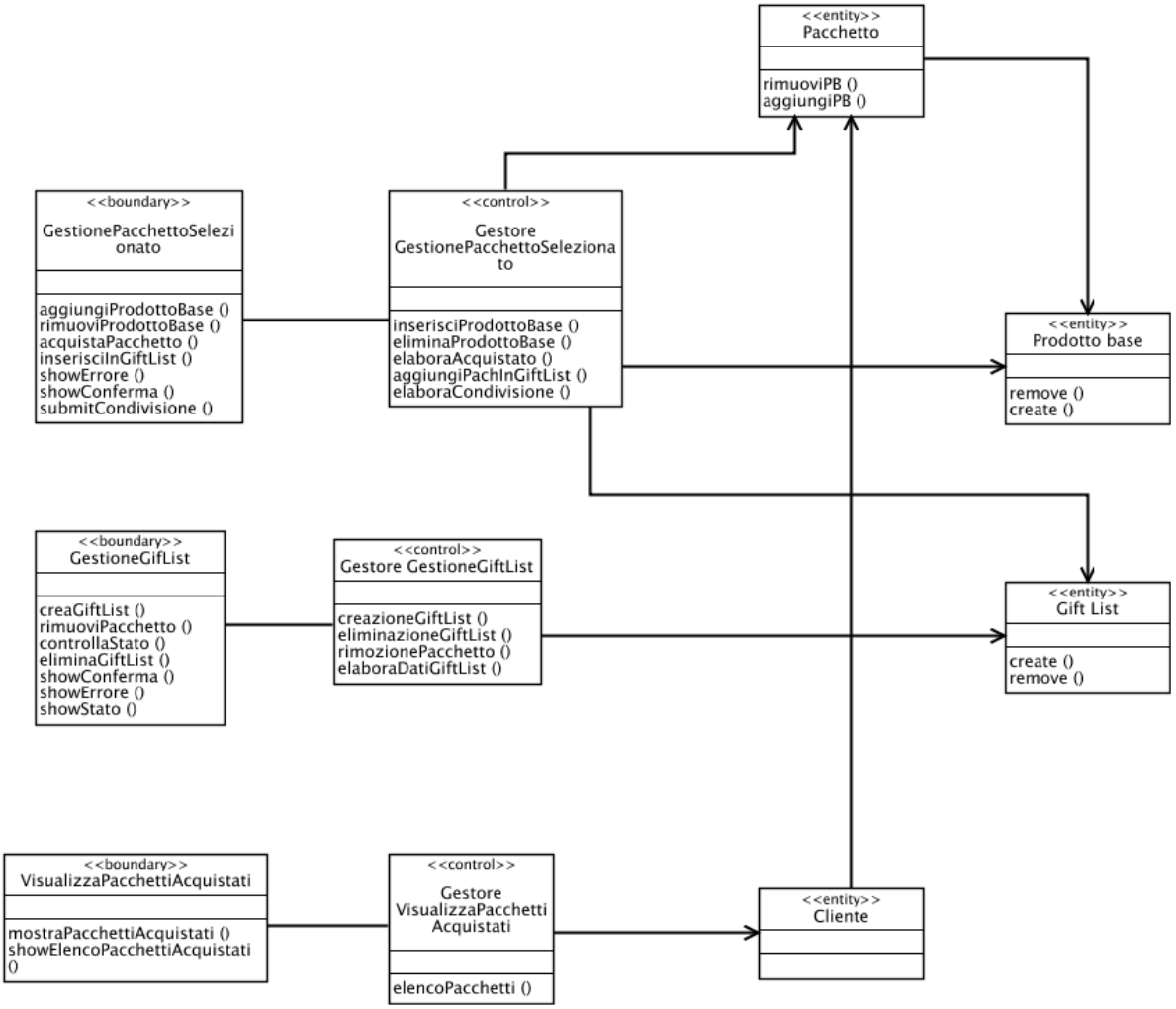
3.3.1 Utente Non Registrato


<<boundary>>
VisualizzaPacchettoAmico
showPacchetto ()
mostraPacchettoAmico ()
<<control>>
Gestore
VisualizzaPachettoAmico
selezionaPacchetto ()
<<entity>>
Pacchetto
findById ()
<<boundary>>
Registrazione
mostraFormCompilazione ()
showFormCompilazione ()
submitDati ()
showConfermaRegitrazione ()
<<control>>
Gestore
Registrazione
registraUtente ()
<<entity>>
Cliente
create ()
remove ()
<<boundary>>
AccessoAreaGiftList
submitCodiceGiftList ()
showGiftList ()
submitAcquistaPBDaGL ()
showConferma ()
showErrore ()
<<control>>
Gestore AccessoGiftList
selezionaGiftList ()
acquistaPBdaPacchetto ()
<<entity>>
GiftList
findById ()
registraAcquisto ()
<<entity>>
ProdottiBaseAcquis
tati
findById ()
create ()

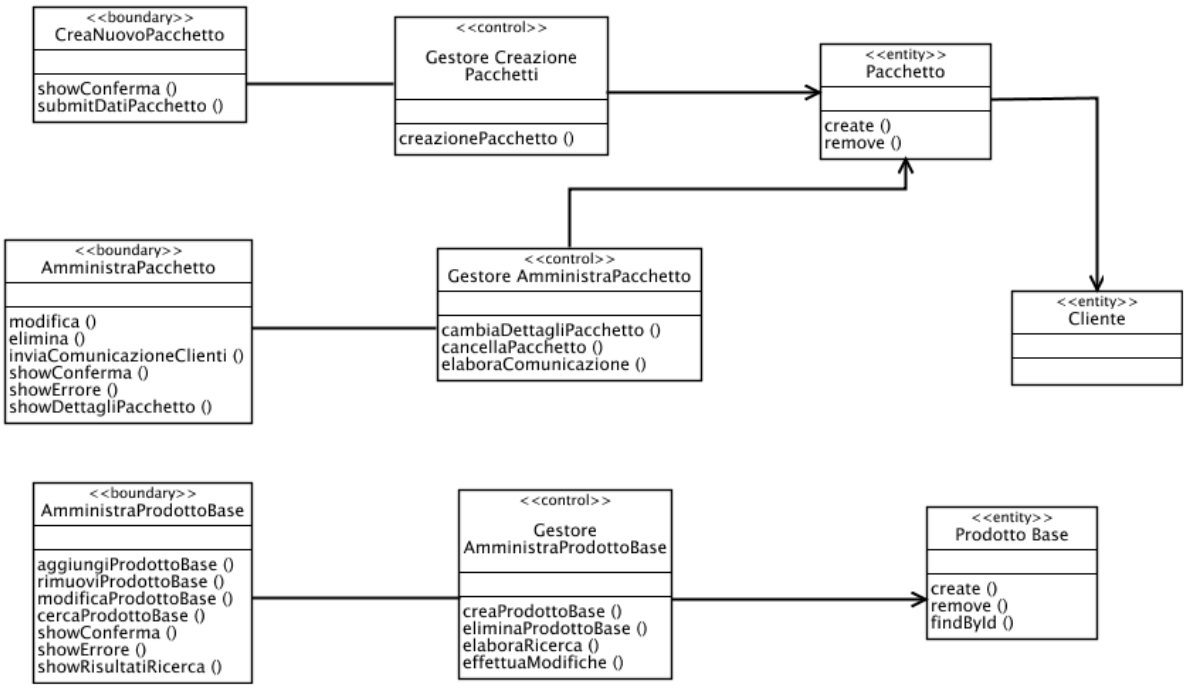
3.3.2 Utente Registrato



3.3.3 Cliente


<<entity>>
Pacchetto
rimuoviPB ()
aggiungiPB ()
<<boundary>>
GestionePacchettoSelezionato
aggiungiProdottoBase ()
rimuoviProdottoBase ()
acquistaPacchetto ()
inserisciInGiftList ()
showErrore ()
showConferma ()
submitCondivisione ()
<<control>>
Gestore GestionePacchettoSelezionato
inserisciProdottoBase ()
eliminaProdottoBase ()
elaboraAcquistato ()
aggiungiPachInGiftList ()
elaboraCondivisione ()
<<entity>>
Prodotto base
remove ()
create ()
<<boundary>>
GestioneGifList
creaGiftList ()
rimuoviPacchetto ()
controllaStato ()
eliminaGiftList ()
showConferma ()
showErrore ()
showStato ()
<<control>>
Gestore GestioneGiftList
creazioneGiftList ()
eliminazioneGiftList ()
rimozionePacchetto ()
elaboraDatiGiftList ()
<<entity>>
Gift List
create ()
remove ()
<<boundary>>
VisualizzaPacchettiAcquistati
mostraPacchettiAcquistati ()
showElencoPacchettiAcquistati ()
<<control>>
Gestore VisualizzaPacchetti Acquistati
elencoPacchetti ()
<<entity>>
Cliente

3.3.4 Dipendente

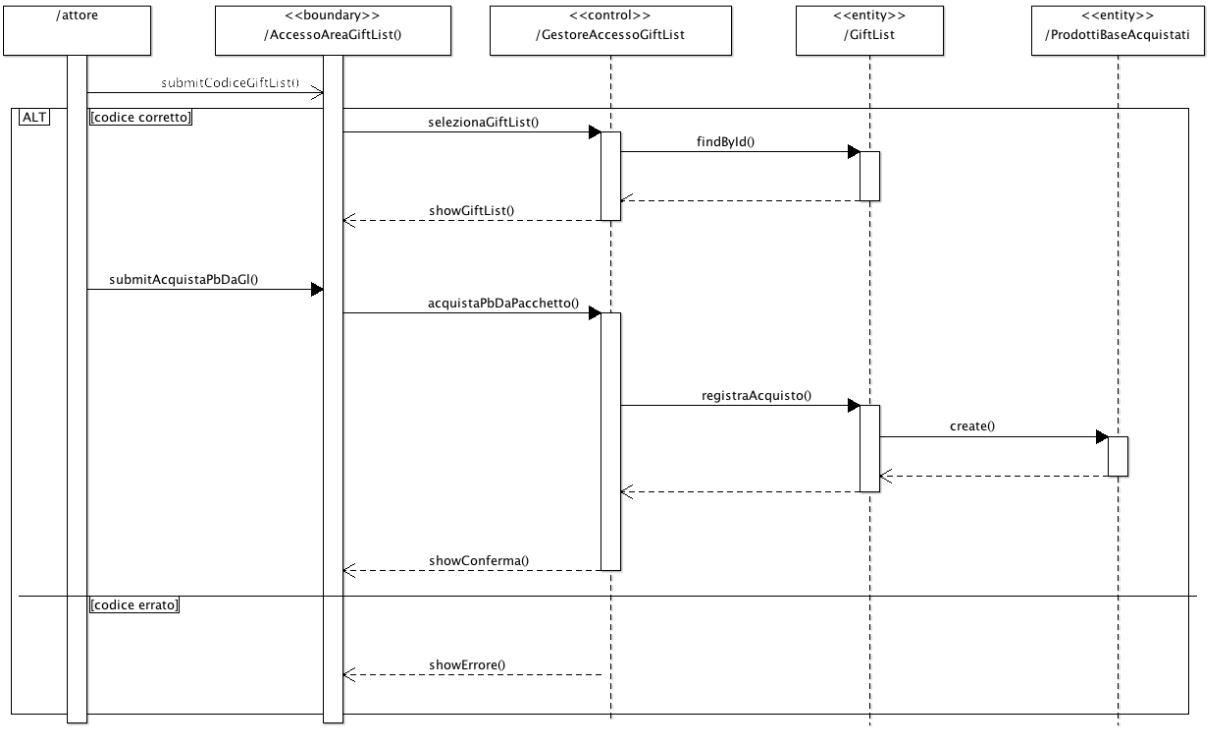


3.4 Diagrammi di sequenza

In questa sezione abbiamo introdotto alcuni diagrammi di sequenza UML, che a differenza dei diagrammi di sequenza proposti nel RASD posseggono un livello di dettaglio maggiore e per quanto possibile più vicino a quello implementativo.

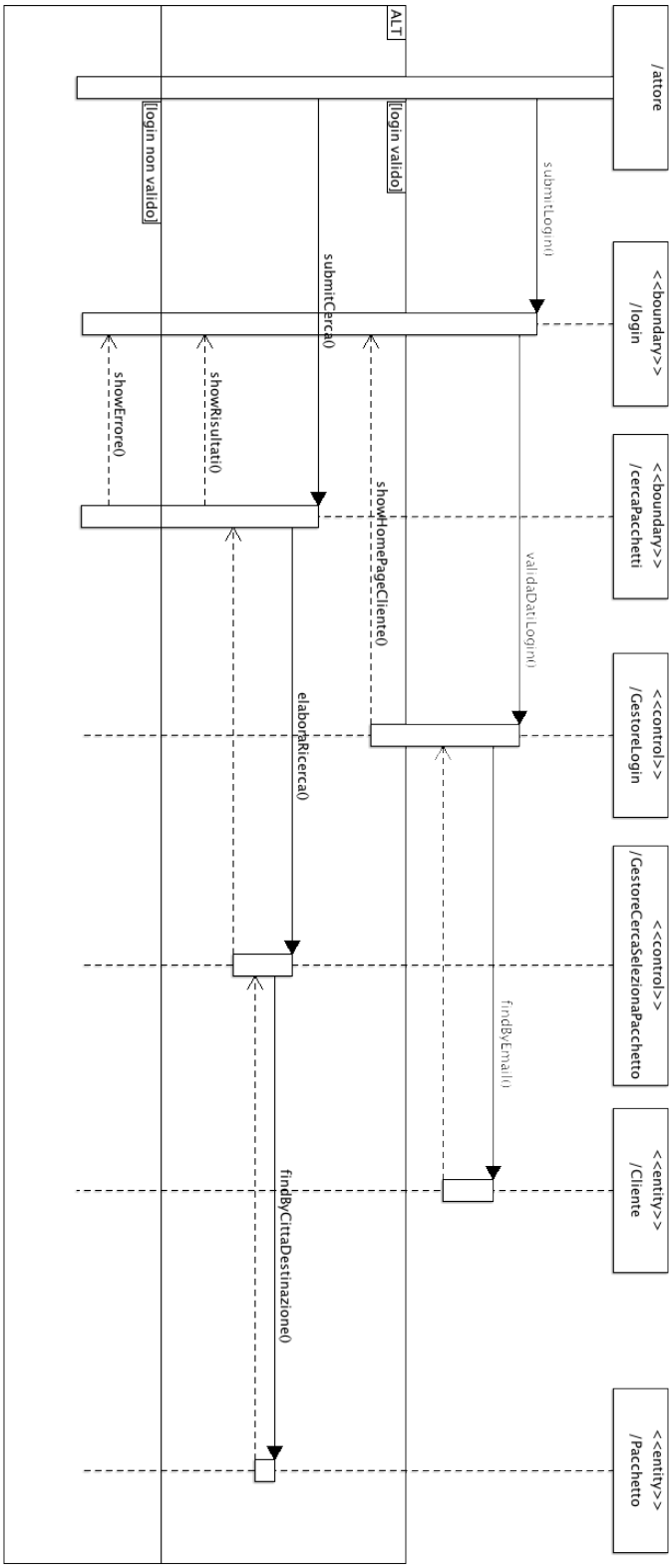
3.4.1 Diagramma 1

Il primo diagramma di sequenza mostra il processo con il quale un utente non registrato può acquistare un prodotto base da una gift list di cui possiede il codice.
Nel caso in cui il codice della gift list sia errato, viene mostrata la schermata di errore.
L'attore invoca il metodo submitCodiceGiftList sulla boundary "AccessoAreaGiftList" inoltrando il codice della GL che vuole visualizzare, esso viene servito dal control "GestoreAccessoGiftList" che interroga la base dei dati mediante l'entity "GiftList".
Una volta selezionato il prodotto base, procede con l'acquisto e il control inoltra la registrazione dell'acquisto creando un istanza nell'entity "ProdottiBaseAcquistati".



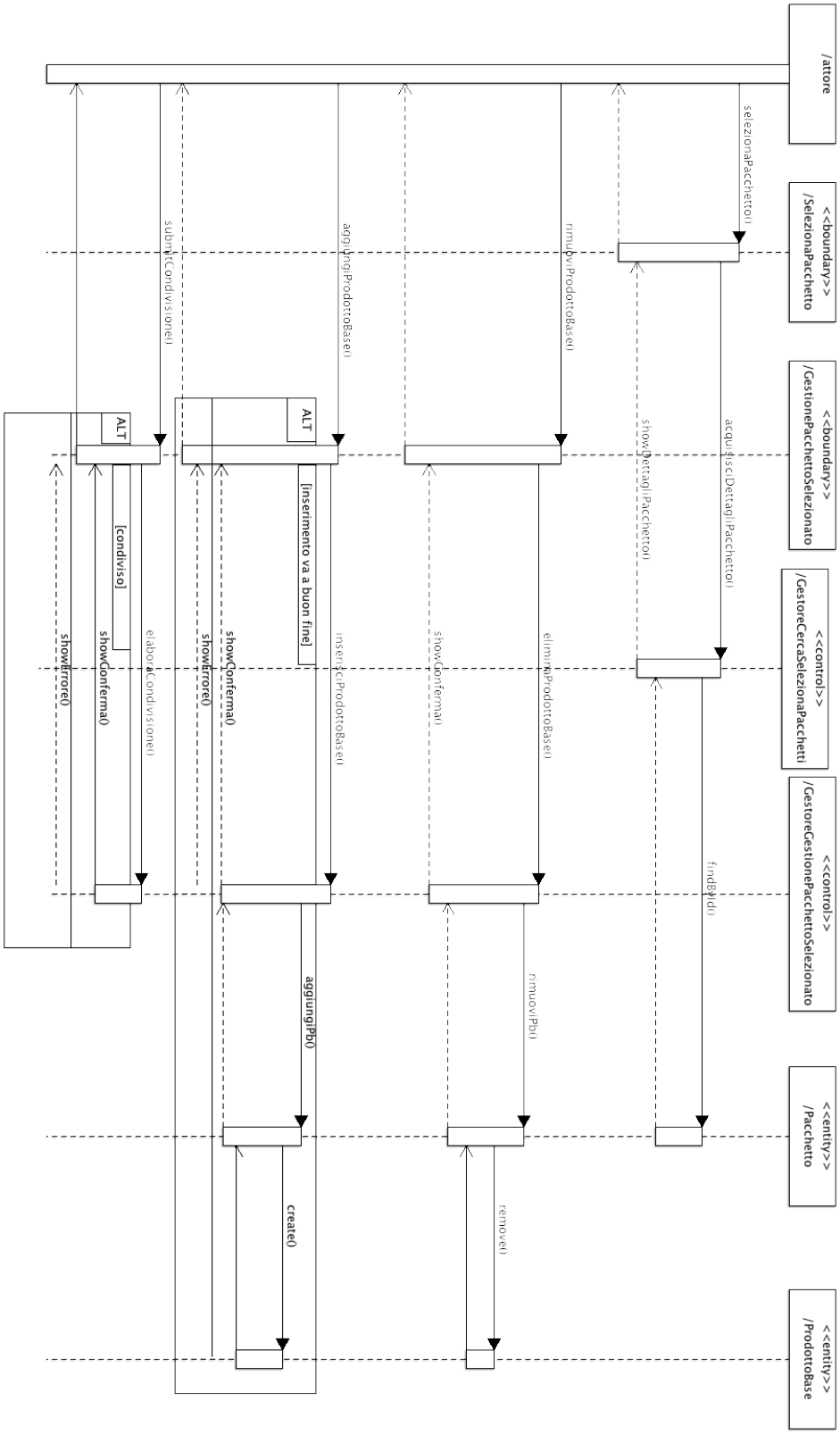
3.4.2 Diagramma 2

Il secondo diagramma di sequenza, mostra il processo di Login da parte di un cliente.
Dopo aver inserito i dati necessari per effettuare l'accesso, il gestore login, utilizzando i dati della entity cliente, controlla la correttezza dei dati inseriti e, in caso negativo, mostra una schermata di errore.
In caso di successo, il sistema permette la ricerca di pacchetti all'interno del sistema.
L'attore inserisce i parametri di ricerca e il sistema elabora i risultati, restituendo i pacchetti richiesti.


ALT
[login valido]
[login non valido]
/attore
<<boundary>>
/login
<<boundary>>
/cercaPacchetti
<<control>>
/GestoreLogin
<<control>>
/GestoreCercaSelezionaPacchetto
<<entity>>
/Cliente
<<entity>>
/Pacchetto
submitLogin()
submitCerca()
showErrore()
showRisultati()
showHomePageCliente()
validaDatiLogin()
elaboraRicerca()
findByEmail()
findByCittaDestinazione()

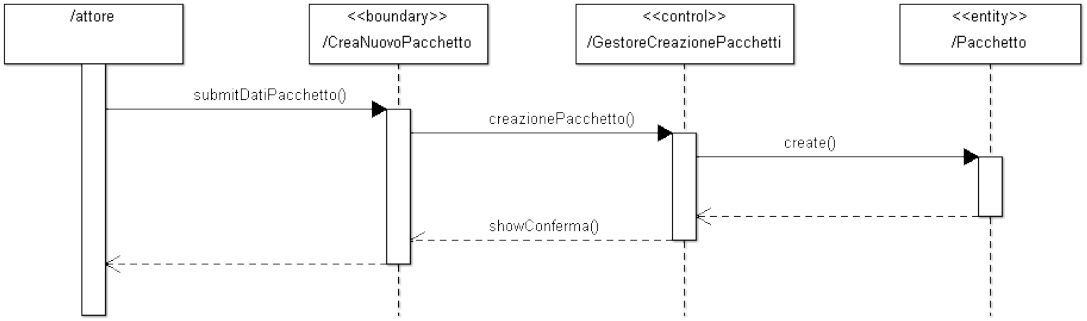
3.4.3 Diagramma 3

Il processo di selezione e condivisione di un pacchetto, aggiunta e rimozione di un prodotto base, è mostrato nel sequence diagram sottostante.
L'attore seleziona, attraverso la boundary "SelezionaPacchetto", il pacchetto e il control "GestoreCercaSelezionaPacchetto" ne acquisisce i dati interrogando l'entità "Pacchetto".
Per rimuovere un prodotto base da un pacchetto selezionato, il cliente effettua un'operazione di rimozione mediante la boundary. Il control "GestoreGestionePacchettoSelezionato" rimuove il prodotto base operando sull'entità Pacchetto e ProdottoBase.
L'attività di inserimento di un prodotto base segue una procedura simile. La principale differenza sta nel fatto che, in caso di errore nell'aggiunta, il control mostra un errore.
La condivisione di un pacchetto viene fatta sempre dal control "GestoreGestionePacchettoSelezionato" che, dopo aver ricevuto un comando dalla boundary, elabora la richiesta e mostra un messaggio di conferma, oppure in caso di insuccesso, un messaggio di errore.


/attore
<<boundary>>
/SelezionaPacchetto
<<boundary>>
/GestionePacchettoSelezionato
<<control>>
/GestoreCercaSelezionaPacchetti
<<control>>
/GestoreGestionePacchettoSelezionato
<<entity>>
/Pacchetto
<<entity>>
/ProdottoBase
selezionaPacchetto()
acquisisciDettagliPacchetto()
showDettagliPacchetto()
rimuoviProdottoBase()
aggiungiProdottoBase()
submitCondivisione()
ALT
[condiviso]
showConferma()
elaboraCondivisione()
showErrore()
ALT
[inserimento va a buon fine]
showConferma()
showErrore()
inserisciProdottoBase()
showConferma()
eliminaProdottoBase()
rimuoviPb()
remove()
findById()
aggiungiPb()
create()

3.4.4 Diagramma 4

L'ultimo sequence diagram mostra l'operazione di creazione di un nuovo pacchetto da parte di un dipendente della compagnia.
L'attore inserisce i dati del pacchetto, confermandone la creazione e il control "GestoreCreazionePacchetti" si occupa di creare il pacchetto operando sull'entità "Pacchetto", quindi vengono infine mostrati i dettagli del pacchetto appena creato e un conferma di avvenuta creazione.



4.Ore di lavoro

Membro del gruppo	**Ore**
Giorgio Conte	17
Lorenzo Di Tucci	14
Leonardo Cavagnis	18