UNIVERSITA' DEGLI STUDI DI PALERMO

FACOLTA' DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Sistema software per la gestione di una libreria universitaria

Tesina di Ingegneria del Software di:
XXX
YYY

Anno Accademico 2001-2002

INDICE ANALITICO

Analisi dei requisiti
(Requirements Analysis Document)

1. Introduzione

Il sistema da progettare si propone di supportare la gestione di una libreria universitaria al fine di migliorarne l'efficienza in tutte le sue attività cioè nella:

- Vendita dei libri
- Gestione del magazzino
- Acquisizione dei libri

Per far ciò si è pensato di strutturare il sistema in modo da non modificare eccessivamente i compiti affidati ai suoi utilizzatori e il loro modo di svolgerli. A tal fine si sono identificate tre figure fondamentali per la sua gestione:

- Addetto alle vendite (commesso) che consulta le disponibilità in magazzino, registra le vendite ed effettua le prenotazioni dei libri in funzione delle esigenze dei clienti
- Addetto al magazzino (magazziniere) il quale deve poter registrare le consegne dei libri specificandone l'ordine
- Addetto agli ordini (gestore ordini) che si occupa di generare ed inoltrare gli ordini in funzione del periodo e delle previsioni di vendita e di gestire il database dei libri inserendo all'occorrenza nuovi testi e rimuovendo quelli non più prodotti. Inoltre si occupa di associare ai libri i corsi universitari che li adottano in modo da avere sempre una visione di quando è più utile incrementare la scorta di magazzino.

1.1 Definizioni, acronimi e abbreviazioni

Utilizzeremo il termine impiegato come soggetto di azioni comuni alle tre figure che sono i possibili utilizzatori del sistema:

- commesso
- magazziniere

- gestore ordini

Distingueremo fra i termini prenotazione e ordine intendendo con il primo la richiesta di uno o più libri da parte di un cliente alla libreria e, con il secondo, quella della libreria ad un rappresentante o fornitore.

Nei casi d'uso e nei screen mock-ups sono presenti dei numeri posti fra parentesi e in colore blu (esempio (2.1.1)). Questi identificano univocamente uno screen mock-up e quindi permettono di risalire velocemente alle finestre cui fanno riferimento i caso d'uso. I numeri vengono associati alle finestre con una struttura ad albero .

2. Sistema corrente

Attualmente possiamo supporre che la libreria sia gestita totalmente in maniera manuale anche se basata su una divisione logica dei compiti analoga a quella identificata per il sistema. Questo causa problemi di comunicazione tra le diverse figure che operano all'interno della libreria e che si ripercuotono nella qualità del servizio offerto ai clienti. Con tale approccio, infatti, non è agevole prevedere le future vendite ne tanto meno gestirle nel modo adeguato facendo un ordine preventivo. Inoltre si hanno difficoltà nella gestione di casi eccezionali.

3. Sistema proposto

3.1 Overview del prodotto

Si propone di realizzare un software che guidi gli impiegati della libreria nel loro lavoro.

Il sistema si basa sulla gestione di un unico database cui ci si può interfacciare in modo diverso in funzione del ruolo che l'utilizzatore ha nella libreria. Questo database conterrà:

- dati relativi al magazzino per gestire i suoi movimenti e per avere sempre una visione aggiornata della giacenze
- dati relativi ai corsi universitari quali nome, numero di studenti, moduli di svolgimento, testi consigliati o richiesti
- dati circa la storia dell'acquisto/vendita dei libri
- dati relativi ai clienti per gestire le prenotazioni
- dati statistici di vendita per facilitare gli ordini dei libri nei periodi di maggiore richiesta

3.2 Requisiti funzionali

Il sistema prevede la possibilità di accedervi in tre modalità:

- Vendita dei libri
- Gestione del magazzino
- Acquisizione dei libri

Si accede al sistema tramite uno qualunque dei terminali (non necessariamente tre). Tutti i terminali sono in grado di funzionare in qualunque modalità sopra menzionata, previa abilitazione o durante la fase di avvio del sistema o durante

l'utilizzo in una modalità differente. Queste si differenziano per l'interfaccia utente ma soprattutto per le operazioni che l'operatore può effettuare.

3.3 Requisiti non funzionali

3.3.1 Interfaccia utente e fattore umano

Il sistema presenta un'interfaccia utente di tipo intuitivo basata su sistema operativo Windows. Questo permette una sua utilizzazione da parte di utenti anche non esperti con l'ausilio di un addestramento minimo, inoltre la distinzione delle tre modalità permette di limitare il campo di azione dell'utente preservando il sistema da perdite e/o modifiche accidentali di dati.

3.3.2 Documentazione

Durante la progettazione del sistema saranno prodotti e rilasciati i seguenti documenti:

Documento	Utenti del documento
Documento di analisi dei requisiti (RAD)	Sviluppatori, clienti
Documento del piano di sviluppo del progetto (SPMP)	Sviluppatori, clienti
Documento dell'architettura software (SDD)	Sviluppatori, analisti
Documento del progetto esecutivo (ODD)	Sviluppatori

3.3.3 Considerazioni Hardware

Il sistema rilasciato verrà installato su un'unica macchina nella quale sarà anche allocato il database Access2002. Poiché il sistema si interfaccia con il database tramite un JDBC-ODBC driver facilmente si può estendere il suo utilizzo ad una generica rete di computer eventualmente dedicando un'unica macchina, che

fungerà da server, per il database. L'utilizzo di questo approccio non pone un limite massimo al numero di terminali collegabili.

Si supporrà di utilizzare macchine dotate di processori Intel o AMD.

3.3.4 Prestazioni Hardware

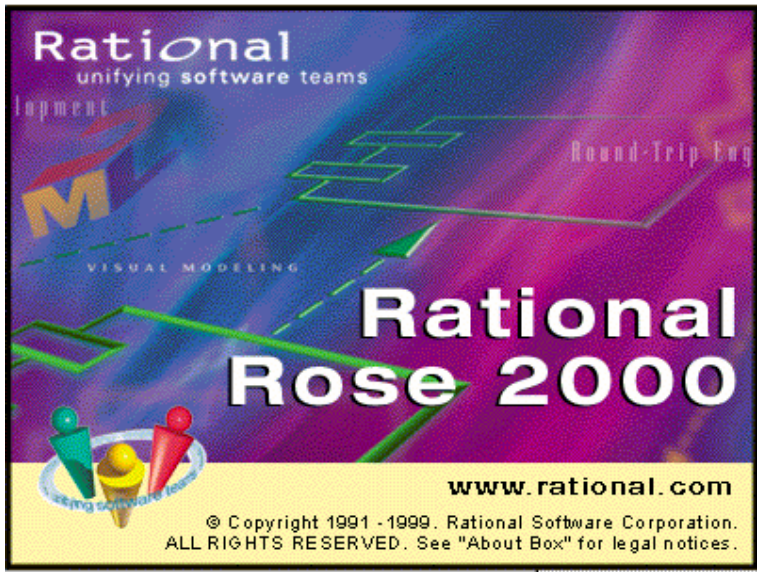
Si prevede che i requisiti minimi imposti dal sistema siano quelli relativi al corretto funzionamento del sistema operativo.

3.3.5 Interfaccia del sistema

Il sistema prevede l'utilizzo di dispositivi di interfacciamento tradizionali di un personal computer (monitor, tastiera e mouse).

3.4 Pseudo-requisiti

Per lo sviluppo del progetto utilizzeremo lo strumento CASE Rational Rose 2000 interfacciandolo con il tool di sviluppo Forte for Java release 3.0 Enterprise Edition. Utilizzeremo come linguaggio di programmazione Java in quanto questo permette facilmente di realizzare l'interfaccia utente, di gestire il database grazie alla libreria JDBC. Inoltre il suo utilizzo è gratuito e non pone vincoli sulla macchina utilizzata essendo multipiattaforma.


Rational
unifying software teams
VISUAL MODELING
Rational
Rose 2000
www.rational.com
© Copyright 1991 - 1999. Rational Software Corporation.
ALL RIGHTS RESERVED. See "About Box" for legal notices.


Sun
microsystems
based on
NetBeans
PointBase
Loading Auto Update ...
FORTE™ FOR JAVA™ RELEASE 3.0
ENTERPRISE EDITION
Copyright © 2001 Sun Microsystems,Inc. All rights reserved. Use is subject to license terms. The PointBase product included herein is for development purposes only and can only be commercially deployed under a separate license from PointBase.

3.5 Modelli del sistema

3.5.1 Scenari

Di seguito riportiamo alcuni scenari tipici che si presentano nell'utilizzo del sistema. Ordinati secondo l'attore partecipante principale.

Scenari di impiegato

Nome dello scenario:	selezione_iniziale_modalita_di_utilizzo
Attori partecipanti:	ugo: commesso
Flusso d'eventi:	1. Ugo avvia il programma di gestione della libreria
	2. Ugo seleziona la modalità di utilizzo vendita tra quelle disponibili (vendita, magazzino, gestione_ordini)
	3. Compare la finestra di gestione delle vendite

Nome dello scenario:	modifica_modalità_di_utilizzo
Attori partecipanti:	ugo: commesso
	pippo: magazziniere
Flusso d'eventi:	1. Ugo sta lavorando al suo terminale in modalità vendita.
	2. Arriva Pippo che ha necessità di lavorare in modalità magazzino. Quindi chiede a Ugo di cedergli momentaneamente la sua postazione.
	3. Pippo clicca sul tasto **Modifica modalità.**
	4. Si apre una finestra con scritto "Sei sicuro di voler cambiare modalità di utilizzo?" e Pippo seleziona **Si**
	5. Appare il form di selezione modalità e Pippo seleziona **Magazzino**

Nome dello scenario:	cerca_libro
Attori partecipanti:	ugo: commesso
	ciccio: cliente
Flusso d'eventi:	1. Ciccio chiede la disponibilità di un libro
	2. Ugo riempe gli appositi campi presenti nel form con i dati del libro (non necessariamente tutti) e clicca sul tasto **Cerca libro.**
	3. Il sistema riordina la lista dei libri mostrando quelli che soddisfano i requisiti
	4. Ugo osserva il numero di copie disponibili in magazzino nella colonna apposita e risponde a Ciccio

Scenari di commesso

Nome dello scenario:	`vendita_libro`
Attori partecipanti:	`ugo: commesso`
	`ciccio: cliente`
Flusso d'eventi:	1. Ciccio vuole acquistare un libro
	2. Ugo, dopo aver verificato la disponibilità del libro, lo seleziona e clicca sul tasto **Vendi**
	3. Il sistema aggiorna il database decrementando la disponibilità del libro

Nome dello scenario:	`nuovo_cliente`
Attori partecipanti:	`ugo: commesso`
	`ciccio: cliente`
Flusso d'eventi:	1. Ciccio vuole prenotare un libro
	2. Ugo clicca sul tasto **Clienti** e si apre la finestra corrispondente
	3. Ugo inserisce nome e cognome di Ciccio, avvia la ricerca e verifica che non appartiene al database
	4. Ugo compila il form con i dati personali di Ciccio e clicca su **Nuovo cliente**
	5. Ciccio viene inserito nel database

Nome dello scenario:	`modifica_cliente`
Attori partecipanti:	`ugo: commesso`
	`ciccio: cliente`
Flusso d'eventi:	1. Ciccio vuole fare aggiornare alcuni suoi dati
	2. Ugo clicca sul tasto **Clienti** e si apre la finestra corrispondente
	3. Ugo inserisce nome e cognome di Ciccio, avvia la ricerca e lo seleziona dalla lista
	4. Ugo modifica i dati nel form con quelli nuovi e clicca su **Modifica cliente**
	5. Il sistema chiede conferma della modifica, Ugo clicca su **Si** e il database viene aggiornato

Nome dello scenario:	`cancella_cliente`
Attori partecipanti:	`ugo: commesso`
	`ciccio: cliente`
Flusso d'eventi:	1. Ciccio vuole essere cancellato dall'archivio clienti
	2. Ugo clicca sul tasto **Clienti** e si apre la finestra corrispondente
	3. Ugo inserisce nome e cognome di Ciccio, avvia la ricerca e lo seleziona
	4. Ugo clicca su **Cancella cliente**
	5. Il sistema chiede conferma della cancellazione, Ugo clicca su **Si** e il database viene aggiornato

Nome dello scenario:	`prenotazione_libro`
Attori partecipanti:	`ugo: commesso`
	`ciccio: cliente`
Flusso d'eventi:	1. Ugo, tramite **Cerca libro**, verifica la disponibilità in magazzino dei libri richiesti da Ciccio e gli comunica che alcuni non sono disponibili ma possono essere prenotati
	2. Ciccio esprime la propria disponibilità alla prenotazione
	3. Ugo clicca su **Clienti** e compare una finestra con il form relativo
	4. Ugo inserisce nome e cognome di Ciccio, avvia la ricerca e verifica che appartiene al database
	5. Ugo seleziona la riga corrispondente a Ciccio
	6. Ugo clicca su **Nuova prenotazione** e si apre la finestra corrispondente
	7. Seleziona un libro da prenotare, specifica il relativo numero di copie e clicca su **Aggiungi libro**. Ripete questo procedura per ogni libro da prenotare e poi clicca su **Visualizza prenotazione** facendo apparire la finestra dedicata con la lista di tutti e soli i libri prenotati specificando di ognuno il relativo numero di copie
	8. Ugo, dopo aver verificato la correttezza della prenotazione, inserisce l'importo lasciato in acconto, clicca su **Conferma** e il database viene aggiornato

Nome dello scenario:	consegna_libro_prenotato
Attori partecipanti:	ugo: commesso
	Ciccio: cliente
Flusso d'eventi:	1. Ciccio consegna a Ugo una ricevuta di prenotazione
	2. Ugo clicca su **Prenotazioni** e apre la finestra col form apposito
	3. Ugo inserisce il numero della ricevuta nell'opportuno campo e clicca su **Cerca prenotazioni**
	4. Ugo seleziona la prenotazione e clicca su **Visualizza prenotazione**
	5. Viene mostrata la situazione corrente della prenotazione, con la lista dei libri prenotati, specificando per ognuno il numero di copie prenotate, il numero di quelle già consegnate al cliente, il numero di copie ordinate ma non ancora consegnate e il numero di quelle disponibili in magazzino. Ciccio esprime la propria disponibilità all'acquisto dei libri disponibili e conferma la prenotazione di quelli prenotati ma non ancora arrivati
	6. Ugo, allora, seleziona un libro disponibile e clicca sul tasto **Vendi**. Ripete tale operazione per tutti i libri della prenotazione disponibili alla vendita

Nome dello scenario:	annulla_prenotazione
Attori partecipanti:	ugo: commesso
	ciccio: cliente
Flusso d'eventi:	1. Ciccio vuole annullare una prenotazione e consegna a Ugo la ricevuta
	2. Ugo clicca su **Prenotazioni,** aprendo la finestra col form apposito, inserisce il numero della ricevuta nell'opportuno campo e clicca su **Cerca prenotazione**
	3. Seleziona la prenotazione e clicca sul tasto **Cancella prenotazione**
	4. Viene chiesta conferma della volontà di cancellare la prenotazione, Ugo clicca su **Si** e il database viene aggiornato
	5. Ugo ritira la ricevuta e restituisce la caparra

Nome dello scenario:	`modifica_prenotazione`
Attori partecipanti:	`ugo: commesso`
	`ciccio: cliente`
Flusso d'eventi:	1. Ciccio chiede a Ugo di modificare una propria prenotazione
	2. Ugo clicca su **Prenotazione** e compare la finestra delle prenotazioni
	3. Seleziona la prenotazione di Ciccio e clicca su **Modifica prenotazione**
	4. Si apre la finestra corrispondente e Ugo seleziona un libro, inserisce il numero di copie desiderate da Ugo e clicca su **Modifica libro**
	5. Ugo clicca su **Visualizza prenotazione** e viene mostrata la lista dei libri prenotati.
	6. Ugo clicca su **Conferma** e il database viene aggiornato

Nome dello scenario:	`visualizza_prenotazione`
Attori partecipanti:	`ugo: commesso`
	`ciccio: cliente`
Flusso d'eventi:	1. Ciccio chiede a Ugo notizie circa una sua prenotazione
	2. Ugo clicca su **Prenotazione** e compare la finestra delle prenotazioni
	3. Seleziona la prenotazione di Ciccio e clicca su **Visualizza prenotazione**
	4. Si apre la finestra corrispondente e Ugo comunica a Ciccio lo stato della prenotazione
	5. Ugo clicca su **Conferma** e ritorna alla finestra delle prenotazioni

Scenari di magazziniere

Nome dello scenario:	consegna_ordine
Attori partecipanti:	pippo: magazziniere
	arturo: fattorino
Flusso d'eventi:	1. Arturo consegna a Pippo un colle relativo a un ordine e la corrispondente bolla d'accompagnamento
	2. Pippo clicca su **Ordini non evasi**, seleziona l'ordine corrispondente e clicca sul tasto **Carica/visualizza ordine**
	3. Si apre una finestra con la lista dei libri appartenenti all'ordine specificando, per ognuno, il numero di copie attese. Pippo selezionato un libro ne immette il numero di copie consegnate e clicca sul tasto **Carica libri**. Ripete questa operazione per ogni libro dell'ordine consegnato e alla fine clicca su **Conferma**
	5. Il sistema aggiorna il database e l'ordine viene considerato dal sistema evaso o non evaso in funzione della completa consegna dell'ordine

Scenari di gestore_ordini

Nome dello scenario:	inserisci_libro
Attori partecipanti:	poldo: gestore_ordini
Flusso d'eventi:	1. Poldo clicca sul tasto **Libri** e appare una finestra con un form apposito dove gestire i dati relativi ai libri
	2. Poldo inserisce i dati di un nuovo libro nell'apposito form e clicca su **Nuovo libro**
	3. La finestra e il database vengono aggiornati

Nome dello scenario:	modifica_libro
Attori partecipanti:	poldo: gestore_ordini
Flusso d'eventi:	1. Poldo clicca sul tasto **Libri** e appare una finestra con un form apposito dove gestire i dati relativi ai libri
	2. Poldo seleziona dalla lista il libro da modificare,compila il form con i soli nuovi dati e clicca sul tasto **Modifica libro**
	3. Il sistema chiede conferma della modifica, Poldo clicca su **Si** e il database viene aggiornato

Nome dello scenario:	`cancella_libro`
Attori partecipanti:	`poldo: gestore_ordini`
Flusso d'eventi:	1. Poldo clicca sul tasto **Libri** e appare una finestra con un form apposito dove gestire i dati relativi ai libri
	2. Poldo seleziona dalla lista il libro da cancellare e clicca sul tasto Cancella libro
	3. Il sistema chiede conferma della cancellazione, Poldo clicca su **Si** e il database viene aggiornato
	4. La finestra si chiude e il database viene aggiornato

Nome dello scenario:	`info_libro`
Attori partecipanti:	`poldo: gestore_ordini`
Flusso d'eventi:	1. Poldo clicca sul tasto **Libri** e appare una finestra con un form apposito dove gestire i dati relativi ai libri
	2. Poldo seleziona dalla lista un libro, clicca sul tasto **Info libro**
	3. Appare una finestra con titolo del libro, autori, casa editrice, edizione, codice identificativo, corsi di adozione, docenti del corso numero di studenti dei corsi, periodi dei corsi.
	4. Poldo clicca su **Indietro** e la finestra si chiude

Nome dello scenario:	`aggiungi_corso_a_libro`
Attori partecipanti:	`poldo: gestore_ordini`
Flusso d'eventi:	1. Poldo clicca sul tasto **Libri** e appare una finestra con un form apposito dove gestire i dati relativi ai libri
	2. Poldo seleziona dalla lista un libro, clicca sul tasto **Info libro**
	3. Appare una finestra con titolo del libro, autori, casa editrice, edizione, codice identificativo, corsi di adozione, docenti del corso numero di studenti dei corsi, periodi dei corsi.
	4. Poldo clicca su **Aggiungi corso** e si apre la finestra LibriCorsi 5. Poldo seleziona dalla prima tabella il nome del corso e dalla seconda il nome del libro e preme sul tasto **Aggiungi relazione** per associare il libro al corso. 6.Il sistema, se necessario aggiorna il database. Ugo clicca su **Gestione libri** e ritorna alla finestra Gestione libri

Nome dello scenario:	inserisci_corso
Attori partecipanti:	poldo: gestore_ordini
Flusso d'eventi:	1. Poldo clicca sul tasto **Corsi** e appare una finestra con un form apposito dove gestire i dati relativi ai corsi
	2. Poldo inserisce i dati di un nuovo corso nell'apposito form e clicca su **Nuovo corso**
	3. La finestra e il database vengono aggiornati

Nome dello scenario:	modifica_corso
Attori partecipanti:	poldo: gestore_ordini
Flusso d'eventi:	1. Poldo clicca sul tasto **Corso** e appare una finestra con un form apposito dove gestire i dati relativi ai corsi
	2. Poldo seleziona dalla lista il corso da modificare,compila il form con i soli nuovi dati e clicca sul tasto **Modifica corso**
	3. Il sistema chiede conferma della modifica, Poldo clicca su **Si** e il database viene aggiornato

Nome dello scenario:	cancella_corso
Attori partecipanti:	poldo: gestore_ordini
Flusso d'eventi:	1. Poldo clicca sul tasto **Corso** e appare una finestra con un form apposito dove gestire i dati relativi ai corsi
	2. Poldo seleziona dalla lista il corso da cancellare e clicca sul tasto **Cancella corso**
	3. Il sistema chiede conferma della cancellazione, Poldo clicca su **Si** e il database viene aggiornato
	4. La finestra si chiude e il database viene aggiornato

Nome dello scenario:	info_corso
Attori partecipanti:	poldo: gestore_ordini
Flusso d'eventi:	1. Poldo clicca sul tasto **Corso** e appare una finestra con un form apposito dove gestire i dati relativi ai corso
	2. Poldo seleziona dalla lista un corso, clicca sul tasto **Info corso**
	3. Appare una finestra con nome del corso, docenti, numero di studenti, periodo e la lista di tutti i libri adottati nel corso.
	4. Poldo clicca su **Indietro** e la finestra si chiude

Nome dello scenario:	aggiungi_libro_a_corso
Attori partecipanti:	poldo: gestore_ordini
Flusso d'eventi:	1. Poldo clicca sul tasto **Corso** e appare una finestra con un form apposito dove gestire i dati relativi ai corso
	2. Poldo seleziona dalla lista un corso, clicca sul tasto **Info corso**
	3. Appare una finestra con nome del corso, docenti, numero di studenti, periodo e la lista di tutti i libri adottati nel corso.
	4. Poldo clicca su **Aggiungi libro** e si apre la finestra LibriCorsi 5. Poldo seleziona dalla prima tabella il nome del corso e dalla seconda il nome del libro e preme sul tasto **Aggiungi relazione** per associare il libro al corso. 6.Il sistema, se necessario aggiorna il database. Ugo clicca su **Gestione corsi** e ritorna alla finestra Gestione corsi

Nome dello scenario:	nuovo_ordine
Attori partecipanti:	poldo: gestore_ordini
Flusso d'eventi:	1. Poldo clicca sul tasto **Nuovo ordine** e si apre una finestra con la l'elenco dei libri dove, per ognuno, viene specificato il numero di copie in magazzino, il numero di quelle prenotate, di quelle ordinate ma non consegnate e il numero di copie del nuovo ordine e i dati relativi al rappresentante di riferimento
	2. Poldo seleziona un libro, inserisce nella casella "Numero copie da ordinare" il numero di copie per ogni libro da ordinare e clicca sul tasto **Aggiungi libro**. Ripete tale operazione per tutti i libri da ordinare
	3. Poldo clicca su **Visualizza ordine** la finestra cambia mostrando soltanto l'elenco dei libri ordinati e la possibilità di salvare, modificare, annullare e stampare l'ordine
	4. Poldo clicca su **Conferma,** la finestra si chiude, il database viene aggiornato e l'ordine viene stampato

Nome dello scenario:	visualizza_ordine
Attori partecipanti:	poldo: gestore_ordini
Flusso d'eventi:	1. Poldo selezione dalla lista un ordine e clicca sul tasto **Visualizza ordine**
	2. Poldo clicca su **Conferma** e la finestra si chiude

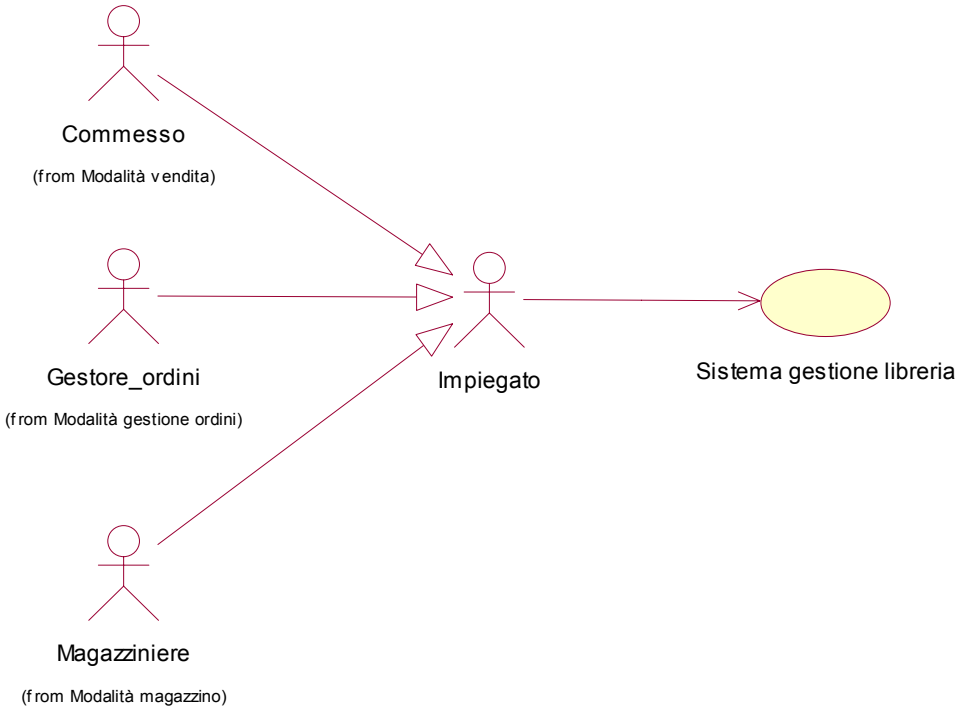
Nome dello scenario:	`cerca_libro`
Attori partecipanti:	`poldo: gestore_ordini`
Flusso d'eventi:	1. Poldo clicca sul tasto **Libri** appare una finestra con un form apposito dove gestire i dati relativi ai libri
	2. Poldo riempe gli appositi campi presenti nel form con i dati di un libro (non necessariamente tutti) e clicca sul tasto **Cerca libro**.
	3. Il sistema riordina la lista dei libri mostrando quelli che soddisfano i requisiti
	4. Poldo clicca sul tasto **Mostra tutti libri** e la lista dei libri viene rivisualizzata

3.5.2 Modelli dei casi d'uso

Di seguito si riportano i casi d'uso del progetto. Il primo diagramma da una vista d'insieme del progetto, i successivi, invece, presenteranno un grado di dettaglio via via crescente.

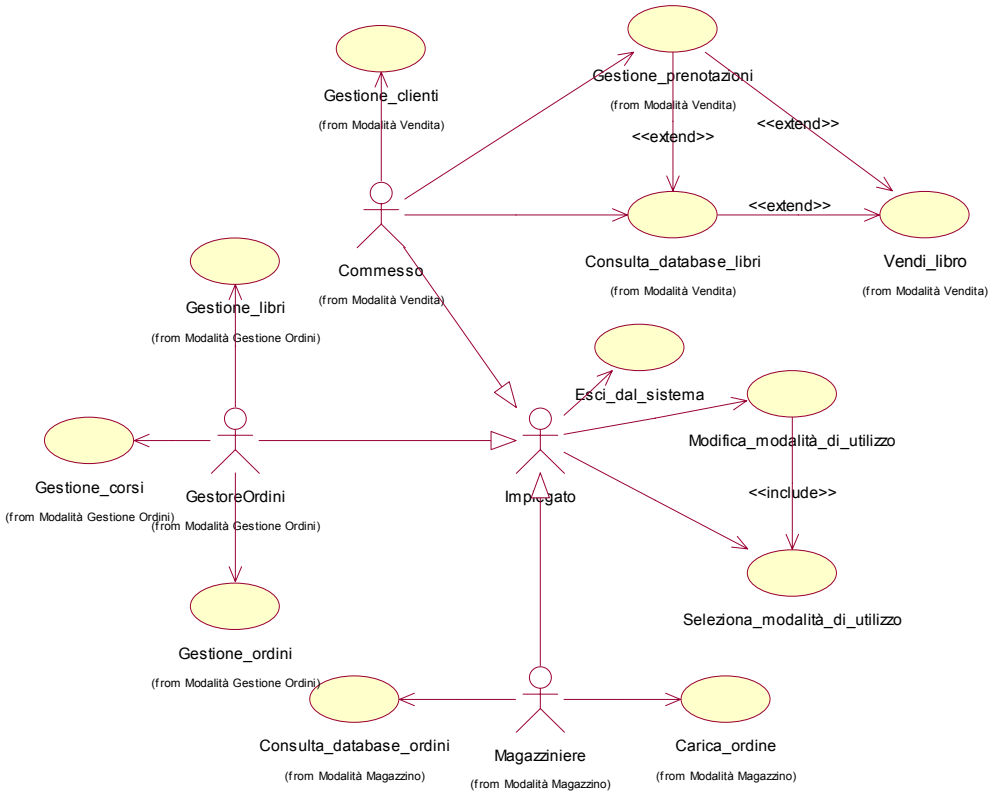
3.5.2.1 Diagramma di contesto del sistema

Il seguente diagramma descrive l'interazione che intercorre tra i diversi attori del sistema.



3.5.2.2 Diagramma delle funzionalità offerte dal sistema

Il seguente diagramma mostra le funzionalità offerte ai singoli attori mantenendo, comunque, un alto livello di astrazione. I casi d'uso più complessi verrano dettagliati con ulteriori diagrammi dei casi d'uso.



Prima di entrare nel dettaglio, verranno mostrati una serie di casi d'uso che sono stati uniformati perché comuni a più figure. Un esempio è il caso d'uso Cerca che può essere utilizzato per i libri, per i clienti, per le prenotazioni e per gli ordini.

Caso d'uso:	Cerca
Attori partecipanti:	Inizializzato da impiegato
Flusso d'eventi:	
Condizione d'ingresso:	1. impiegato, trovandosi in una finestra che permette la ricerca ((2)(2.1)(2.1.1)(2.2)(3)(3.1)(3.1.1)(3.2)(4)), riempe gli appositi campi presenti nel form con i dati da ricercare (non necessariamente tutti) e clicca sul tasto **Cerca**
Condizione d'uscita:	2. SISTEMA riordina la lista nella finestra mostrando quelli che soddisfano i requisiti

Caso d'uso:	Mostra_tutti
Attori partecipanti:	Inizializzato da impiegato
Flusso d'eventi:	
Condizione d'ingresso:	1. impiegato, trovandosi in una finestra che permette la ricerca ((2)(2.1)(2.1.1)(2.2)(3)(3.1)(3.1.1)(3.2)(4)), dopo avere effettuato una ricerca, volendo visualizzare tutto l'elenco (in funzione della finestra in cui si trova) clicca sul tasto **Mostra tutti**
Condizione d'uscita:	2. SISTEMA mostra tutto l'elenco

Caso d'uso:	Indietro
Attori partecipanti:	Inizializzato da impiegato
Flusso d'eventi:	
Condizione d'ingresso:	1. impiegato, trovandosi in una qualunque finestra in cui è presente ((2.1)(2.1.1)(2.1.2)(2.2)(3.1)(3.1.1)(3.2)(3.2.1)(4.1)), clicca sul tasto **Indietro**
Condizione d'uscita:	2. SISTEMA visualizza la finestra di livello immediatamente superiore

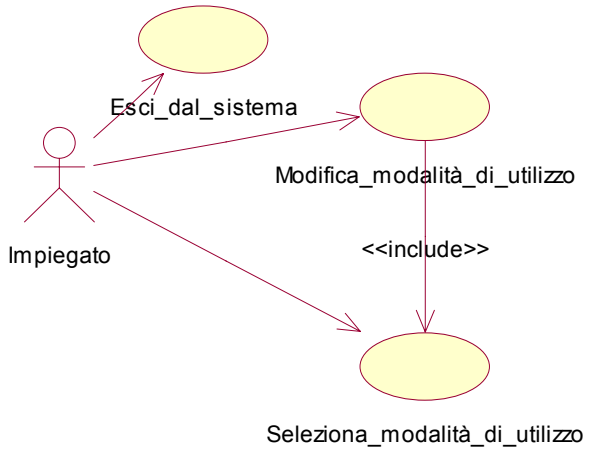
Caso d'uso:	Nuovo
Attori partecipanti:	Inizializzato da impiegato
Flusso d'eventi:	
Condizione d'ingresso:	1. impiegato, trovandosi in una finestra che permette l'immissione di una nuova voce ((2.1)(3.1)(3.1.1)), riempe gli appositi campi presenti nel form con i nuovi dati e clicca sul tasto **Nuovo**
Condizione d'uscita:	2. SISTEMA riordina la lista nella finestra evidenziando nella lista il nuovo dato inserito

Caso d'uso:	`Cancella`
Attori partecipanti:	Inizializzato da `impiegato`
Flusso d'eventi:	
Condizione d'ingresso:	1. `impiegato`, trovandosi in una finestra che permette la cancellazione di una voce del database ((2.1)(2.2)(3.1)), seleziona una voce dalla lista e clicca sul tasto **Cancella**
	2. SISTEMA chiede conferma della cancellazione del dato
	3. `impiegato` fa la sua scelta
Condizione d'uscita:	4. SISTEMA, se necessario, aggiorna il database

Caso d'uso:	`Modifica`
Attori partecipanti:	Inizializzato da `impiegato`
Flusso d'eventi:	
Condizione d'ingresso:	1. `impiegato`, trovandosi in una finestra che permette la modifica di una voce del database ((2.1)(2.2)(3.1)), seleziona la voce, ne modifica i dati interessati e clicca sul tasto **Modifica**
	2. SISTEMA chiede conferma della modifica del dato
	3. `impiegato` fa la sua scelta
Condizione d'uscita:	4. SISTEMA, se necessario, aggiorna il database evidenziando nella lista il dato modificato

3.5.2.2.1 Diagramma delle funzionalità di impiegato

Queste funzionalità sono comuni a tutte e tre le figure (commesso, gestore ordini, magazziniere) operanti nella libreria e per questo vengono generalizzate tramite la figura di impiegato. Di seguito vengono riportati i casi d'uso relativi.



Caso d'uso:	`Selezione_modalita_vendita_libri`
Attori partecipanti:	Inizializzato da `impiegato`
Flusso d'eventi:	
Condizione d'ingresso:	1. `impiegato` avvia il programma di gestione della libreria
	2. SISTEMA visualizza la finestra scelta della modalità di utilizzo (1)
	3. `impiegato` clicca sul tasto **Vendita libri**
Condizione d'uscita:	4. SISTEMA visualizza la finestra vendita libri (2)

Caso d'uso:	`Selezione_modalita_gestione_ordini`
Attori partecipanti:	Inizializzato da `impiegato`
Flusso d'eventi:	
Condizione d'ingresso:	1. `impiegato` avvia il programma di gestione della libreria
	2. SISTEMA visualizza la finestra scelta della modalità di utilizzo (1)
	3. `impiegato` clicca sul tasto **Gestione ordini**
Condizione d'uscita:	4. SISTEMA visualizza la finestra gestione ordini (3)

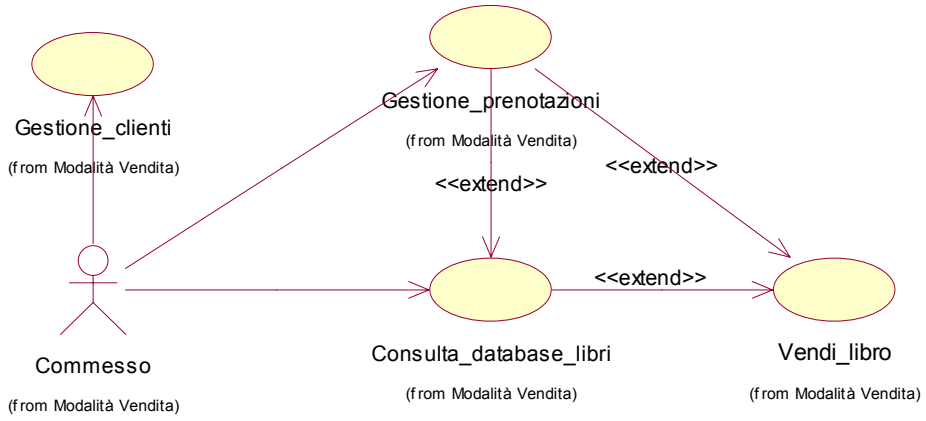
Caso d'uso:	Selezione_modalita_magazzino
Attori partecipanti:	Inizializzato da impiegato
Flusso d'eventi:	
Condizione d'ingresso:	1. impiegato avvia il programma di gestione della libreria
	2. SISTEMA visualizza la finestra scelta della modalità di utilizzo (1)
	3. impiegato clicca sul tasto **Magazzino**
Condizione d'uscita:	4. SISTEMA visualizza la finestra magazzino (4)

Caso d'uso:	Modifica_modalità_di_utilizzo
Attori partecipanti:	Inizializzato da impiegato
Flusso d'eventi:	
Condizione d'ingresso:	1. impiegato, trovandosi nella finestra iniziale di una qualunque modalità ((2)(3)(4)), clicca sul tasto **Modifica modalità.**
	2. SISTEMA visualizza una finestra con scritto "Sei sicuro di voler cambiare modalità?"
	3.impiegato seleziona **Si**
	4. SISTEMA visualizza la finestra scelta della modalità di utilizzo (1)
	5. impiegato seleziona una nuova modalità di utilizzo tra quelle disponibili (vendita libri, gestione ordini, magazzino)
Condizione d'uscita:	6. SISTEMA visualizza la finestra della modalità selezionata

Caso d'uso:	Esci
Attori partecipanti:	Inizializzato da impiegato
Flusso d'eventi:	
Condizione d'ingresso:	1. impiegato, trovandosi nella finestra scelta della modalità di utilizzo (1) o in una qualunque finestra iniziale ((2)(3)(4)) clicca sul tasto **Esci**
	2. SISTEMA visualizza una finestra con scritto "Sei sicuro di voler uscire?"
	3. impiegato risponde alla domanda
Condizione d'uscita:	4. SISTEMA, se necessario, chiude il programma

3.5.2.2.2 Diagramma delle funzionalità di Commesso

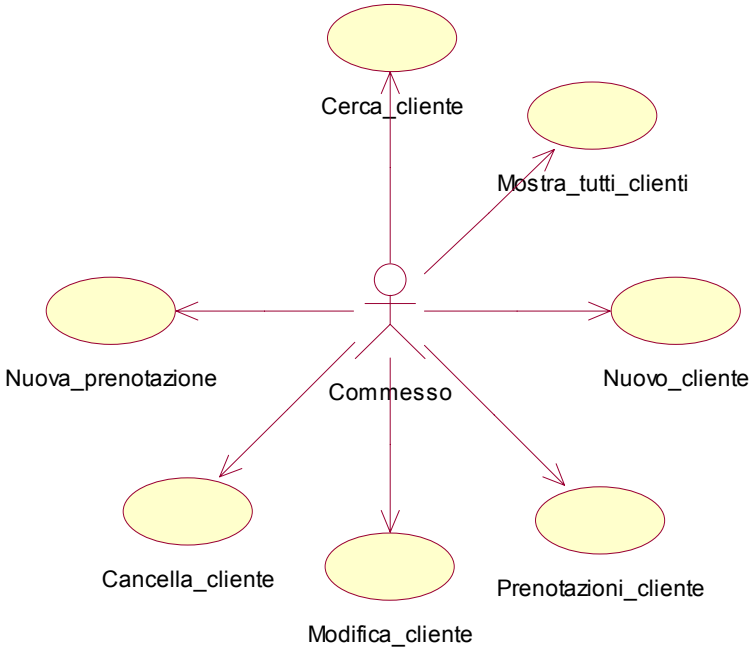
Questi casi d'uso sono a carattere generale e verranno a loro volta espansi sotto forma di altri diagrammi dei casi d'uso.



Caso d'uso:	`Gestione_clienti`
Attori partecipanti:	Inizializzato da `commesso`
Flusso d'eventi:	
Condizione d'ingresso:	1. `commesso` sta lavorando nel pagina vendita libri (2) e clicca sul tasto **Clienti**
Condizione d'uscita:	2. SISTEMA apre la finestra clienti (2.1) visualizzando una lista con nome, cognome, indirizzo, data di nascita, numero di telefono di tutti i clienti

Caso d'uso:	`Gestione_prenotazioni`
Attori partecipanti:	Inizializzato da `commesso`
Flusso d'eventi:	
Condizione d'ingresso:	1. `commesso` sta lavorando nel pagina vendita libri (2) e clicca sul tasto **Prenotazioni**
Condizione d'uscita:	2. SISTEMA apre la finestra prenotazioni (2.1) visualizzando una lista di tutte le prenotazioni specificandone numero prenotazioni, data emissione, evasa, data consegna, totale, acconto, saldo.

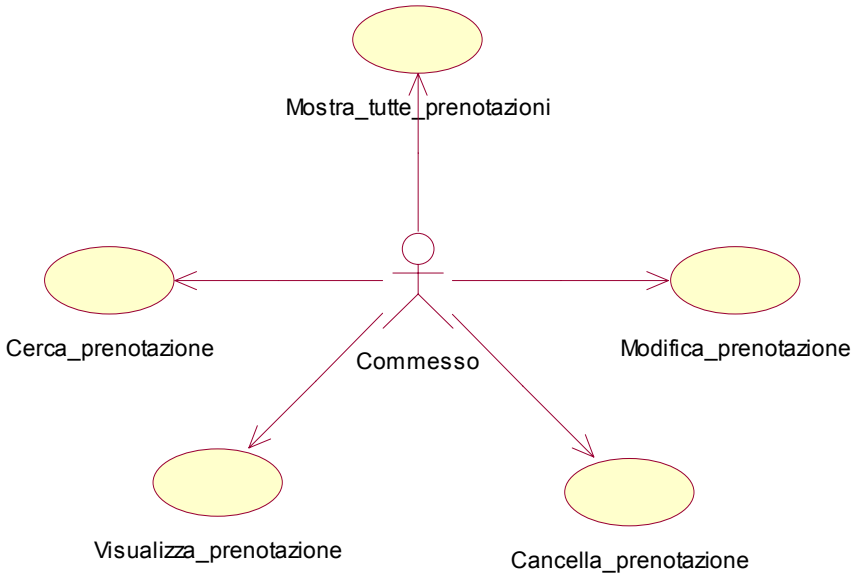
3.5.2.2.2.1 Diagramma delle funzionalità del caso d'uso Gestione_clienti



Caso d'uso:	Nuova_prenotazione
Attori partecipanti:	Inizializzato da commesso
Flusso d'eventi:	
Condizione d'ingresso:	1. commesso sta lavorando nel pagina clienti (2.1). Seleziona dalla lista il nome di un cliente e clicca sul tasto **Nuova prenotazione**
	2. SISTEMA apre la pagina corrispondente (2.1.1)
	3. commesso, dopo aver selezionato un libro dall'elenco e dopo averne inserito il numero di copie da prenotare, clicca su **Aggiungi libro** e itera il procedimento per ogni libro da inserire nella prenotazione. Quando finito e clicca su **Visualizza prenotazione**
	4. SISTEMA apre la finestra conferma prenotazione libri (2.1.1.1), visualizzando la lista di tutti i libri facenti parte della prenotazione specificando per ognuno le copie prenotate, quelle consegnate e quelle disponibili alla consegna 5. commesso inserisce nel form apposito l'importo dell'acconto clicca su **Conferma**
Condizione d'uscita:	6. SISTEMA aggiorna il database e ritorna alla pagina vendita libri (2)

Caso d'uso:	`Prenotazioni cliente`
Attori partecipanti:	Inizializzato da `commesso`
Flusso d'eventi:	
Condizione d'ingresso:	1. `commesso` sta lavorando nel pagina clienti (2.1). Seleziona dalla lista il nome di un cliente e clicca sul tasto **Prenotazioni cliente**
Condizione d'uscita:	2. SISTEMA apre la finestra corrispondente (2.1.2) visualizzando nome, cognome, data di nascita, indirizzo, numero di telefono e prenotazioni effettuate

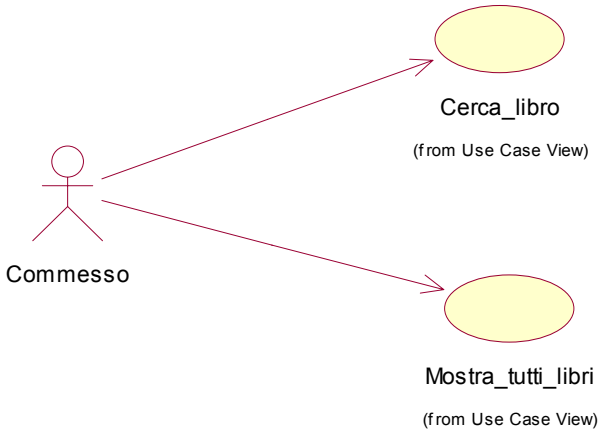
3.5.2.2.2.2 Diagramma delle funzionalità del caso d'uso Gestione_prenotazioni



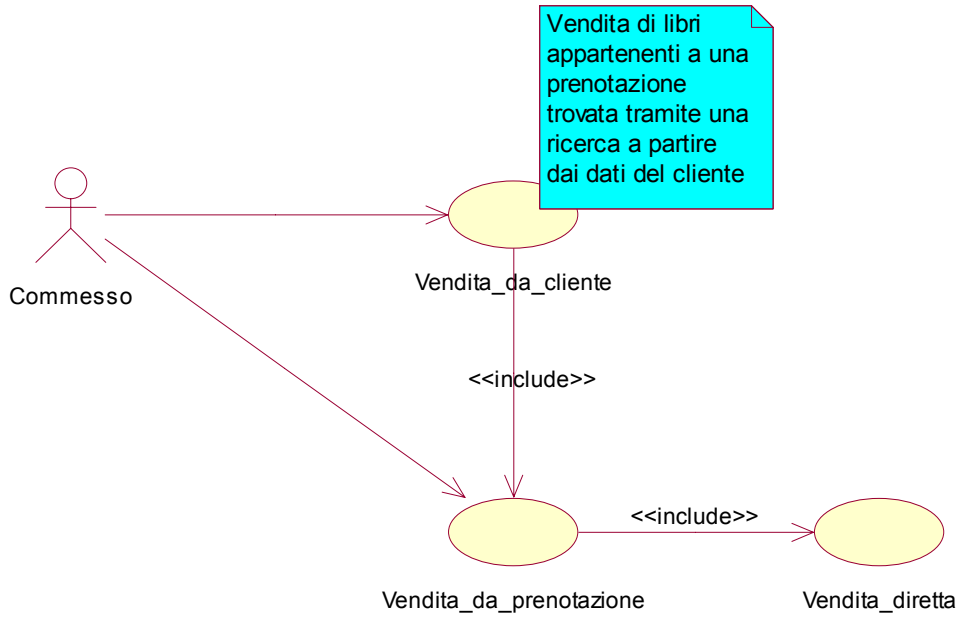
Caso d'uso:	`Modifica_prenotazione`
Attori partecipanti:	Inizializzato da `commesso`
Flusso d'eventi:	
Condizione d'ingresso:	1. `commesso`, trovandosi in una finestra che permette la modifica di una prenotazione ((2.1.1.1)(2.1.2)(2.2)), seleziona una prenotazione e clicca sul tasto **Modifica prenotazione** per visualizzarla
	2. SISTEMA apre la pagina corrispondente (2.1.1)
	3. `commesso` modifica la prenotazione dei libri utilizzando il tasto **Aggiungi** per aggiungere un nuovo libro alla prenotazione, **Rimuovi** per rimuovere un libro dalla prenotazione e **Modifica** per modificare il numero di copie di un certo libro. Quando finito clicca su **Visualizza prenotazione**
	2. SISTEMA apre la finestra conferma prenotazione libri (2.1.1.1), visualizzando la lista di tutti i libri facenti parte della prenotazione specificando per ognuno le copie prenotate, quelle consegnate e quelle disponibili alla consegna 5. `commesso` clicca su **Conferma**
Condizione d'uscita:	6. SISTEMA aggiorna il database e ritorna alla pagina vendita libri (2)

Caso d'uso:	`Visualizza prenotazione`
Attori partecipanti:	Inizializzato da `commesso`
Flusso d'eventi:	
Condizione d'ingresso:	1. `commesso` sta lavorando nel pagina prenotazioni (2.2). Seleziona dalla lista la prenotazione di un cliente e clicca sul tasto **Visualizza prenotazione**
Condizione d'uscita:	2. SISTEMA apre la finestra conferma prenotazione libri (2.1.1.1), visualizzando i dati della prenotazione e la lista di tutti i libri specificando per ognuno le copie prenotate, quelle consegnate e quelle disponibili alla consegna

3.5.2.2.2.3 Diagramma delle funzionalità del caso d'uso Consulta_database_libri



3.5.2.2.2.4 Diagramma delle funzionalità del caso d'uso Vendita_libro

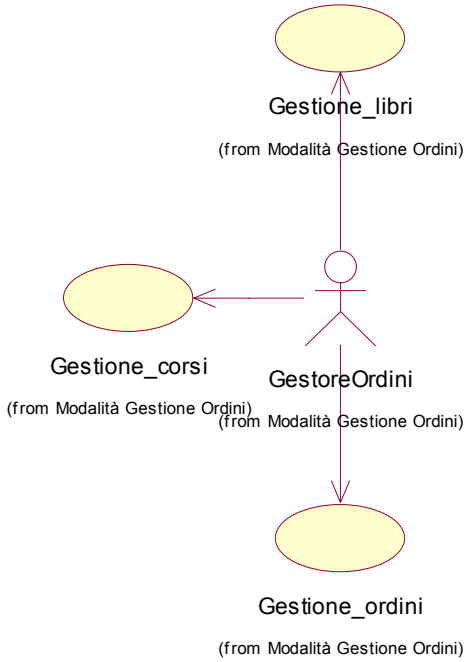


Caso d'uso:	`Vendita_diretta`
Attori partecipanti:	Inizializzato da `commesso`
Flusso d'eventi:	
Condizione d'ingresso:	1. `commesso` seleziona un libro da vendere e clicca sul tasto **Vendi**
Condizione d'uscita:	2. SISTEMA aggiorna il database decrementando la disponibilità del libro
Eccezione:	Se il libro non è disponibile alla vendita o il numero di copie in magazzino è minore o uguale a quello delle copie prenotate SISTEMA mostra un messaggio di errore

Caso d'uso:	Vendita da prenotazione
Attori partecipanti:	Inizializzato da commesso
Flusso d'eventi:	
Condizione d'ingresso:	1. commesso, trovandosi nella finestra prenotazioni (2.2), seleziona una prenotazione e clicca sul tasto **Visualizza prenotazione**
	2. SISTEMA apre la finestra conferma prenotazione libri (2.1.1.1), visualizzando la lista di tutti i libri facenti parte della prenotazione specificando per ognuno le copie prenotate, quelle consegnate e quelle disponibili alla consegna
	3. commesso seleziona un libro prenotato e disponibile alla vendita e clicca sul tasto **Vendi**
	4. SISTEMA decrementa sia il numero di copie prenotate, sia il numero di copie disponibili in magazzino
	5. commesso terminata la fase di vendita clicca sul tasto **Conferma**
Condizione d'uscita:	6. SISTEMA aggiorna il database, considerando la prenotazione evasa nel caso in cui tutti i suoi libri sono stati consegnati e ritorna alla finestra di vendita libri (2)
Eccezione:	Se il libro selezionato non è disponibile alla vendita o risultano già consegnate le copie prenotate SISTEMA visualizza un messaggio di errore non permettendone la vendita

Caso d'uso:	Vendita da cliente
Attori partecipanti:	Inizializzato da commesso
Flusso d'eventi:	
Condizione d'ingresso:	1. commesso, trovandosi nella finestra clienti (2.1), seleziona un cliente e clicca sul tasto **Prenotazioni cliente**
	2.SISTEMA apre la finestra info cliente
	3. commesso seleziona una prenotazione e clicca sul tasto **Visualizza**
	4. SISTEMA apre la finestra conferma prenotazione libri (2.1.1.1), visualizzando la lista di tutti i libri facenti parte della prenotazione specificando per ognuno le copie prenotate, quelle consegnate e quelle disponibili alla consegna
	5. commesso seleziona un libro prenotato e disponibile alla vendita e clicca sul tasto **Vendi**
	6. SISTEMA decrementa sia il numero di copie prenotate, sia il numero di copie disponibili in magazzino
	7. commesso terminata la fase di vendita clicca sul tasto **Conferma**
Condizione d'uscita:	8. SISTEMA aggiorna il database, considerando la prenotazione evasa nel caso in cui tutti i suoi libri sono stati consegnati e ritorna alla finestra di vendita libri (2)
Eccezione:	Se il libro selezionato non è disponibile alla vendita o risultano già consegnate le copie prenotate SISTEMA visualizza un messaggio di errore non permettendone la vendita

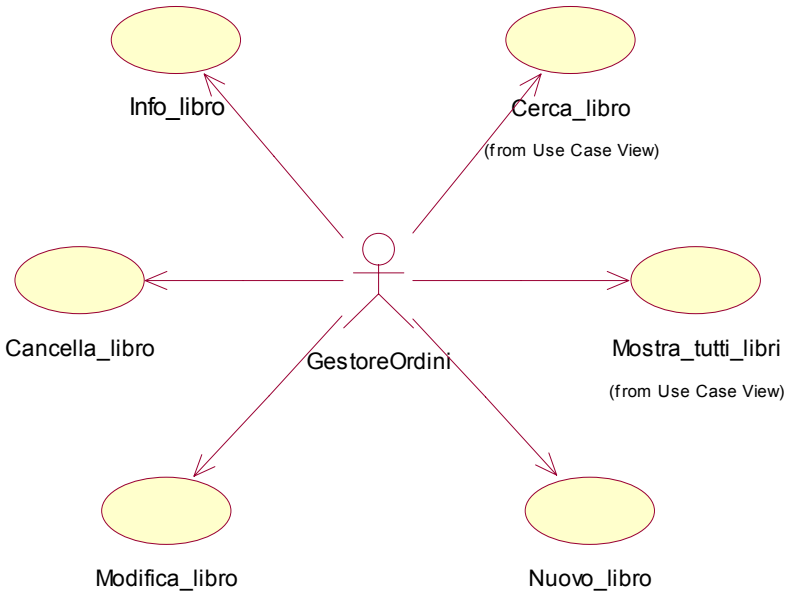
3.5.2.2.3 Diagramma delle funzionalità di Gestore_ordini



Caso d'uso:	`Gestione_libri`
Attori partecipanti:	Inizializzato da `gestore_ordini`
Flusso d'eventi:	
Condizione d'ingresso:	1. `gestore_ordini` sta lavorando nel pagina gestione ordini (3) e clicca sul tasto **Libri**
Condizione d'uscita:	2. SISTEMA apre la finestra gestione libri (3.1)

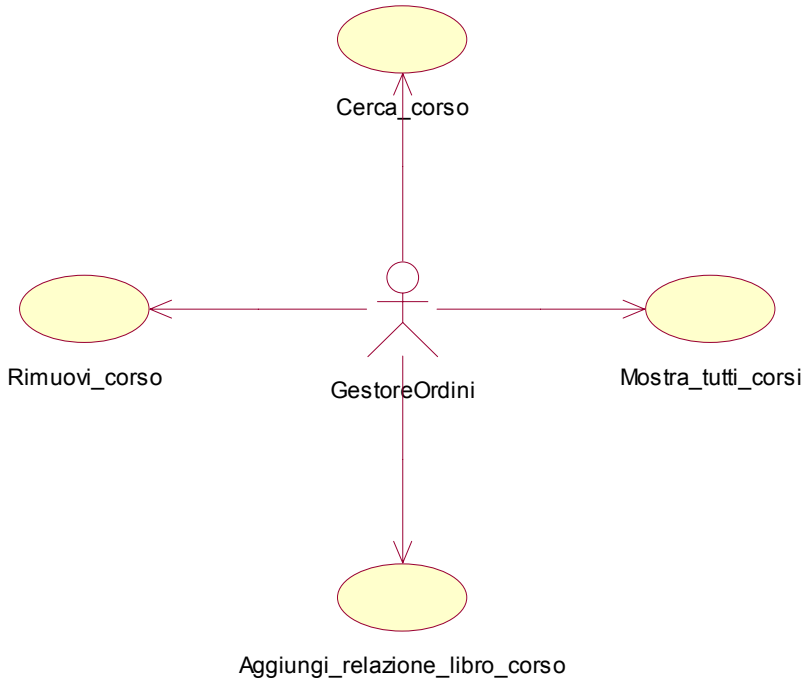
Caso d'uso:	`Gestione_corsi`
Attori partecipanti:	Inizializzato da `gestore_ordini`
Flusso d'eventi:	
Condizione d'ingresso:	1. `gestore_ordini` sta lavorando nel pagina gestione ordini (3) e clicca sul tasto **Corsi**
Condizione d'uscita:	2. SISTEMA apre la finestra gestione corsi (3.3)

3.5.2.2.3.1 Diagramma delle funzionalità del caso d'uso Gestione_libri



Caso d'uso:	`Info_libro`
Attori partecipanti:	Inizializzato da `gestore_ordini`
Flusso d'eventi:	
Condizione d'ingresso:	1. gestore_ordini, trovandosi nella finestra gestione libri (3.1), seleziona dalla lista un libro e clicca sul tasto **Info libro**
	2. SISTEMA visualizza una finestra con un form apposito con i dati relativi al libro (titolo, autori, casa editrice, edizione, anno, tipologia/argomento), la lista dei corsi in cui è adottato, il numero di studenti di ogni corso e il periodo dei corsi
Condizione d'uscita:	3. `gestore_ordini` clicca su **Indietro** 4. SISTEMA apre la finestra gestione libri (3.1)

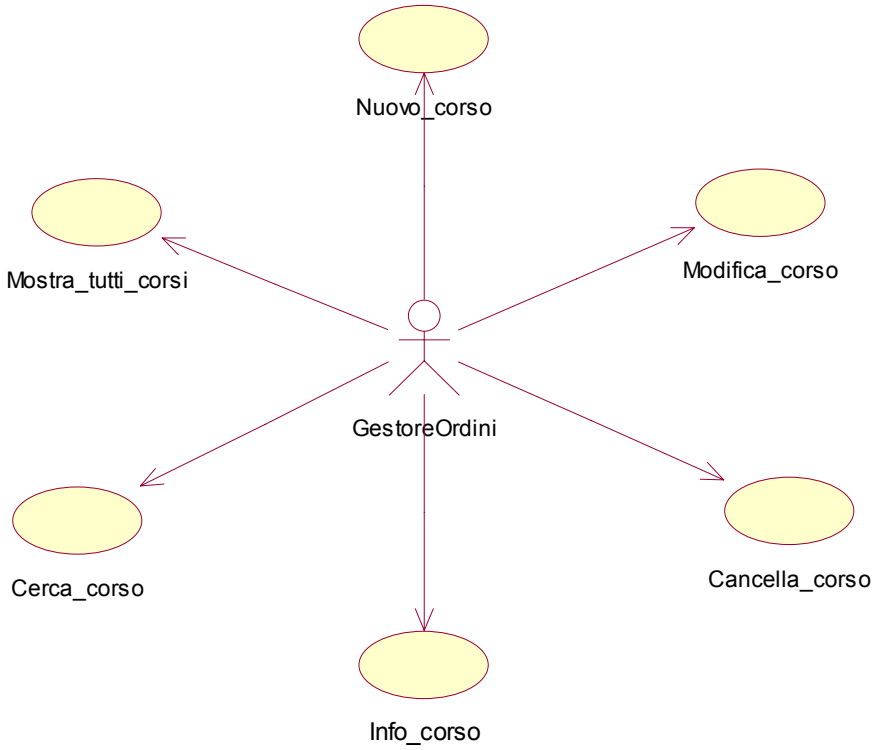
3.5.2.2.3.1.1 Diagramma delle funzionalità del caso d'uso Info_libro



Caso d'uso:	Rimuovi corso
Attori partecipanti:	Inizializzato da gestore ordini
Flusso d'eventi:	
Condizione d'ingresso:	1. gestore_ordini, trovandosi nella finestra info libro (3.1.1), seleziona dalla lista un corso e clicca sul tasto **Rimuovi corso**
	2. SISTEMA chiede conferma dell'operazione
	3. gestore ordini fa la sua scelta
Condizione d'uscita:	4. SISTEMA, se necessario, aggiorna il database

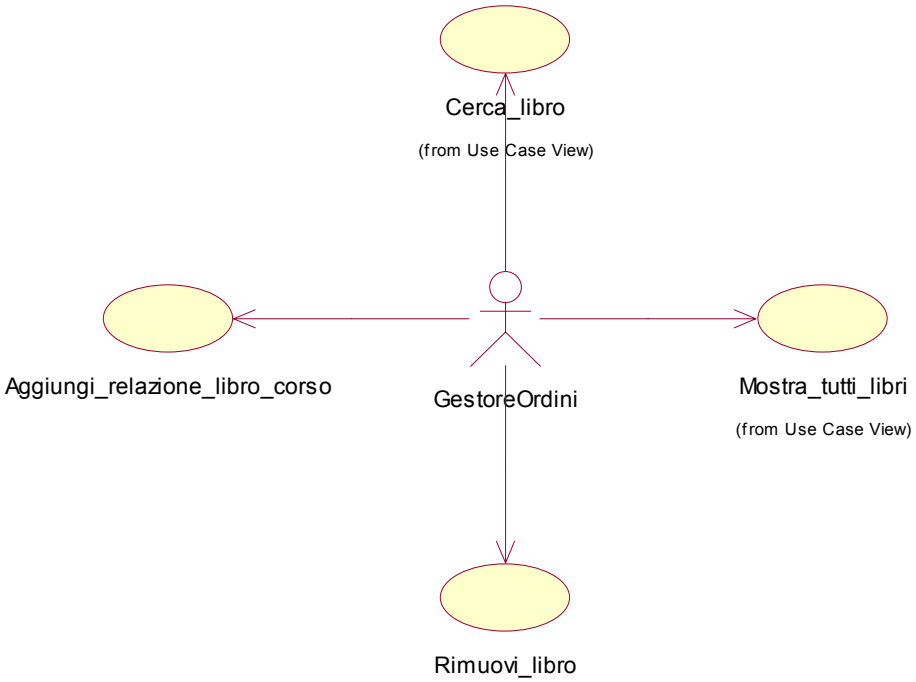
Caso d'uso:	`Aggiungi_relazione_libro_corso`
Attori partecipanti:	Inizializzato da `gestore_ordini`
Flusso d'eventi:	
Condizione d'ingresso:	1. gestore_ordini, trovandosi nella finestra info libro (3.1.1) clicca sul tasto **Aggiungi corso**
	2. SISTEMA visualizza la finestra libri corsi (3.1.1.1) con un due tabelle una con la lista di tutti i libri e l'altra con quella di tutti i corsi
	3. `gestore_ordini` seleziona un libro e un corso e clicca sul tasto **Aggiungi relazione**
	4. SISTEMA se necessario aggiorna il database
	5. `gestore_ordini` clicca su **Gestione libri**
Condizione d'uscita:	6. SISTEMA apre la finestra gestione libri (3.1)

3.5.2.2.3.2 Diagramma delle funzionalità del caso d'uso Gestione_corsi



Caso d'uso:	`Info_corso`
Attori partecipanti:	Inizializzato da `gestore_ordini`
Flusso d'eventi:	
Condizione d'ingresso:	1. gestore_ordini, trovandosi nella finestra gestione corsi (3.2), seleziona dalla lista un corso e clicca sul tasto **Info corso**
	2. SISTEMA visualizza una finestra con un form apposito con i dati relativi al corso (nome corso, docente, periodo, numero di studenti) e la lista dei libri adottati
	3. `gestore_ordini` clicca su **Indietro**
Condizione d'uscita:	4. SISTEMA apre la finestra gestione corsi (3.2)

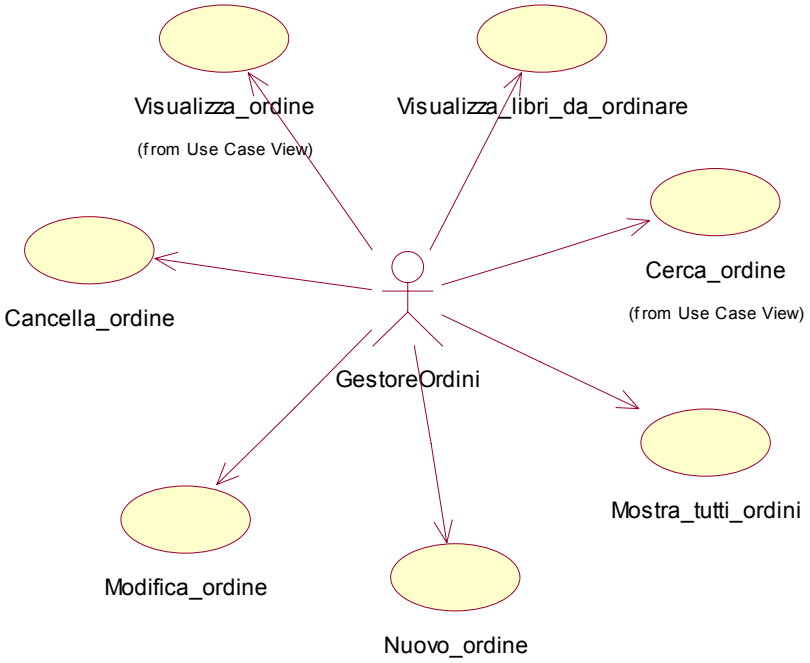
3.5.2.2.3.2.1 Diagramma delle funzionalità del caso d'uso Info_libro



Caso d'uso:	`Rimuovi libro`
Attori partecipanti:	Inizializzato da `gestore ordini`
Flusso d'eventi:	
Condizione d'ingresso:	1. gestore_ordini, trovandosi nella finestra info corso (3.2.1), seleziona dalla lista un libro e clicca sul tasto **Rimuovi libro**
	2. SISTEMA chiede conferma dell'operazione
	3. `gestore ordini` fa la sua scelta
Condizione d'uscita:	4. SISTEMA, se necessario, aggiorna il database

Caso d'uso:	`Aggiungi relazione libro corso`
Attori partecipanti:	Inizializzato da `gestore_ordini`
Flusso d'eventi:	
Condizione d'ingresso:	1. gestore_ordini, trovandosi nella finestra info corso (3.2.1) clicca sul tasto **Aggiungi libro**
	2. SISTEMA visualizza la finestra libri corsi (3.1.1.1) con un due tabelle una con la lista di tutti i libri e l'altra con quella di tutti i corsi
	3. `gestore_ordini` seleziona un libro e un corso e clicca sul tasto **Aggiungi relazione**
	4. SISTEMA se necessario aggiorna il database
	5. `gestore_ordini` clicca su **Gestione corsi** (3.2)
Condizione d'uscita:	6. SISTEMA apre la finestra gestione corsi

3.5.2.2.3.3 Diagramma delle funzionalità del caso d'uso Gestione_ordini



Caso d'uso:	`Visualizza_libri_da_ordinare`
Attori partecipanti:	Inizializzato da `gestore_ordini`
Flusso d'eventi:	
Condizione d'ingresso:	1. `gestore_ordini` clicca sul tasto **Visualizza libri da ordinare** per visualizzare la lista di tutti i libri che sono stati prenotti e non sono disponibili alla vendita
Condizione d'uscita:	2. SISTEMA visualizza la lista dei libri in questione

Caso d'uso:	Nuovo_ordine
Attori partecipanti:	Inizializzato da gestore_ordini
Flusso d'eventi:	
Condizione d'ingresso:	1. gestore_ordini si trova nella finestra modalità gestione ordini (3) e clicca sul tasto **Nuovo ordine**
	2. SISTEMA apre la finestra nuovo ordine (3.3) attribuento automaticamente al nuovo ordine un numero identificativo e la data di emissione. SISTEMA visualizza inoltre l'elenco di tutti i libri specificandone per ognuno il rappresentante a cui inoltrare l'ordine, il numero di copie da nuovo da ordinare, il numero di copie in magazzino, il numero di copie già ordinate ma non consegnate e il numero di copie prenotate
	3. gestore_ordini, dopo aver selezionato un libro dall'elenco e dopo averne inserito il numero di copie da ordinare, clicca su **Aggiungi libro** e itera il procedimento per ogni libro da inserire nell'ordine. Quando finito clicca su **Visualizza ordine**
	4. SISTEMA mostra la finestra conferma ordine libri (3.3.1), visualizzando soltanto l'elenco dei libri appartenenti all'ordine dando la possibilità di annullare, modificare o confermare l'ordine
	5. gestore_ordini fa la sua scelta
Condizione d'uscita:	6. SISTEMA, se necessario, aggiorna il database e stampa l'ordine e ritorna alla pagina di gestione ordini (3)

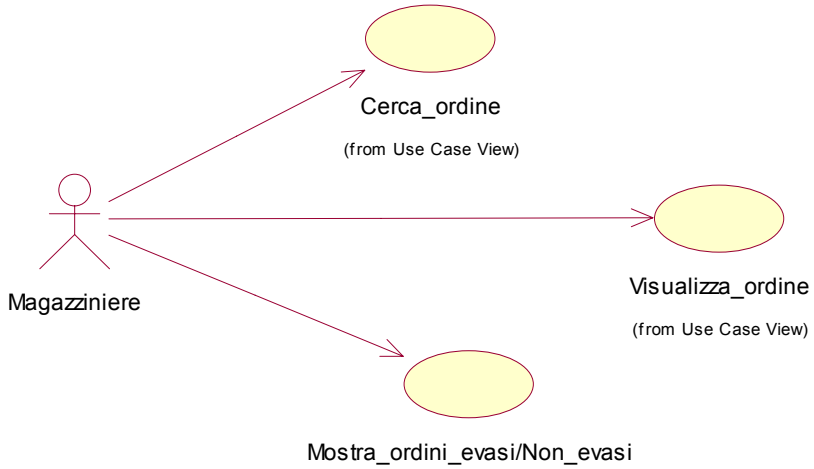
Caso d'uso:	Modifica_ordine
Attori partecipanti:	Inizializzato da gestore_ordini
Flusso d'eventi:	
Condizione d'ingresso:	1. gestore_ordini, trovandosi in una finestra che permette la modifica di un ordine ((3)(3.3.1)), seleziona dalla lista un ordine e clicca sul tasto **Modifica ordine**
	2. SISTEMA apre la pagina corrispondente (3.3)
	3. gestore_ordini modifica l'ordine dei libri utilizzando il tasto **Aggiungi** per aggiungere un nuovo libro all'ordine, **Rimuovi** per rimuovere un libro dall'ordine e **Modifica** per modificare il numero di copie di un certo libro. Quando finito clicca su **Visualizza ordine**
	4. SISTEMA apre la finestra conferma prenotazione libri (3.3.1), visualizzando la lista di tutti i libri facenti parte dell'ordine specificando per ognuno il numero di copie nell'ordine, il numero di quelle in magazzino, il numero di quelle già ordinate ma non ancora consegnate e il numero di quelle prenotate
	5. commesso clicca su **Conferma**
Condizione d'uscita:	6. SISTEMA stampa l'ordine, aggiorna il database e ritorna alla pagina gestione ordini(3)

Caso d'uso:	Visualizza_ordine
Attori partecipanti:	Inizializzato da gestore_ordini
Flusso d'eventi:	
Condizione d'ingresso:	1. gestore_ordini, trovandosi in una pagina che permette la visualizzazione di un ordine ((3)(3.3)), clicca sul tasto **Visualizza ordine** per visualizzare dettagliatamente l'ordine selezionato
Condizione d'uscita:	2. SISTEMA apre la finestra conferma prenotazione libri (3.3.1), visualizzando la lista di tutti i libri facenti parte dell'ordine specificando per ognuno il numero di copie nell'ordine, il numero di quelle in magazzino, il numero di quelle già ordinate ma non ancora consegnate e il numero di quelle prenotate

3.5.2.2.4 Diagramma delle funzionalità di Magazziniere

Caso d'uso:	Consegna ordine
Attori partecipanti:	Inizializzato da magazziniere
Flusso d'eventi:	
Condizione d'ingresso:	1. magazziniere sta lavorando nella finestra modalità magazzino (4) e clicca su **Ordini non evasi**
	2. SISTEMA visualizza nella lista presente nella pagina della modalita magazzino solo gli ordini non evasi
	3. magazziniere seleziona un ordine e clicca sul tasto **Carica/visualizza ordine**
	4. SISTEMA apre la finestra carica/visualizza ordine (4.1) con la lista dei dati dei libri appartenenti all'ordine specificandone la quantità attesa di ognuno.
	5. magazziniere seleziona un libro dalla lista indicandone il numero di copie consegnate e clicca su **Carica libri**. Ripete tale operazione per ogni libro consegnato appartenente all'ordine e, quando finito, clicca su **Conferma**
Condizione d'uscita:	6. SISTEMA aggiorna il database e l'ordine viene considerato dal sistema evaso o non evaso in funzione del fatto che i libri caricati siano tutti quelli dell'ordine o solo una parte e si riporta alla finestra di modalità magazzino (4.1)

3.5.2.2.4.1 Diagramma delle funzionalità del caso d'uso Consulta_database_ordini



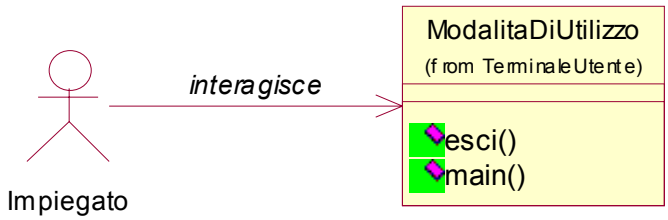
Caso d'uso:	`Mostra ordini evasi/Non evasi`
Attori partecipanti:	Inizializzato da `magazziniere`
Flusso d'eventi:	
Condizione d'ingresso:	1. `magazziniere` sta lavorando nella finestra modalità magazzino (4) e clicca su **Ordini evasi / Ordini non evasi**
	2. SISTEMA visualizza nella lista presente nella pagina della modalita magazzino solo gli ordini evasi / ordini non evasi

3.5.3 Modelli degli oggetti

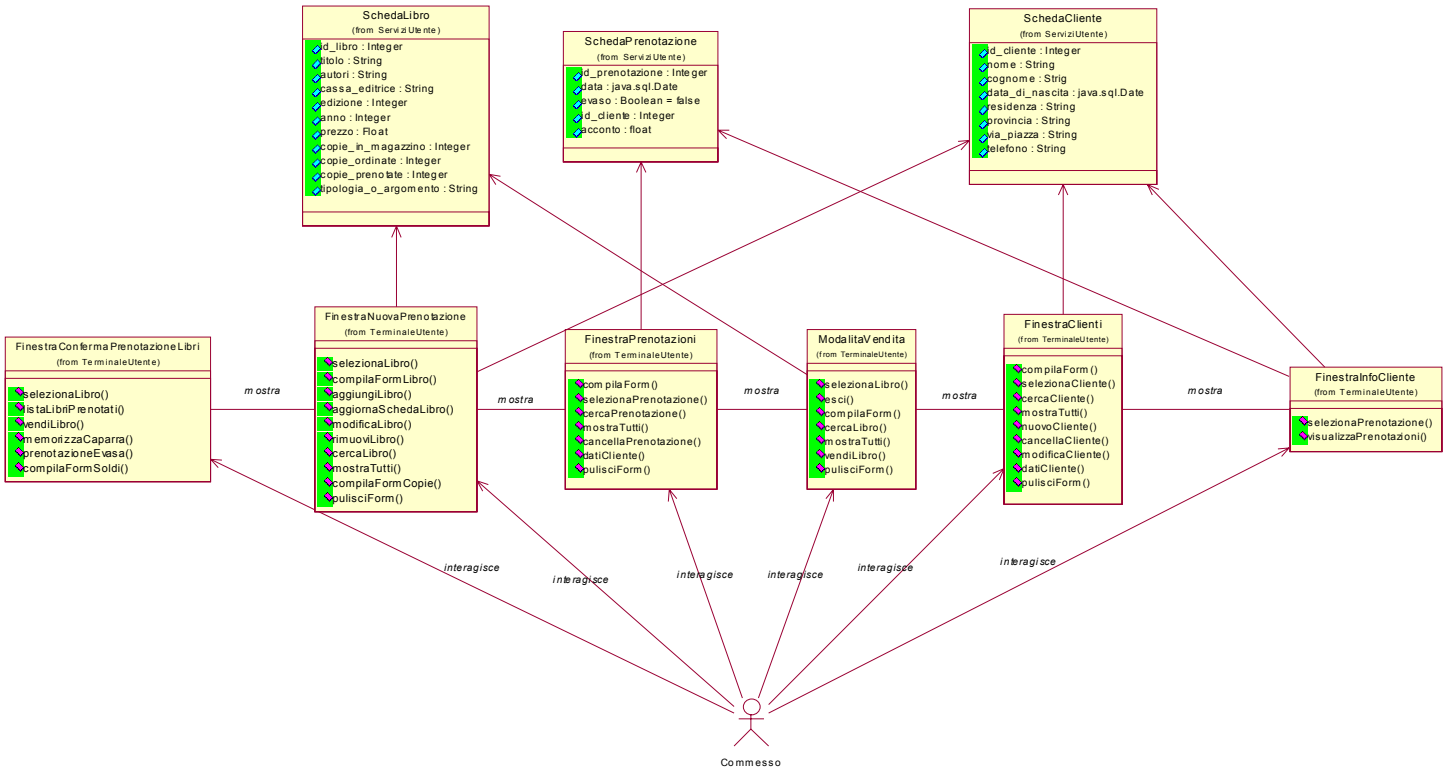
3.5.3.1 Diagramma delle classi del dominio

Anche per lo sviluppo in fase di analisi del diagramma delle classi si è preferito procedere separando i diagrammi in funzione dell'attore principale. Seguiranno, quindi, quattro diagrammi, ognuno relativo a un particolare attore.

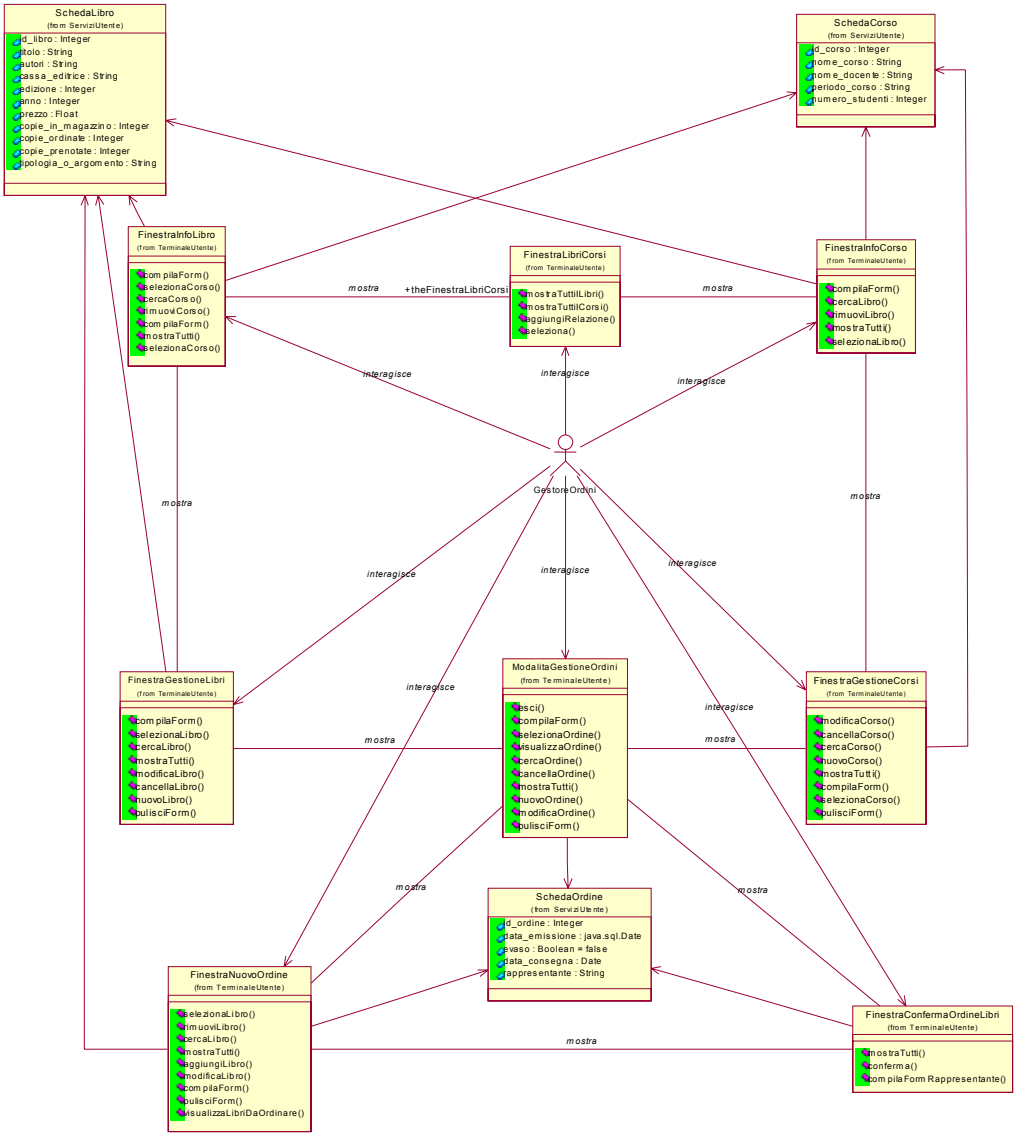
3.5.3.1.1 Diagramma delle classi del dominio di Impiegato



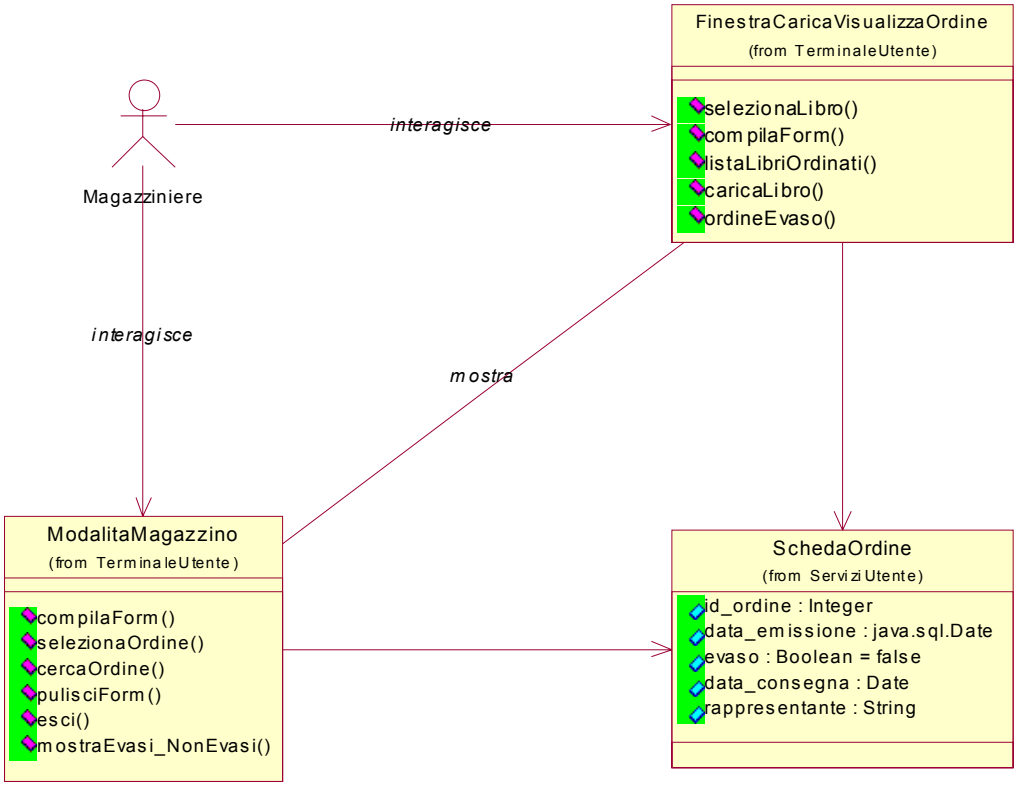
3.5.3.1.2 Diagramma delle classi del dominio di Commesso



3.5.3.1.3 Diagramma delle classi del dominio di Gestore_ordini



3.5.3.1.4 Diagramma delle classi del dominio di Magazziniere



3.5.4 Modelli dinamici

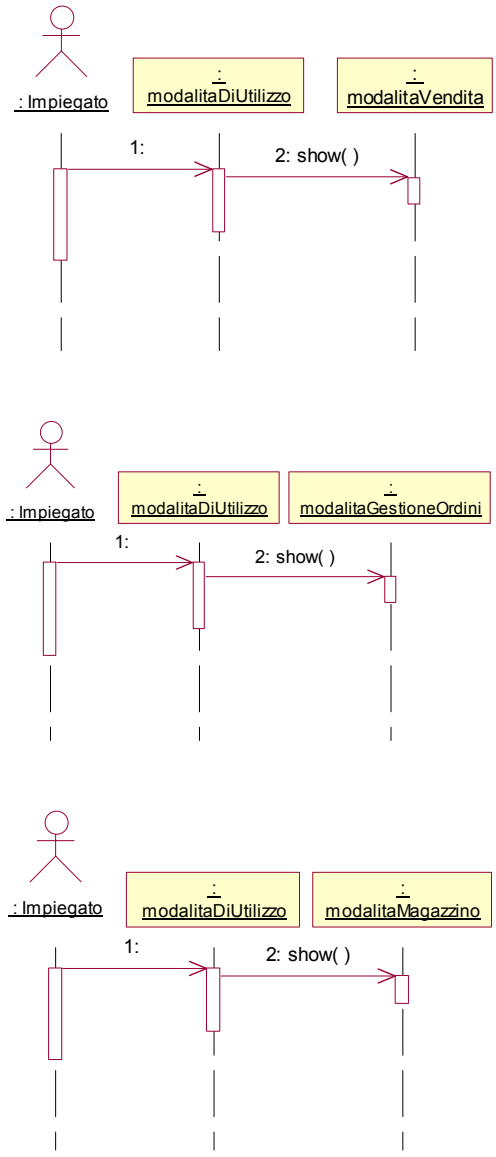
3.5.4.1 Diagrammi di sequenza

Qui si riportano i diagrammi di sequenza specificando, per ognuno, l'attore principale in quanto avremo modelli dinamici che descrivono casi d'uso analoghi ma compiuti da attori differenti.

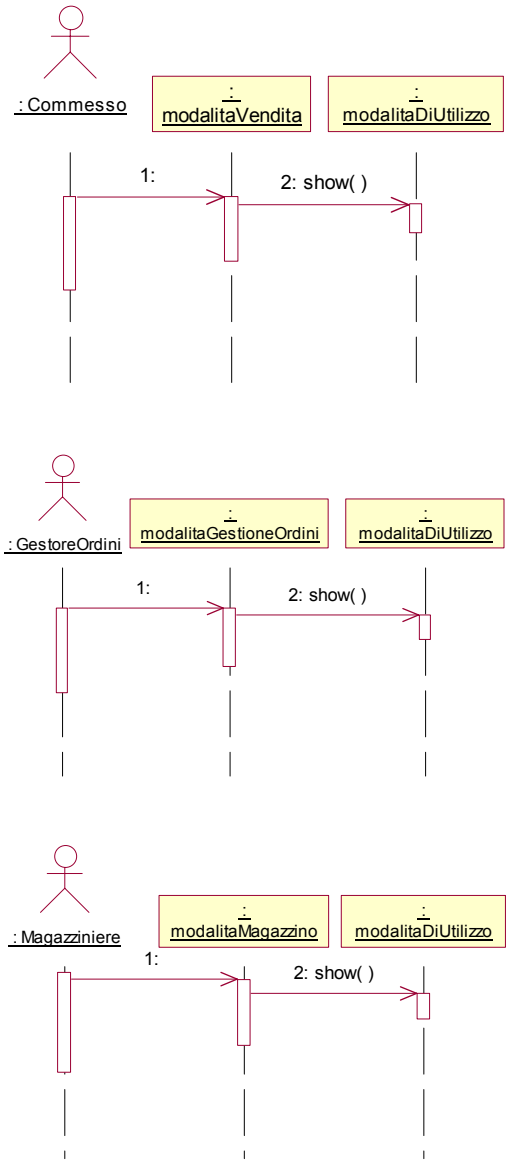
Per evitare di perdere di vista l'obiettivo, cioè l'azione compiuta dall'attore, assumeremo sempre che si stia lavorando nella finestra associata all'operazione. Nella sezione **3.5.5** verranno mostrati i possibili percorsi che permettono di raggiungere la finestra interessata all'azione.

L'assenza del nome in un object message significherà l'utilizzazione di un metodo correlato ad una azione strettamente legata all'interfaccia grafica (pressione di un bottone).

3.5.4.1.1 Scelta modalità di utilizzo (impiegato)

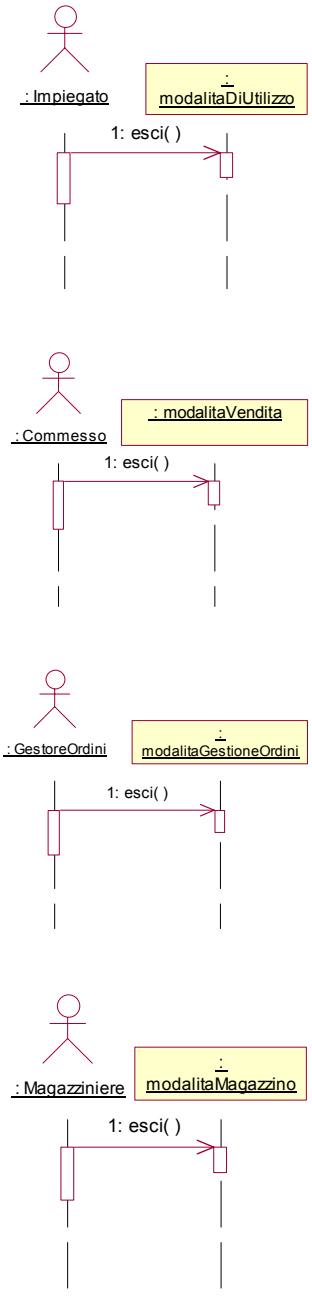


3.5.4.1.2 Modifica modalità di utilizzo

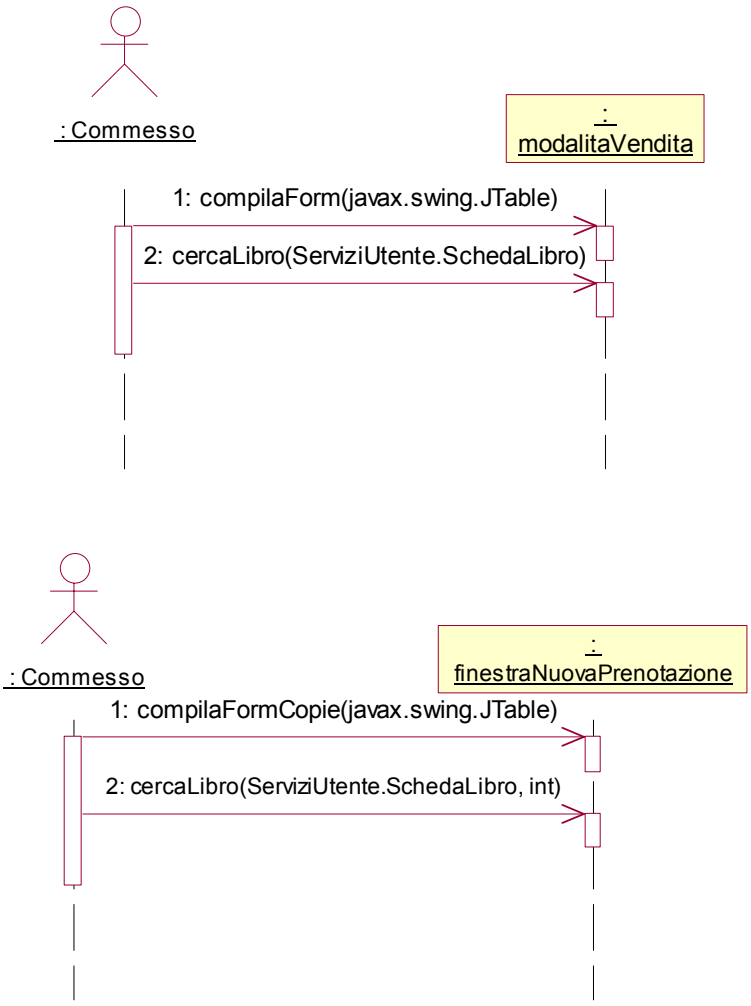


Essendo che il caso d'uso `modifica_modalita_di_utilizzo` include quello `seleziona_modalita_di_utilizzo`, i sequence diagram di questa sezione rappresentano il flusso di azioni da compiere per aprire la finestra modalità di utilizzo da dove si effettuerà una nuova scelta seguendo il sequence diagram **3.5.4.1.1**.

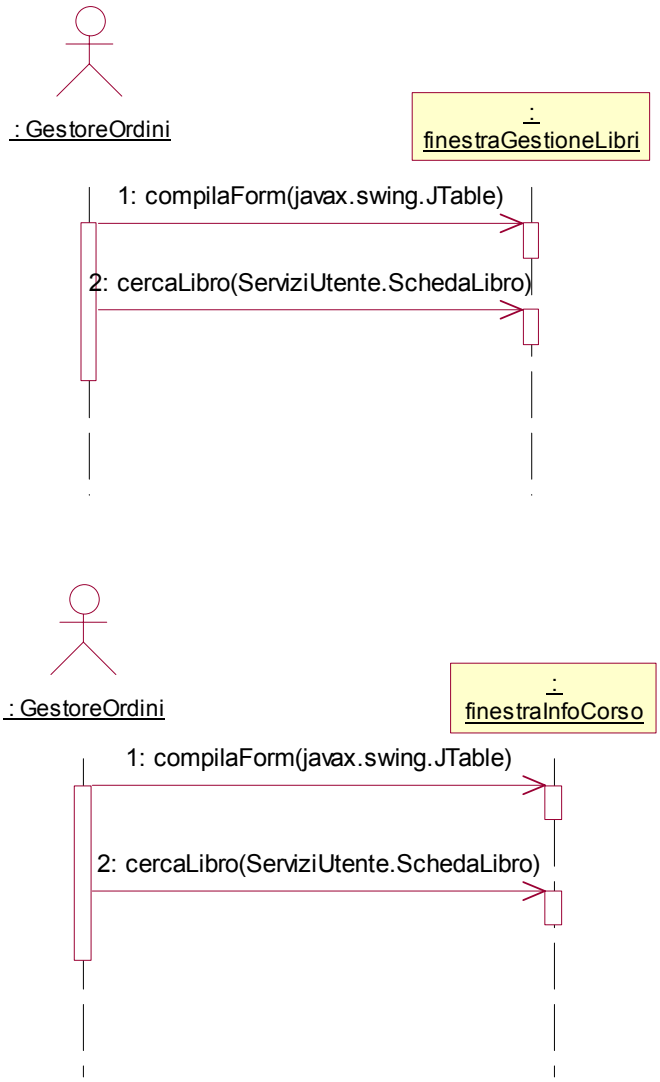
3.5.4.1.3 Esci dal sistema



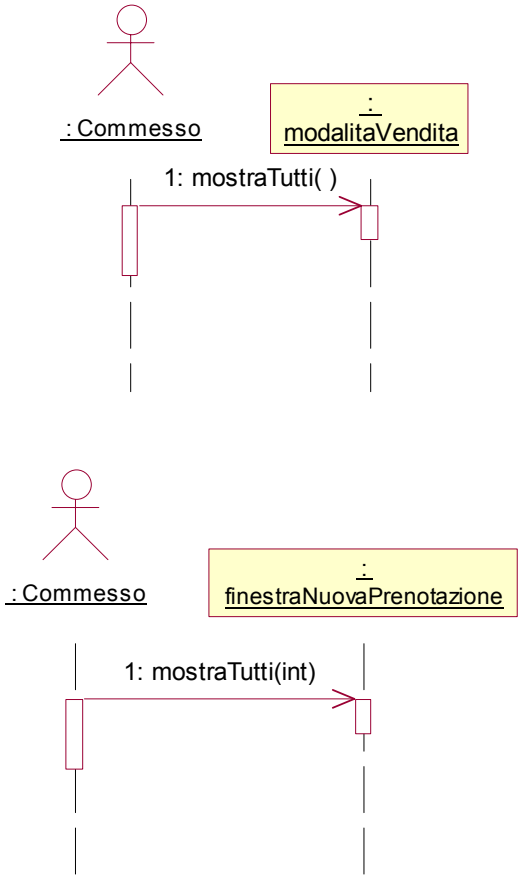
3.5.4.1.4 Cerca libro (commesso)



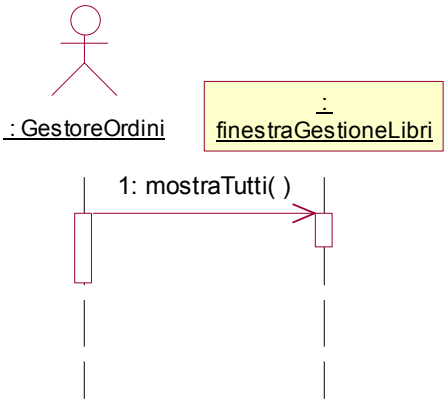
3.5.4.1.5 Cerca libro (gestore ordini)

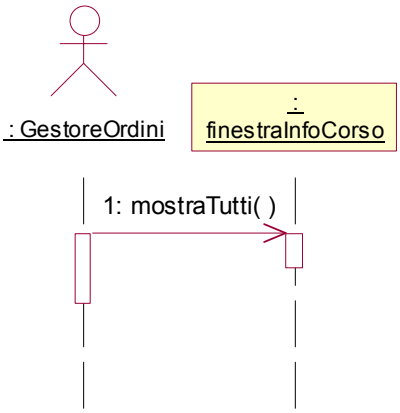


3.5.4.1.6 Mostra tutti i libri (commesso)

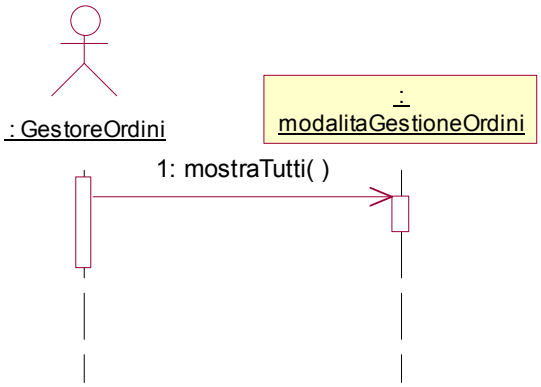


3.5.4.1.7 Mostra tutti i libri (gestore ordini)

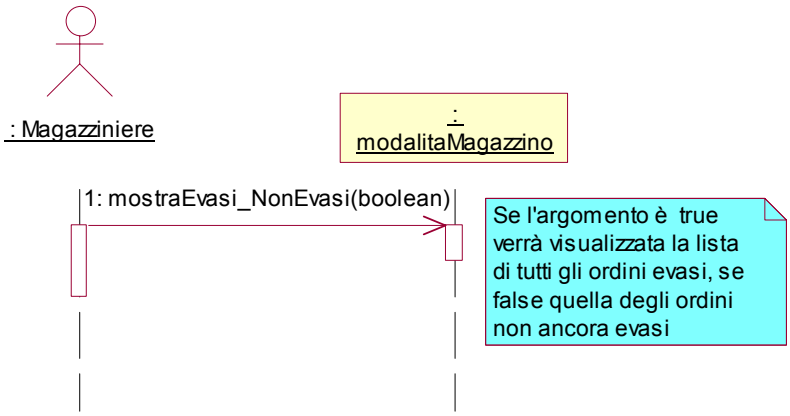




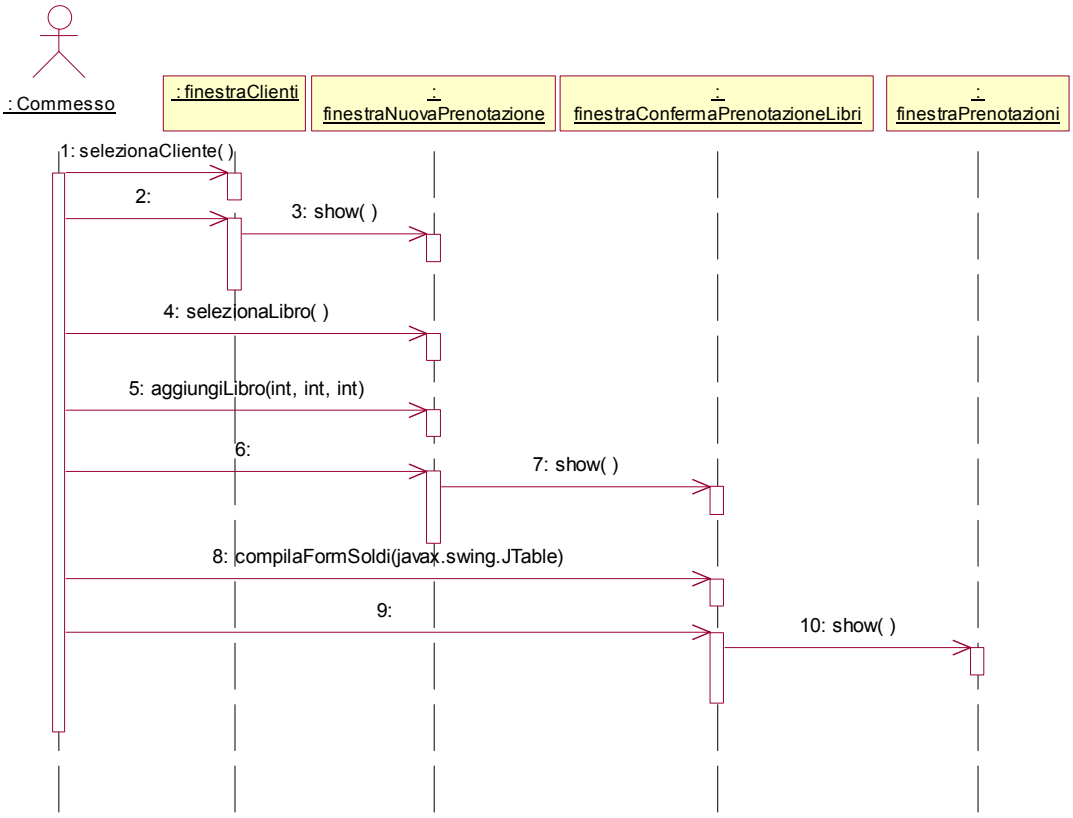
3.5.4.1.8 Mostra tutti gli ordini (gestore ordini)



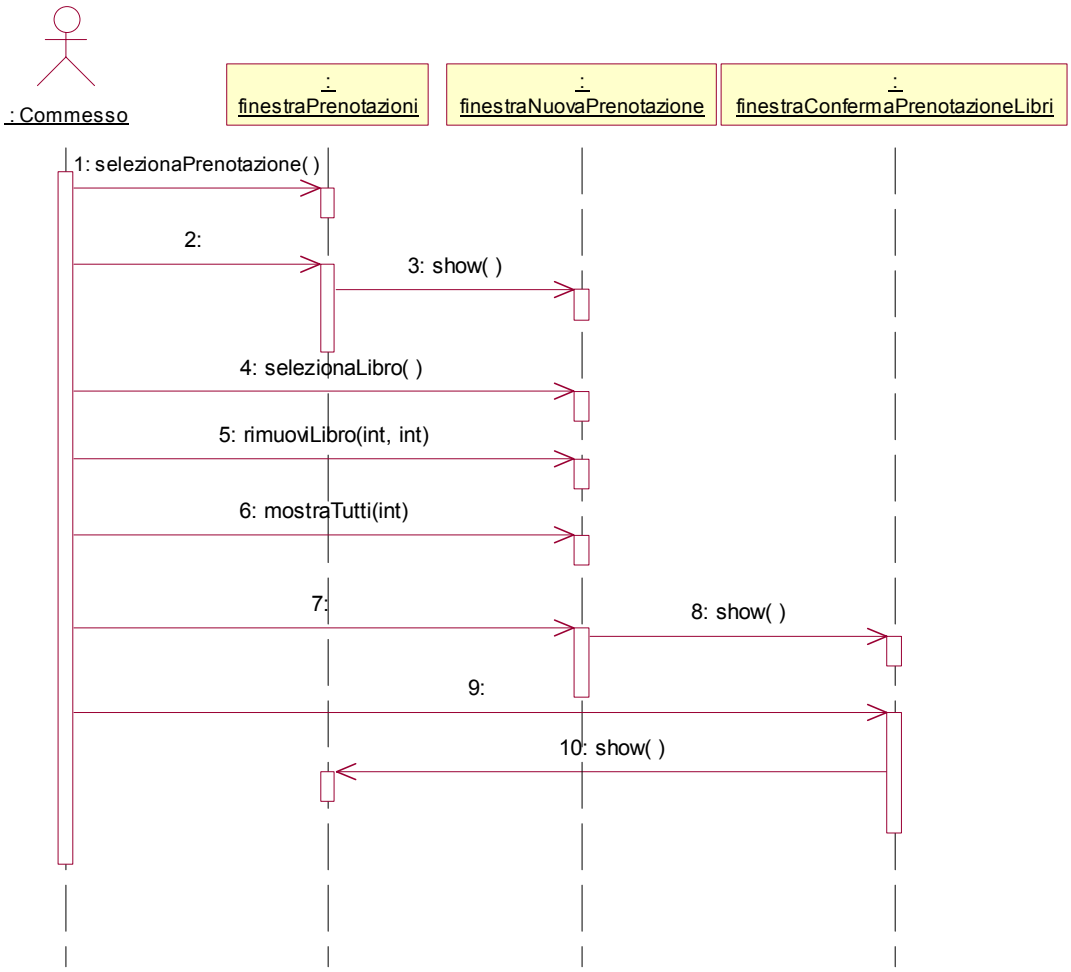
3.5.4.1.9 Mostra ordini Evasi/Non evasi (magazziniere)



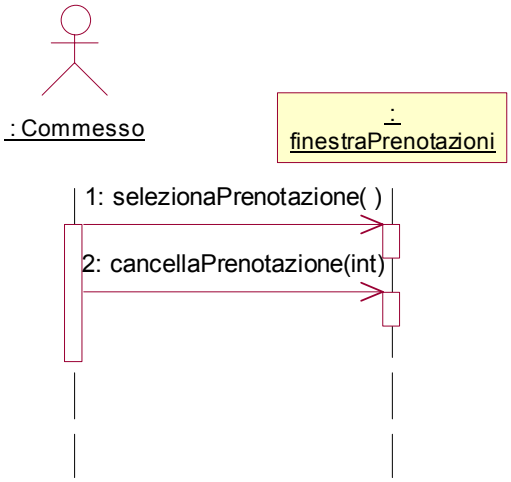
3.5.4.1.10 Nuova prenotazione (commesso)



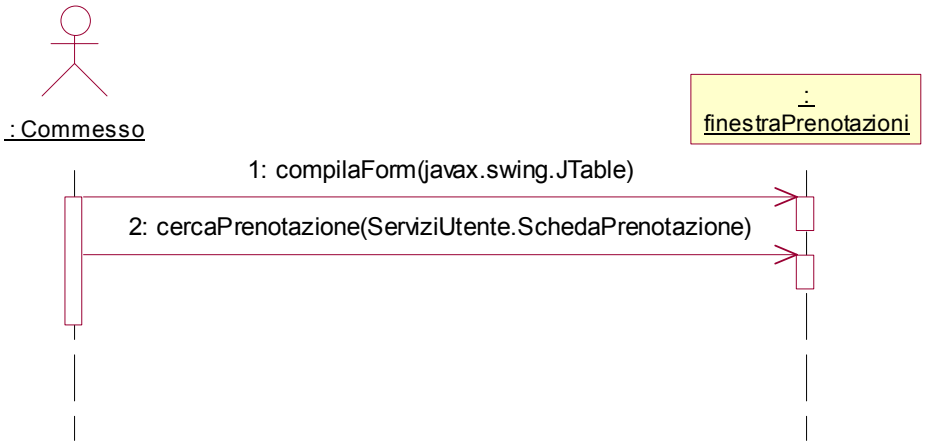
3.5.4.1.11 Modifica prenotazione (commesso)



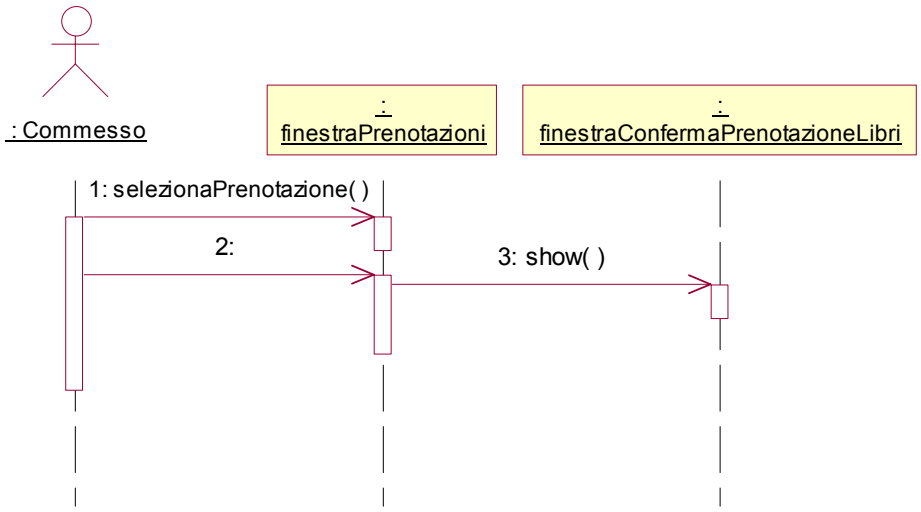
3.5.4.1.12 Cancella prenotazione (commesso)



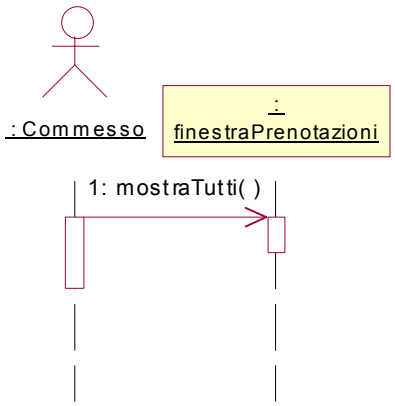
3.5.4.1.13 Cerca prenotazione (commesso)



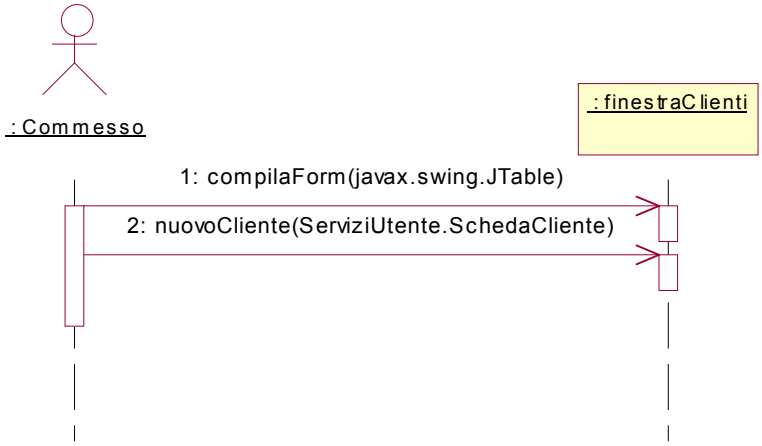
3.5.4.1.14 Visualizza prenotazione (commesso)



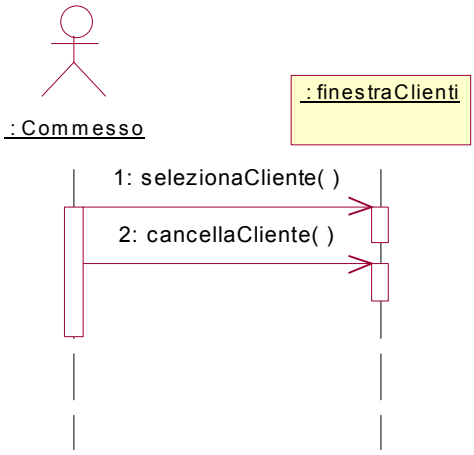
3.5.4.1.15 Mostra tutte le prenotazioni (commesso)



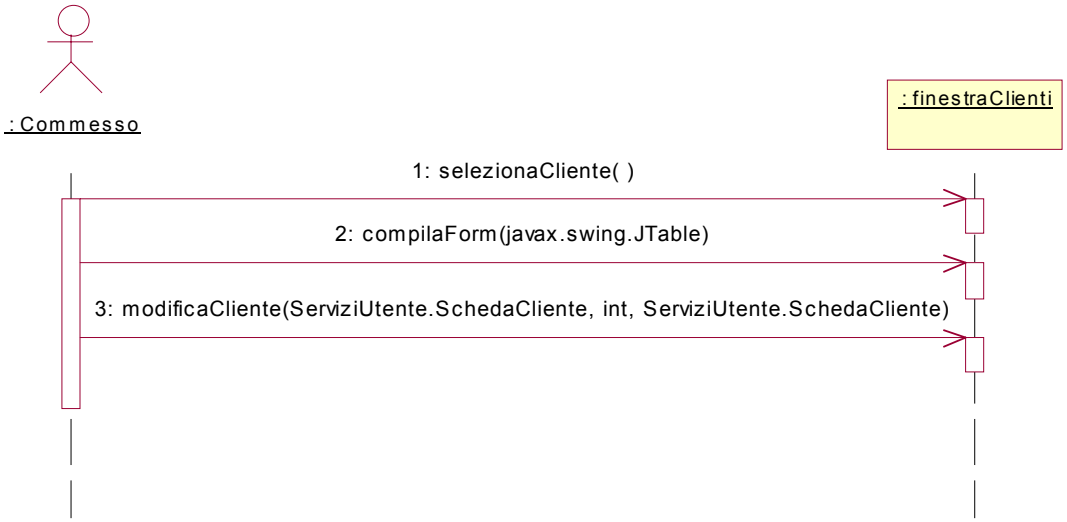
3.5.4.1.16 Nuovo cliente (commesso)



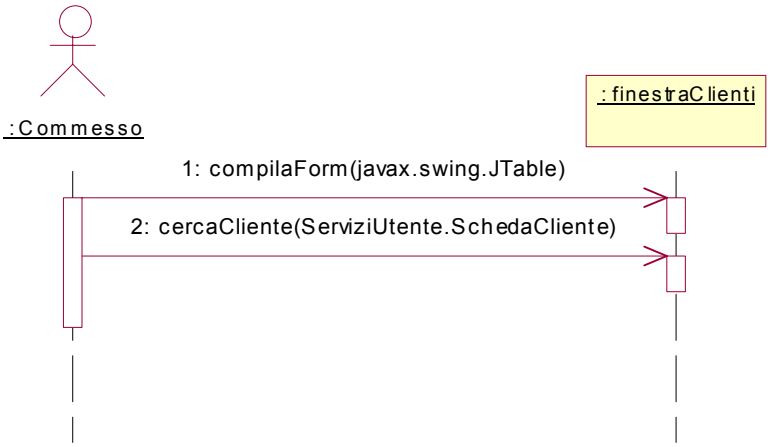
3.5.4.1.17 Cancella cliente (commesso)



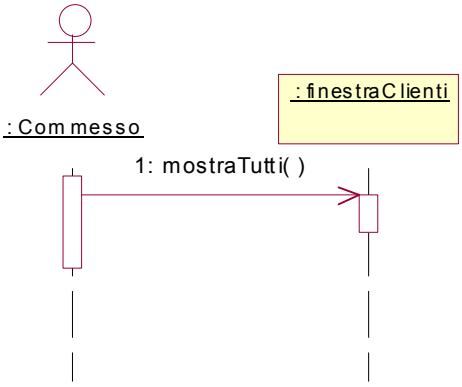
3.5.4.1.18 Modifica cliente (commesso)



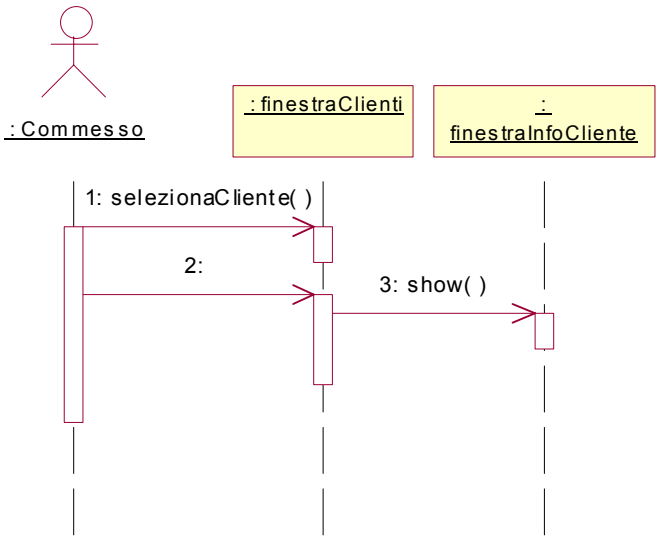
3.5.4.1.19 Cerca cliente (commesso)



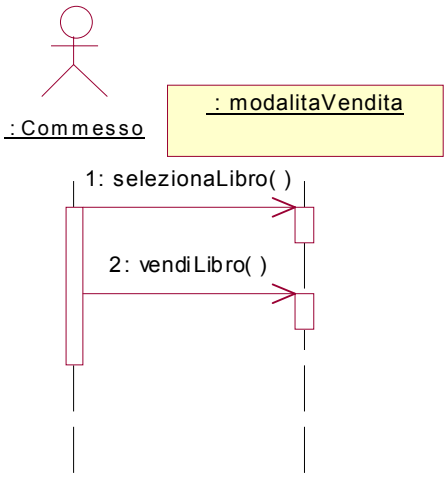
3.5.4.1.20 Mostra tutti i clienti (commesso)



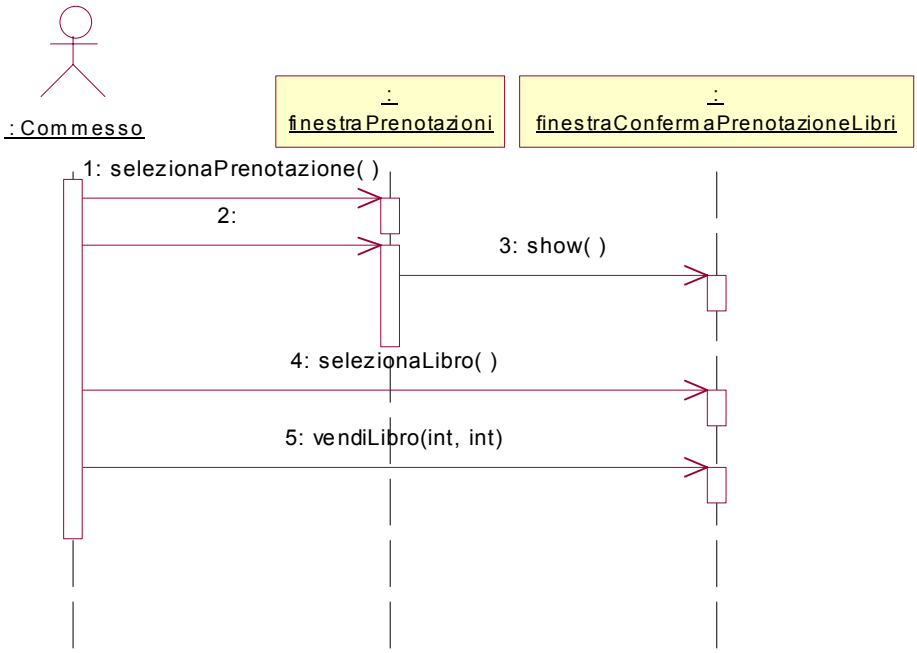
3.5.4.1.21 Info cliente (commesso)



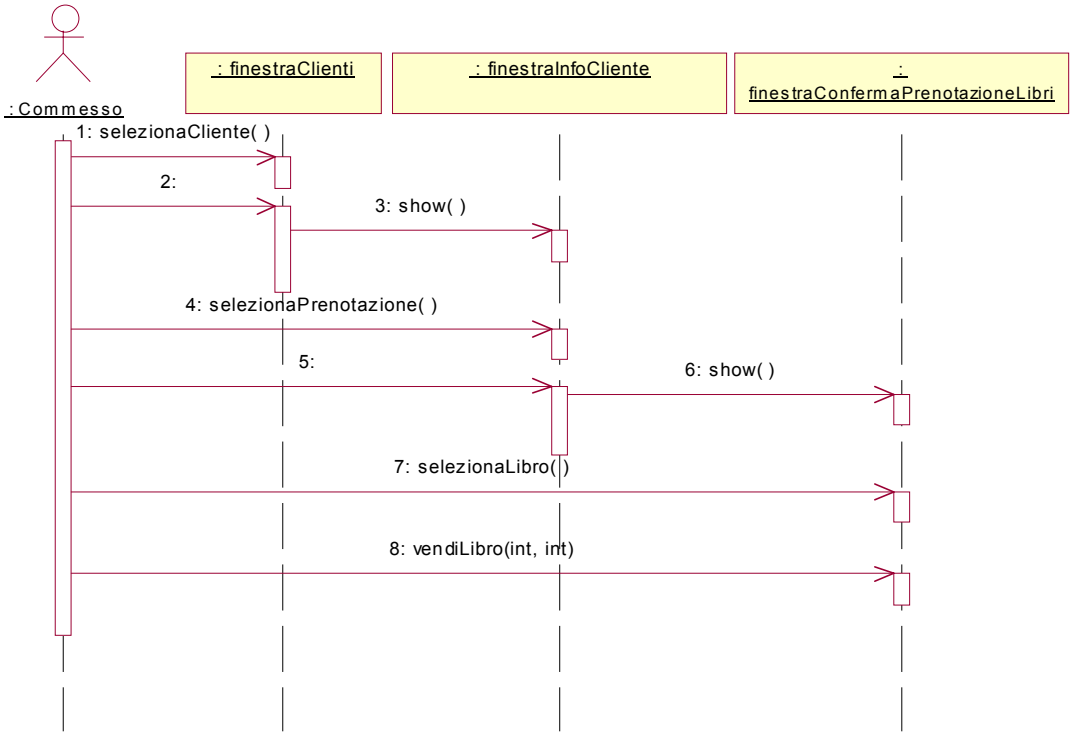
3.5.4.1.22 Vendita diretta (commesso)



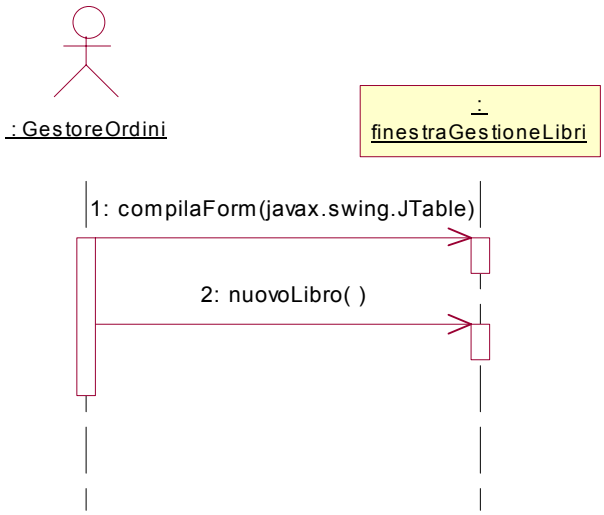
3.5.4.1.23 Vendita da prenotazione (commesso)



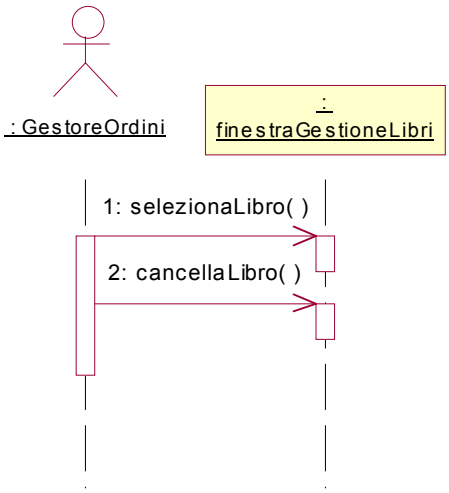
3.5.4.1.24 Vendita da cliente (commesso)



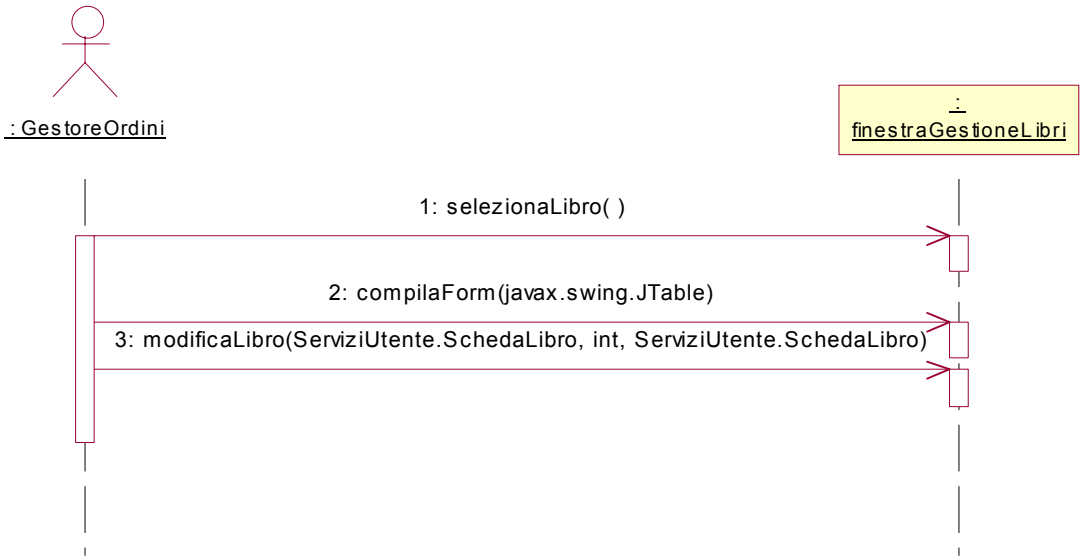
3.5.4.1.25 Nuovo libro (gestore ordini)



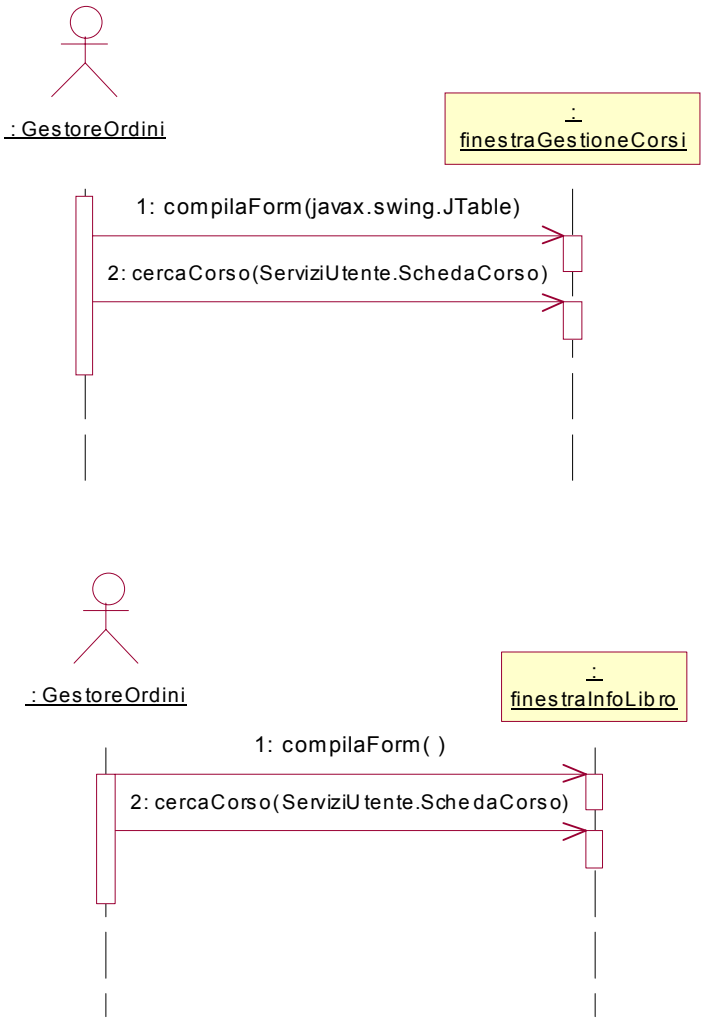
3.5.4.1.26 Cancella libro (gestore ordini)



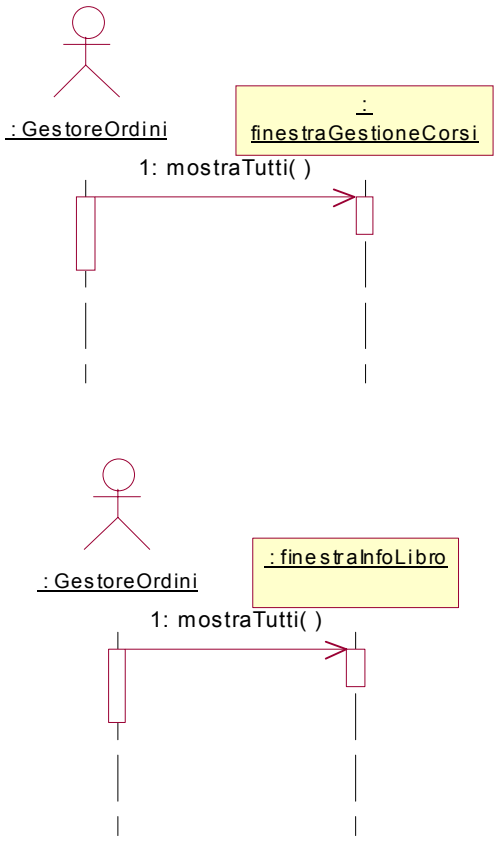
3.5.4.1.27 Modifica libro (gestore ordini)



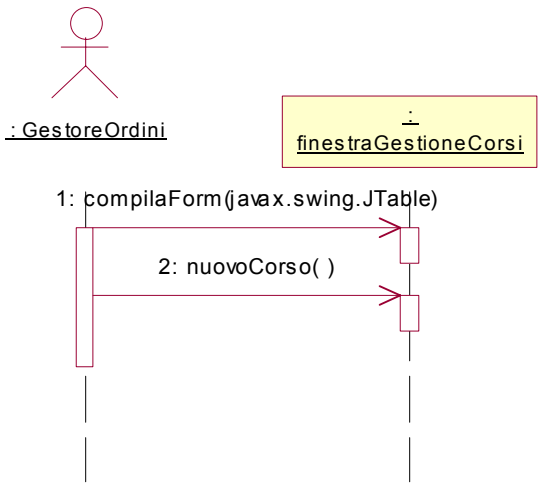
3.5.4.1.28 Cerca corso (gestore ordini)



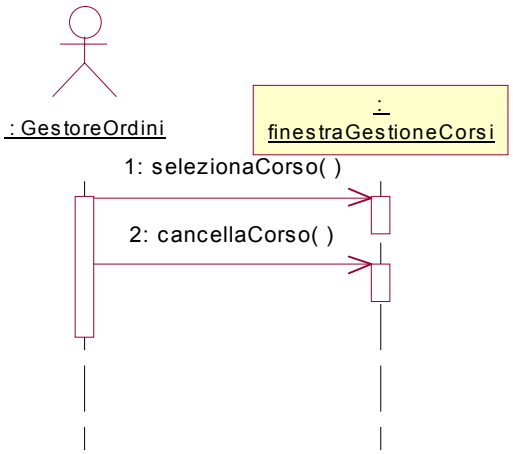
3.5.4.1.29 Mostra tutti i corsi (gestore ordini)



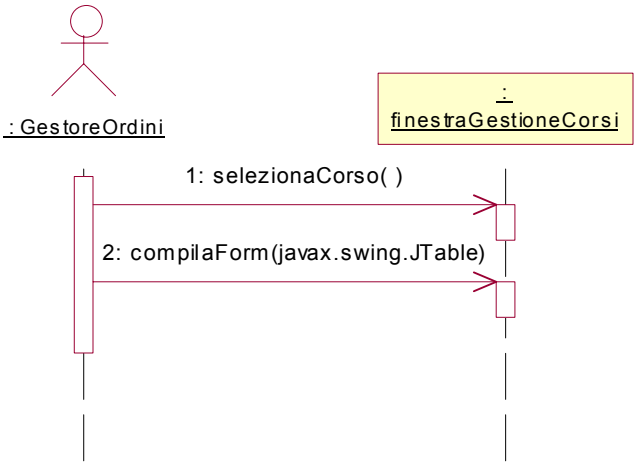
3.5.4.1.30 Nuovo corso (gestore ordini)



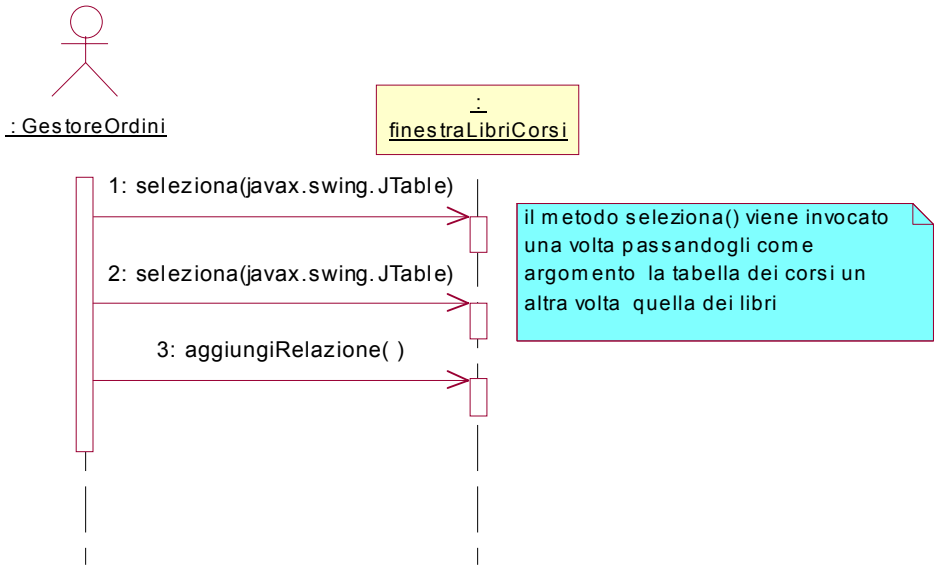
3.5.4.1.31 Cancella corso (gestore ordini)



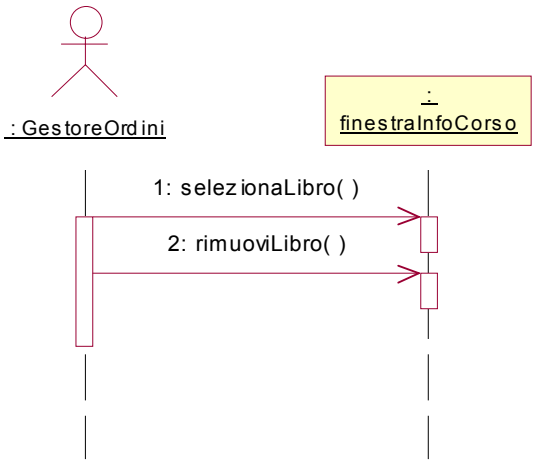
3.5.4.1.32 Modifica corso (gestore ordini)

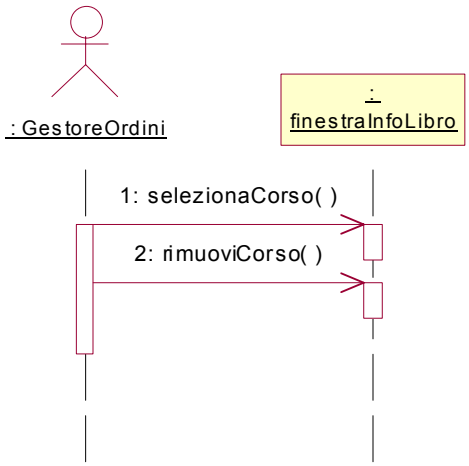


3.5.4.1.33 Aggiungi relazione libro corso (gestore ordini)



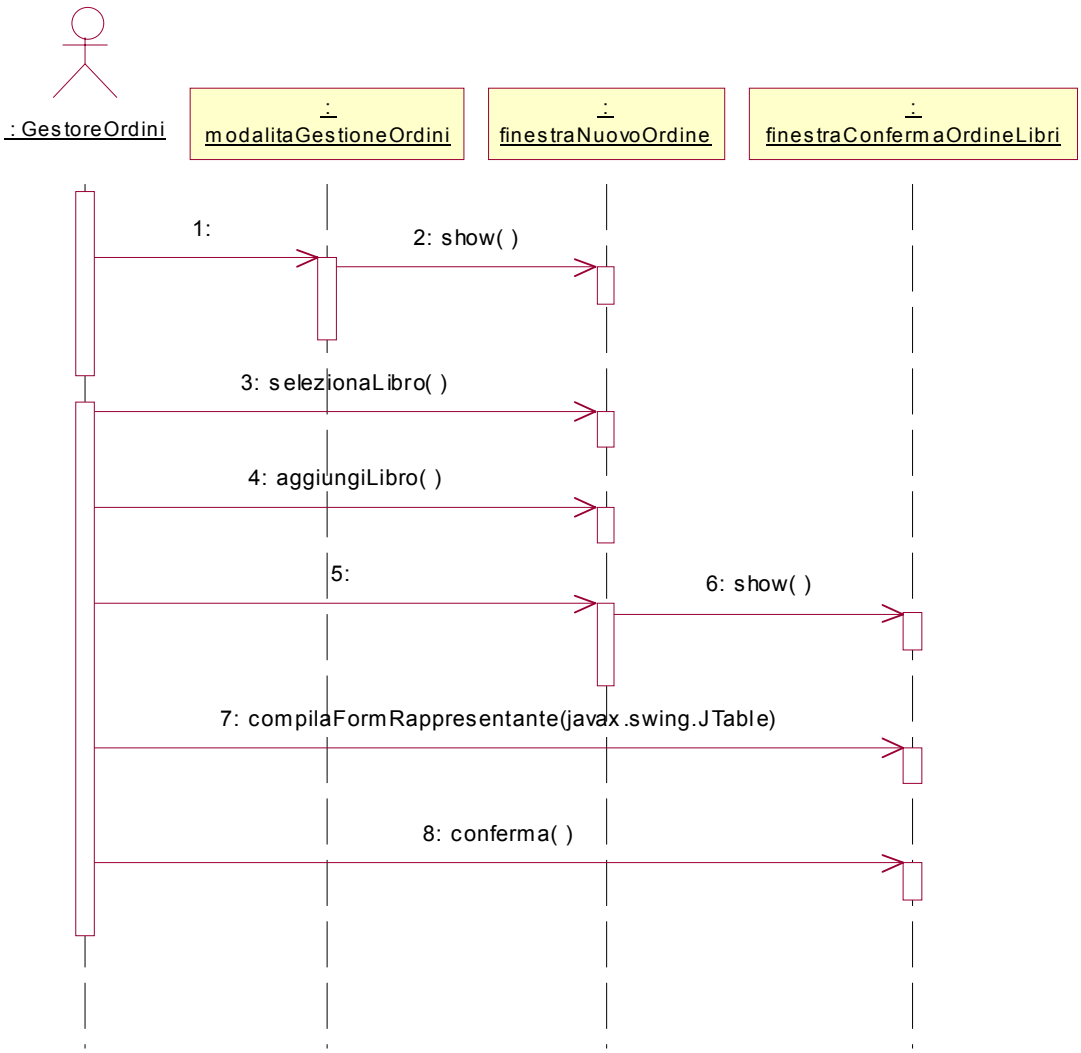
3.5.4.1.34 Rimuovi relazione libro corso (gestore ordini)



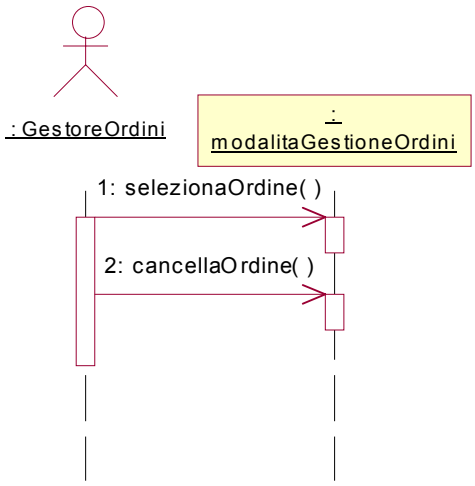


Questi sequence diagram descrivono le due diverse modalità per rimuovere una relazione che intercorre tra un libro e un corso che lo adotta. Nella prima si rimuove un libro associato a un corso mentre nella seconda un corso associato a un libro. Entrambe le operazioni rimuovono la stessa tupla dal database.

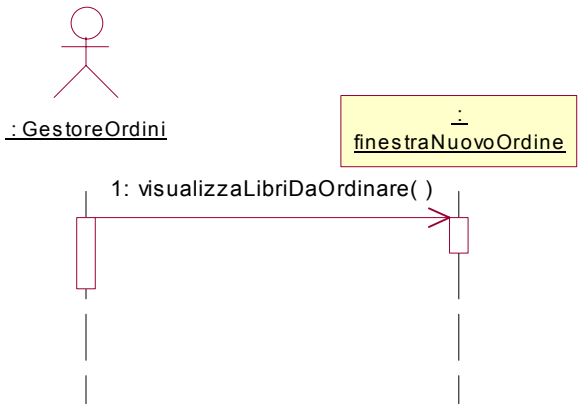
3.5.4.1.35 Nuovo ordine (gestore ordini)



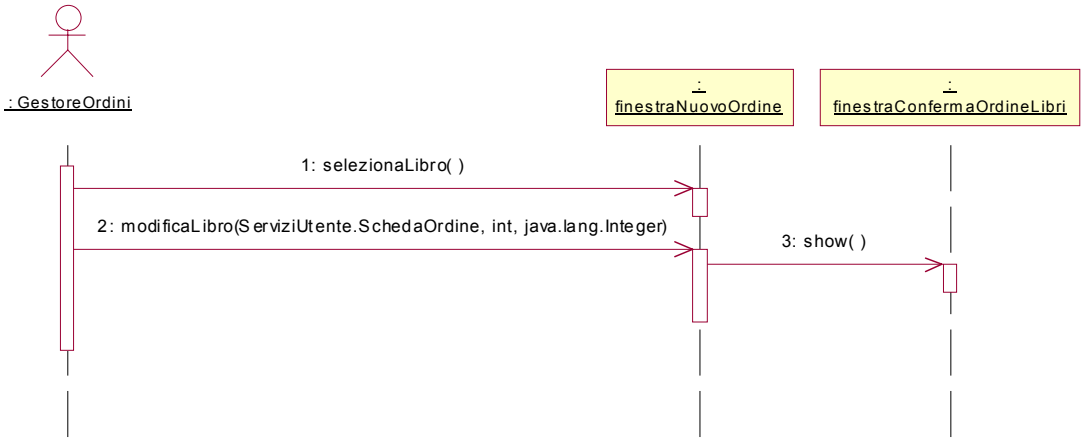
3.5.4.1.36 Cancella ordine (gestore ordini)



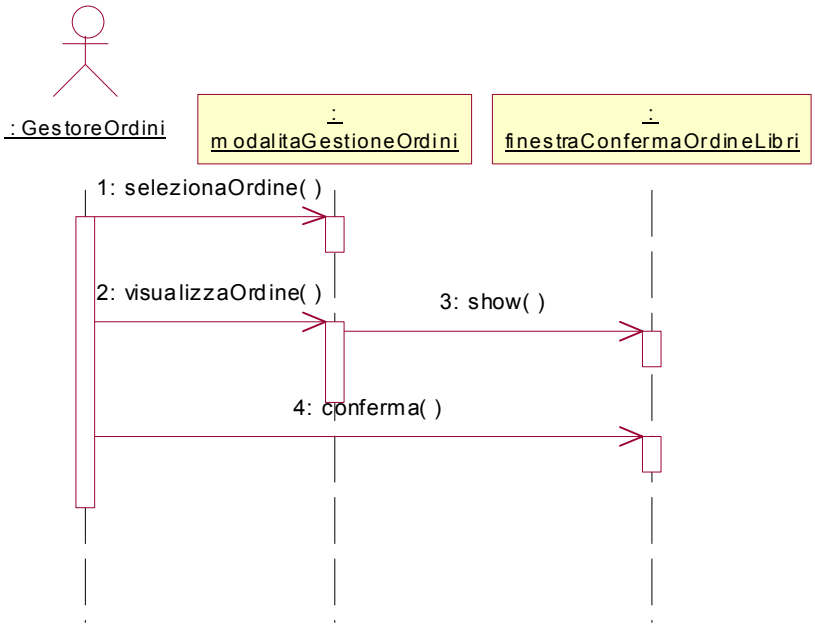
3.5.4.1.37 Visualizza libri da ordinare (gestore ordini)



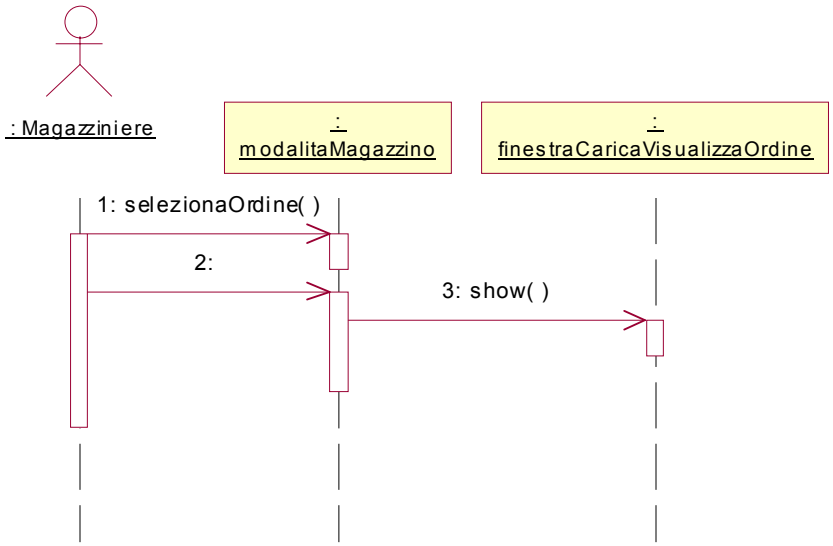
3.5.4.1.38 Modifica ordine (gestore ordini)



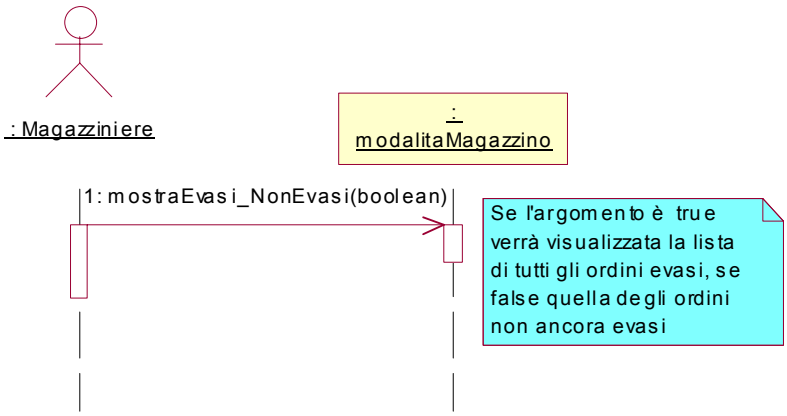
3.5.4.1.39 Visualizza ordine (gestore ordini)



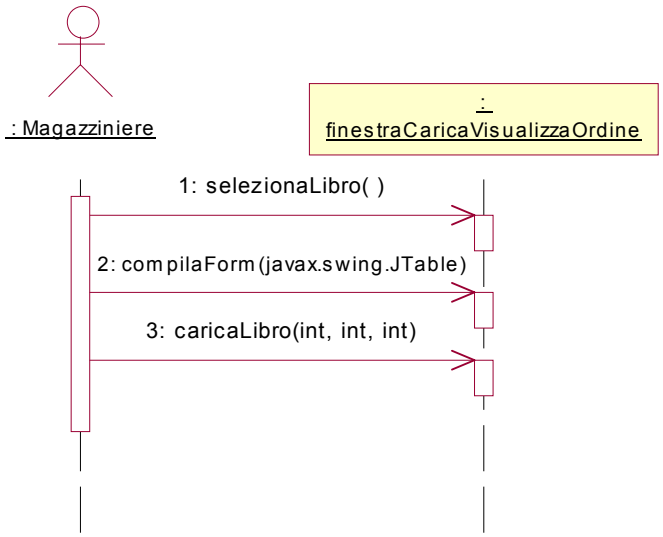
3.5.4.1.40 Visualizza ordine (magazziniere)



3.5.4.1.41 Ordini evasi/non evasi(magazziniere)

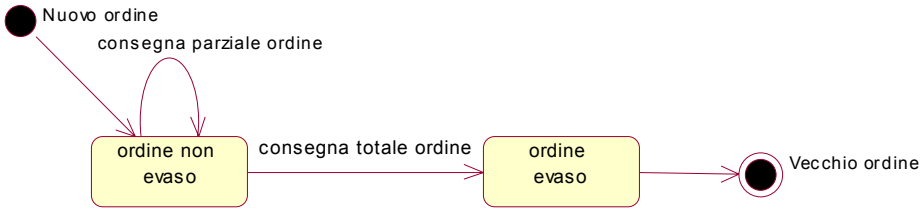


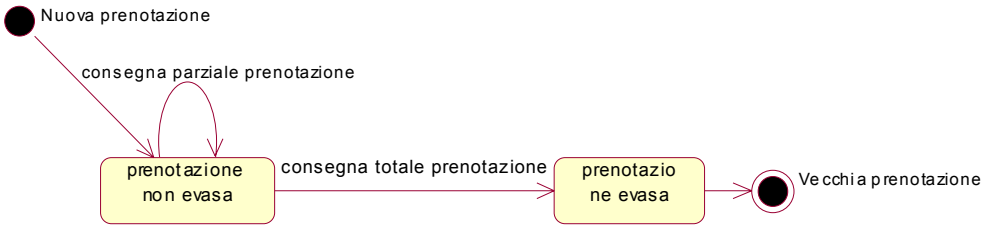
3.5.4.1.42 Carica Ordine (magazziniere)



3.5.4.2 Diagrammi di statechart

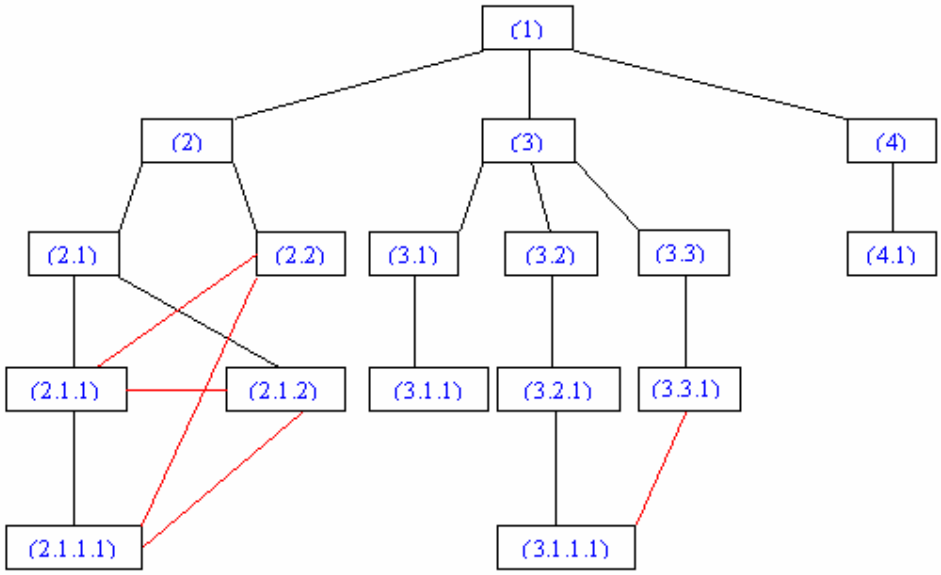
Questi diagrammi illustrano come avviene la transizione di stato delle classi SchedaPrenotazione e SchedaOrdine utilizzando per entrambi l'attributo evaso che può assumere i valori si e no.



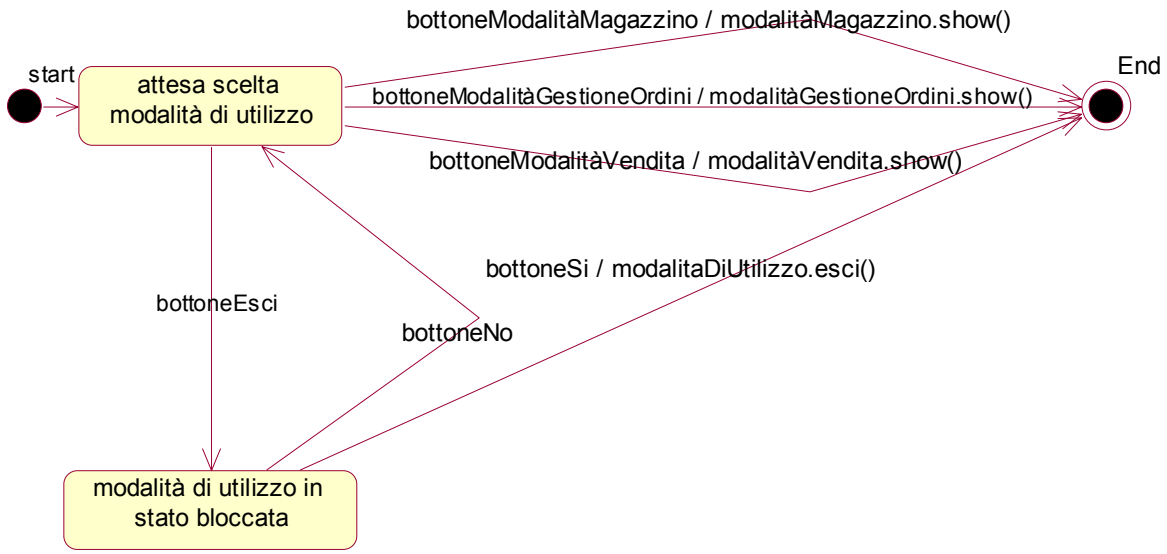


3.5.5 Interfaccia utente - navigational paths and screen mock-ups

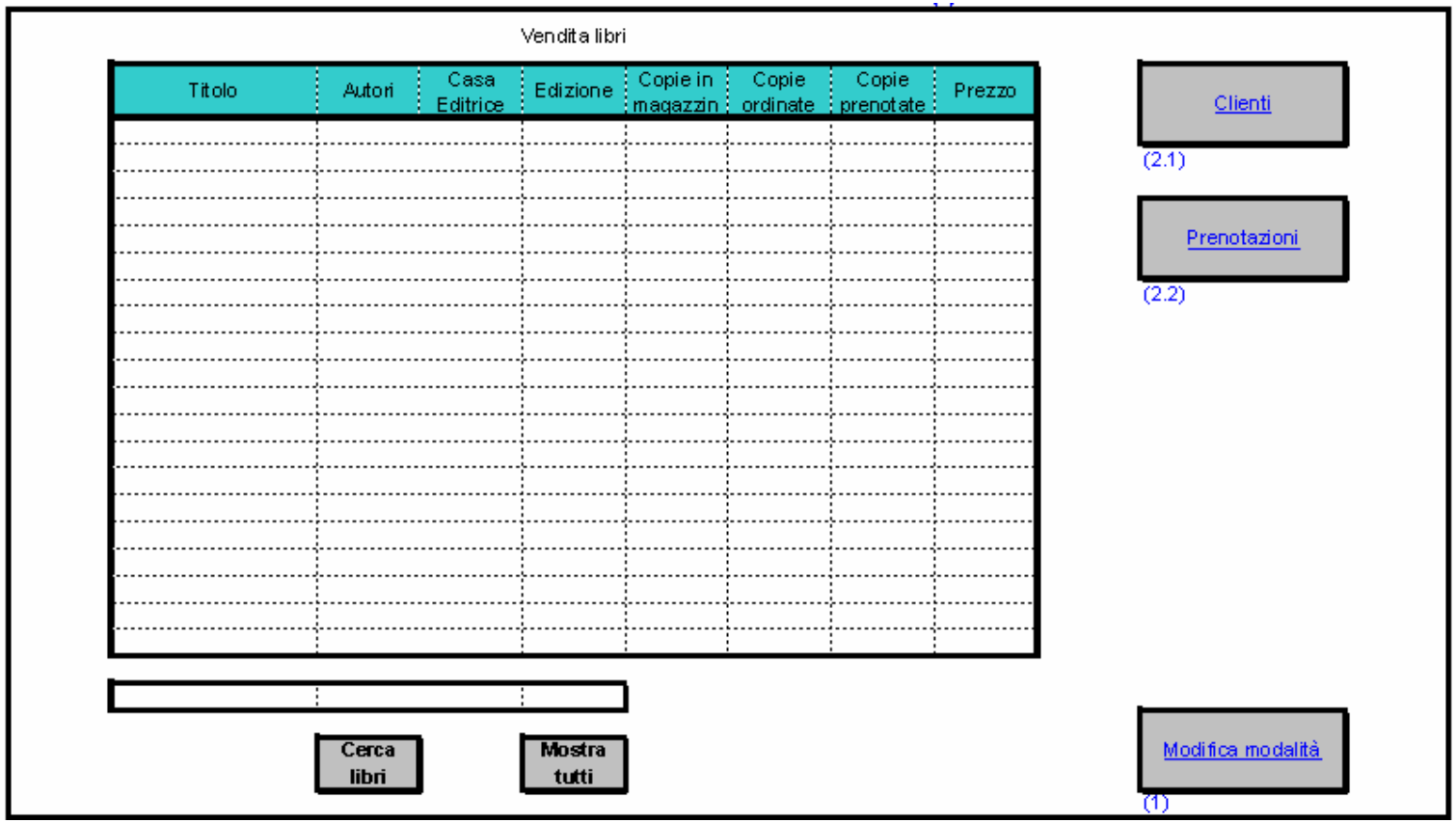
Per la descrizione dell'interfaccia utente verranno utilizzati i diagrammi di attività che illustreranno le transizioni che avvengono nel sistema in seguito a sollecitazioni esterne. Si procederà mostrando prima uno screen mock-up e poi il relativo diagramma delle attività. Per semplificare la comprensione delle transizioni verrà associato un numero ad ogni finestra (esempio (2.3) (4.5) (3.1)) e questo sarà riportato vicino ad ogni singolo bottone che attiverà la finestra. Qui di seguito viene riportato la struttura ad albero secondo la quale si attivano le finestre. La linea nera rappresenta un legame di tipo diretto fra una finestra padre e quella figlio, mentre quella rossa rappresenta chiamate fra finestre non imparentate ma utili per un dinamico utilizzo del software.

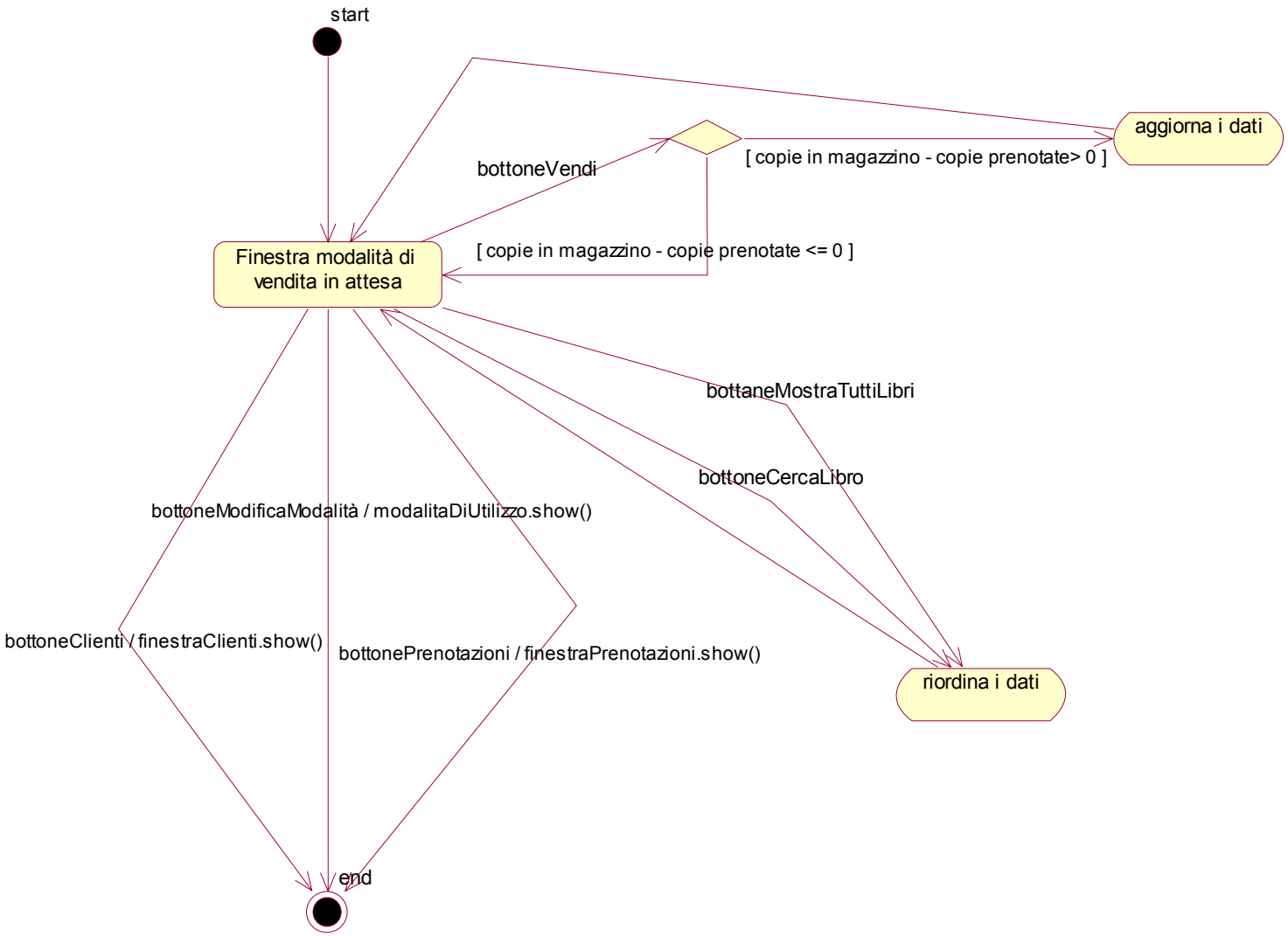


3.5.5.1 Modalità di utilizzo (1)

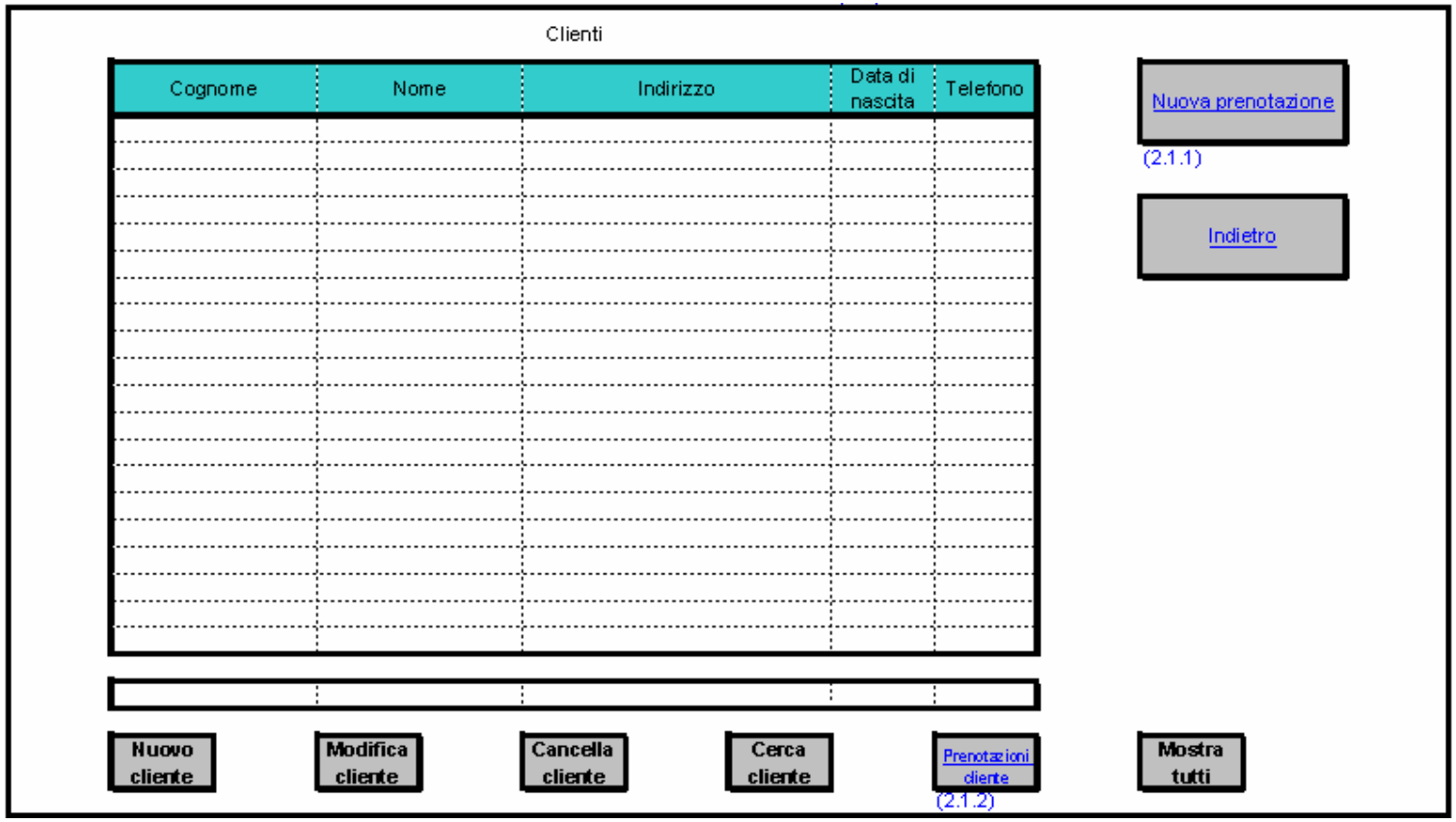

start
attesa scelta
modalità di utilizzo
bottoneModalitàMagazzino / modalitàMagazzino.show()
bottoneModalitàGestioneOrdini / modalitàGestioneOrdini.show()
bottoneModalitàVendita / modalitàVendita.show()
End
bottoneEsci
bottoneSi / modalitaDiUtilizzo.esci()
bottoneNo
modalità di utilizzo in
stato bloccata

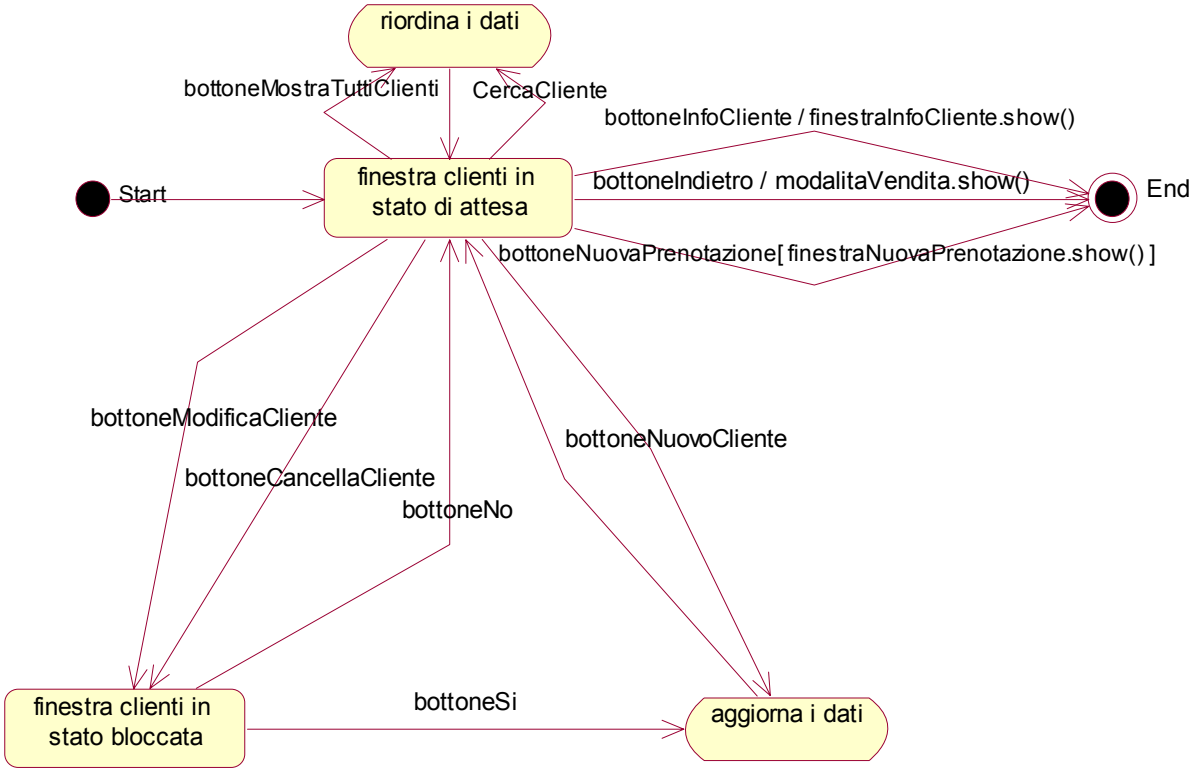
3.5.5.2 Finestra modalità vendita libri (2)



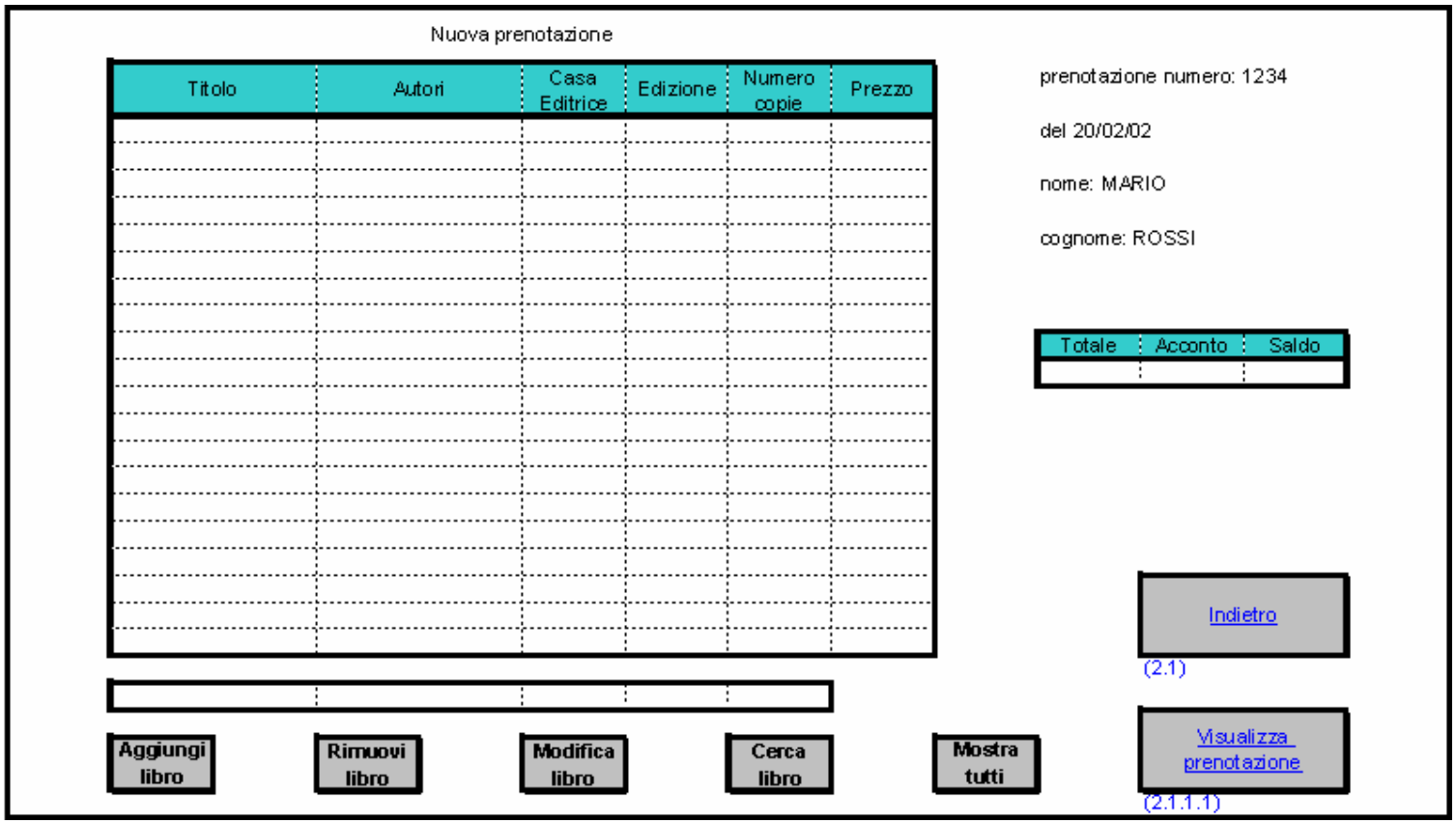

start
aggiorna i dati
bottoneVendi
[copie in magazzino - copie prenotate> 0]
Finestra modalità di vendita in attesa
[copie in magazzino - copie prenotate <= 0]
bottaneMostraTuttiLibri
bottoneCercaLibro
bottoneModificaModalità / modalitaDiUtilizzo.show()
bottoneClienti / finestraClienti.show()
bottonePrenotazioni / finestraPrenotazioni.show()
riordina i dati
end

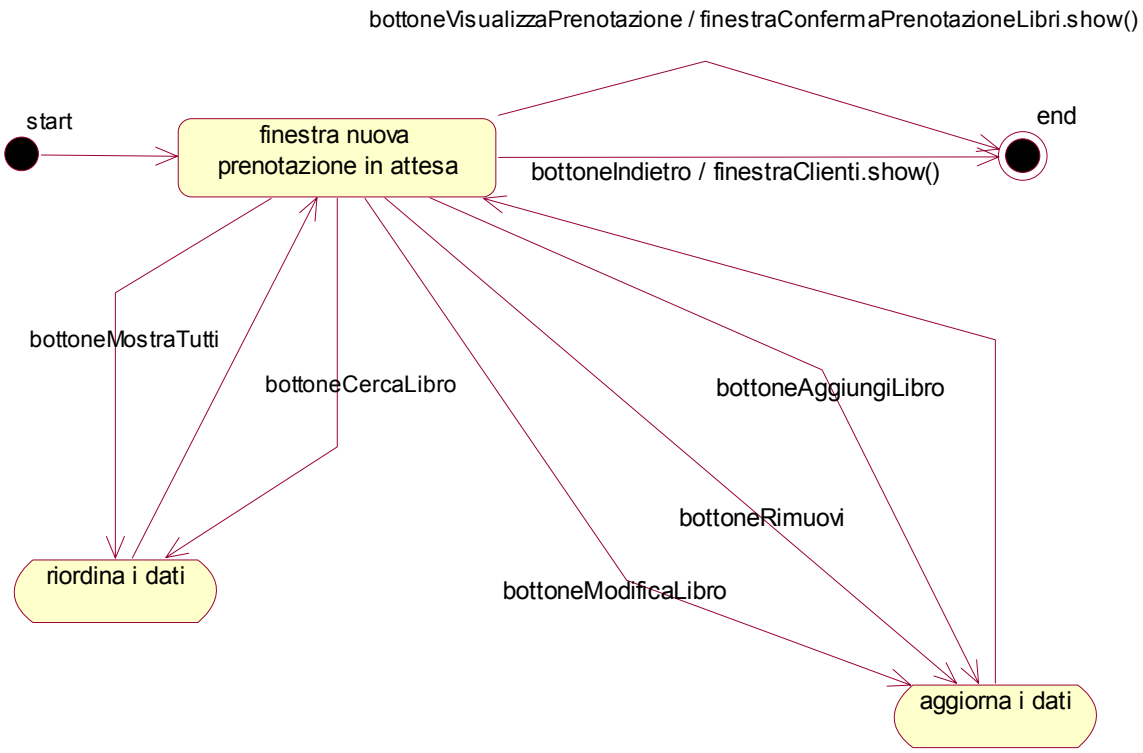
3.5.5.3 Finestra gestione clienti (2.1)



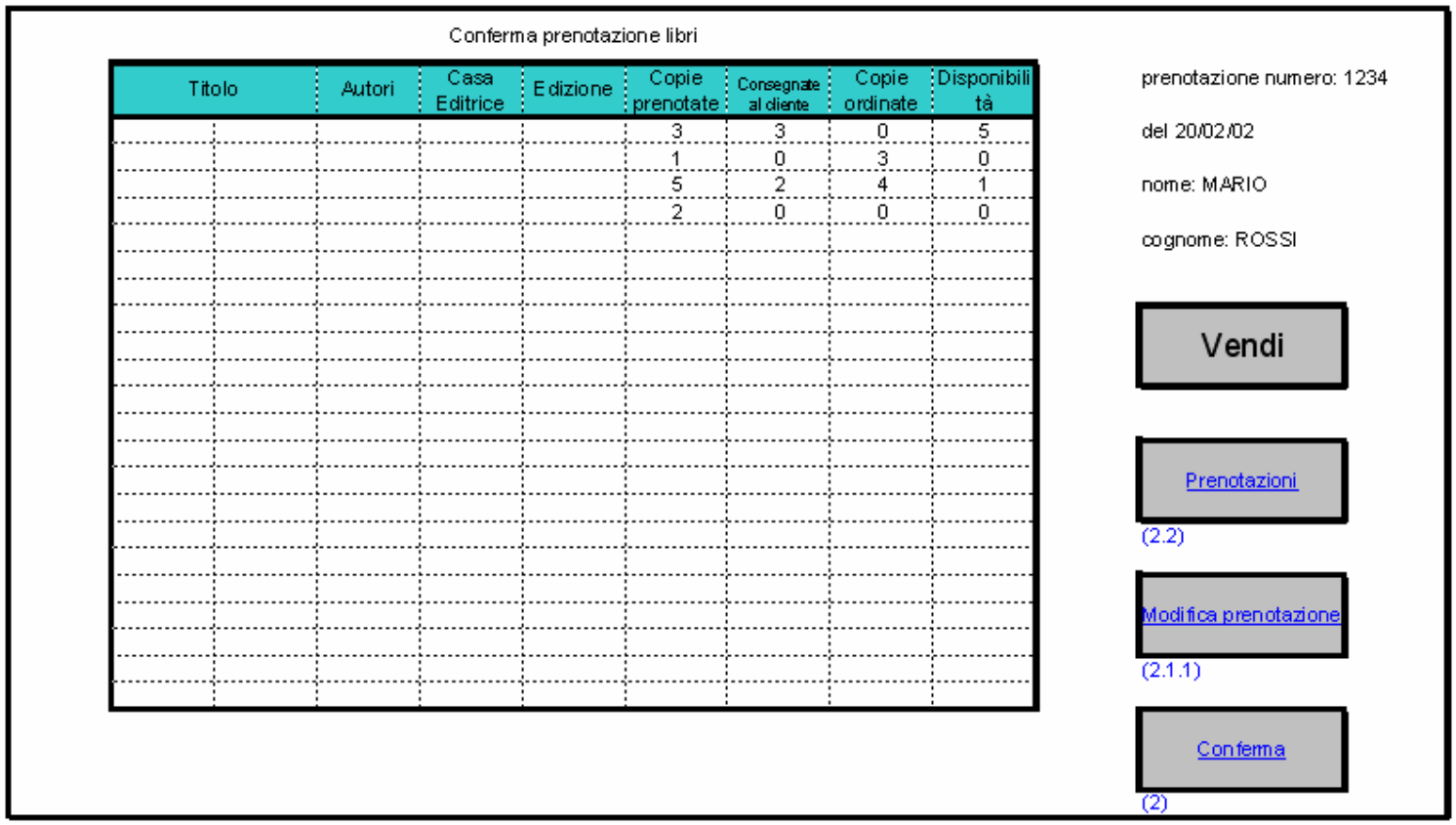

riordina i dati
bottoneMostraTuttiClienti
CercaCliente
bottoneInfoCliente / finestraInfoCliente.show()
Start
finestra clienti in stato di attesa
bottoneIndietro / modalitaVendita.show()
End
bottoneNuovaPrenotazione[finestraNuovaPrenotazione.show()]
bottoneModificaCliente
bottoneNuovoCliente
bottoneCancellaCliente
bottoneNo
finestra clienti in stato bloccata
bottoneSi
aggiorna i dati

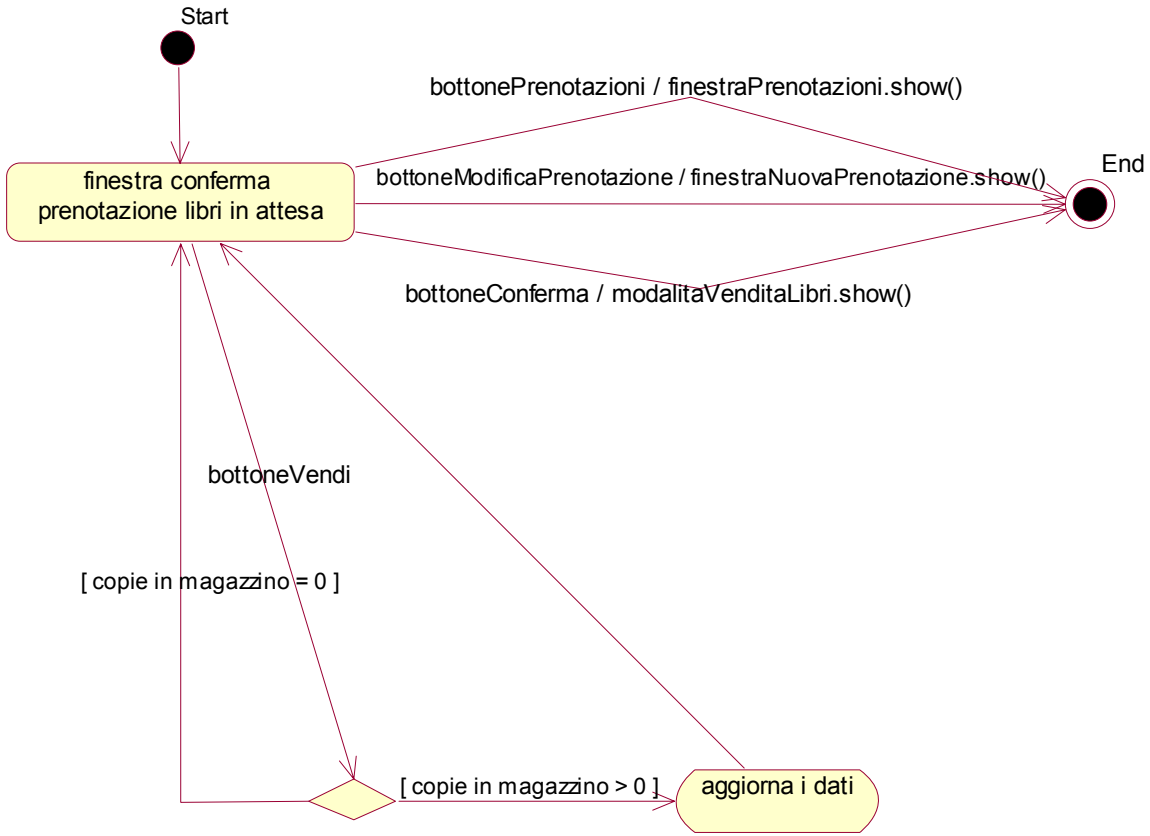
3.5.5.4 Finestra nuova prenotazione (2.1.1)



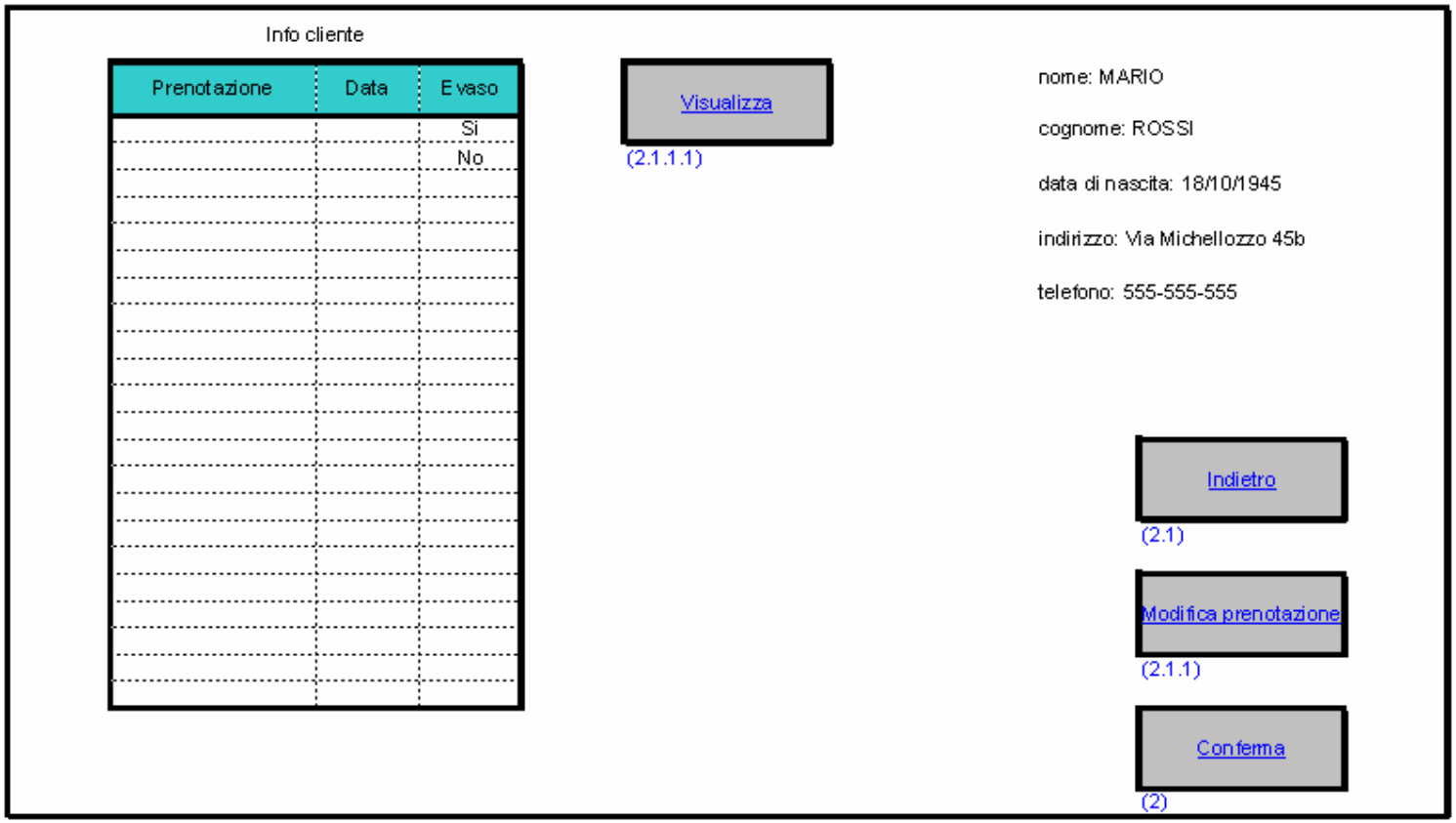

bottoneVisualizzaPrenotazione / finestraConfermaPrenotazioneLibri.show()
start
finestra nuova
prenotazione in attesa
end
bottoneIndietro / finestraClienti.show()
bottoneMostraTutti
bottoneCercaLibro
bottoneAggiungiLibro
bottoneRimuovi
bottoneModificaLibro
riordina i dati
aggiorna i dati

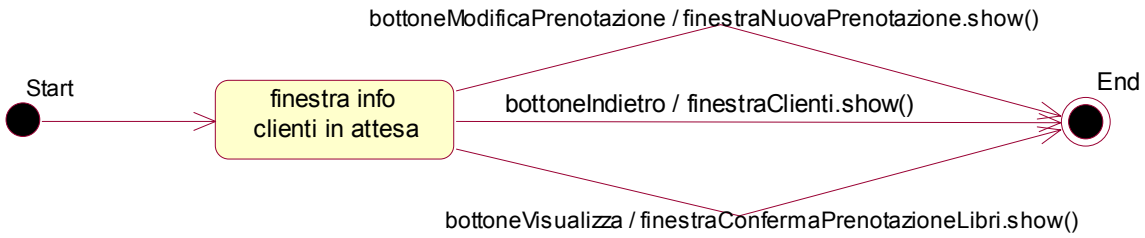
3.5.5.5 Finestra conferma prenotazione libri (2.1.1.1)



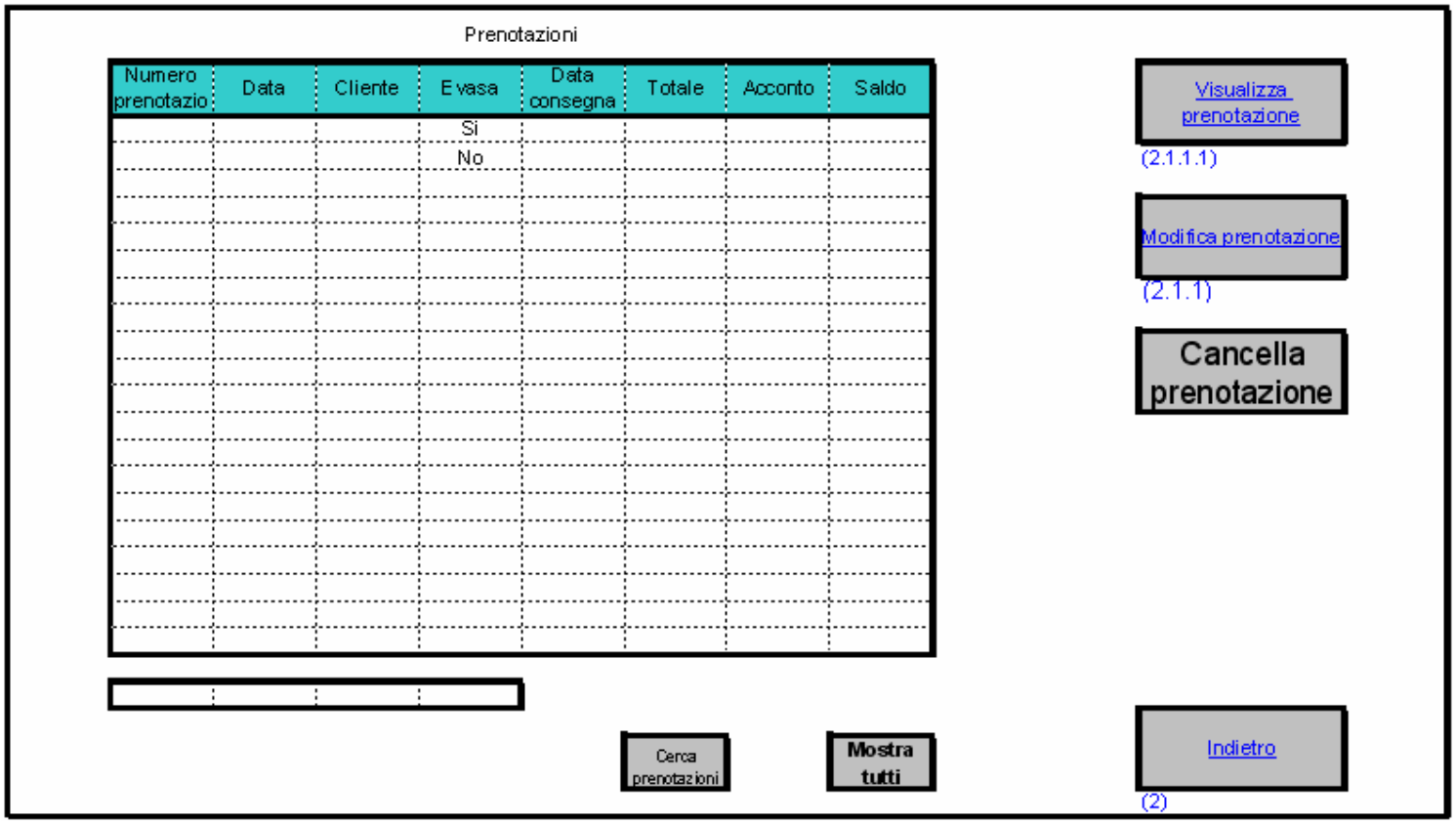

Start
bottonePrenotazioni / finestraPrenotazioni.show()
finestra conferma
prenotazione libri in attesa
bottoneModificaPrenotazione / finestraNuovaPrenotazione.show()
End
bottoneConferma / modalitaVenditaLibri.show()
bottoneVendi
[copie in magazzino = 0]
[copie in magazzino > 0]
aggiorna i dati

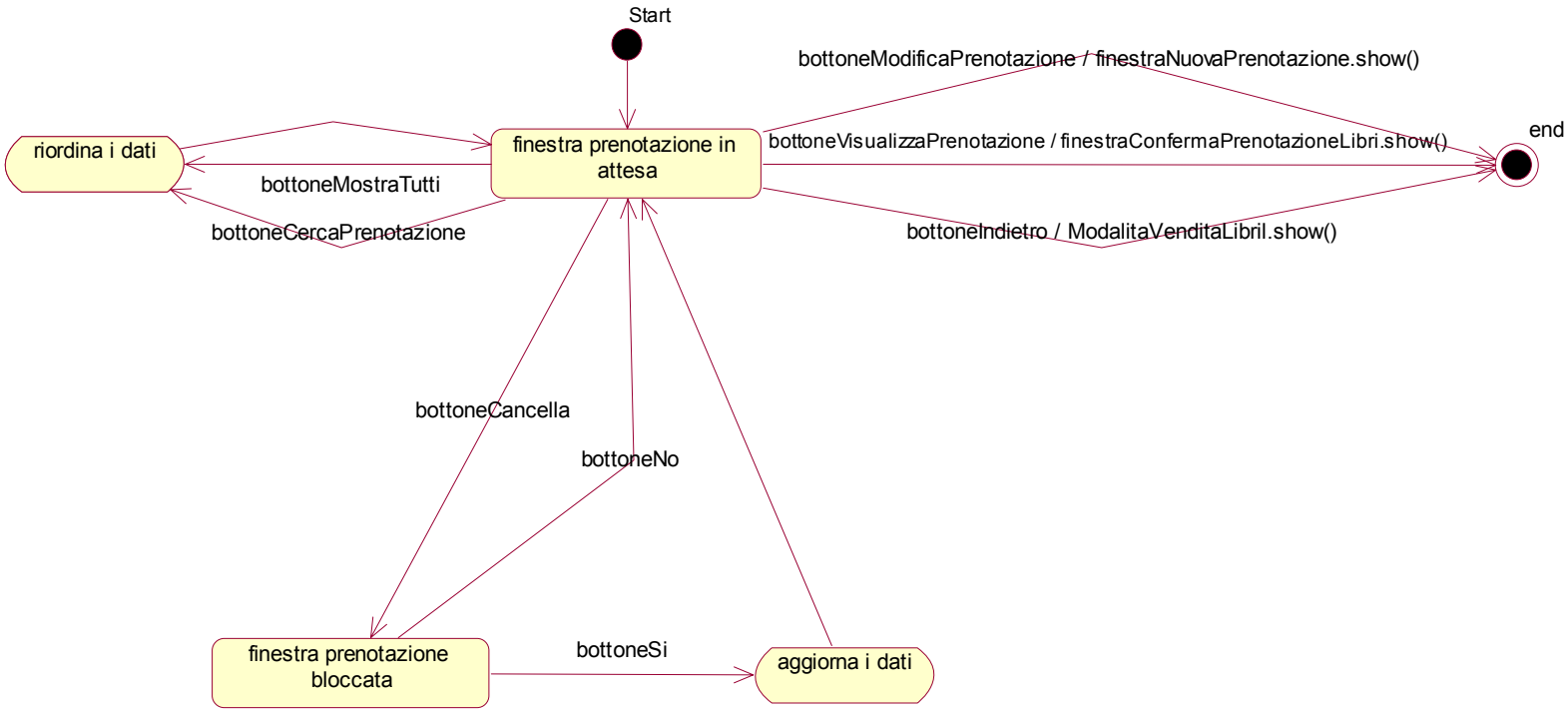
3.5.5.6 Finestra info cliente (2.1.2)



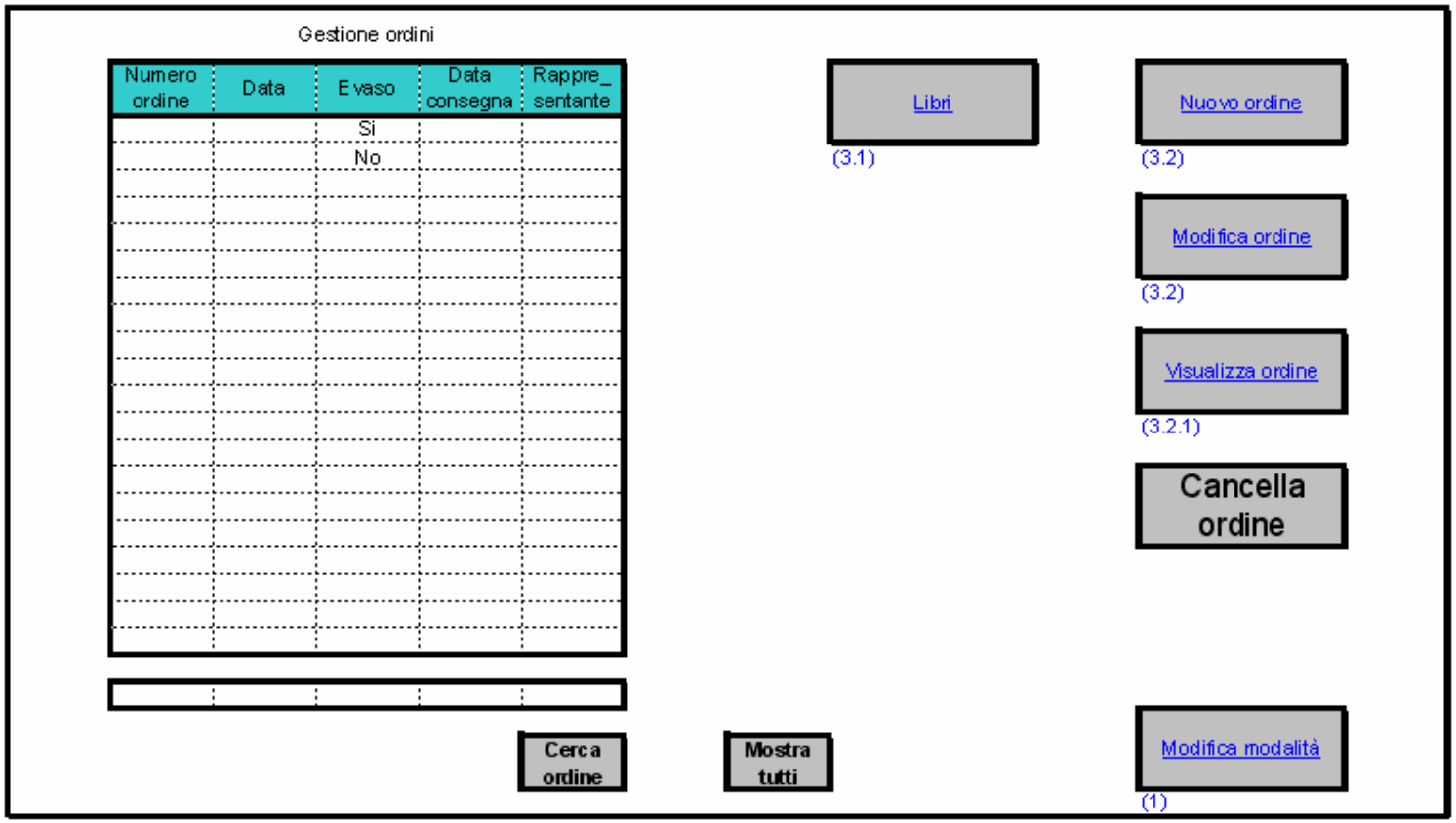

bottoneModificaPrenotazione / finestraNuovaPrenotazione.show()
Start
finestra info
clienti in attesa
bottoneIndietro / finestraClienti.show()
End
bottoneVisualizza / finestraConfermaPrenotazioneLibri.show()

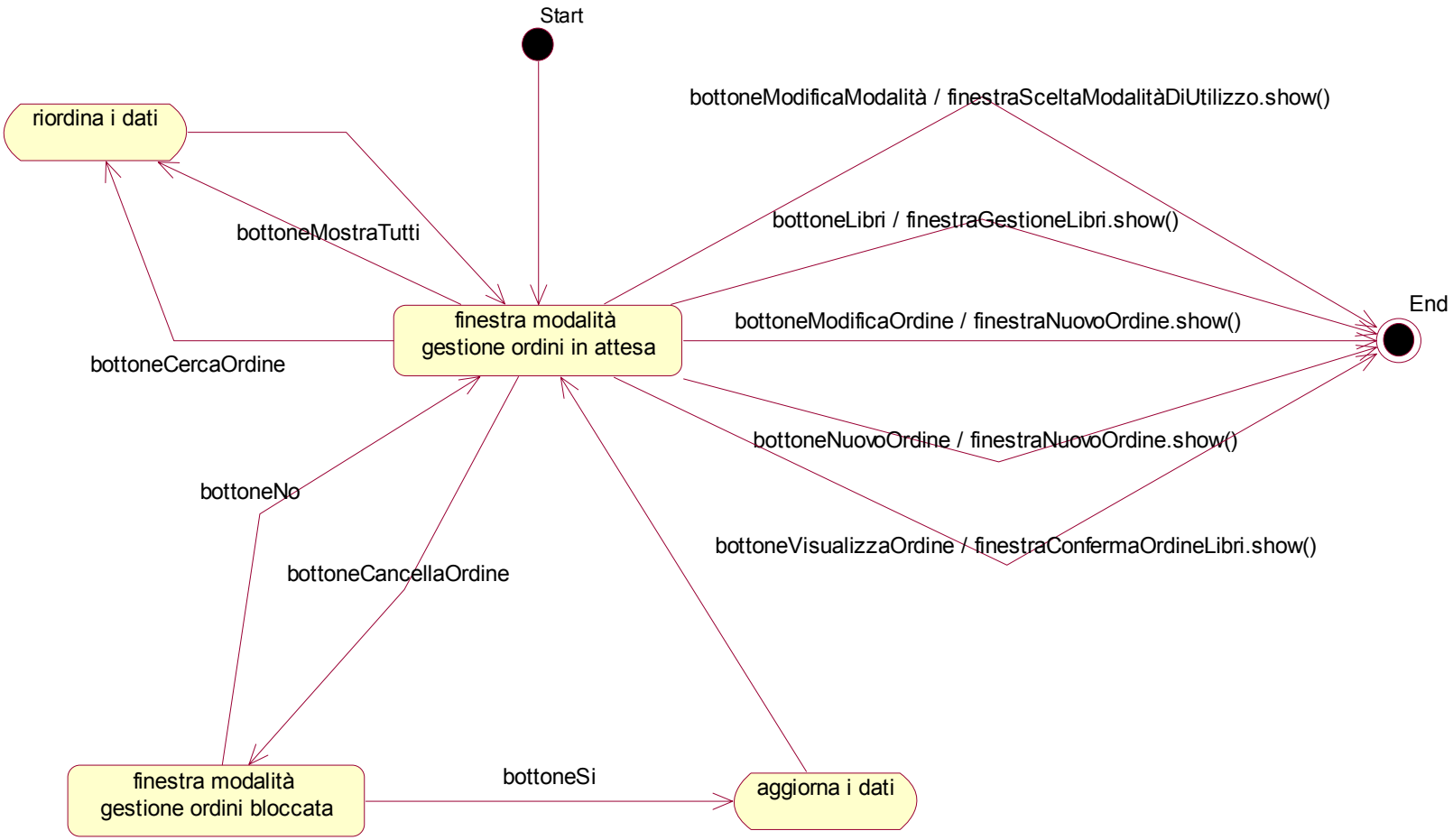
3.5.5.7 Finestra prenotazioni (2.2)



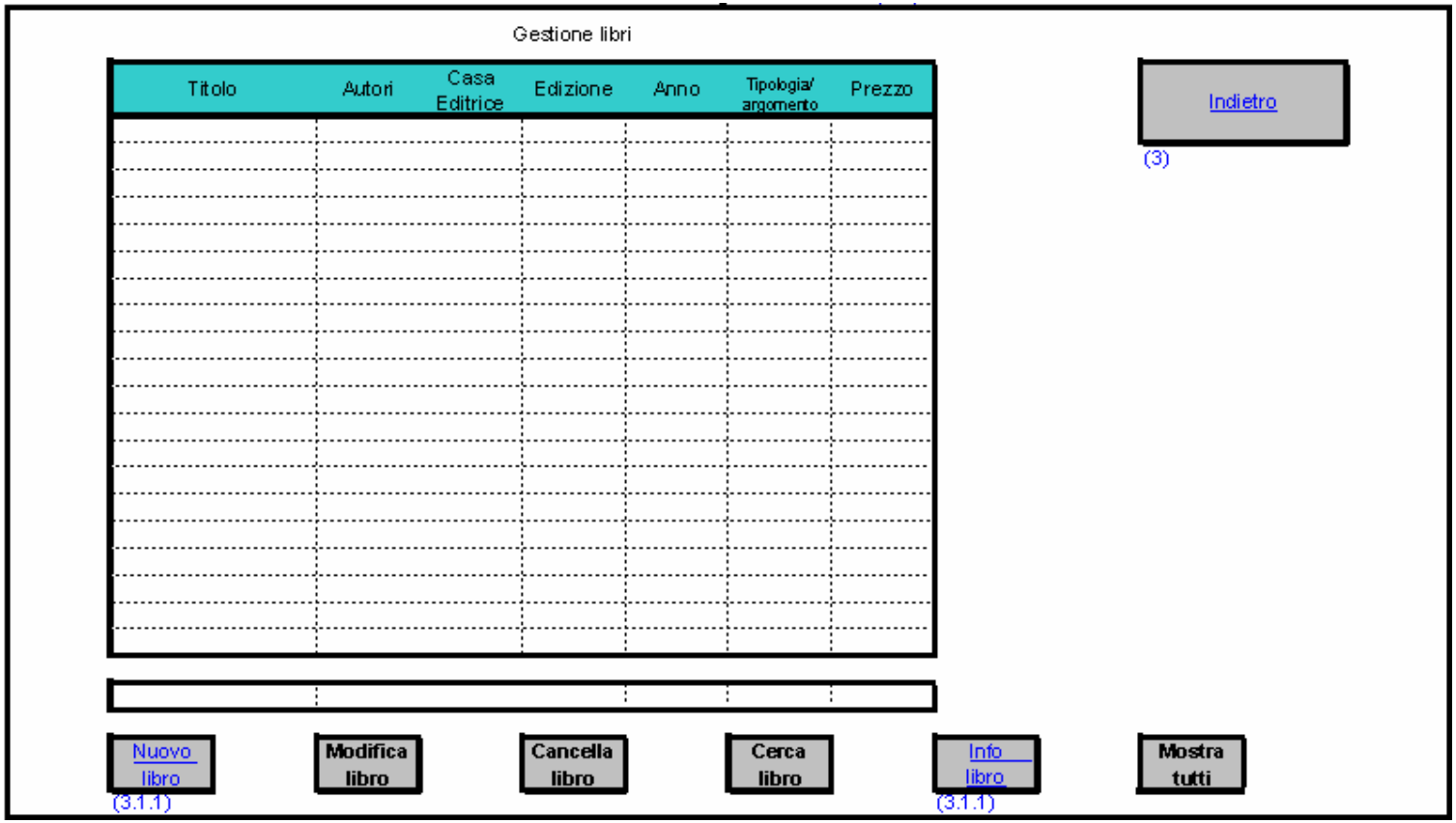

Start
bottoneModificaPrenotazione / finestraNuovaPrenotazione.show()
riordina i dati
finestra prenotazione in attesa
bottoneVisualizzaPrenotazione / finestraConfermaPrenotazioneLibri.show()
end
bottoneMostraTutti
bottoneCercaPrenotazione
bottoneIndietro / ModalitaVenditaLibril.show()
bottoneCancella
bottoneNo
finestra prenotazione bloccata
bottoneSi
aggiorna i dati

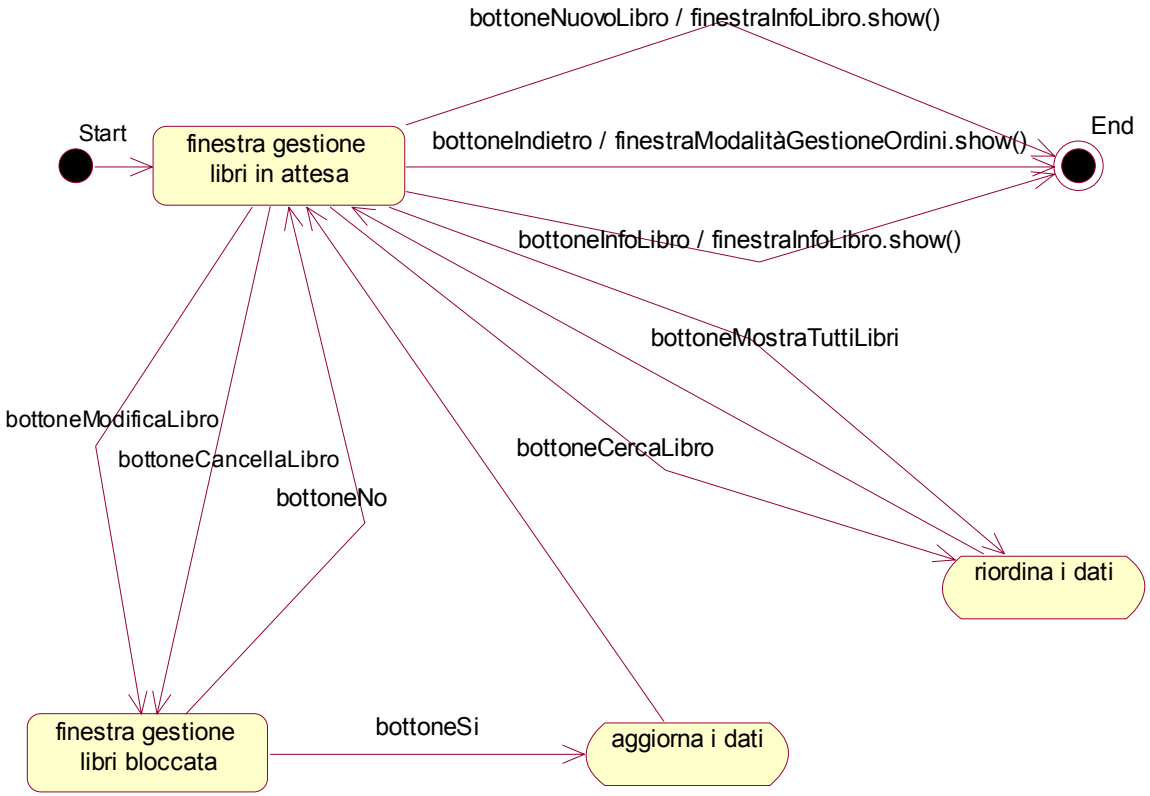
3.5.5.8 Finestra modalità gestione ordini (3)



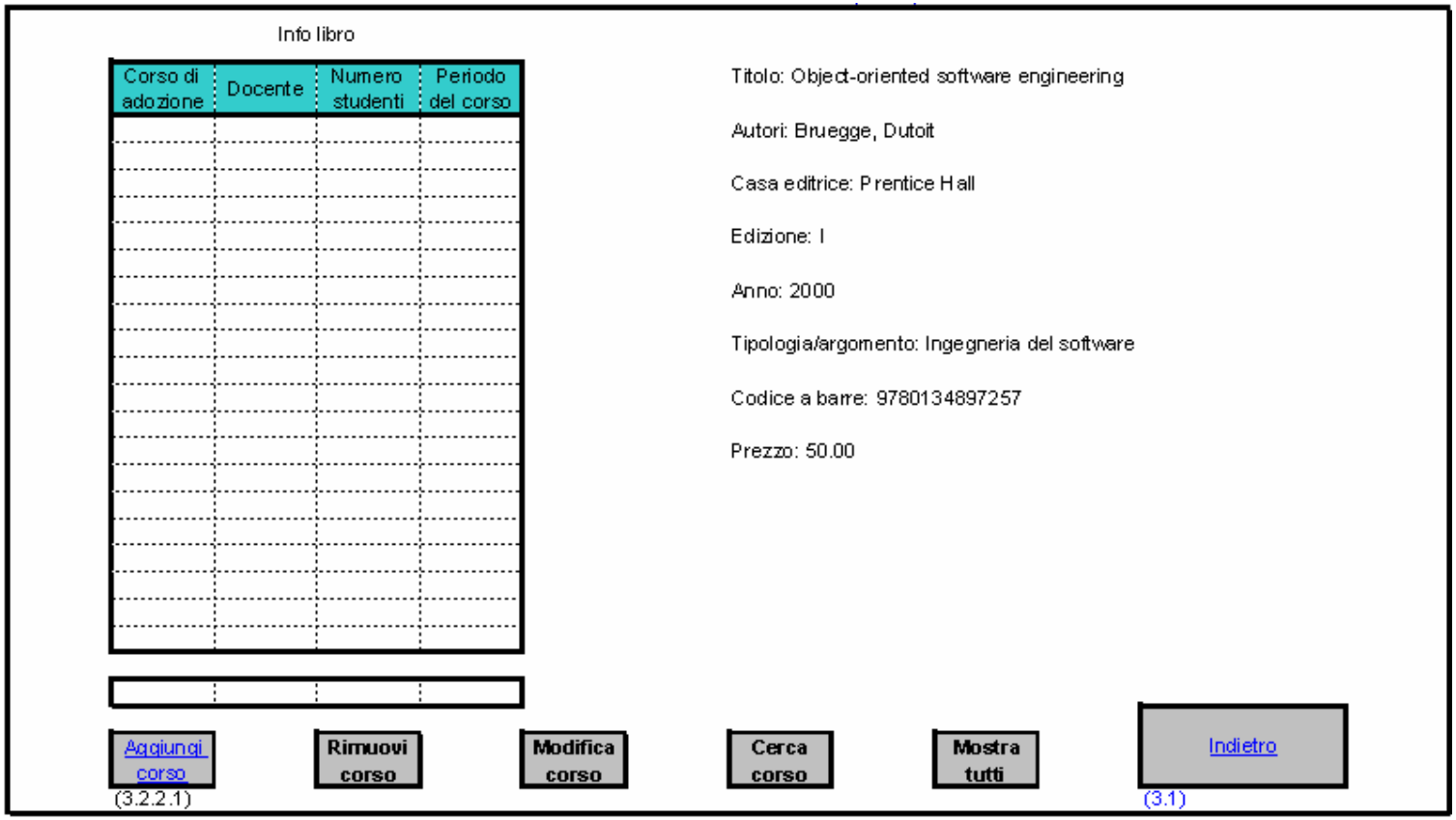

Start
riordina i dati
bottoneModificaModalità / finestraSceltaModalitàDiUtilizzo.show()
bottoneMostraTutti
bottoneLibri / finestraGestioneLibri.show()
End
finestra modalità
gestione ordini in attesa
bottoneModificaOrdine / finestraNuovoOrdine.show()
bottoneCercaOrdine
bottoneNuovoOrdine / finestraNuovoOrdine.show()
bottoneNo
bottoneVisualizzaOrdine / finestraConfermaOrdineLibri.show()
bottoneCancellaOrdine
finestra modalità
gestione ordini bloccata
bottoneSi
aggiorna i dati

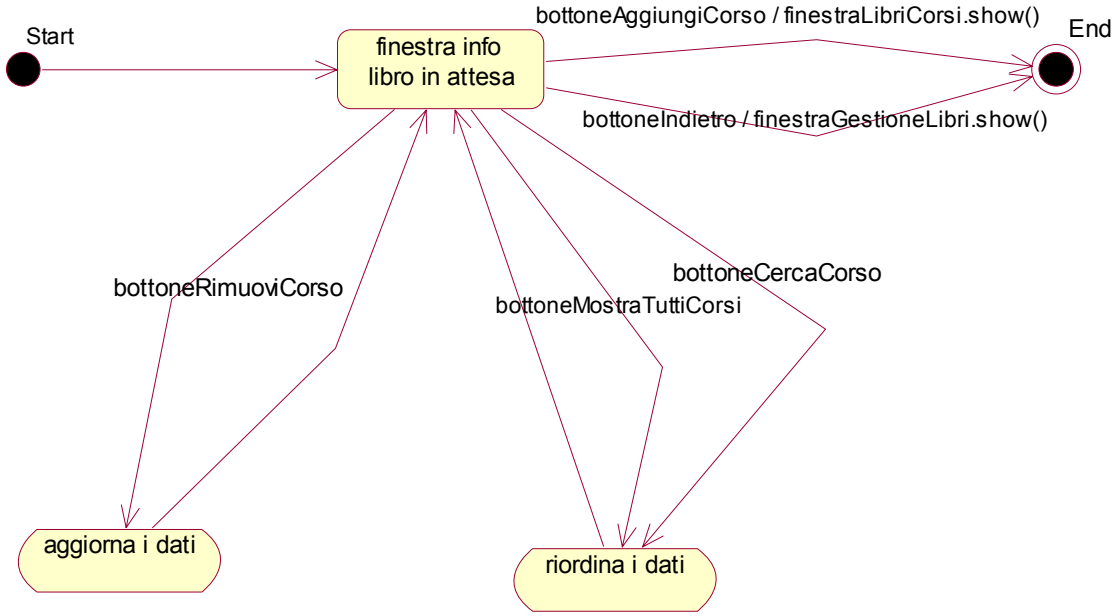
3.5.5.9 Finestra gestione libri (3.1)



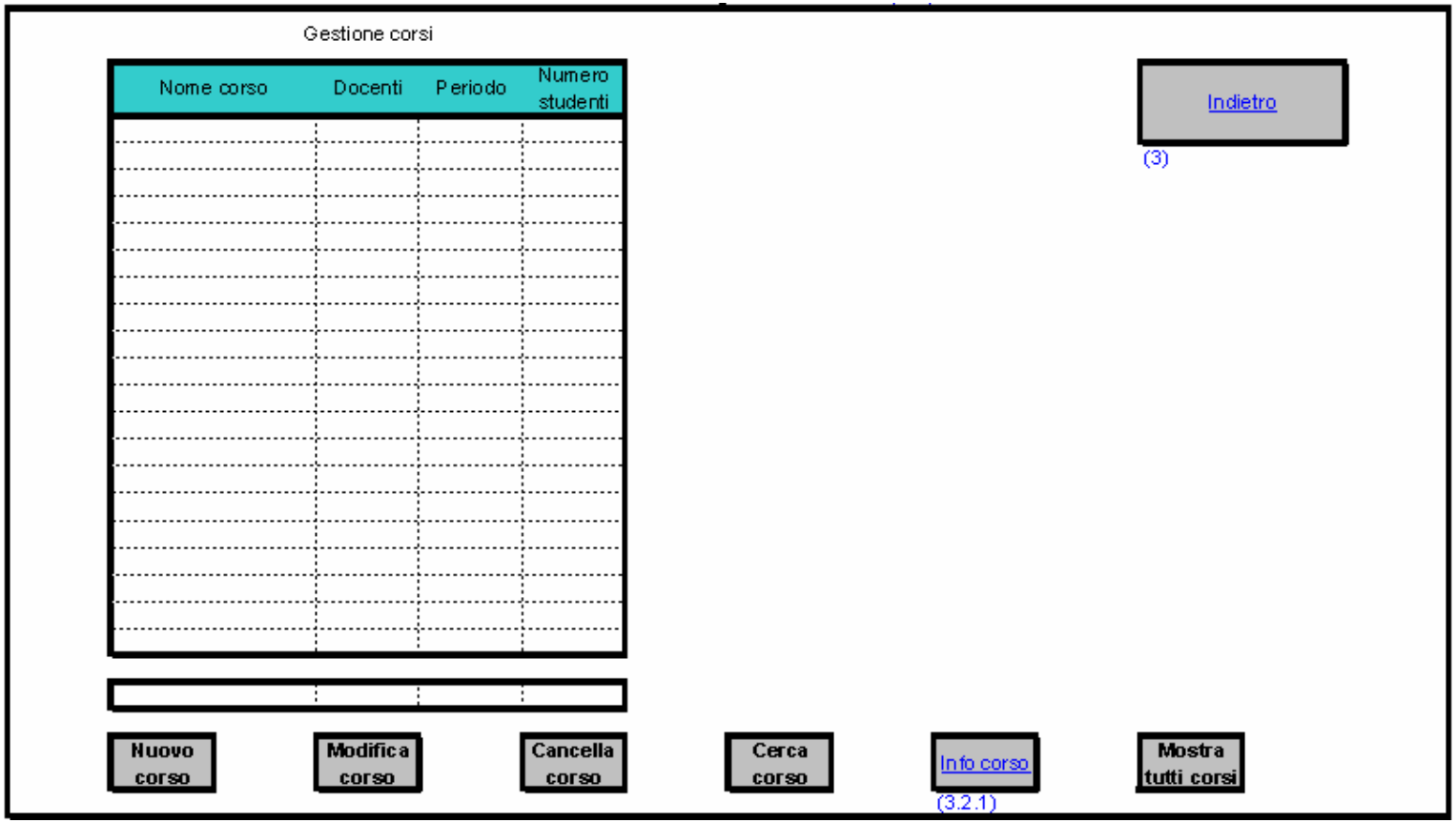

bottoneNuovoLibro / finestraInfoLibro.show()
Start
finestra gestione libri in attesa
bottoneIndietro / finestraModalitàGestioneOrdini.show()
End
bottoneInfoLibro / finestraInfoLibro.show()
bottoneMostraTuttiLibri
bottoneModificaLibro
bottoneCancellaLibro
bottoneNo
bottoneCercaLibro
riordina i dati
finestra gestione libri bloccata
bottoneSi
aggiorna i dati

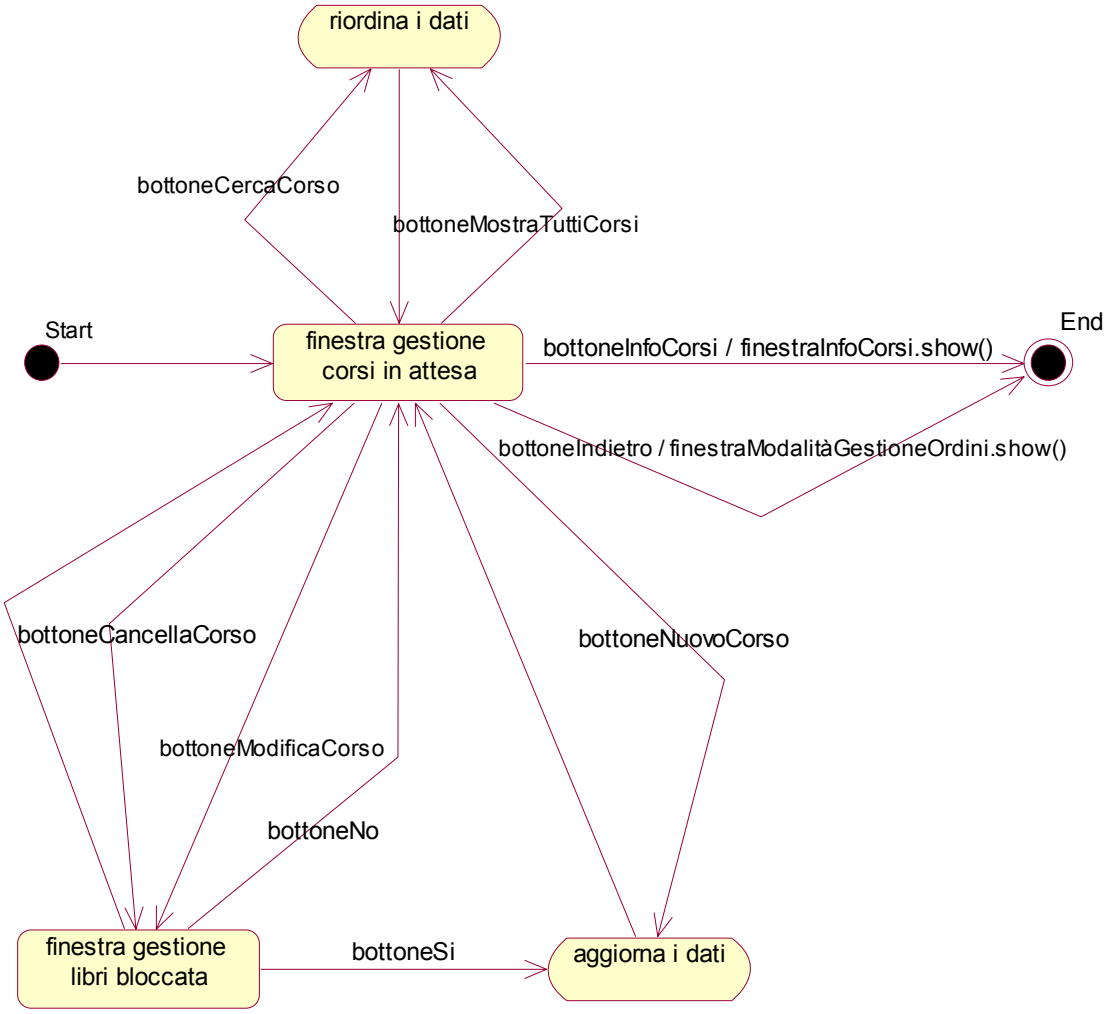
3.5.5.10 Finestra info libro (3.1.1)



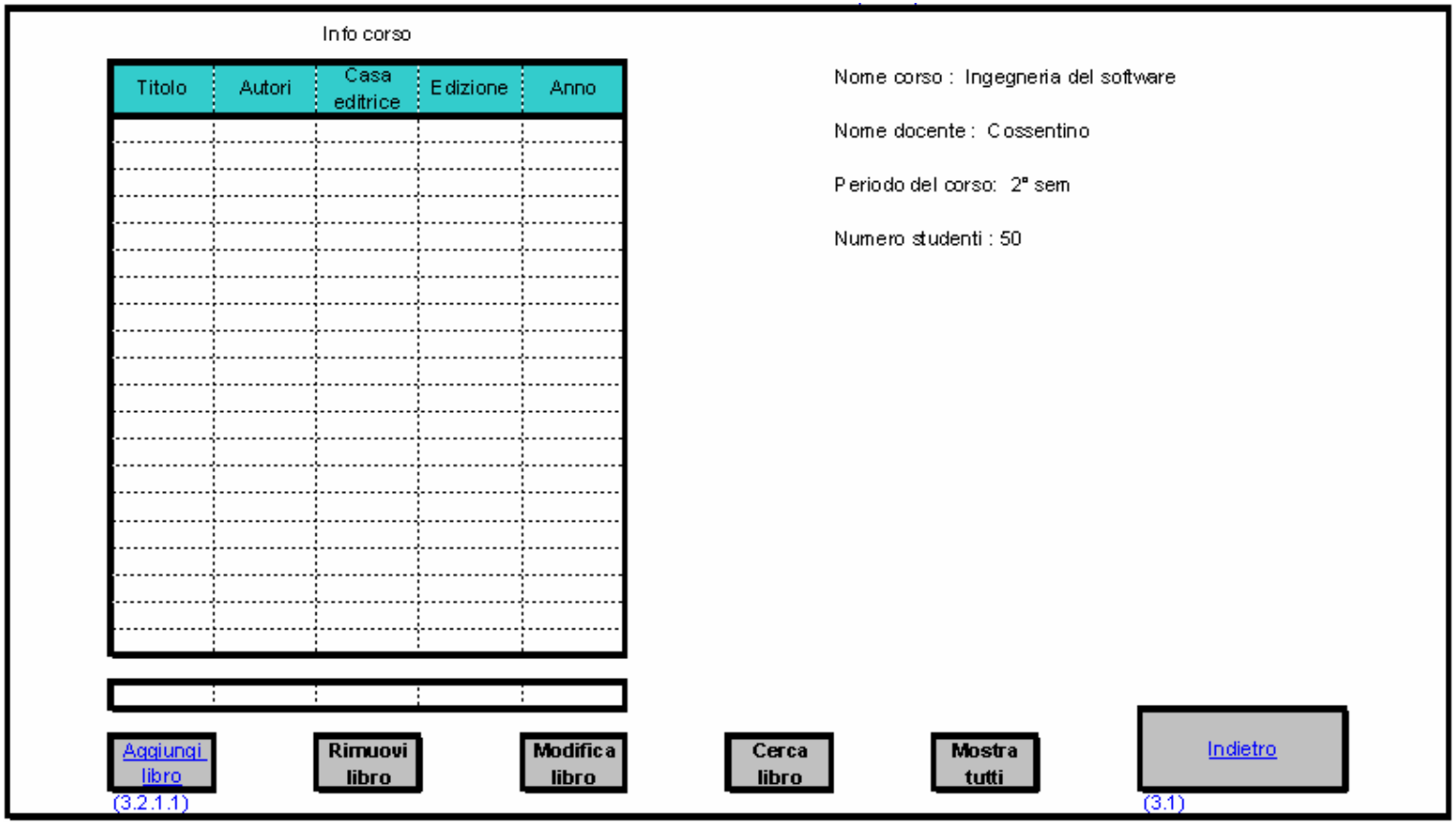

Start
finestra info
libro in attesa
bottoneAggiungiCorso / finestraLibriCorsi.show()
End
bottoneIndietro / finestraGestioneLibri.show()
bottoneRimuoviCorso
bottoneMostraTuttiCorsi
bottoneCercaCorso
aggiorna i dati
riordina i dati

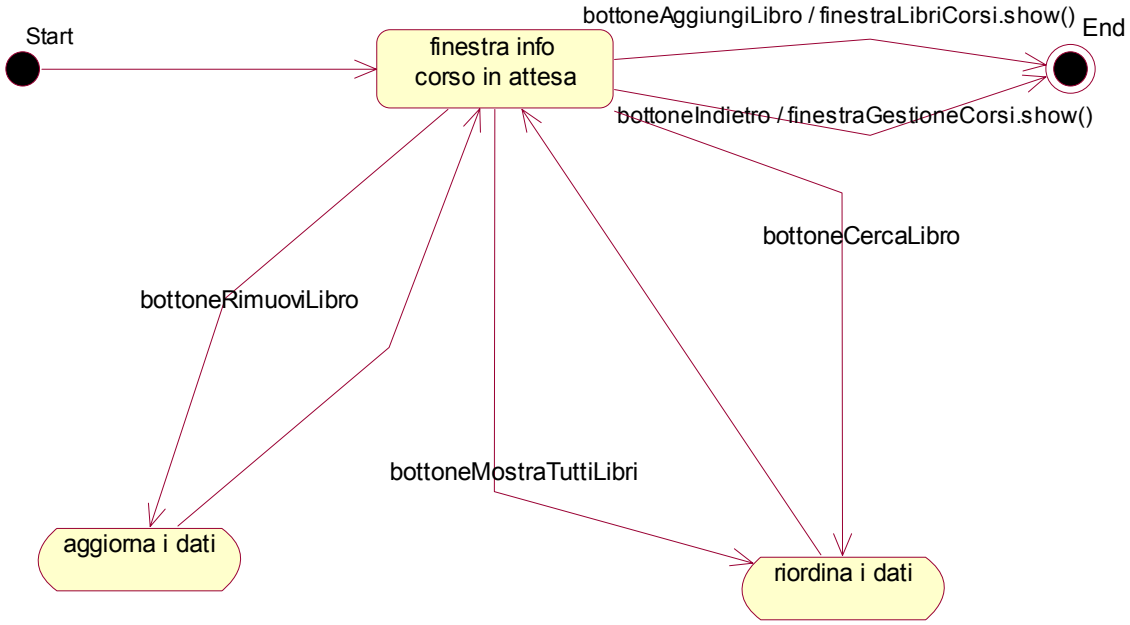
3.5.5.11 Finestra gestione corsi (3.2)



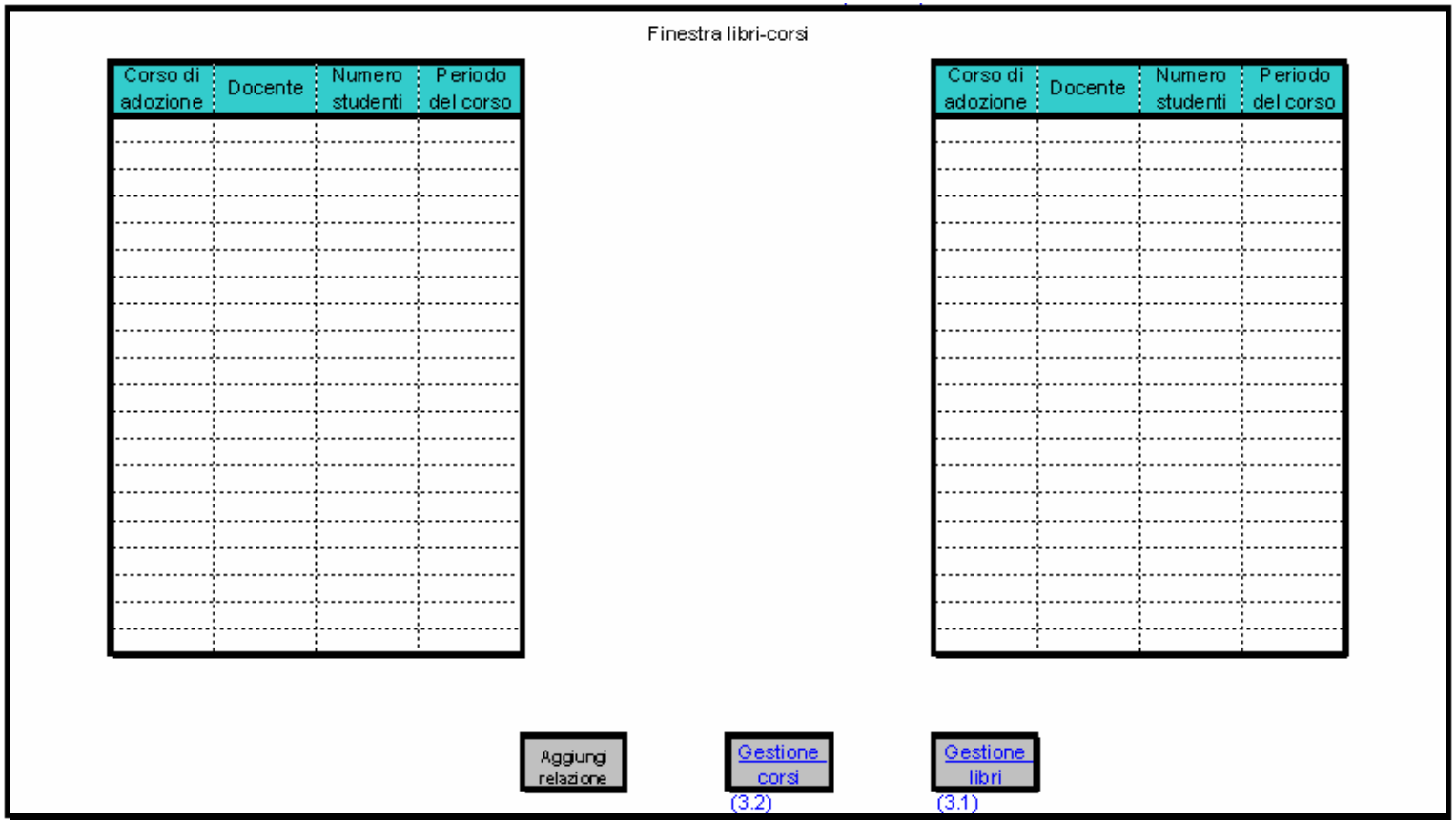

riordina i dati
bottoneCercaCorso
bottoneMostraTuttiCorsi
Start
finestra gestione
corsi in attesa
bottoneInfoCorsi / finestraInfoCorsi.show()
End
bottoneIndietro / finestraModalitàGestioneOrdini.show()
bottoneCancellaCorso
bottoneNuovoCorso
bottoneModificaCorso
bottoneNo
finestra gestione
libri bloccata
bottoneSi
aggiorna i dati

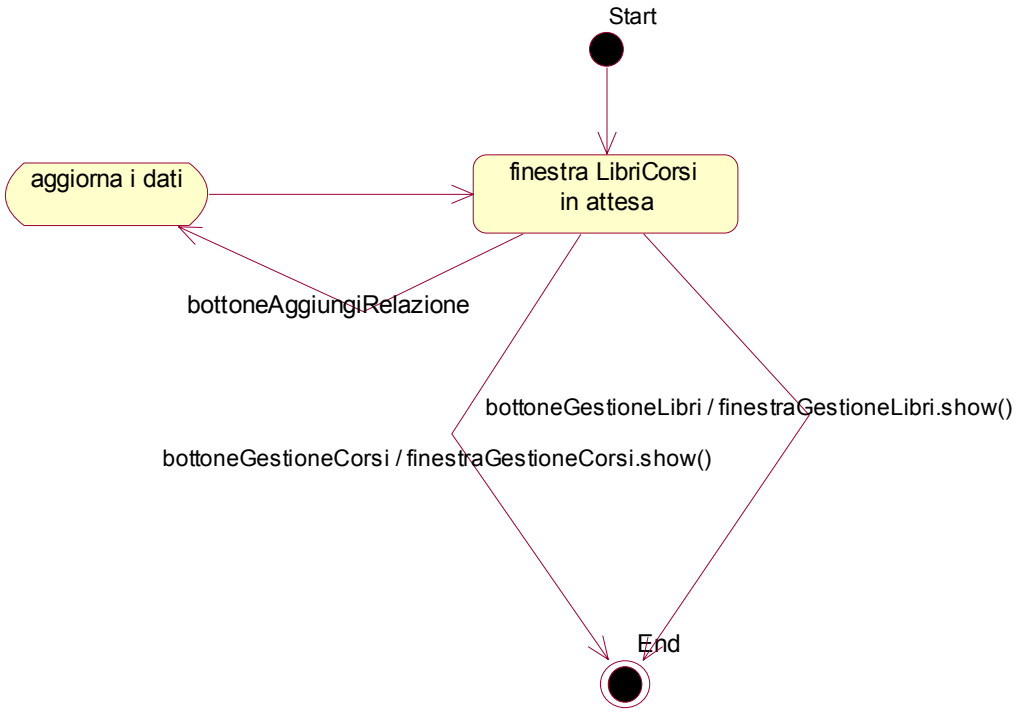
3.5.5.12 Finestra info corso (3.2.1)



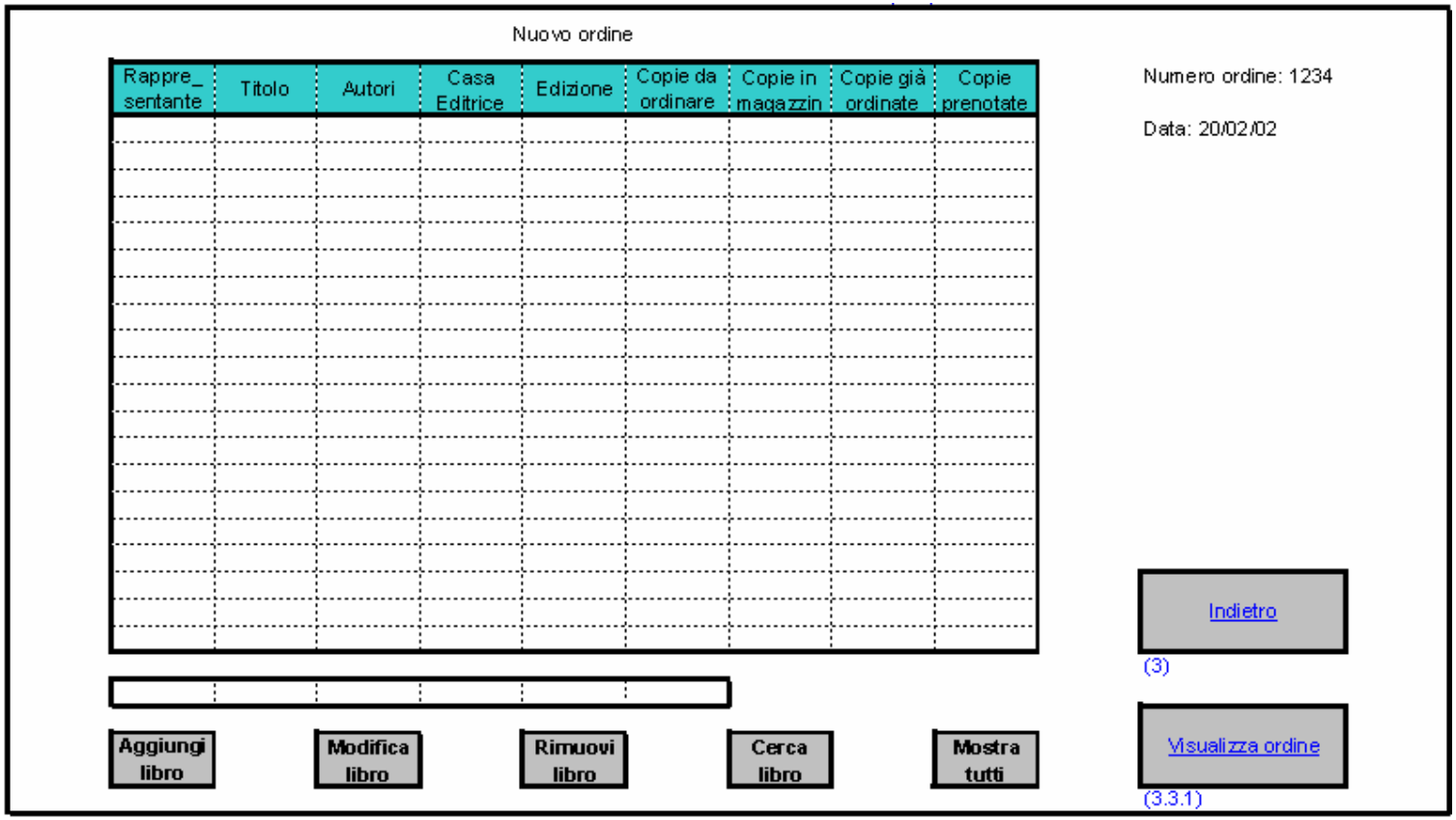

Start
finestra info
corso in attesa
bottoneAggiungiLibro / finestraLibriCorsi.show()
End
bottoneIndietro / finestraGestioneCorsi.show()
bottoneCercaLibro
bottoneRimuoviLibro
bottoneMostraTuttiLibri
aggiorna i dati
riordina i dati

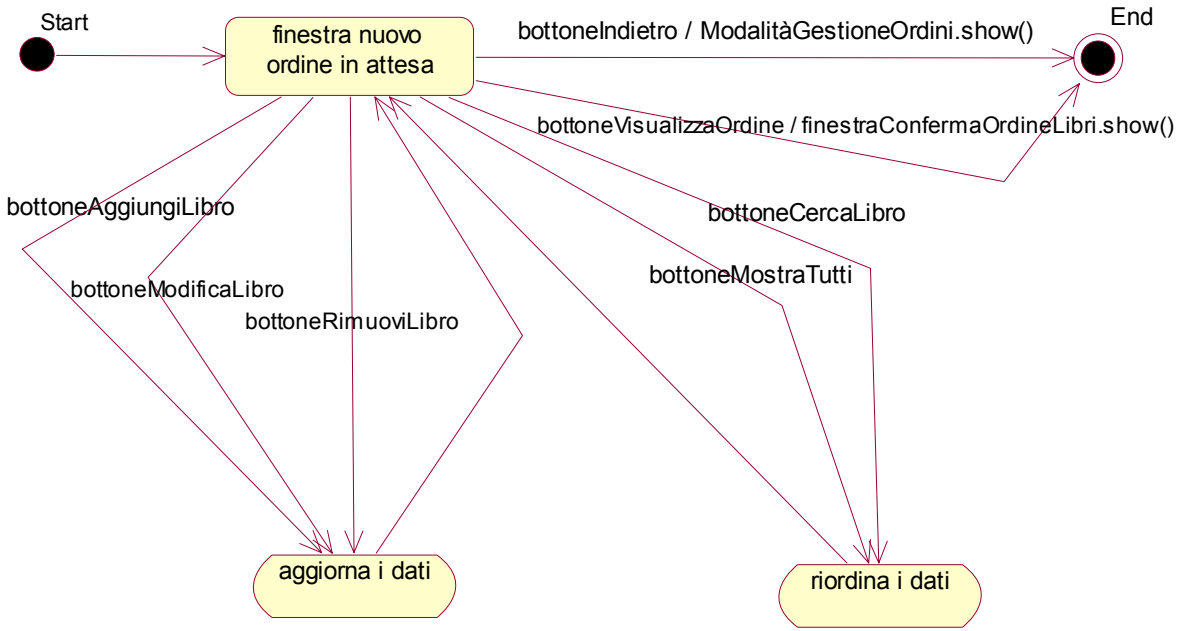
3.5.5.13 Finestra libri corsi (3.2.1.1)



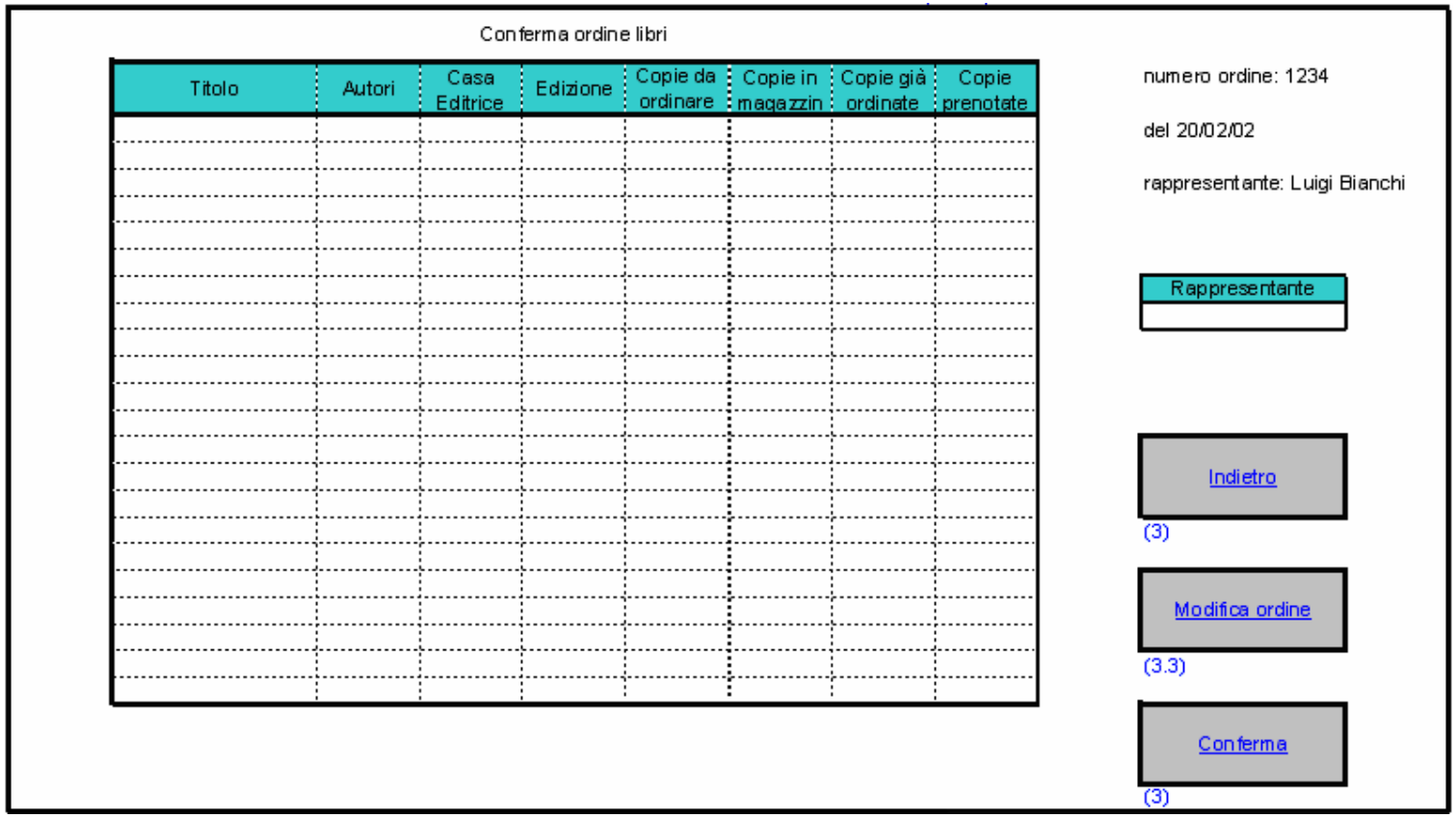

Start
aggiorna i dati
finestra LibriCorsi
in attesa
bottoneAggiungiRelazione
bottoneGestioneLibri / finestraGestioneLibri.show()
bottoneGestioneCorsi / finestraGestioneCorsi.show()
End

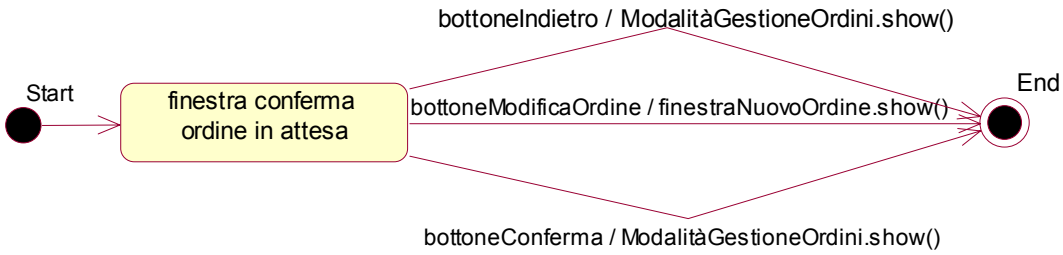
3.5.5.14 Finestra nuovo ordine (3.3)



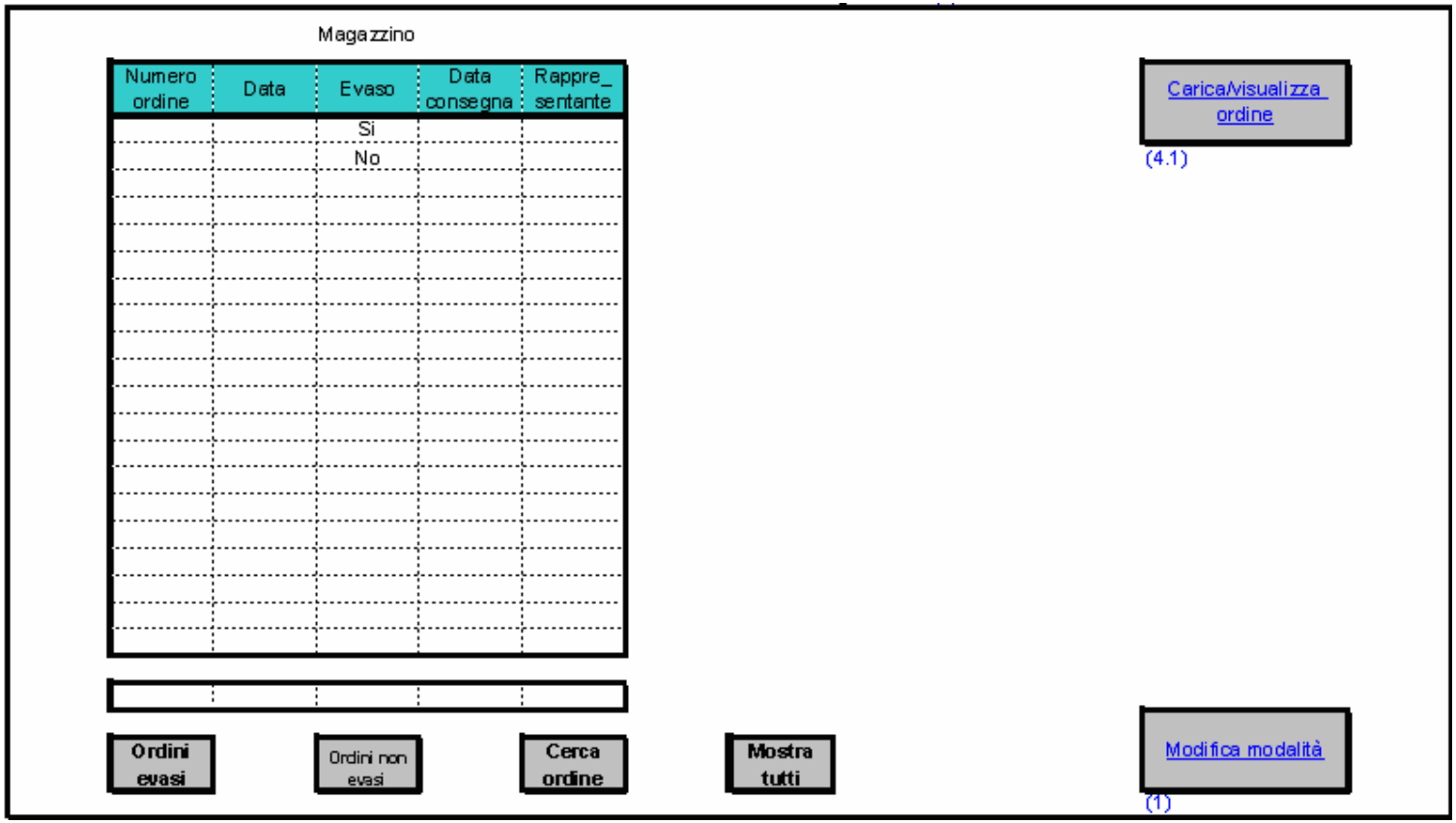

Start
finestra nuovo ordine in attesa
bottoneIndietro / ModalitàGestioneOrdini.show()
End
bottoneVisualizzaOrdine / finestraConfermaOrdineLibri.show()
bottoneAggiungiLibro
bottoneCercaLibro
bottoneModificaLibro
bottoneMostraTutti
bottoneRimuoviLibro
aggiorna i dati
riordina i dati

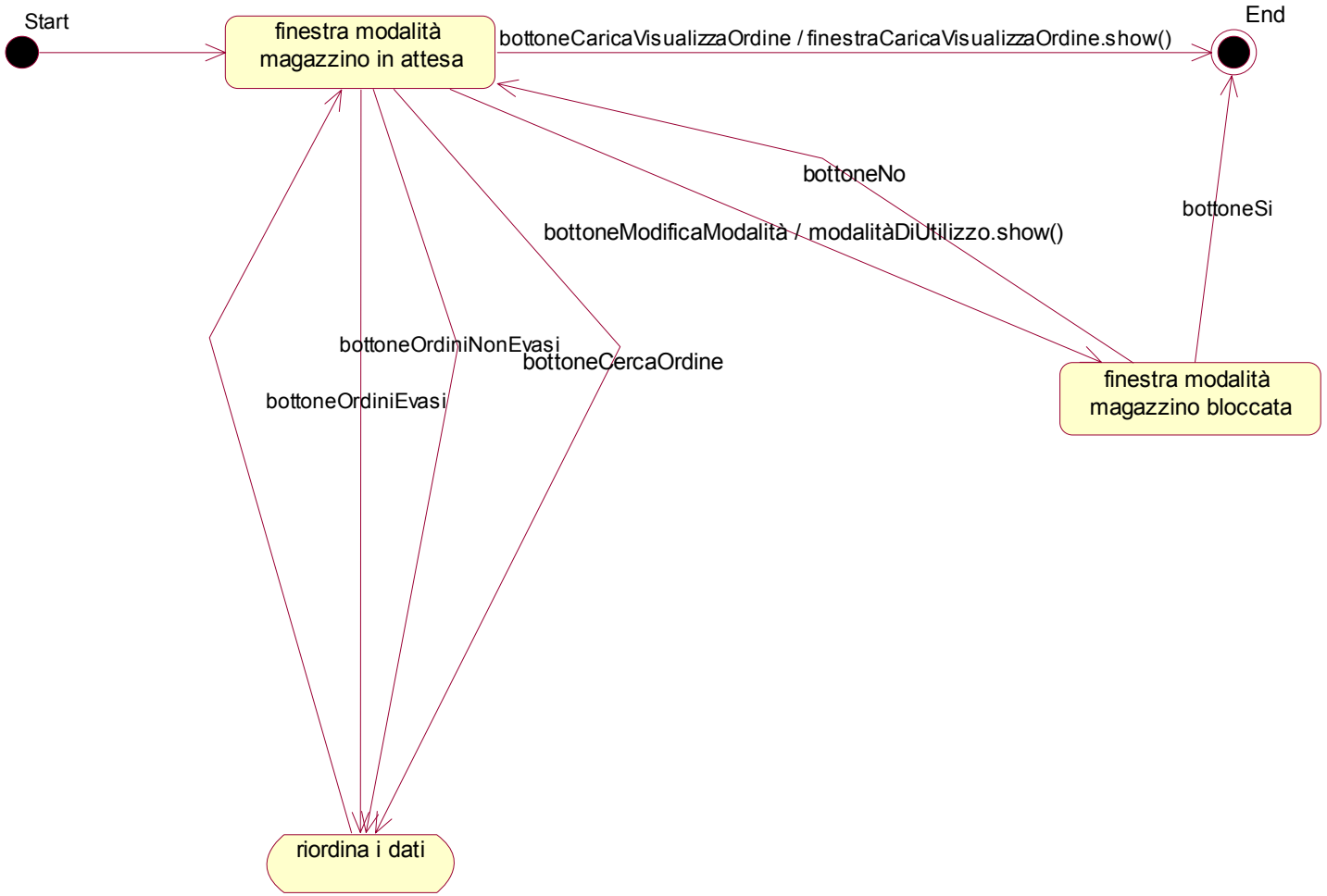
3.5.5.15 Finestra conferma ordine libri (3.3.1)



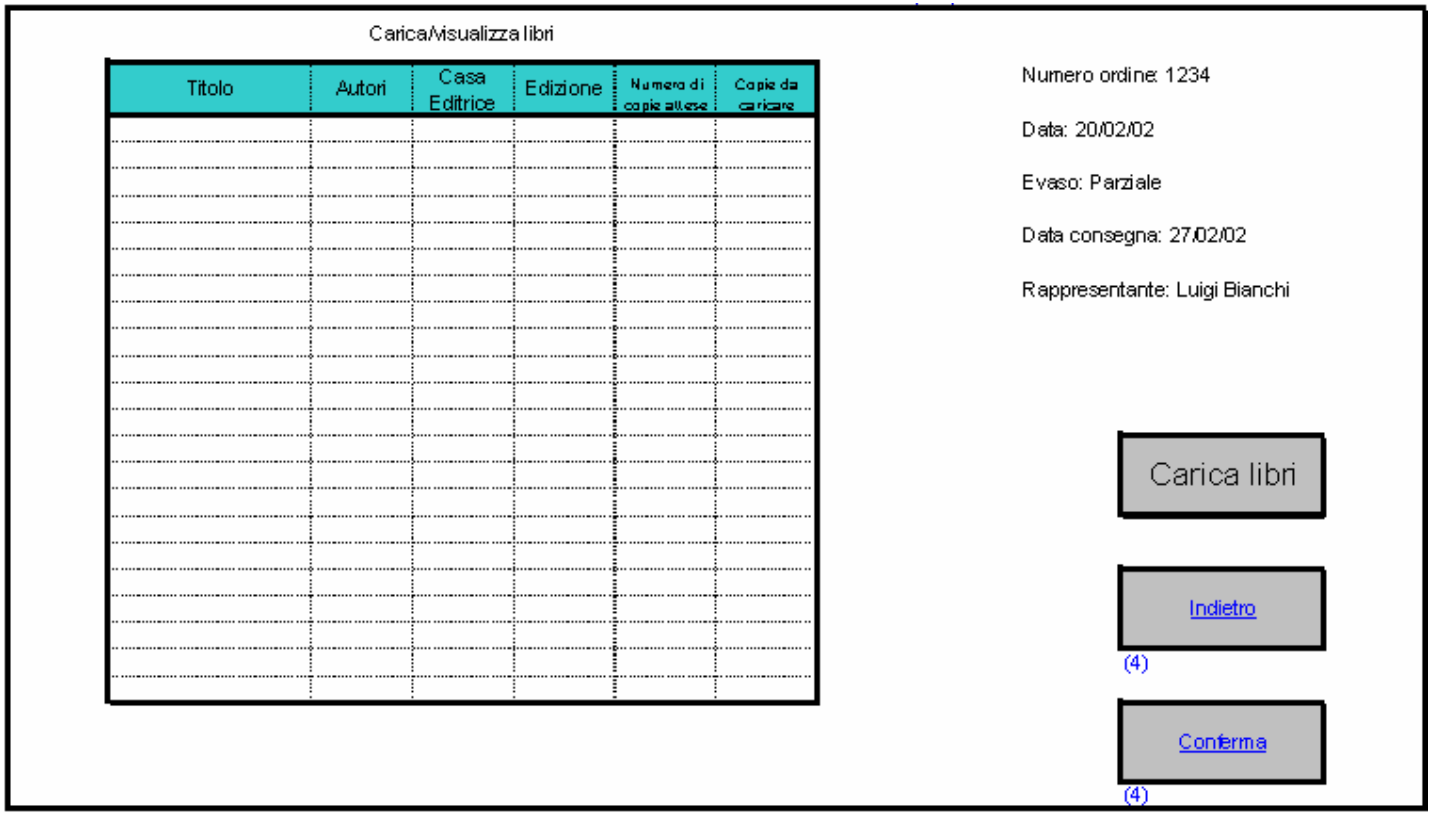

bottoneIndietro / ModalitàGestioneOrdini.show()
Start
finestra conferma ordine in attesa
bottoneModificaOrdine / finestraNuovoOrdine.show()
End
bottoneConferma / ModalitàGestioneOrdini.show()

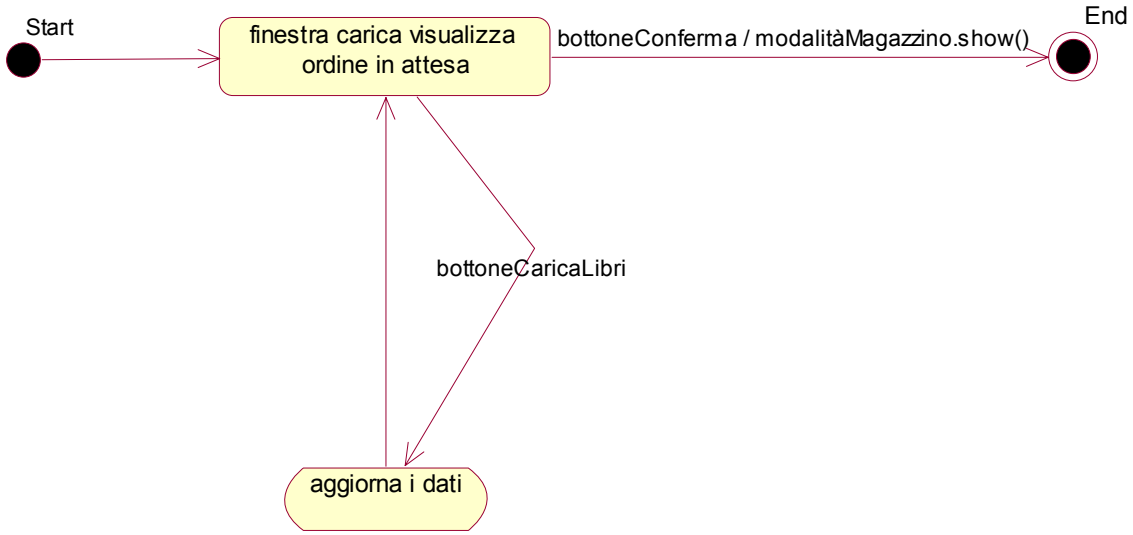
3.5.5.16 Modalità magazzino (4)




Start
finestra modalità magazzino in attesa
bottoneCaricaVisualizzaOrdine / finestraCaricaVisualizzaOrdine.show()
End
bottoneNo
bottoneSi
bottoneModificaModalità / modalitàDiUtilizzo.show()
bottoneOrdiniNonEvasi
bottoneCercaOrdine
bottoneOrdiniEvasi
finestra modalità magazzino bloccata
riordina i dati

3.5.5.17 Finestra carica/visualizza ordini (4.1)




Start
finestra carica visualizza ordine in attesa
bottoneConferma / modalitàMagazzino.show()
End
bottoneCaricaLibri
aggiorna i dati

Architettura del sistema
(System Design Document)

3. Architettura software proposta

3.1 Overview

Il sistema sarà suddiviso in due sottosistemi secondo le funzionalità da essi offerte. In particolare il primo è composto dalle classi che permettono l'interazione tra l'utente e il sistema, mentre il secondo, composto da classi analoghe alle entità del database, funge da supporto allo scambio di informazioni tra i metodi delle classi del primo sottosistema. Si è operata tale scelta in quanto, pur essendoci nel sistema tre attori con funzionalità diverse, il loro modo di operare risulta del tutto analogo.

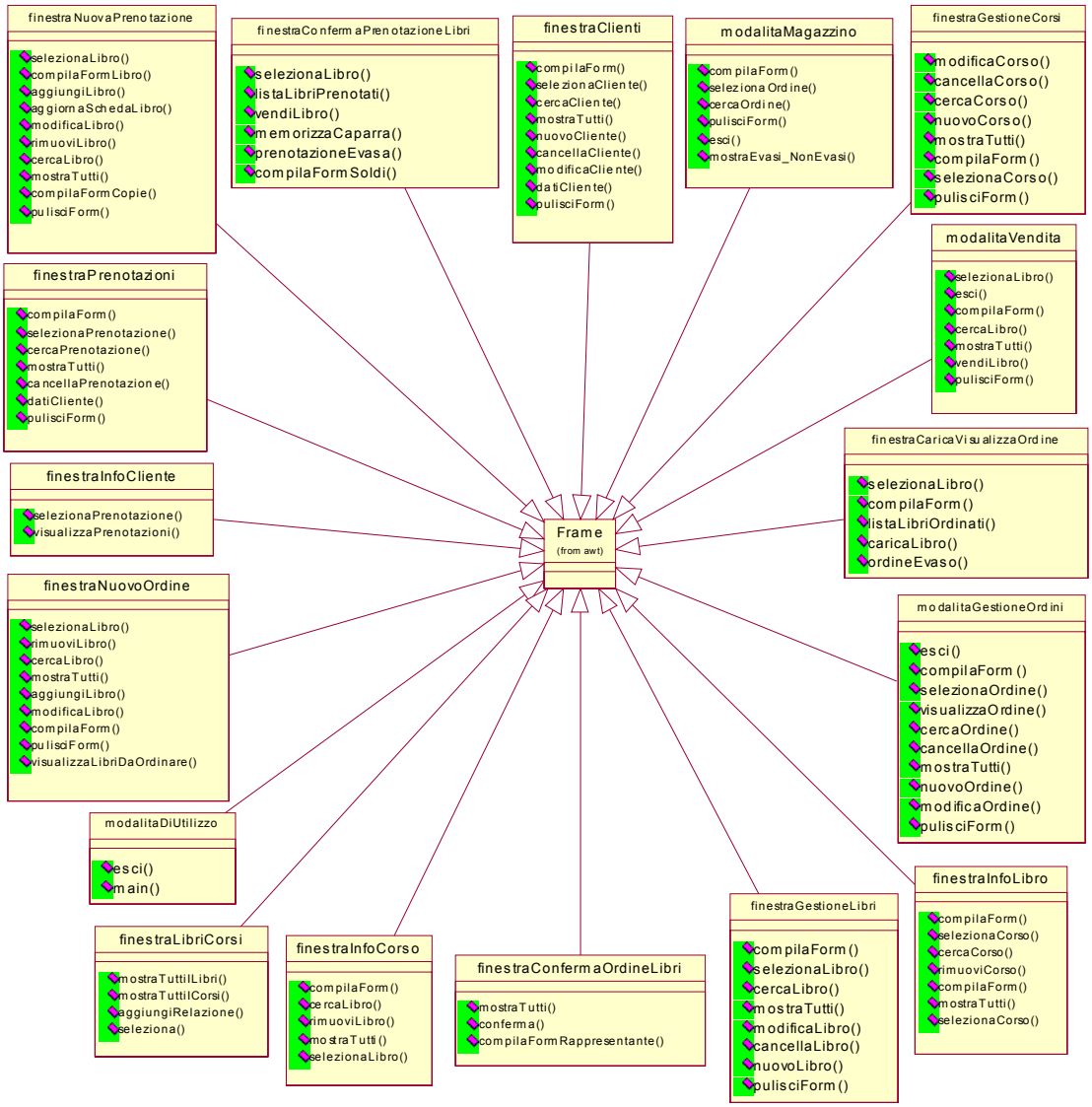
3.2 Decomposizione in sottosistemi

I sottosistemi individuati sono:

- Terminali utente
- Servizi utenti

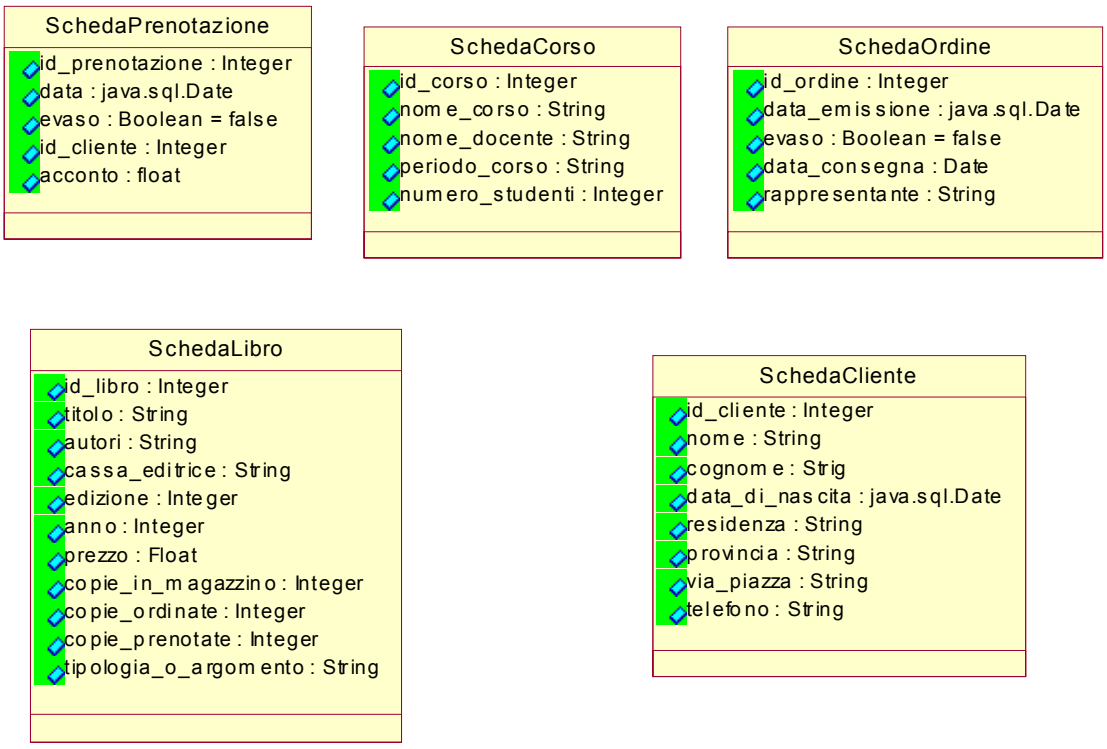
3.2.1 Terminali utente

Si occupa di tutti gli aspetti relativi alle operazioni che un utente può svolgere mentre sta lavorando. A tal fine è stato identificato un diagramma delle classi relativo al package terminale utente.


finestraNuovaPrenotazione
selezionaLibro()
compilaFormLibro()
aggiungiLibro()
aggiornaSchedaLibro()
modificaLibro()
rimuoviLibro()
cercaLibro()
mostraTutti()
compilaFormCopie()
pulisciForm()
finestraConfermaPrenotazioneLibri
selezionaLibro()
listaLibriPrenotati()
vendiLibro()
memorizzaCaparra()
prenotazioneEvasa()
compilaFormSoldi()
finestraClienti
compilaForm()
selezionaCliente()
cercaCliente()
mostraTutti()
nuovoCliente()
cancellaCliente()
modificaCliente()
datiCliente()
pulisciForm()
modalitaMagazzino
compilaForm()
selezionaOrdine()
cercaOrdine()
pulisciForm()
esci()
mostraEvasi_NonEvasi()
finestraGestioneCorsi
modificaCorso()
cancellaCorso()
cercaCorso()
nuovoCorso()
mostraTutti()
compilaForm()
selezionaCorso()
pulisciForm()
finestraPrenotazioni
compilaForm()
selezionaPrenotazione()
cercaPrenotazione()
mostraTutti()
cancellaPrenotazione()
datiCliente()
pulisciForm()
modalitaVendita
selezionaLibro()
esci()
compilaForm()
cercaLibro()
mostraTutti()
vendiLibro()
pulisciForm()
finestraInfoCliente
selezionaPrenotazione()
visualizzaPrenotazioni()
finestraCaricaVisualizzaOrdine
selezionaLibro()
compilaForm()
listaLibriOrdinati()
caricaLibro()
ordineEvaso()
Frame
(from awt)
finestraNuovoOrdine
selezionaLibro()
rimuoviLibro()
cercaLibro()
mostraTutti()
aggiungiLibro()
modificaLibro()
compilaForm()
pulisciForm()
visualizzaLibriDaOrdinare()
modalitaGestioneOrdini
esci()
compilaForm()
selezionaOrdine()
visualizzaOrdine()
cercaOrdine()
cancellaOrdine()
mostraTutti()
nuovoOrdine()
modificaOrdine()
pulisciForm()
modalitaDiUtilizzo
esci()
main()
finestraInfoLibro
compilaForm()
selezionaCorso()
cercaCorso()
rimuoviCorso()
compilaForm()
mostraTutti()
selezionaCorso()
finestraGestioneLibri
compilaForm()
selezionaLibro()
cercaLibro()
mostraTutti()
modificaLibro()
cancellaLibro()
nuovoLibro()
pulisciForm()
finestraLibriCorsi
mostraTuttiILibri()
mostraTuttiICorsi()
aggiungiRelazione()
seleziona()
finestraInfoCorso
compilaForm()
cercaLibro()
rimuoviLibro()
mostraTutti()
selezionaLibro()
finestraConfermaOrdineLibri
mostraTutti()
conferma()
compilaFormRappresentante()

3.2.2 Servizi utente

Questo sottosistema fornisce delle classi, analoghe alle entità del database, che permettono una più semplice manipolazione dei dati acquisiti dal database. Per far ciò viene utilizzata la libreria JDBC per accedere al database e query SQL per la sua interrogazione. Le classi scheda ordine, scheda libro, scheda corso, scheda cliente e scheda prenotazione non presentano relazioni tra di loro ma, come visto nelle sezioni **3.5.3.1.1**, **3.5.3.1.2** e **3.5.3.1.3**, sono di supporto alle classi del sottosistema terminale utente.



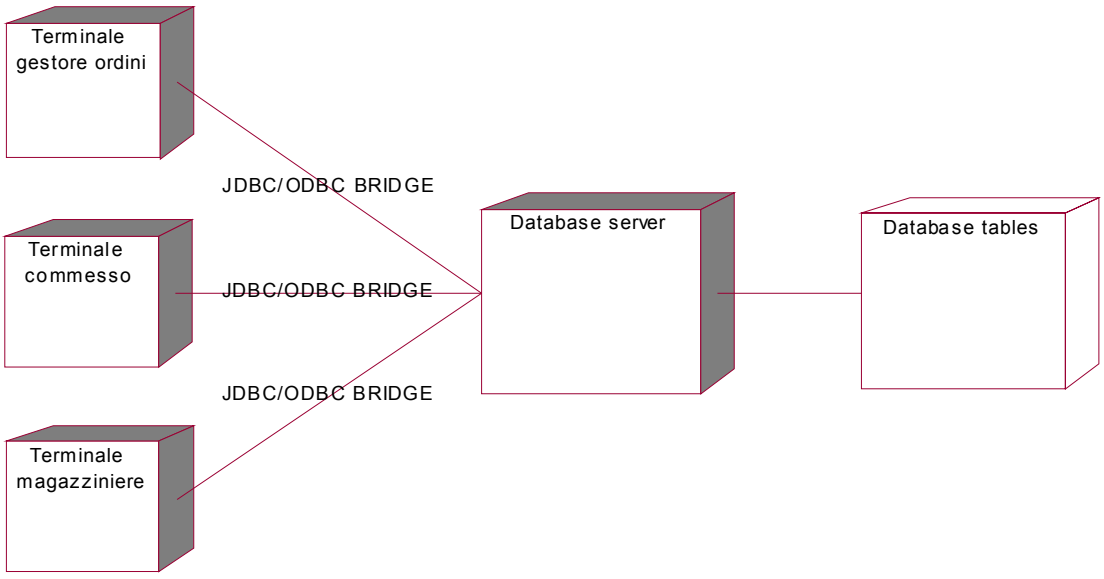
3.3 Dislocazione hardware/software

Il sistema progettato funziona su un'unica macchina dove l'utente, all'occorrenza, può selezionare una qualunque modalità d'uso tra le tre previste. L'utilizzo, comunque, della libreria JDBC rende il sistema estremamente versatile e permette una sua utilizzazione anche su una rete di computer con la sola modifica del driver di connessione al database.

Il sistema è stato testato con esito positivo su una rete Fast Ethernet 10/100 composta da due personal computer collegati con scheda di rete PCI LAN 10/100 Digicom tramite cavo UTP cat. 5 a 8 fili incrociato e connettori RJ45.

Data l'ambito del sistema (una libreria) si può quindi supporre il suo utilizzo in una rete locale con gerarchia peer to peer o anche con una architettura di tipo client-server. Gli elaboratori utilizzabili sono personal computer con processori Intel o Amd e sistema operativo della famiglia Windows.

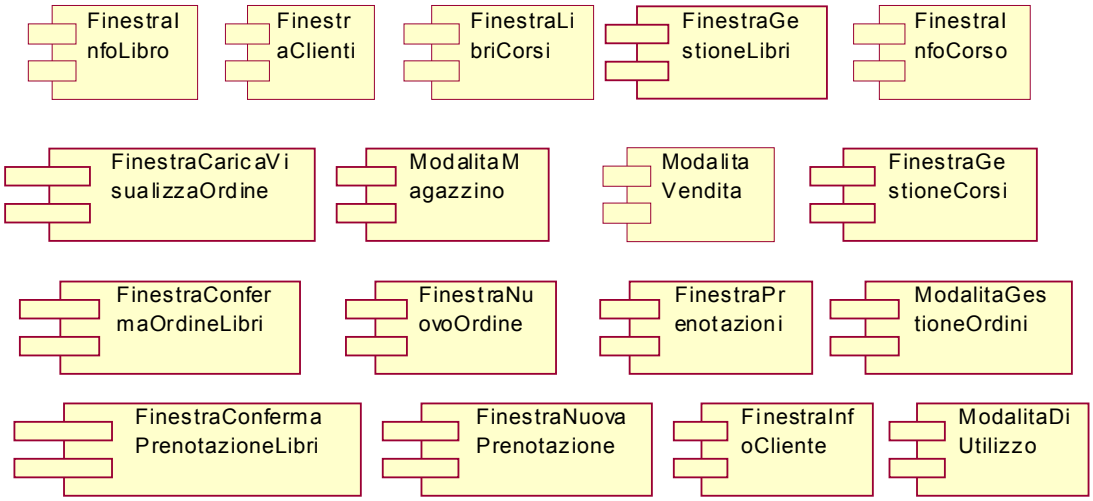
3.3.1 Diagramma di Deployment del progetto



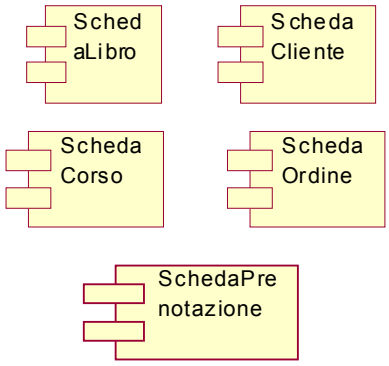
3.3.2 Diagrammi delle componenti del sistema

Il CASE tool Rational Rose non permette, in Java, l'associazione di più classi a un'unica componente, ma genera automaticamente un package immagine a livello di component view semplicemente trasportando il package di cui si vuole generare il codice dal logical view al component view. Il package immagine avrà tante component quante sono le classi e ad ognuna associa l'omonima classe.

3.3.2.1 Package TerminaleUtente



3.3.2.2 Package ServiziUtente



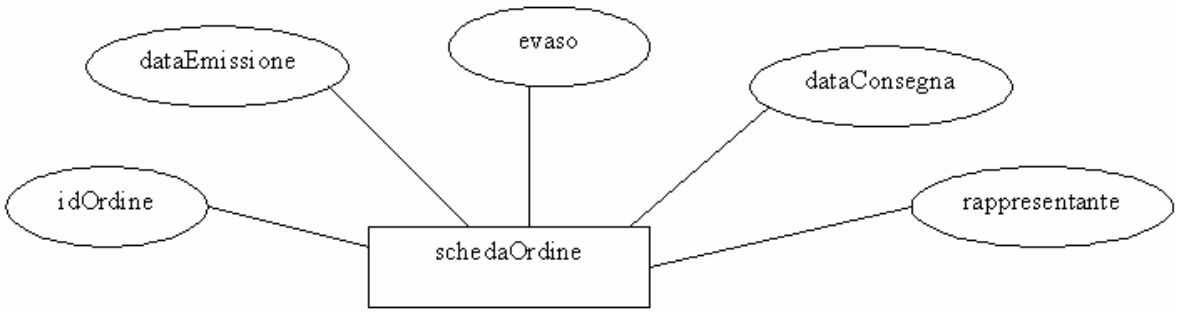
3.4 Gestione dei dati persistenti

Attraverso i diagrammi entità relazione (ERD), si darà un'ipotetica schematizzazione delle entità individuate nel progetto, dei loro attributi e delle loro relazioni.

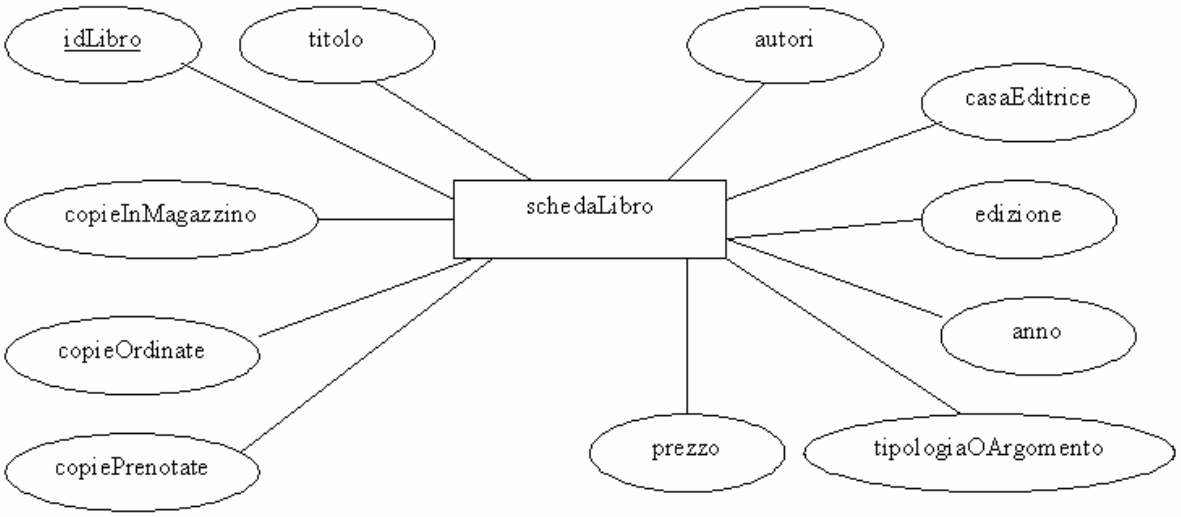
3.4.1 Progetto concettuale

Lo schema concettuale è la rappresentazione più astratta dei dati e delle relazioni.

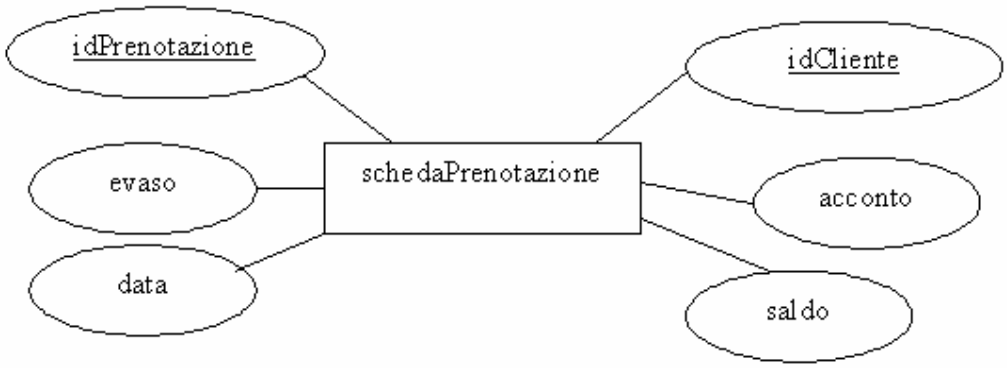
3.4.1.1 Entità schedaOrdine



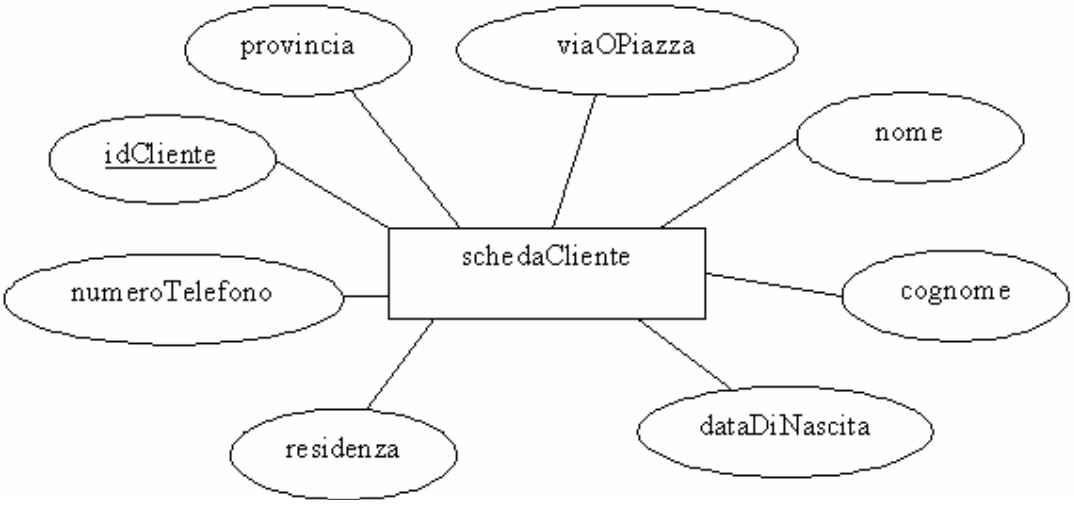
3.4.1.2 Entità schedaLibro



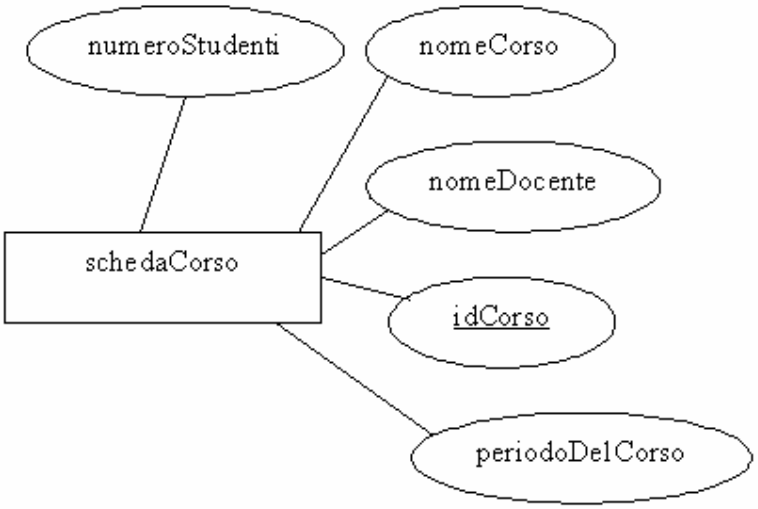
3.4.1.3 Entità schedaPrenotazione



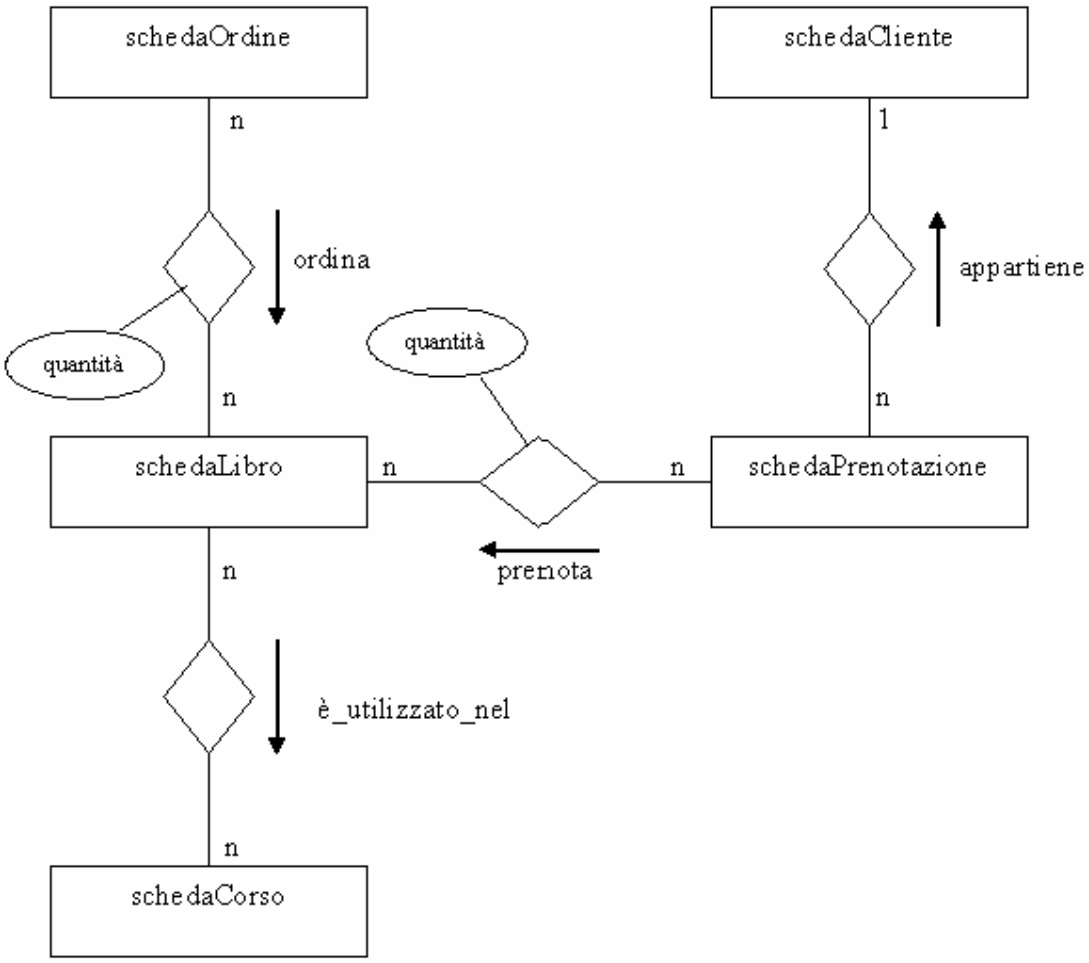
3.4.1.4 Entità schedaCliente



3.4.1.5 Entità schedaCorso



3.4.2 Relazioni tra le entità



3.4.3 Progetto logico

Per la tipologia e per la quantità di dati si è scelta l'implementazione di un database relazionale.

In seguito sono riportate le tabelle relative alle entità ERD.

schedaLibro	
Campo	**Tipo**
idLibro	contatore
titolo	testo (50) (richiesto: si, indice: Si)
autori	testo (30) (richiesto: si, indice: Si)
casaEditrice	testo (20) (richiesto: si, indice: Si)
edizione	byte (richiesto: no, indice: No)
anno	intero (richiesto: no, indice: No)
prezzo	valuta (richiesto: no, indice: No)
copieInMagazzino	intero (richiesto: no, indice: No)
copieOrdinate	intero (richiesto: no, indice: No)
copiePrenotate	intero (richiesto: no, indice: No)
tipologiaOArgomento	testo (30) (richiesto: no, indice: Si)

schedaOrdine	
Campo	**Tipo**
idOrdine	contatore
dataEmissione	data (richiesto: si, indice: Si)
Evaso	{si, no} =no (richiesto si, indice Si)
dataConsegna	data (valido se "evaso=si") (richiesto: no, indice: Si)
rappresentante	testo (20) (richiesto: si, indice: Si)

schedaPrenotazione	
Campo	**Tipo**
idPrenotazione	Contatore
data	data (richiesto: si, indice: Si)
evaso	{si, no } =no (richiesto: si, indice: Si)
idCliente	Intero lungo (richiesto: si, indice: Si)

schedaCliente	
Campo	**Tipo**
idCliente	contatore
nome	testo (15) (richiesto: si, indice: Si)
cognome	testo (15) (richiesto: si, indice: Si)
dataDiNascita	data (richiesto: no, indice: No)
residenza	testo (20) (richiesto: no, indice: No)
provincia	testo (10) (richiesto: no, indice: No)
viaOPiazza	testo (30) (richiesto: no, indice: No)
numeroTelefono	testo (15) (richiesto: no, indice: No)

schedaCorso	
Campo	**Tipo**
idCorso	contatore
nomeCorso	testo (20) (richiesto: si, indice: Si)
nomeDocente	testo (15) (richiesto: no, indice: No)
periodoCorso	testo (15) (richiesto: no, indice: No)
numeroStudenti	intero (richiesto: no, indice: No)

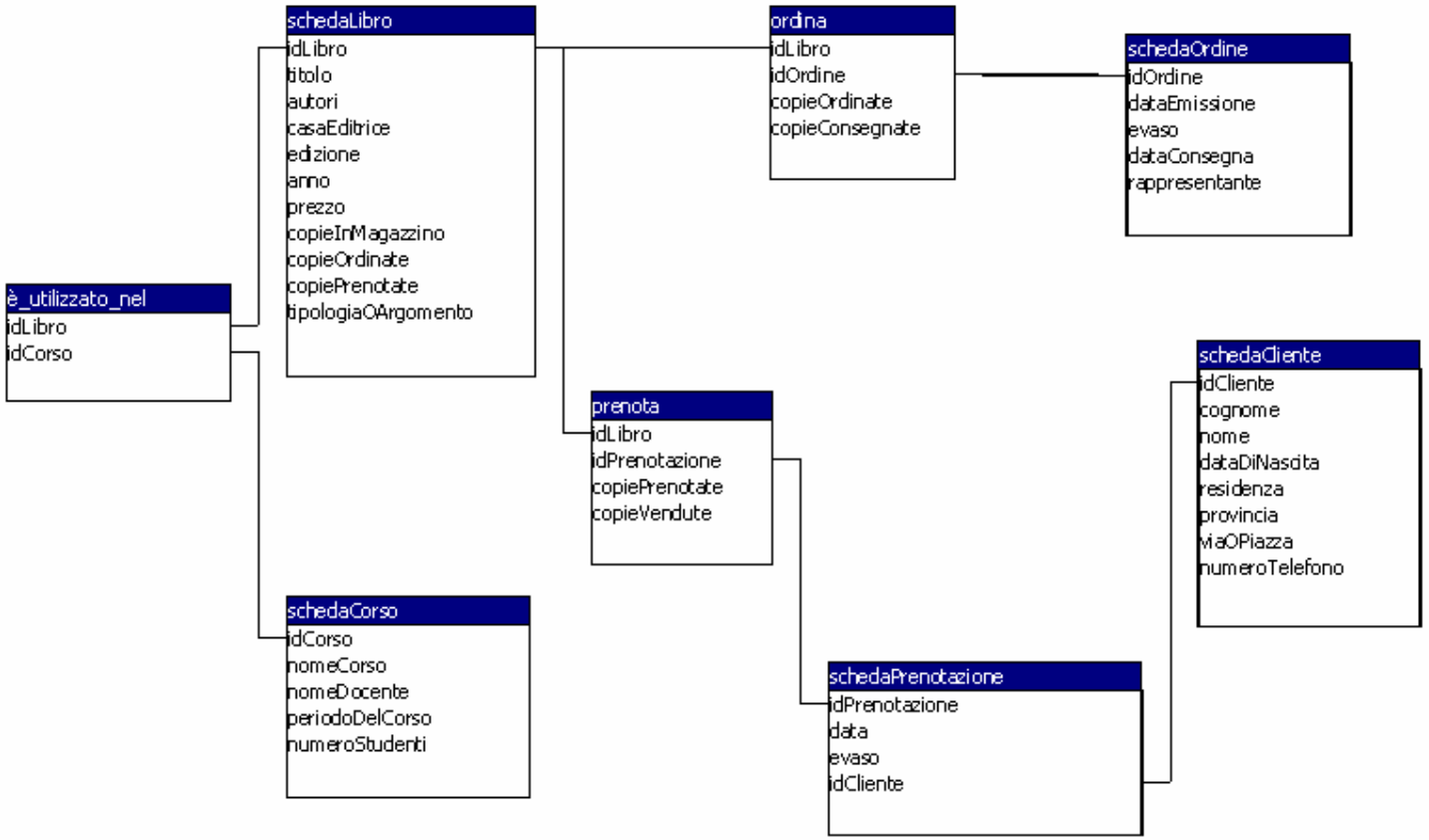
Poichè le relazioni tra le entità sono del tipo N : N, nasce l'esigenza di inserire delle tabelle che permettono di identificare univocamente le relazioni.

ordina	
Campo	**Tipo**
idLibro	intero
idOrdine	intero
copieOrdinate	intero (richiesto: si, indice: No)
copieConsegnate	intero (richiesto: si, indice: No)

prenota	
Campo	**Tipo**
idLibro	intero
idPrenotazione	intero
copiePrenotate	intero (richiesto: si, indice: No)
copieVendute	intero (richiesto: si, indice: No)

è_utilizzato_nel	
Campo	**Tipo**
idLibro	intero
idCorso	intero

In ultimo si riporta il diagramma che mostra come si legano le entità del progetto logico.



3.5 Controllo di flusso del software

Il sistema utilizza un controllo di flusso del tipo event – driver, cioè il sistema si pone in stato di attesa fino all'avvenimento di un evento esterno a cui rispondere in modo opportuno. Il sistema non presenta l'esecuzione di thread concorrenti in quanto ogni terminale ha un unico operatore che svolge il proprio lavoro in modo sequenziale.

Gli unici casi di concorrenza si hanno nell'accesso contemporaneo al database da parte dei diversi terminali. La gestione di questi accessi concorrenti risulta, comunque, di competenza del database management system (DBMS).

Documento del piano di sviluppo del progetto

1. Introduzione

Il sistema proposto si prefigge di assistere lo svolgimento dei compiti dei dipendenti di una libreria. In particolare si è tenuto conto di una sua possibile utilizzazione da parte di utenti, anche non esperti, con l'ausilio di un addestramento minimo. La distinzione fra le tre figure nasce dall'esigenza di limitare il campo di azione dell'utente preservando il sistema da perdite e/o modifiche accidentali di dati.

1.1 Documenti per il cliente

Il progetto produrrà un software che dovrà realizzare tutte le funzionalità specificate nel documento di analisi dei requisiti (RAD).

Si elenca in seguito tutta la documentazione che sarà consegnata al cliente:

- documento del piano di sviluppo del progetto (SPMP), che definisce il processo tecnico e la produzione del sistema
- documento di analisi dei requisiti (RAD), che descrive le funzionalità del sistema da realizzare. Questo documento è creato interagendo con gli esperti del dominio ed è approvato dal cliente
- documento dell'architettura software (SDD), che descrive obbiettivi, architettura, decomposizione ad alto livello e allocazione hardware/software.
- Documento del progetto esecutivo (ODD), che descrive il sistema in termini di oggetti, loro metodi e attributi.

1.2 Evoluzione della pianificazione del progetto

La pianificazione delle attività, l'allocazione delle risorse e la gestione delle previsioni sono curate grazie all'ausilio del tool Microsoft Project 98. Eventuali cambiamenti nella pianificazione saranno notificati e questo documento sarà aggiornato.

2. Organizzazione del progetto

2.1 Progetto di riferimento

Il progetto è iniziato il 16 Agosto 2002 ed è terminato il 31 Ottobre 2002.

E' stata adottata una metodologia object-oriented sviluppato tramite il CASE tool Rational Rose 2000 Enterprise Edition utilizzando UML come linguaggio di riferimento e Java come linguaggio di programmazione.

3. Strumenti utilizzati

Di seguito sono riportati gli strumenti utilizzati nella progettazione del prodotto:

- Rational Rose 2000 Enterprice Edition, utilizzato nella progettazione dei modelli del sistema (RAD), nella scomposizione del sistema (SDD) e nella definizione degli oggetti del sistema (ODD)
- Java™ 2 SDK, Standard Edition Version 1.4.0, utilizzato per implementare gli oggetti del sistema
- Forte™ for Java™ release 3.0 Enterprice Edition, tool di sviluppo per software Java
- Microsoft Project 98, utilizzato per curare la pianificazione del progetto

- Microsoft Access 2002, utilizzato per la realizzazione esemplificativa del database
- Microsoft Word 2002, utilizzato per la produzione della documentazione cartacea
- Microsoft Excel 2002, utilizzato per la creazione di schemi, tabelle e screen mock-ups di supporto alla documentazione cartacea
- Microsoft Paint 98, utilizzato per l'elaborazione e il trattamento delle immagini inserite nel documento

4. Vista complessiva sulla pianificazione del progetto

Progetto esecutivo
(Object Design Document)

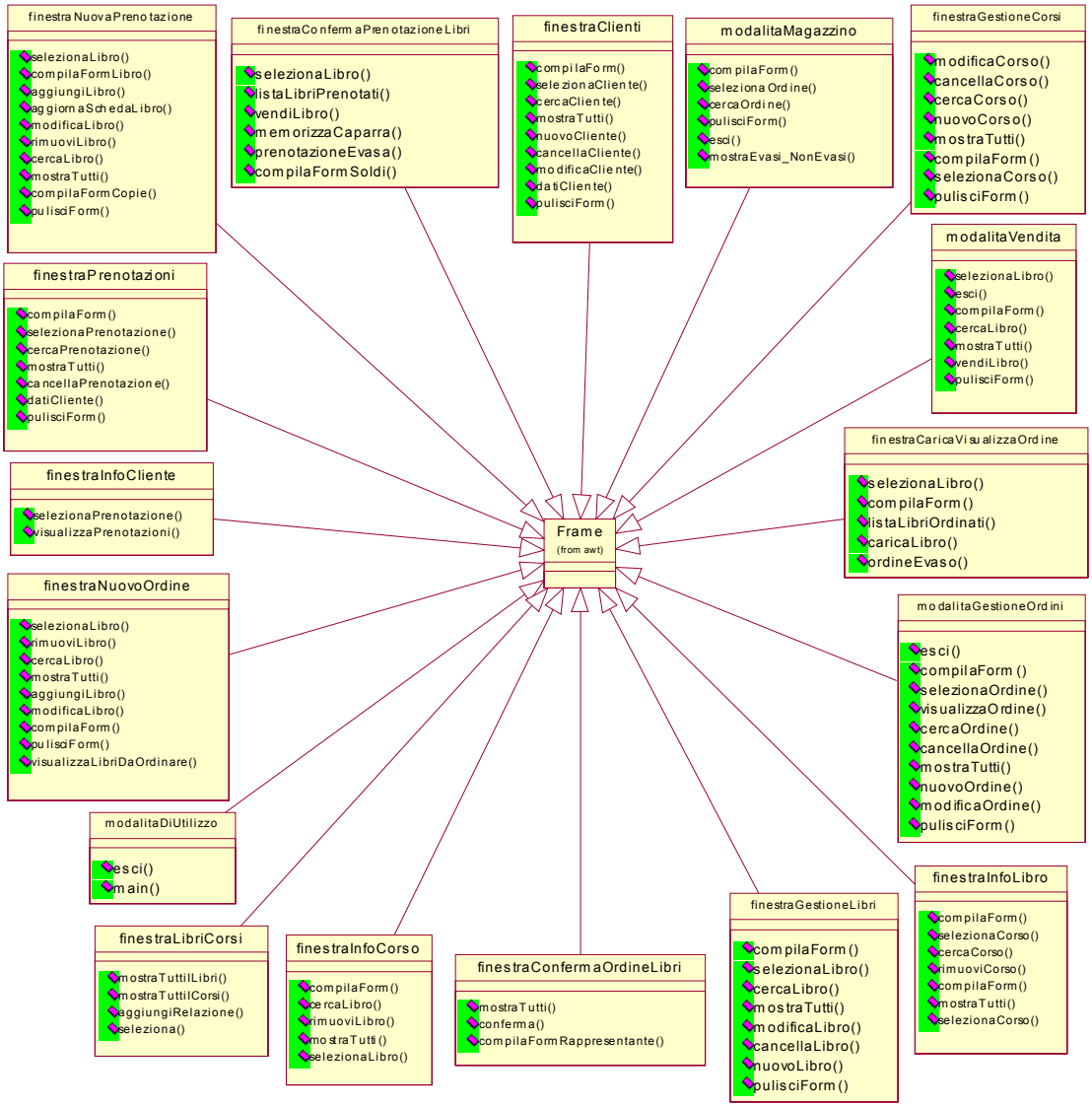
1. Struttura delle classi del sistema

Il sistema è stato decomposto in due package TerminaleUtente e ServiziUtente.

Procederemo descrivendo le classi secondo questa suddivisione.

1.1 Package TerminaleUtente

Contiene le seguenti classi:



1.1.1 Classe modalitaDiUtilizzo

Tale classe è l'unica a possedere un metodo main che permette l'esecuzione del programma. E' un'estensione della classe Frame e genera la finestra che permette la selezione della modalità di utilizzo, tra le tre possibili, tramite la pressione del relativo bottone. Da anche la possibilità all'utente di terminare l'esecuzione del programma con il relativo tasto di uscita.

Metodi:

public void **esci**()

Viene utilizzato per terminare l'esecuzione del programma. Prima della chiusura chiede conferma dell'azione.

public void **main**()

E' il metodo principale del programma che permette la sua esecuzione.

Richiama una nuova istanza della classe modalitaDiUtilizzo rendendola attiva e visibile.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;


public class modalitaDiUtilizzo extends Frame
{


   public modalitaDiUtilizzo()
   {
   }


   /**
   @roseuid 3D9031FB029E
   */
   public void esci()
   {
   }


   /**
   @roseuid 3D91C1490190
   */
   public void main()
   {
   }
}
```

1.1.2 Classe modalitaVendita

E' un'estensione della classe Frame e genera la finestra principale della modalità vendita. Permette di visualizzare, tramite una tabella, la lista di tutti i libri del database. Offre l'opportunità di cercare un particolare testo tramite query, effettuabili inserendo le chiavi di ricerca nell'opportuno form e cliccando sul tasto dedicato. Tale finestra inoltre permette di gestire la vendita dei testi e di attivare le finestre relative alla gestione dei clienti e delle prenotazioni. Da anche la possibilità all'utente di terminare l'esecuzione del programma con il relativo tasto di uscita.

Metodi:

public int **selezionaLibro**()

E' utilizzato per individuare quale riga della tabella dei libri è selezionata e ritorna il suo indice.

public void **esci**()

Viene utilizzato per terminare l'esecuzione del programma. Prima della chiusura chiede conferma dell'azione.

public SchedaLibro **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaLibro (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public void **cercaLibro**(ServiziUtente.SchedaLibro libro)

E' utilizzato per effettuare una ricerca di libri all'interno del database. E' predisposto ad accettare in ingresso un oggetto SchedaLibro (precedentemente creato tramite compilaForm()) ed a partire dai dati in esso contenuti (non necessariamente tutti) effettua la query e ne visualizza il risultato nella tabella dei libri.

public void **mostraTutti**()

Invoca il metodo cercaLibro() passandogli, come parametro SchedaLibro, dati che permettono la visualizzazione di tutti i libri memorizzati nel database.

public void **vendiLibro**()

Permette di decrementare nel database la disponibilità di un libro ogni volta che ne viene venduta una copia. Effettua inoltre un controllo sull'effettiva disponibilità del libro, cioè controlla se le copie in magazzino non risultano già prenotate e quindi momentaneamente non vendibili. Tale metodo può essere attivato tramite il bottone ad esso dedicato e solo dopo aver selezionato dalla tabella la riga del libro da decrementare.

public void **pulisciForm**(javax.swing.JTable jTable)

Viene richiamato ogni volta che un metodo di interrogazione del database ha successo. E' utilizzato per svuotare i campi della tabella di ricerca, per facilitare il lavoro dell'utente se deve effettuare un'altra operazione che utilizza la stessa tabella di interrogazione.

Codice Java generato:

```
package TerminaleUtente;


import TerminaleUtente.*;
import java.awt.Frame;
import ServiziUtente.SchedaLibro;
import javax.swing.JTable;


public class modalitaVendita extends Frame
{
   public finestraClienti theFinestraClienti;
   public finestraPrenotazioni theFinestraPrenotazioni;
   public SchedaLibro theSchedaLibro;


   public modalitaVendita()
   {
   }


   /**
   @roseuid 3D902E63026C
   */
   public int selezionaLibro()
   {
   }


   /**
   @roseuid 3D90301D01EA
   */
   public void esci()
   {
   }
```

```
/**
@roseuid 3D90369903AC
*/
public SchedaLibro compilaForm(javax.swing.JTable table)
{
}

/**
@roseuid 3D95B1ED0316
*/
public void cercaLibro(ServiziUtente.SchedaLibro libro)
{
}

/**
@roseuid 3DAC1A020078
*/
public void mostraTutti()
{
}

/**
@roseuid 3DAC1AA003CA
*/
public void vendiLibro()
{
}

/**
@roseuid 3DAC1ADB0276
*/
```

```
    public void pulisciForm(javax.swing.JTable jTable)
    {
    }
}
```

1.1.3 Classe finestraClienti

E' un'estensione della classe Frame e genera la finestra per la gestione dei clienti. Permette di visualizzare, tramite una tabella, la lista di tutti i clienti della libreria. Offre l'opportunità di cercare, inserire o modificare un cliente tramite query, effettuabili compilando opportunamente il form di ricerca e cliccando sul tasto corrispondente all'azione da intraprendere. Inoltre è possibile cancellare un cliente e inizializzare una nuova prenotazione selezionando un cliente dalla tabella e cliccando sul tasto relativo.

Metodi:

public SchedaCliente **compilaForm**(javax.swing.JTable jTable1)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaCliente (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public int **selezionaCliente**()

E' utilizzato per individuare quale riga della tabella dei clienti è selezionata e ritorna il suo indice.

public void **cercaCliente**(ServiziUtente.SchedaCliente cliente)

E' utilizzato per effettuare una ricerca di libri all'interno del database. E' predisposto ad accettare in ingresso un oggetto SchedaCliente (precedentemente

creato tramite compilaForm()) ed a partire dai dati in esso contenuti (non necessariamente tutti) effettua la query e ne visualizza il risultato nella tabella dei libri.

public void **mostraTutti**()

Invoca il metodo cercaCliente() passandogli, come parametro SchedaCliente, dati che permettono la visualizzazione di tutti i libri memorizzati nel database.

public void **nuovoCliente**(ServiziUtente.SchedaCliente cliente)

Il metodo permette di memorizzare un nuovo cliente nel database. Il cliente viene passato al metodo tramite il parametro SchedaCliente, ottenuto tramite compilaForm(). Verifica inoltre la completezza dei dati inseriti.

public void **cancellaCliente**()

Permette, previa conferma, la cancellazione dal database del cliente precedentemente selezionato nella tabella clienti.

public void **modificaCliente**(ServiziUtente.SchedaCliente nuoviDati, int riga, ServiziUtente.SchedaCliente vecchiDati)

Permette di modificare i dati relativi ad un cliente selezionato. In particolare confronta i vecchiDati (presenti nella tabella clienti) con i nuoviDati inseriti nella tabella di ricerca. Prima di effettuare la modifica nel database chiede conferma dell'operazione.

public SchedaCliente **datiCliente**(int riga)

Questo metodo ha in ingresso un intero che rappresenta l'indice di una riga della tabella clienti e ritorna un oggetto SchedaClienti dove sono memorizzati i dati estratti dalla stessa riga della tabella

public void **pulisciForm**(javax.swing.JTable jTable)

Viene richiamato ogni volta che un metodo di interrogazione del database ha successo. E' utilizzato per svuotare i campi della tabella di ricerca, per facilitare il lavoro dell'utente se deve effettuare un'altra operazione che utilizza la stessa tabella di interrogazione.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;
import ServiziUtente.SchedaCliente;
import javax.swing.JTable;


public class finestraClienti extends Frame
{
   public finestraInfoCliente theFinestraInfoCliente;
   public SchedaCliente theSchedaCliente;


   public finestraClienti()
   {
   }


   /**
   @roseuid 3D901A1002E4
   */
   public SchedaCliente compilaForm(javax.swing.JTable jTable1)
   {
   }


   /**
   @roseuid 3D901A9402DA
   */
   public int selezionaCliente()
   {
   }


   /**
```

```
    @roseuid 3DAC1DDC01E0
    */
    public void cercaCliente(ServiziUtente.SchedaCliente cliente)
    {
    }

    /**
    @roseuid 3DAC1E010262
    */
    public void mostraTutti()
    {
    }

    /**
    @roseuid 3DAC1E160028
    */
    public void nuovoCliente(ServiziUtente.SchedaCliente cliente)
    {
    }

    /**
    @roseuid 3DAC1E310230
    */
    public void cancellaCliente()
    {
    }

    /**
    @roseuid 3DAC1E5300AA
    */
    public  void  modificaCliente(ServiziUtente.SchedaCliente  nuoviDati,
int riga, ServiziUtente.SchedaCliente vecchiDati)
```

```
    {
    }

    /**
    @roseuid 3DAC1E9C02D0
    */
    public SchedaCliente datiCliente(int riga)
    {
    }

    /**
    @roseuid 3DAC1ED4019A
    */
    public void pulisciForm(javax.swing.JTable jTable)
    {
    }
}
```

1.1.4 Classe finestraInfoCliente

E' un'estensione della classe Frame e genera la finestra che permette, tramite una tabella, la visualizzazione delle prenotazioni di un cliente. Viene richiamata passandogli un oggetto di tipo SchedaCliente che diventa il soggetto delle operazioni della finestra. Permette la visualizzazione o la modifica delle prenotazioni del cliente, richiamando le opportune finestre .

Metodi:

public int **selezionaPrenotazione**()

E' utilizzato per individuare quale riga della tabella delle prenotazioni del cliente è selezionata e ritorna il suo indice.

public void **visualizzaPrenotazioni**(int idCliente)

Permette la visualizzazione, nella tabella della finestra, di tutte le prenotazioni del cliente. L'idCliente, che è l'intero che passiamo come parametro, è la chiave che permette la ricerca delle prenotazioni del cliente.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;
import ServiziUtente.SchedaCliente;
import ServiziUtente.SchedaPrenotazione;


public class finestraInfoCliente extends Frame
{
   public SchedaCliente theSchedaCliente;
   public SchedaPrenotazione theSchedaPrenotazione;


   public finestraInfoCliente()
   {
   }


   /**
   @roseuid 3D902B8B012C
   */
   public int selezionaPrenotazione()
   {
   }


   /**
   @roseuid 3DAC1CEA035C
   */
   public void visualizzaPrenotazioni(int idCliente)
   {
   }
}
```

1.1.5 Classe finestraPrenotazioni

E' un'estensione della classe Frame e genera la finestra per la gestione delle prenotazioni. Permette di visualizzare, tramite una tabella, la lista di tutte le prenotazioni della libreria. Offre l'opportunità di cercare, modificare o cancellare una prenotazione tramite query, effettuabili compilando opportunamente il form di ricerca e cliccando sul tasto corrispondente all'azione da intraprendere.

Metodi:

public SchedaPrenotazione **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaPrenotazione (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public int **selezionaPrenotazione**()

E' utilizzato per individuare quale riga della tabella delle prenotazioni del cliente è selezionata e ritorna il suo indice.

public void **cercaPrenotazione**(ServiziUtente.SchedaPrenotazione prenotazione)

E' utilizzato per cercare prenotazioni all'interno del database. E' predisposto ad accettare in ingresso un oggetto SchedaPrenotazione (precedentemente creato

tramite compilaForm()) ed a partire dai dati in esso contenuti (non necessariamente tutti) effettua la query e ne visualizza il risultato nella tabella delle prenotazioni.

public void **mostraTutti**()

Invoca il metodo cercaPrenotazione() passandogli, come parametro SchedaPrenotazione, dati che permettono la visualizzazione di tutte le prenotazioni memorizzate nel database.

public void **cancellaPrenotazione**(int idPrenotazione)

Permette, previa conferma, la cancellazione dal database della prenotazione passata come parametro tramite il suo id.

public SchedaCliente **datiCliente**(int riga)

Questo metodo ha in ingresso un intero che rappresenta l'indice di una riga della tabella prenotazioni e ritorna un oggetto SchedaClienti che contiene i dati del cliente relativo alla prenotazione. Tali dati sono ottenuti tramite una query usando come chiave di ricerca l'idCliente che è contenuto nella seconda colonna della tabella.

public void **pulisciForm**(javax.swing.JTable jTable)

Viene richiamato ogni volta che un metodo di interrogazione del database ha successo. E' utilizzato per svuotare i campi della tabella di ricerca, per facilitare il lavoro dell'utente se deve effettuare un'altra operazione che utilizza la stessa tabella di interrogazione.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;
import ServiziUtente.SchedaPrenotazione;
import javax.swing.JTable;
import ServiziUtente.SchedaCliente;


public class finestraPrenotazioni extends Frame
{
   public finestraNuovaPrenotazione theFinestraNuovaPrenotazione;
   public SchedaPrenotazione theSchedaPrenotazione;


   public finestraPrenotazioni()
   {
   }


   /**
   @roseuid 3D9022DB021C
   */
   public SchedaPrenotazione compilaForm(javax.swing.JTable table)
   {
   }


   /**
   @roseuid 3D90238502B2
   */
   public int selezionaPrenotazione()
   {
   }
```

```
    /**
    @roseuid 3DAC1B6D01FE
    */
    public void cercaPrenotazione(ServiziUtente.SchedaPrenotazione
prenotazione)
    {
    }

    /**
    @roseuid 3DAC1B970280
    */
    public void mostraTutti()
    {
    }

    /**
    @roseuid 3DAC1BA901AE
    */
    public void cancellaPrenotazione(int idPrenotazione)
    {
    }

    /**
    @roseuid 3DAC1C090262
    */
    public SchedaCliente datiCliente(int riga)
    {
    }

    /**
    @roseuid 3DAC1C37038E
    */
```

```
    public void pulisciForm(javax.swing.JTable jTable)
    {
    }
}
```

1.1.6 Classe finestraNuovaPrenotazione

E' un'estensione della classe Frame e genera la finestra che permette all'utente di selezionare quali libri inserire in una prenotazione e in quale quantità. Questa finestra viene richiamata sia se si genera una nuova prenotazione, sia se si vuole modificare una prenotazione già esistente. Il suo costruttore richiede come parametri la SchedaCliente corrispondente al cliente che effettua la prenotazione e l'idPrenotazione. Permette di visualizzare, tramite una tabella, la lista di tutti i libri. Offre l'opportunità di cercare un libro, aggiungere, modificare o rimuovere le copie di un libro dalla prenotazione tramite query, effettuabili compilando opportunamente il form di ricerca e cliccando sul tasto corrispondente all'azione da intraprendere.

Metodi:

public int **selezionaLibro**()

E' utilizzato per individuare quale riga della tabella dei clienti è selezionata e ritorna il suo indice.

Public SchedaLibro **compilaFormLibro**(javax.swing.Jtable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaLibro (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

Public void **aggiungiLibro**(int idLibro, int idPrenotazione, int copiePrenotate)

Permette di aggiungere un libro, passato al metodo tramite il parametro idLibro, alla prenotazione idPrenotazione nel numero di copie specificato dal parametro copiePrenotate. Tale metodo aggiorna il database sia nel caso in cui il libro non appartiene alla prenotazione (nuovo inserimento), sia nel caso in cui deve incrementare il numero di copie.

public void **aggiornaSchedaLibro**(int idLibro, int copiePrenotate)

Viene richiamato ogni qual volta vi è la necessità di aggiornare la quantità di copie prenotate di un particolare libro specificato dal parametro idLibro. Viene invocato sia per aumentare che per diminuire il numero di prenotazioni per un dato libro passandogli come copiePrenotate rispettivamente un numero positivo o negativo.

public void **modificaLibro**(int idLibro, int idPrenotazione, int copiePrenotate)

Serve per sostituire il numero di copie prenotate del libro idLibro appartenente alla prenotazione idPrenotazione con il nuovo numero copiePrenotate.

public void **rimuoviLibro**(int idLibro, int idPrenotazione)

Rimuove l'appartenenza del libro idLibro dalla prenotazione idPrenotazione.

public void **cercaLibro**(ServiziUtente.SchedaLibro libro, int idPrenotazione)

E' utilizzato per effettuare una ricerca di libri all'interno del database. E' predisposto ad accettare in ingresso un oggetto SchedaLibro (precedentemente creato tramite compilaForm()) e l'idPrenotazione della prenotazione. Effettua la query e ne visualizza il risultato nella tabella dei libri.

public void **mostraTutti**(int idPrenotazione)

Invoca il metodo cercaLibro() passandogli, come parametro SchedaLibro, dati che permettono la visualizzazione di tutti i libri memorizzati nel database.

public int **compilaFormCopie**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica cella cui contiene il valore del numero di copie da inserire. Tale tabella viene passata come parametro al metodo che ritorna tipo int .

public void **pulisciForm**(javax.swing.JTable jTable)

Viene richiamato ogni volta che un metodo di interrogazione del database ha successo. E' utilizzato per svuotare i campi della tabella di ricerca, per facilitare il lavoro dell'utente se deve effettuare un'altra operazione che utilizza la stessa tabella di interrogazione.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;
import ServiziUtente.SchedaLibro;
import ServiziUtente.SchedaCliente;
import javax.swing.JTable;


public class finestraNuovaPrenotazione extends Frame
{
   public finestraConfermaPrenotazioneLibri
theFinestraConfermaPrenotazioneLibri;
   public SchedaLibro theSchedaLibro;
   public SchedaCliente theSchedaCliente;


   public finestraNuovaPrenotazione()
   {
   }


   /**
   @roseuid 3D901B1C0122
   */
   public int selezionaLibro()
   {
   }


   /**
   @roseuid 3D906B2D0334
   */
   public SchedaLibro compilaFormLibro(javax.swing.JTable table)
   {
```

```
    }

    /**
    @roseuid 3DAC22E402EE
    */
    public void aggiungiLibro(int idLibro, int idPrenotazione, int
copiePrenotate)
    {
    }

    /**
    @roseuid 3DAC230801C2
    */
    public void aggiornaSchedaLibro(int idLibro, int copiePrenotate)
    {
    }

    /**
    @roseuid 3DAC231C030C
    */
    public void modificaLibro(int idLibro, int idPrenotazione, int
copiePrenotate)
    {
    }

    /**
    @roseuid 3DAC2339003C
    */
    public void rimuoviLibro(int idLibro, int idPrenotazione)
    {
    }
```

```
    /**
    @roseuid 3DAC235702C6
    */
    public void cercaLibro(ServiziUtente.SchedaLibro libro, int
idPrenotazione)
    {
    }

    /**
    @roseuid 3DAC237C015E
    */
    public void mostraTutti(int idPrenotazione)
    {
    }

    /**
    @roseuid 3DAC23D0017C
    */
    public int compilaFormCopie(javax.swing.JTable table)
    {
    }

    /**
    @roseuid 3DAC23FF015E
    */
    public void pulisciForm(javax.swing.JTable jTable)
    {
    }
}
```

1.1.7 Classe finestraConfermaPrenotazioneLibri

E' un'estensione della classe Frame e genera la finestra che permette all'utente di visualizzare i soli libri appartenenti a una data prenotazione. Il suo costruttore richiede come parametri la SchedaPrenotazione corrispondente alla prenotazione da visualizzare. Offre l'opportunità di inserire l'acconto e uscire dalla finestra oppure, di cancellare la prenotazione o ancora di modificarla richiamando il costruttore della classe finestraNuovaPrenotazione passandogli come parametro la prenotazione visualizzata.

Metodi:

public int **selezionaLibro**()

E' utilizzato per individuare quale riga della tabella dei libri è selezionata e ritorna il suo indice.

public void **listaLibriPrenotati**(int idPrenotazione)

Permette la visualizzazione, nella tabella della finestra, di tutti i libri della prenotazione passatagli come parametro.

public void **vendiLibro**(int riga, int idPrenotazione)

Dopo aver effettuato un controllo sull'effettiva disponibilità del libro in magazzino, permette di decrementare nel database della prenotazione idPrenotazione il numero di copie del libro indicato dalla parametro riga ed inoltre decrementa anche la sua disponibilità in magazzino. Tale metodo può essere attivato tramite il bottone ad

esso dedicato e solo dopo aver selezionato dalla tabella la riga del libro da decrementare.

public void **memorizzaCaparra**(int idPrenotazione, float caparra)

Memorizza nell'opportuno campo di schedaPrenotazione, identificata tramite idPrenotazione, l'importo lasciato come acconto dal cliente passato con il parametro caparra.

public void **prenotazioneEvasa**(int idPrenotazione)

Richiamato quando tutti i libri della prenotazione idPrenotazione sono stati venduti, pone il campo evasa di schedaPrenotazione a true.

public float **compilaFormSoldi**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica cella cui contiene l'importo lasciato in acconto alla prenotazione dal cliente. Tale tabella viene passata come parametro al metodo che ritorna tipo float.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;
import javax.swing.JTable;


public class finestraConfermaPrenotazioneLibri extends Frame
{


   public finestraConfermaPrenotazioneLibri()
   {
   }


   /**
   @roseuid 3D902BB20050
   */
   public int selezionaLibro()
   {
   }


   /**
   @roseuid 3DAC205C012C
   */
   public void listaLibriPrenotati(int idPrenotazione)
   {
   }


   /**
   @roseuid 3DAC20B60302
   */
   public void vendiLibro(int riga, int idPrenotazione)
```

```
    {
    }

    /**
    @roseuid 3DAC20CD037A
    */
    public void memorizzaCaparra(int idPrenotazione, float caparra)
    {
    }

    /**
    @roseuid 3DAC20E702BC
    */
    public void prenotazioneEvasa(int idPrenotazione)
    {
    }

    /**
    @roseuid 3DAC210F03C0
    */
    public float compilaFormSoldi(javax.swing.JTable table)
    {
    }
}
```

1.1.8 Classe modalitaGestioneOrdini

E' un'estensione della classe Frame e genera la finestra principale della modalità gestione ordini. Permette di visualizzare, tramite una tabella, la lista di tutti gli ordini del database. Offre l'opportunità di cercare un particolare ordine tramite query, effettuabili inserendo le chiavi di ricerca nell'opportuno form e cliccando sul tasto dedicato. Tale finestra permette di avviare la classe finestraNuovoOrdine sia per la creazione di un nuovo ordine sia per la modifica di uno gia esistente. Inoltre permette di visualizzare nel dettaglio un ordine richiamando la classe finestraConfermaOrdineLibri, da l'opportunità di cancellare un ordine esistente e permette all'utente di terminare l'esecuzione del programma con il relativo tasto di uscita.

Metodi:

public void **esci**()

Viene utilizzato per terminare l'esecuzione del programma. Prima della chiusura chiede conferma dell'azione.

public SchedaOrdine **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaOrdine (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public int **selezionaOrdine**()

E' utilizzato per individuare quale riga della tabella degli ordini è selezionata e ritorna il suo indice.

public void **visualizzaOrdine**()

Richiamare il costruttore della classe finestraConfermaOrdineLibri passando come parametro l'ordine selezionato nella tabella ordini della finestra. Controlla inoltre se un nella tabella vi è un ordine selezionato altrimenti avverte l'utente con un opportuno messaggio.

public void **cercaOrdine**(ServiziUtente.SchedaOrdine ordine)

E' utilizzato per cercare ordini all'interno del database. E' predisposto ad accettare in ingresso un oggetto SchedaOrdine (precedentemente creato tramite compilaForm()) ed a partire dai dati in esso contenuti (non necessariamente tutti) effettua la query e ne visualizza il risultato nella tabella delle prenotazioni.

public void **cancellaOrdine**(int id)

Permette, previa conferma, la cancellazione dal database dell'ordine passata come parametro tramite il suo id.

public void **mostraTutti**()

Invoca il metodo cercaOrdine() passandogli, tramite il parametro SchedaOrdine, dati che permettono la visualizzazione di tutte gli ordini memorizzati nel database.

public void **nuovoOrdine**()

Il metodo permette di memorizzare un nuovo ordine nel database. La classe genera automaticamente i dati iniziali di un oggetto SchedaOrdine settando l'attributo dataEmissione alla data in cui viene inizializzato l'ordine e l'attributo evaso a no. Richiama infine il metodo costruttore della finestraNuovoOrdine passando come parametro il nuovo ordine.

public void **modificaOrdine**()

Permette di modificare i dati relativi all'ordine selezionato richiamando il metodo costruttore della finestraNuovoOrdine passando come parametro l'ordine da modificare.

public void **pulisciForm**(javax.swing.JTable jTable)

Viene richiamato ogni volta che un metodo di interrogazione del database ha successo. E' utilizzato per svuotare i campi della tabella di ricerca, per facilitare il lavoro dell'utente se deve effettuare un'altra operazione che utilizza la stessa tabella di interrogazione.

Codice Java generato:

```
package TerminaleUtente;


import TerminaleUtente.*;
import java.awt.Frame;
import ServiziUtente.SchedaOrdine;
import javax.swing.JTable;


public class modalitaGestioneOrdini extends Frame
{
   public finestraGestioneLibri theFinestraGestioneLibri;
   public finestraGestioneCorsi theFinestraGestioneCorsi;
   public finestraNuovoOrdine theFinestraNuovoOrdine;
   public finestraConfermaOrdineLibri theFinestraConfermaOrdineLibri;
   public SchedaOrdine theSchedaOrdine;


   public modalitaGestioneOrdini()
   {
   }


   /**
   @roseuid 3D9031090190
   */
   public void esci()
   {
   }


   /**
   @roseuid 3D9063440118
   */
   public SchedaOrdine compilaForm(javax.swing.JTable table)
```

```
{
}

/**
@roseuid 3D9067B802BC
*/
public int selezionaOrdine()
{
}

/**
@roseuid 3DAC16BE0258
*/
public void visualizzaOrdine()
{
}

/**
@roseuid 3DAC16D502D0
*/
public void cercaOrdine(ServiziUtente.SchedaOrdine ordine)
{
}

/**
@roseuid 3DAC171101E0
*/
public void cancellaOrdine(int id)
{
}

/**
```

```
    @roseuid 3DAC17C9032A
    */
    public void mostraTutti()
    {
    }

    /**
    @roseuid 3DAC17DC0398
    */
    public void nuovoOrdine()
    {
    }

    /**
    @roseuid 3DAC17F10352
    */
    public void modificaOrdine()
    {
    }

    /**
    @roseuid 3DAC1846017C
    */
    public void pulisciForm(javax.swing.JTable jTable)
    {
    }
}
```

1.1.9 Classe finestraNuovoOrdine

E' un'estensione della classe Frame e genera la finestra che permette all'utente di selezionare quali libri inserire in una data prenotazione specificandone il numero di copie. Questa finestra viene richiamata sia che si generi un nuovo ordine, sia che si voglia modificare un ordine già esistente. Il suo costruttore richiede come parametro SchedaOrdine che corrisponde all'ordine che si vuole generare. Permette di visualizzare, tramite una tabella, la lista di tutti i libri o quella dei soli libri di cui vi è maggiore urgenza di ordine (libri prenotati da clienti, ma non ancora ordinati). Offre l'opportunità di cercare un libro, aggiungere, modificare o rimuovere le copie di un libro dall'ordine tramite query, effettuabili compilando opportunamente il form di ricerca e cliccando sul tasto corrispondente all'azione da intraprendere.

Metodi:

public int **selezionaLibro**()

E' utilizzato per individuare quale riga della tabella dei libri è selezionata e ritorna il suo indice.

public void **rimuoviLibro**()

Rimuove il libro selezionato nella tabella dall'ordine identificato nella finestra.

public void **cercaLibro**(ServiziUtente.SchedaLibro libro)

E' utilizzato per effettuare una ricerca di libri all'interno del database. E' predisposto ad accettare in ingresso un oggetto SchedaLibro (precedentemente creato tramite compilaForm()) ed a partire dai dati in esso contenuti (non necessariamente tutti) effettua la query e ne visualizza il risultato nella tabella dei libri.

public void **mostraTutti**()

Invoca il metodo cercaLibro() passandogli, come parametro SchedaLibro, dati che permettono la visualizzazione di tutti i libri memorizzati nel database.

public void **aggiungiLibro**()

Permette di aggiungere un libro, selezionato dalla tabella, all'ordine della finestra specificandone nel form dedicato il numero di copie da ordinare. Tale metodo aggiorna il database sia nel caso in cui il libro non appartiene all'ordine (nuovo inserimento), sia nel caso in cui deve incrementare il numero di copie.

public void **modificaLibro**(ServiziUtente.SchedaOrdine ordine, int id,
java.lang.Integer copie_nuove)

Serve per sostituire il numero di copie ordinate del libro id appartenente all'ordine passato come parametro con il nuovo numero copie_nuove.

public SchedaLibro **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un

oggetto di tipo SchedaLibro (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public void **pulisciForm**(javax.swing.JTable jTable)

Viene richiamato ogni volta che un metodo di interrogazione del database ha successo. E' utilizzato per svuotare i campi della tabella di ricerca, per facilitare il lavoro dell'utente se deve effettuare un'altra operazione che utilizza la stessa tabella di interrogazione.

public void **visualizzaLibriDaOrdinare**()

E' utilizzato per mostrare all'utente quali libri necessitano di essere ordinati con maggiore urgenza essendo stati prenotati ma non essendo stati ancora ordinati in quantità tale da soddisfare le richieste. L'esito di tale metodo è la visualizzazione nella tabella dei libri dei soli testi che soddisfano tale requisito.

Codice Java generato:

```
package TerminaleUtente;

import TerminaleUtente.*;
import java.awt.Frame;
import ServiziUtente.SchedaLibro;
import ServiziUtente.SchedaOrdine;
import javax.swing.JTable;

public class finestraNuovoOrdine extends Frame
{
   public finestraConfermaOrdineLibri theFinestraConfermaOrdineLibri;
   public SchedaLibro theSchedaLibro;
   public SchedaOrdine theSchedaOrdine;

   public finestraNuovoOrdine()
   {
   }

   /**
   @roseuid 3D90659C033E
   */
   public int selezionaLibro()
   {
   }

   /**
   @roseuid 3DAC15450212
   */
   public void rimuoviLibro()
   {
```

```
    }

    /**
    @roseuid 3DAC156A003C
    */
    public void cercaLibro(ServiziUtente.SchedaLibro libro)
    {
    }

    /**
    @roseuid 3DAC1585008C
    */
    public void mostraTutti()
    {
    }

    /**
    @roseuid 3DAC159C021C
    */
    public void aggiungiLibro()
    {
    }

    /**
    @roseuid 3DAC15D001B8
    */
    public void modificaLibro(ServiziUtente.SchedaOrdine ordine, int id,
java.lang.Integer copie_nuove)
    {
    }

    /**
```

```
    @roseuid 3DAC160302E4
    */
    public SchedaLibro compilaForm(javax.swing.JTable table)
    {
    }

    /**
    @roseuid 3DAC164B005A
    */
    public void pulisciForm(javax.swing.JTable jTable)
    {
    }

    /**
    @roseuid 3DB1037601FE
    */
    public void visualizzaLibriDaOrdinare()
    {
    }
}
```

1.1.10 Classe finestraConfermaOrdineLibri

E' un'estensione della classe Frame e genera la finestra che permette all'utente di visualizzare i soli libri appartenenti a un dato ordine. Il suo costruttore richiede come parametri la SchedaOrdine corrispondente all'ordine da visualizzare. Offre l'opportunità di inserire il nome del rappresentante cui inoltrare l'ordine, di uscire dalla finestra oppure, di cancellare l'ordine o ancora di modificarlo richiamando il costruttore della classe finestraNuovaOrdineLibri passandogli come parametro l'ordine visualizzato.

Metodi:

public void **mostraTutti**(ServiziUtente.SchedaOrdine ordine)

Permette la visualizzazione, nella tabella della finestra, di tutti i libri dell'ordine passatogli come parametro.

public void **conferma**()

Termina la procedura di creazione o modifica dell'ordine verificando l'inserimento del nome del rappresentante, altrimenti segnalato con un relativo messaggio di errore, e memorizzandolo nell'opportuno campo di SchedaOrdine. Infine richiama il costruttore della classe modalitaGestioneOrdini

public String **compilaFormRappresentante**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica cella cui contiene il nome del rappresentante da inserire. Tale tabella viene passata come parametro al metodo che ritorna tipo string.

Codice Java generato:

```
package TerminaleUtente;


import TerminaleUtente.*;
import java.awt.Frame;
import ServiziUtente.SchedaOrdine;
import javax.swing.JTable;


public class finestraConfermaOrdineLibri extends Frame
{
   public SchedaOrdine theSchedaOrdine;


   public finestraConfermaOrdineLibri()
   {
   }


   /**
   @roseuid 3DAC13D0024E
   */
   public void mostraTutti(ServiziUtente.SchedaOrdine ordine)
   {
   }


   /**
   @roseuid 3DB102C70384
   */
   public void conferma()
   {
   }


   /**
```

```
    @roseuid 3DB65A990050
    */
    public String compilaFormRappresentante(javax.swing.JTable table)
    {
    }
}
```

1.1.11 Classe finestraGestioneLibri

E' un'estensione della classe Frame e genera la finestra per la gestione dei libri. Permette di visualizzare, tramite una tabella, la lista di tutti i libri del database. Offre l'opportunità di cercare, inserire o modificare un libro tramite query, effettuabili compilando opportunamente il form di ricerca e cliccando sul tasto corrispondente all'azione da intraprendere. Inoltre è possibile richiamare la finestra che mostra tutti i corsi associati a un dato libro.

Metodi:

public SchedaLibro **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaLibro (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public int **selezionaLibro**()

E' utilizzato per individuare quale riga della tabella dei libri è selezionata e ritorna il suo indice.

public void **cercaLibro**(ServiziUtente.SchedaLibro libro)

E' utilizzato per effettuare una ricerca di libri all'interno del database. E' predisposto ad accettare in ingresso un oggetto SchedaLibro (precedentemente creato tramite

compilaForm()) ed a partire dai dati in esso contenuti (non necessariamente tutti) effettua la query e ne visualizza il risultato nella tabella dei libri.

public void **mostraTutti**()

Invoca il metodo cercaLibro() passandogli, come parametro SchedaLibro, dati che permettono la visualizzazione di tutti i libri memorizzati nel database.

public void **modificaLibro**(ServiziUtente.SchedaLibro nuoviDati, int riga, ServiziUtente.SchedaLibro vecchiDati)

Permette di modificare i dati relativi ad un libro selezionato. In particolare confronta i vecchiDati (presenti nella tabella libri) con i nuoviDati inseriti nella tabella di ricerca. Prima di effettuare la modifica nel database chiede conferma dell'operazione.

public void **cancellaLibro**()

Permette, previa conferma, la cancellazione dal database del libro precedentemente selezionato nella tabella dei libri.

public void **nuovoLibro**()

Il metodo permette di memorizzare un nuovo libro nel database. I dati relativi al libro vengono acquisiti tramite il metodo compilaForm() e verifica inoltre la completezza dei dati inseriti.

public void **pulisciForm**(javax.swing.JTable jTable)

Viene richiamato ogni volta che un metodo di interrogazione del database ha successo. E' utilizzato per svuotare i campi della tabella di ricerca, per facilitare il

lavoro dell’utente se deve effettuare un'altra operazione che utilizza la stessa tabella di interrogazione.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;
import ServiziUtente.SchedaLibro;
import javax.swing.JTable;


public class finestraGestioneLibri extends Frame
{
   public finestraInfoLibro theFinestraInfoLibro;
   public SchedaLibro theSchedaLibro;


   public finestraGestioneLibri()
   {
   }


   /**
   @roseuid 3D90371D02C6
   */
   public SchedaLibro compilaForm(javax.swing.JTable table)
   {
   }


   /**
   @roseuid 3D906A2801AE
   */
   public int selezionaLibro()
   {
   }


   /**
```

```
    @roseuid 3D95B26503C0
    */
    public void cercaLibro(ServiziUtente.SchedaLibro libro)
    {
    }

    /**
    @roseuid 3DABE39A03AC
    */
    public void mostraTutti()
    {
    }

    /**
    @roseuid 3DAC18A400FA
    */
    public void modificaLibro(ServiziUtente.SchedaLibro nuoviDati, int
riga, ServiziUtente.SchedaLibro vecchiDati)
    {
    }

    /**
    @roseuid 3DAC18F90032
    */
    public void cancellaLibro()
    {
    }

    /**
    @roseuid 3DAC192B005A
    */
    public void nuovoLibro()
```

```
    {
    }

    /**
    @roseuid 3DAC19820280
    */
    public void pulisciForm(javax.swing.JTable jTable)
    {
    }
}
```

1.1.12 Classe finestraInfoLibro

E' un'estensione della classe Frame e genera la finestra che permette, tramite una tabella, la visualizzazione dei corsi associati ad un libro. Viene richiamata passandogli un oggetto di tipo SchedaLibro che diventa il soggetto delle operazioni della finestra. Permette di rimuovere un corso o di aggiungerlo richiamando la finestraLibriCorsi.

Metodi:

public SchedaCorso **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaCorso (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public void **selezionaCorso**()

E' utilizzato per individuare quale riga della tabella dei corsi è selezionata e ritorna il suo indice.

public void **cercaCorso**(ServiziUtente.SchedaCorso corso)

E' utilizzato per effettuare una ricerca di Corsi all'interno del database che hanno una relazione con il libro soggetto della finestra. E' predisposto ad accettare in ingresso un oggetto SchedaCorso (precedentemente creato tramite compilaForm())

ed a partire dai dati in esso contenuti (non necessariamente tutti) effettua la query e ne visualizza il risultato nella tabella dei libri.

public void **rimuoviCorso**()

Permette la rimozione della relazione tra il libro rappresentato nella finestra e il corso selezionato nella tabella.

public SchedaCorso **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaCorso (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public void **mostraTutti**()

Invoca il metodo cercaCorso() passandogli, come parametro SchedaCorso, dati che permettono la visualizzazione di tutti i corsi memorizzati nel database e associati al libro soggetto della finestra.

public int **selezionaCorso**()

E' utilizzato per individuare quale riga della tabella dei libri è selezionata e ritorna il suo indice.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;
import ServiziUtente.SchedaLibro;
import ServiziUtente.SchedaCorso;
import javax.swing.JTable;


public class finestraInfoLibro extends Frame
{
   public finestraLibriCorsi theFinestraLibriCorsi;
   public SchedaLibro theSchedaLibro;
   public SchedaCorso theSchedaCorso;


   public finestraInfoLibro()
   {
   }


   /**
   @roseuid 3D906AE4006E
   */
   public void compilaForm()
   {
   }


   /**
   @roseuid 3D906E0F01A4
   */
   public void selezionaCorso()
   {
   }
```

```
/**
@roseuid 3DAC10500398
*/
public void cercaCorso(ServiziUtente.SchedaCorso corso)
{
}

/**
@roseuid 3DAC10BA01EA
*/
public void rimuoviCorso()
{
}

/**
@roseuid 3DAC10DF0348
*/
public SchedaCorso compilaForm(javax.swing.JTable table)
{
}

/**
@roseuid 3DAC113A02E4
*/
public void mostraTutti()
{
}

/**
@roseuid 3DAC11560096
*/
```

```
    public int selezionaCorso()
    {
    }
}
```

1.1.13 Classe finestraLibriCorsi

E' un'estensione della classe Frame e genera la finestra che permette, tramite due tabelle, di associare un libro a un corso e viceversa. Permette di richiamare finestraGestioneCorsi e FinestraGestioneLibri.

Metodi:

public void **mostraTuttiILibri**()

Permette di visualizzare, nella seconda tabella della finestra, la lista di tutti i libri memorizzati nel database.

public void **mostraTuttiICorsi**()

Permette di visualizzare, nella prima tabella della finestra, la lista di tutti i corsi memorizzati nel database.

public void **aggiungiRelazione**()

Acquisisce le righe selezionate nelle due tabelle e, dopo aver chiesto conferma dell'operazione, memorizza la relazione libro corso se non ancora presente nel database.

public int **seleziona**(javax.swing.JTable table)

E' utilizzato per individuare quale riga della tabella table è selezionata e ritorna il suo indice. Viene richiamata da aggiungiRelazione() passandogli sia la tabella dei corsi che quella dei libri.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;
import javax.swing.JTable;


public class finestraLibriCorsi extends Frame
{


   public finestraLibriCorsi()
   {
   }


   /**
   @roseuid 3DAC28EB00AA
   */
   public void mostraTuttiILibri()
   {
   }


   /**
   @roseuid 3DAC28FD015E
   */
   public void mostraTuttiICorsi()
   {
   }


   /**
   @roseuid 3DAC2919038E
   */
   public void aggiungiRelazione()
```

```
    {
    }

    /**
    @roseuid 3DAC292C005A
    */
    public int seleziona(javax.swing.JTable table)
    {
    }
}
```

1.1.14 Classe finestraGestioneCorsi

E' un'estensione della classe Frame e genera la finestra per la gestione dei corsi. Permette di visualizzare, tramite una tabella, la lista di tutti i corsi del database. Offre l'opportunità di cercare, inserire o modificare un corso tramite query, effettuabili compilando opportunamente il form di ricerca e cliccando sul tasto corrispondente all'azione da intraprendere. Inoltre è possibile richiamare la finestra che mostra tutti i libri associati a un dato corso.

Metodi:

public void **modificaCorso**(ServiziUtente.SchedaCorso nuoviDati, int riga, ServiziUtente.SchedaCorso vecchiDati)

Permette di modificare i dati relativi ad un corso selezionato. In particolare confronta i vecchiDati (presenti nella tabella corsi) con i nuoviDati inseriti nella tabella di ricerca. Prima di effettuare la modifica nel database chiede conferma dell'operazione.

public void **cancellaCorso**()

Permette, previa conferma, la cancellazione dal database del corso precedentemente selezionato nella tabella dei libri.

public void **cercaCorso**(ServiziUtente.SchedaCorso corso)

E' utilizzato per effettuare una ricerca di corsi all'interno del database. E' predisposto ad accettare in ingresso un oggetto SchedaCorso (precedentemente creato tramite compilaForm()) ed a partire dai dati in esso contenuti (non

necessariamente tutti) effettua la query e ne visualizza il risultato nella tabella dei corsi.

public void **nuovoCorso**()

Il metodo permette di memorizzare un nuovo corso nel database. I dati relativi al libro vengono acquisiti tramite il metodo compilaForm() e verifica inoltre la completezza dei dati inseriti.

public void **mostraTutti**()

Invoca il metodo cercaCorso() passandogli, come parametro SchedaCorso, dati che permettono la visualizzazione di tutti i corsi memorizzati nel database.

public SchedaCorso **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaCorso (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public int **selezionaCorso**()

E' utilizzato per individuare quale riga della tabella dei corsi è selezionata e ritorna il suo indice.

public void **pulisciForm**(javax.swing.JTable jTable)

Viene richiamato ogni volta che un metodo di interrogazione del database ha successo. E' utilizzato per svuotare i campi della tabella di ricerca, per facilitare il

lavoro dell’utente se deve effettuare un'altra operazione che utilizza la stessa tabella di interrogazione.

Codice Java generato:

```
package TerminaleUtente;

import java.awt.Frame;
import ServiziUtente.SchedaCorso;
import javax.swing.JTable;

public class finestraGestioneCorsi extends Frame
{
   public finestraInfoCorso theFinestraInfoCorso;
   public SchedaCorso theSchedaCorso;

   public finestraGestioneCorsi()
   {
   }

   /**
   @roseuid 3DAC276A0320
   */
   public void modificaCorso(ServiziUtente.SchedaCorso nuoviDati, int
riga, ServiziUtente.SchedaCorso vecchiDati)
   {
   }

   /**
   @roseuid 3DAC278E0154
   */
   public void cancellaCorso()
   {
   }
```

```
/**
@roseuid 3DAC279D014A
*/
public void cercaCorso(ServiziUtente.SchedaCorso corso)
{
}

/**
@roseuid 3DAC27B0029E
*/
public void nuovoCorso()
{
}

/**
@roseuid 3DAC27BF033E
*/
public void mostraTutti()
{
}

/**
@roseuid 3DAC27E000B4
*/
public SchedaCorso compilaForm(javax.swing.JTable table)
{
}

/**
@roseuid 3DAC283E032A
*/
```

```
    public int selezionaCorso()
    {
    }

    /**
    @roseuid 3DAC28670014
    */
    public void pulisciForm(javax.swing.JTable jTable)
    {
    }
}
```

1.1.15 Classe finestraInfoCorso

E' un'estensione della classe Frame e genera la finestra che permette, tramite una tabella, la visualizzazione dei libri associati a un corso. Viene richiamata passandogli un oggetto di tipo SchedaCorso che diventa il soggetto delle operazioni della finestra. Permette di rimuovere un libro o di aggiungerlo richiamando la finestraLibriCorsi.

Metodi:

public SchedaLibro **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaLibro (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public void **cercaLibro**(ServiziUtente.SchedaLibro libro)

E' utilizzato per effettuare una ricerca dei libri all'interno del database che hanno una relazione con il corso soggetto della finestra. E' predisposto ad accettare in ingresso un oggetto SchedaLibro (precedentemente creato tramite compilaForm()) ed a partire dai dati in esso contenuti (non necessariamente tutti) effettua la query e ne visualizza il risultato nella tabella dei libri.

public void **rimuoviLibro**()

Permette la rimozione della relazione tra il corso rappresentato nella finestra e il libro selezionato nella tabella.

public void **mostraTutti**()

Invoca il metodo cercaLibro() passandogli, come parametro SchedaLibro, dati che permettono la visualizzazione di tutti i libri memorizzati nel database e associati al libro soggetto della finestra.

public int **selezionaLibro**()

E' utilizzato per individuare quale riga della tabella dei libri è selezionata e ritorna il suo indice.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;
import ServiziUtente.SchedaCorso;
import ServiziUtente.SchedaLibro;
import javax.swing.JTable;


public class finestraInfoCorso extends Frame
{
   public finestraLibriCorsi theFinestraLibriCorsi;
   public SchedaCorso theSchedaCorso;
   public SchedaLibro theSchedaLibro;


   public finestraInfoCorso()
   {
   }


   /**
   @roseuid 3DABE29B00D2
   */
   public SchedaLibro compilaForm(javax.swing.JTable table)
   {
   }


   /**
   @roseuid 3DABE2C0000A
   */
   public void cercaLibro(ServiziUtente.SchedaLibro libro)
   {
   }
```

```
    /**
    @roseuid 3DAC24E203C0
    */
    public void rimuoviLibro()
    {
    }

    /**
    @roseuid 3DAC252201FE
    */
    public void mostraTutti()
    {
    }

    /**
    @roseuid 3DAC253200D2
    */
    public int selezionaLibro()
    {
    }
}
```

1.1.16 Classe modalitaMagazzino

E' un'estensione della classe Frame e genera la finestra principale della modalità magazzino. Permette di visualizzare, tramite una tabella, la lista di tutti gli ordini evasi e non evasi del database. Offre l'opportunità di cercare un particolare ordine tramite query, effettuabili inserendo le chiavi di ricerca nell'opportuno form e cliccando sul tasto dedicato. Tale finestra permette di avviare la classe finestraCaricaVisualizzaOrdine per caricare un ordine. Permette inoltre all'utente di terminare l'esecuzione del programma con il relativo tasto di uscita.

Metodi:

public SchedaOrdine **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica riga in cui le colonne contengono le singole chiavi di ricerca. Tale tabella viene passata come parametro al metodo che ritorna un oggetto di tipo SchedaCorso (vedi package ServiziUtente) dove vengono memorizzati i valori inseriti.

public int **selezionaOrdine**()

E' utilizzato per individuare quale riga della tabella degli ordini è selezionata e ritorna il suo indice.

public void **cercaOrdine**(ServiziUtente.SchedaOrdine ordine)

E' utilizzato per cercare ordini all'interno del database. E' predisposto ad accettare in ingresso un oggetto SchedaOrdine (precedentemente creato tramite

compilaForm()) ed a partire dai dati in esso contenuti (non necessariamente tutti) effettua la query e ne visualizza il risultato nella tabella delle prenotazioni.

public void **pulisciForm**(javax.swing.JTable table)

Viene richiamato ogni volta che un metodo di interrogazione del database ha successo. E' utilizzato per svuotare i campi della tabella di ricerca, per facilitare il lavoro dell'utente se deve effettuare un'altra operazione che utilizza la stessa tabella di interrogazione.

public void **esci**()

Viene utilizzato per terminare l'esecuzione del programma. Prima della chiusura chiede conferma dell'azione.

public void **mostraEvasi_NonEvasi**(boolean b)

Permette la visualizzazione, nella tabella della finestra, di tutti gli ordini evasi o non evasi. Se viene passato come parametro true visualizza tutti gli ordini evasi altrimenti visualizza quelli non evasi.

Codice Java generato:

```
package TerminaleUtente;


import java.awt.Frame;
import ServiziUtente.SchedaOrdine;
import javax.swing.JTable;


public class modalitaMagazzino extends Frame
{
   public SchedaOrdine theSchedaOrdine;
   public finestraCaricaVisualizzaOrdine
theFinestraCaricaVisualizzaOrdine;


   public modalitaMagazzino()
   {
   }


   /**
   @roseuid 3D9063AB00D2
   */
   public SchedaOrdine compilaForm(javax.swing.JTable table)
   {
   }


   /**
   @roseuid 3D9070CD0168
   */
   public int selezionaOrdine()
   {
   }
```

```
    /**
    @roseuid 3DAC12770032
    */
    public void cercaOrdine(ServiziUtente.SchedaOrdine ordine)
    {
    }

    /**
    @roseuid 3DAC135A0302
    */
    public void pulisciForm(javax.swing.JTable table)
    {
    }

    /**
    @roseuid 3DAD05F60226
    */
    public void esci()
    {
    }

    /**
    @roseuid 3DAD1AB70366
    */
    public void mostraEvasi_NonEvasi(boolean b)
    {
    }
}
```

1.1.17 Classe finestraCaricaVisualizzaOrdine

E' un'estensione della classe Frame e genera la finestra che permette di caricare e di visualizzare, tramite una tabella, la lista di tutti i libri di un ordine, passatogli come parametro. La carica di un libro avviene selezionandolo dalla tabella e compilando la cella "copie da caricare" e cliccando sul tasto apposito. Controlla se i libri caricati evadono l'ordine e se il controllo è positivo modifica il campo evaso di SchedaOrdine nel database a true .

Metodi:

public int **selezionaLibro**()

E' utilizzato per individuare quale riga della tabella dei libri è selezionata e ritorna il suo indice.

public int **compilaForm**(javax.swing.JTable table)

E' utilizzato per catturare l'input utente nel form di ricerca. L'input viene inserito in una tabella di un'unica cella. Tale tabella viene passata come parametro al metodo che ritorna un intero che rappresenta il numero di copie del libro da caricare.

public void **listaLibriOrdinati**(int idOrdine)

Visualizza nelle tabella la lista di tutti i libri appartenenti all'ordine specificato dall'idOrdine

public void **caricaLibro**(int riga, int idOrdine, int copie)

Viene utilizzato per caricare di un libro identificato dalla riga della tabella e che fa parte dell'ordine idOrdine. Il numero di copie da caricare sarà posto uguale al parametro copie.

public void **ordineEvaso**(int idOrdine)

Richiamato quando tutti i libri dell'ordine idOrdine sono stati caricati, pone il campo evaso di schedaOrdine a true.

Codice Java generato:

```
package TerminaleUtente;


import TerminaleUtente.*;
import java.awt.Frame;
import ServiziUtente.SchedaOrdine;
import javax.swing.JTable;


public class finestraCaricaVisualizzaOrdine extends Frame
{
   public SchedaOrdine theSchedaOrdine;


   public finestraCaricaVisualizzaOrdine()
   {
   }


   /**
   @roseuid 3D90723D01D6
   */
   public int selezionaLibro()
   {
   }


   /**
   @roseuid 3D907255024E
   */
   public int compilaForm(javax.swing.JTable table)
   {
   }


   /**
```

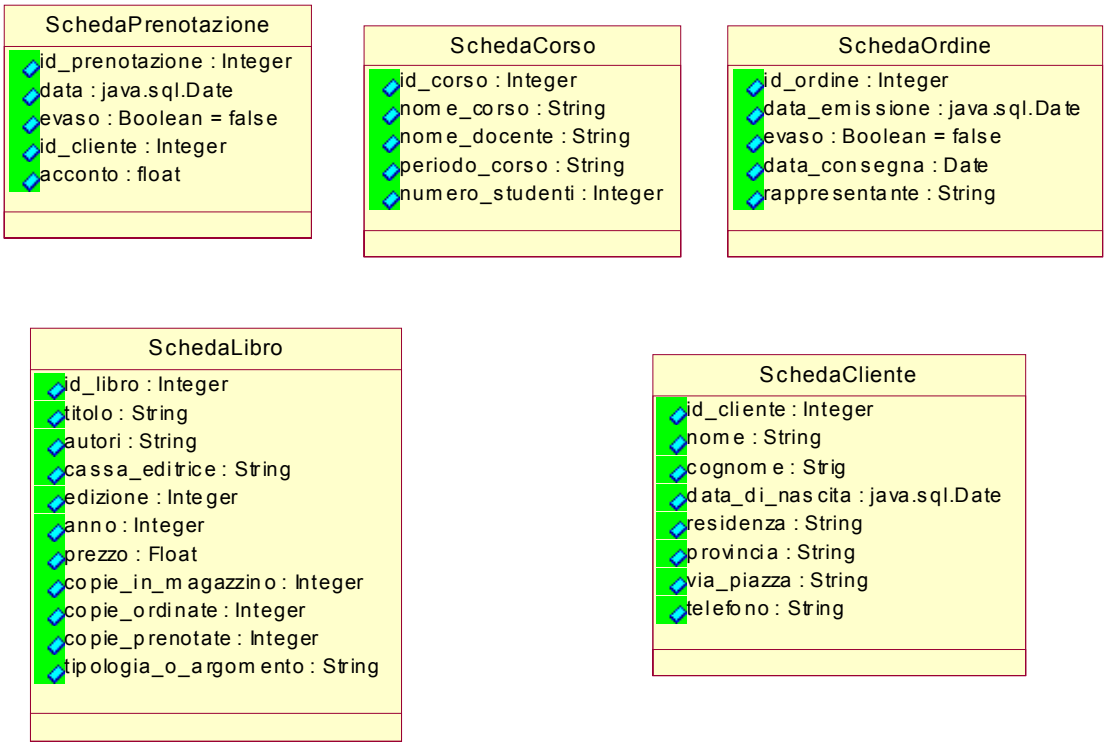
```
    @roseuid 3DABE69C03AC
    */
    public void listaLibriOrdinati(int idOrdine)
    {
    }

    /**
    @roseuid 3DABE6C20046
    */
    public void caricaLibro(int riga, int idOrdine, int copie)
    {
    }

    /**
    @roseuid 3DABE84E0168
    */
    public void ordineEvaso(int idOrdine)
    {
    }
}
```

1.2 Package ServiziUtente

Contiene le seguenti classi:



1.2.1 Classe SchedaLibro

Classe per la manipolazione dei dati dei libri costruita con gli stessi campi della entità schedaLibro del database.

Attributi:

public Integer id_libro;

public String titolo;

public String autori;

public String cassa_editrice;

public Integer edizione;

public Integer anno;

public Float prezzo;

public Integer copie_in_magazzino;

public Integer copie_ordinate;

public Integer copie_prenotate;

public String tipologia_o_argomento;

Codice Java generato:

```
package ServiziUtente;



public class SchedaLibro
{
   public Integer id_libro;
   public String titolo;
   public String autori;
   public String cassa_editrice;
   public Integer edizione;
   public Integer anno;
   public Float prezzo;
   public Integer copie_in_magazzino;
   public Integer copie_ordinate;
   public Integer copie_prenotate;
   public String tipologia_o_argomento;

   public SchedaLibro()
   {
   }
}
```

1.2.2 Classe SchedaOrdine

Classe per la manipolazione dei dati degli ordini costruita con gli stessi campi della entità schedaOrdine del database.

Attributi:

```
public Integer id_ordine;

public java.sql.Date data_emissione;

public Boolean evaso = false;

public java.sql.Date data_consegna;

public String rappresentante;
```

Codice Java generato:

```
package ServiziUtente;


import java.sql.Date;


public class SchedaOrdine
{
   public Integer id_ordine;
   public Date data_emissione;
   public Boolean evaso = false;
   public Date data_consegna;
   public String rappresentante;


   public SchedaOrdine()
   {
   }
}
```

1.2.3 Classe SchedaCorso

Classe per la manipolazione dei dati dei corsi costruita con gli stessi campi della entità schedaCorso del database.

Attributi:

```
public Integer id_corso;

public String nome_corso;

public String nome_docente;

public String periodo_corso;

public Integer numero_studenti;
```

Codice Java generato:

```
package ServiziUtente;



public class SchedaCorso
{
   public Integer id_corso;
   public String nome_corso;
   public String nome_docente;
   public String periodo_corso;
   public Integer numero_studenti;


   public SchedaCorso()
   {
   }
}
```

1.2.4 Classe SchedaCliente

Classe per la manipolazione dei dati dei clienti costruita con gli stessi campi della entità schedaCliente del database.

Attributi:

```
public Integer id_cliente;
public String nome;
public Strig cognome;
public java.sql.Date data_di_nascita;
public String residenza;
public String provincia;
public String via_piazza;
public String telefono;
```

Codice Java generato:

```
package ServiziUtente;


import java.sql.Date;


public class SchedaCliente
{
   public Integer id_cliente;
   public String nome;
   public Strig cognome;
   public Date data_di_nascita;
   public String residenza;
   public String provincia;
   public String via_piazza;
   public String telefono;


   public SchedaCliente()
   {
   }
}
```

1.2.5 Classe SchedaPrenotazione

Classe per la manipolazione dei dati delle prenotazioni costruita con gli stessi campi della entità schedaPrenotazione del database.

Attributi:

```
public Integer id_prenotazione;

public java.sql.Date data;

public Boolean evaso = false;

public Integer id_cliente;

public float acconto;
```

Codice Java generato:

```
package ServiziUtente;


import java.sql.Date;


public class SchedaPrenotazione
{
   public Integer id_prenotazione;
   public Date data;
   public Boolean evaso = false;
   public Integer id_cliente;
   public float acconto;


   public SchedaPrenotazione()
   {
   }
}
```