

UNIVERSITA' DEGLI STUDI DI PALERMO

FACOLTA' DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA
DIPARTIMENTO DI INFORMATICA



Libreria G & R

**Programma di gestione di una libreria
universitaria**

**Tesina di Ingegneria del Software di :
XXX
YYY**

Anno Accademico 2002/2003

INDICE SINTETICO

Piano di sviluppo del progetto (SPMP).....7

Analisi dei requisiti del software (RAD).....34

Architettura del sistema (SDD).....135

Progetto esecutivo (Object Design).....174

Appendice A.....312

INDICE

Software Project Management Plan

PREFAZIONE	pag. 8
1 INTRODUZIONE	pag. 9
1.1 Documenti Prodotti	pag. 9
1.2 OverView del prodotto	pag. 10
1.3 Evoluzione della pianificazione del progetto	pag. 12
1.4 Materiale Tecnico	pag. 12
1.5 Acronimi e definizioni	pag. 13
2. Organizzazione del progetto	pag. 13
2.1 attività e fasi di lavoro	pag. 14
2.1.1 Pianificazione	pag. 14
2.1.2 Requirements analysis	pag. 15
2.1.3 System Design	pag. 15
2.1.4 Presentazione del progetto ed approvazione del cliente	pag. 15
2.1.5 Fase di Object Design	pag. 16
2.1.6 Presentazione prima versione del tool	pag. 16
2.1.7 Fase di test del sistema	pag. 16
2.1.8 Revisione finale	pag. 16
2.1.9 Presentazione del prodotto al cliente e consegna	pag. 17
2.2 Risorse umane coinvolte nel progetto	pag. 17
3. Strumenti Utilizzati	pag. 18
4. Vista complessiva sulla pianificazione del progetto	pag. 20
5 Diagrammi di pianificazione del progetto	pag. 21

Requirements Analysis Document

1.INTRODUZIONE	pag. 35
1.1.Descrizione del sistema	pag. 35
1.2.Scopo del sistema	pag. 35

2.SISTEMA CORRENTE	pag. 36
3.SISTEMA PROPOSTO	pag. 38
3.1.Panoramica generale	pag. 38
3.2 Requisiti funzionali	pag. 39
3.2.1 Gestione Libri	pag. 40
3.2.2 Gestione Magazzino	pag. 41
3.2.3 Gestione Vendite	pag. 41
3.2.4 Gestione Acquisti	pag. 42
3.2.5 Gestione Ordini	pag. 42
3.2.6 Gestione Prenotazione Cliente	pag. 43
3.2.7 Gestione manageriale	pag. 44
3.3 Requisiti non funzionali	pag. 46
3.3.1 Interfaccia utente	pag. 46
3.3.2 Documentazione	pag. 47
3.3.3 Considerazioni hardware	pag. 47
3.3.4 Prestazioni del sistema	pag. 48
3.3.5 Gestione degli errori e tolleranza ai guasti	pag. 48
3.3.6 Sicurezza	pag. 48
3.4 Pseudo-Requisiti	pag. 49
3.5 Modelli del sistema	pag. 50
3.5.1 Scenari	pag. 50
3.5.2 Modello dei casi d'uso del sistema	pag. 58
3.5.2.1 Attori	pag. 58
3.5.2.2 Diagramma principale di casi d'uso	pag. 59
3.5.2.3 Diagramma funzionalità cliente	pag. 60
3.5.2.3 Diagr. funzionalità addetto vendite	pag. 61
3.5.2.4 Diagr. funzionalità addetto magazzino	pag. 66
3.5.2.5 Diagr. funzionalità addetto agli ordini	pag. 68
3.5.2.6 Diagr. .funzionalità amministratore	pag. 72

3.5.3 Modello Oggetto	pag. 77
3.5.3.2 Class Diagram	pag. 77
3.5.3.2.1 Diagr. classi server	pag. 77
3.5.3.2.2 Diagr.classi cliente	pag. 78
3.5.3.2.3 Diagr. classi addetto magazzino	pag. 79
3.5.3.2.4 Diagr. classi addetto vendite	pag. 80
3.5.3.2.5 Diagr.classi addetto ordini	pag. 81
3.5.3.2.6 Diagr.classi amministratore	pag. 82
3.5.3.3 Progetto concettuale database	pag. 85
3.5.4 Modello dinamico del sistema	pag. 99
3.5.4.1 Sequence diagram	pag. 99
3.5.4.2 State diagram	pag. 119
3.5.5 Interfacce Utente	pag. 125

System Design Document

1. Obiettivi del sistema	pag. 136
2. Architettura Software	pag. 136
2.1 Overview	pag. 136
2.2 Decomposizione del sistema	pag. 141
2.2.1 Package Cliente	pag. 142
2.2.2 Package FinestraAddettoMagazzino	pag. 143
2.2.3 Package Finestra AddettoOrdini	pag. 144
2.2.4 Package FinestraAddettoVendite	pag. 145
2.2.5 Package Amministratore	pag. 146
2.3 Hardware-Software Mapping	pag. 147
2.3.1 Diagramma di Deployment del sistema	pag. 148
2.3.2 Diagramma delle componenti del sistema	pag. 148
2.4 Data Management	pag. 151
2.4.1 Progetto logico del Database	pag. 151

2.4.2 Progetto fisico del Database	pag. 165
2.5 Controllo degli accessi al sistema e sicurezza	pag. 168
2.6 Global Software Control	pag. 171
2.6.1 External control flow	pag. 171
2.6.2 Controllo interno	pag. 172
2.6.3 Concurrent Control	pag. 172
2.7 Boundary Condition	pag. 173
Classi del sistema	pag. 174
Descrizione delle classi	pag. 186
Appendice A	pag. 312

Software Project Management Plan

**Documento del piano di sviluppo del
progetto**

Prefazione

Questo documento definisce il processo tecnico e la pianificazione della produzione del sistema.

Esso è rivolto a: sviluppatori, analisti e cliente.

Si tratta di un documento di controllo per il progetto del programma di gestione della libreria universitaria G&R.

1. Introduzione

Libreria G&R è un software di gestione di una libreria universitaria .

Il progetto si pone lo scopo di semplificarne ed ottimizzarne il controllo, fornendo funzioni diverse per i suoi impiegati a lavoro su vari terminali che trattano informazioni differenti.

L'obiettivo finale è dunque quello di fornire uno strumento software che permetta di avere sempre una visione completa e coerente di tutte le attività che hanno luogo nella libreria: dalla semplice ricerca di un libro alla situazione del magazzino ,dalle vendite all'amministrazione , facendo sì che le varie attività possano essere svolte in modo ordinato e senza che si intralcino a vicenda

1.1 Documenti prodotti

Si elenca in seguito la documentazione del progetto che sarà prodotta:

- **Documento del piano di sviluppo del progetto (SPMP):** che definisce il processo tecnico e la pianificazione della produzione del sistema (questo documento).
- **Documento di analisi dei requisiti (RAD):** che descrive le funzionalità del sistema da realizzare, esso è formato da quattro tipi di modelli: modello use case, modello oggetto, modello funzionale e modello dinamico. Questo documento è creato interagendo con gli esperti del dominio ed è approvato dal cliente.
- **Documento dell'architettura software (SDD):** che descrive gli obiettivi, l'architettura, la decomposizione ad alto livello del sistema, l'allocazione hardware/software. Questo documento fornisce la base per l'implementazione del sistema e per la stesura del progetto esecutivo (ODD).
- **Documento del Progetto esecutivo (ODD):** che descrive il sistema in termini di oggetti, dei loro metodi e degli attributi. Questo documento viene redatto durante la fase di implementazione del sistema.
- **Codice sorgente di tutto il sistema.**
- **Pacchetto di installazione del tool:** sarà fornito un CD contenente il programma che permette di installare e configurare il tool nella macchina dell'utente.

1.2 Overview del progetto

Riportiamo adesso una tabella di riferimento che indica le attività principali che si sono affrontate al fine di poter progettare e realizzare il sistema proposto.

Schedulazione del progetto		
Data	Fase Progetto	Attività
08/10/2002 20/11/2002	Consolidamento delle nozioni e della conoscenza di: <ul style="list-style-type: none">- progettazione software Object-Oriented .- MsProject- Rational Rose- Java.	
21/11/2002		Discussione con i membri del progetto dei risultati ottenuti
22/11/2002	Inizio fase di analisi del sistema	
23/11/2002 12/12/2002	Analisi dei requisiti e schedulazione iniziale delle attività del progetto.	
13/12/2002		Presentazione del progetto ed approvazione del cliente
16/12/2002 20/01/2003	Sviluppo ed implementazione del progetto	
21/01/2003		Presentazione della prima versione del prodotto ed approvazione del cliente
22/01/2003 30/02/2003	Alfa test e correzioni interne al prodotto ed al progetto	
31/02/2003 07/02/2003	Beta Test e correzioni al prodotto ed al progetto	
10/02/2003 19/02/2003	Revisione finale del progetto	

19/02/2003		Consegna del prodotto al cliente.
------------	--	-----------------------------------

1.3 Evoluzione della pianificazione del progetto

Per poter pianificare correttamente ed effettuare in modo semplice cambiamenti nella pianificazione, per la gestione ed allocazione delle risorse necessarie alla realizzazione del prodotto e per la gestione delle previsioni si è utilizzato il tool Microsoft Project

Il progetto utilizza una metodologia di progettazione orientata agli oggetti ed UML per lo sviluppo del software.

Il prodotto finale e le sue versioni intermedie sono state testate su PC che utilizzano sistema operativo Windows XP e Windows 2000.

1.4 Materiale di riferimento

Per la realizzazione del sistema sono stati utilizzati i seguenti materiali di riferimento:

- Bruegge-Dutoit : *Object-Oriented Software Engineering: Conquering Complex and Changing System* (Corso di Ingegneria del Software).
- Tutto il materiale di esempio trovato sul sito di Ingegneria del Software del professore M. Cossentino. Tale materiale può essere recuperato collegandosi al sito internet http://www.csai.unipa.it/cossentino/isw0102/#regole_esame e sfruttando i collegamenti ivi presenti.

- Lemay-Perkins : *Java 1.1 Guida Completa* (Manuale di java).
- Articoli vari su Java dai siti <http://www.mokabyte.com> e <http://www.javaworld.com>
- Guida in linea di Jbuilder2 client/server
- Documentazione del JDK1.2 e JDK1.3
- Guida in linea di Rational Rose Enterprise Edition
- Guida in linea di Ms-Project 98.

1.5 Acronimi e Definizioni

GUI - Graphical User Interface

JDK - Java Development Kit

ODD - Object Design Document

RAD - Requirements Analysis Document

ROSE – Tool di progettazione e sviluppo software.

SDD - System Design Document

SPMP - Software Project Management Plan

UML - Unified Modeling Notation

2. Organizzazione del progetto

Il progetto è iniziato l'8 ottobre 2002 ed è terminato il 19 febbraio 2003.

Durante tale periodo sono state svolte diverse attività. Presentiamo di seguito quelle principali che hanno delle durate precise e che terminano (in genere) con la creazione del prodotto in versioni intermedie. Tali versioni intermedie del prodotto (eccetto il prodotto finale stesso che si reputa essere completo e funzionante) servono per mostrare il sistema al cliente. Tali attività sono di seguito riportate.

- riassunto e presentazione risultati dello studio preliminare: 21 novembre 2002;
- presentazione del progetto e approvazione del cliente: 13 dicembre 2003;
- presentazione e dimostrazione prima versione del tool: 21 gennaio 2003;
- Consegna del prodotto: 19 febbraio 2003

2.1 Attività e fasi di lavoro

Riportiamo di seguito l'insieme dettagliato delle varie attività e fasi di lavoro che si sono affrontate per poter realizzare il sistema software progettato.

2.1.1 Pianificazione

Questa fase è legata alla descrizione generale del progetto. Viene data così una panoramica delle sue attività e la dipendenza delle risorse che verranno utilizzate in esso.

Questa prima fase porta alla stesura di questo documento.

2.1.2 Requirements Analysis

Durante questa attività viene analizzato il problema e viene esaminato il sistema in termini delle funzionalità che deve fornire cercando di mantenere le proprietà di consistenza e di completezza.

Questa attività porta alla determinazione di un insieme di modelli che descrivono il sistema che si vuole realizzare e che il cliente desidera avere.

Il documento che viene prodotto è il RAD (Requirements Analysis Document) che presenta quattro modelli principali per il sistema:

- modello degli use case : descrive le funzionalità che il sistema deve offrire all'utente;
- modello oggetto : che descrive la struttura del sistema in termini di oggetti;
- modello funzionale : descrive l'interazione fra i vari oggetti che realizzano le funzionalità del sistema;
- modello dinamico: descrive l'evoluzione dinamica del sistema.

2.1.3 System Design

Lo scopo principale di questa attività è di stabilire l'architettura del sistema attraverso la sua decomposizione in sottosistemi e l'allocazione delle sue parti sulle componenti hardware.

2.1.4 Presentazione del progetto e approvazione del cliente

Presentazione del progetto al cliente e approvazione delle funzionalità del

sistema. L'incontro è avvenuto in data 13 dicembre 2003.

2.1.5 Fase di Object Design

L'attività di implementazione mira a realizzare i moduli software dei vari package di cui si compone il sistema. Questa fase produce il documento (ODD).

2.1.6 Presentazione prima versione del tool

Viene presentata la prima versione del tool al cliente . Eventuali revisioni vengono concordate con il cliente. L'incontro è stato fissato per il 21 gennaio 2003.

2.1.7 Fase di test del sistema

Durante tale attività si passa a testare il sistema cercando di trovare ed eliminare imperfezioni e malfunzionamenti dello stesso.

2.1.8 Revisione finale

Attività di controllo finale sulla qualità della documentazione da fornire al cliente e preparazione del CD di installazione del prodotto.

2.1.9 Presentazione del prodotto al cliente e consegna

In questa fase viene presentato il prodotto al cliente e viene fatta una dimostrazione del prodotto. Tale fase termina con la consegna del prodotto e l'utilizzo da parte del cliente stesso.

2.2 Risorse umane coinvolte nel progetto

Il progetto è stato affidato ad un singolo team composto dalle seguenti figure professionali:

- Capo progetto: coordina il progetto e lo pianifica secondo le esigenze ed i tempi del committente.

Coordina anche gli incontri con il cliente e gli incontri fra i membri del team.

- Analista: si occupa di fornire un'adeguata analisi del problema insieme a degli esperti del dominio e cerca di tradurre in specifiche tecniche le richieste del cliente.
- Programmatore Java : ha il compito di sviluppare i componenti del sistema.
- Programmatore GUI: ha il compito di sviluppare le interfacce grafiche del sistema.
- Tester: ha il compito di testare il sistema e di comunicare ai programmatori eventuali errori trovati.

- Progettista software: ha il compito di studiare il sistema ed effettuarne la decomposizione in sottosistemi , si occupa della fase del sistem design e partecipa anche nella fase di realizzazione del sistema.

Nel diagramma di Gantt (riportato in seguito in questo documento) viene presentato il modo in cui tali risorse umane sono state allocate alle varie attività.

I costi orari dei componenti dello staff tecnico sono riportati nell'analisi dei costi.

3. Strumenti utilizzati

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

- Microsoft Project 98: per curare la pianificazione del progetto.
- Rational Rose Enterprise Edition: strumento CASE utilizzato nella progettazione dei modelli del sistema (RAD), nella scomposizione del sistema (SDD) e nella definizione degli oggetti del sistema (ODD).
- JBuilder2 client/server suite ver 2.0: strumento utilizzato nell'implementazione degli oggetti del sistema;
- Microsoft Word 2002 (pacchetto Office Xp): utilizzato per creare tutta la documentazione del sistema.

- Jasc Paint Shop Pro ver. 7 Eval : per catturare le immagini che sono state inserite nella documentazione.
- Microsoft Access 2002(pacchetto Office Xp): usato per realizzare il database della libreria universitaria.

4. Vista complessiva sulla pianificazione del progetto

Riportiamo adesso tutte le informazioni di pianificazione del progetto che sono state utilizzate per pianificare e gestire le fasi di produzione del sistema.

Tale attività di pianificazione è stata realizzata utilizzando gli strumenti messi a disposizione dal tool Microsoft Project 98. Sono stati di grande aiuto soprattutto la carta di Gantt e le varie prospettive dalle quali è possibile osservare tutte le fasi di pianificazione.

Risorse del progetto

Mostriamo adesso le risorse che vengono utilizzate nel nostro progetto e gli attributi più importanti di esse.

ID	Nome risorsa	Gruppo	Unità max	Punta	Tariffa std.	Tariffa str.	Costo	Lavoro
1	Capo Progetto		100%	100%	€ 260,00/g	€ 0,00/h	€ 3.510,00	108 h
2	Analista		100%	100%	€ 130,00/g	€ 0,00/h	€ 2.925,00	180 h
3	Programmatore Java		100%	100%	€ 104,00/g	€ 0,00/h	€ 3.380,00	280 h
4	Programmatore GUI		100%	100%	€ 104,00/g	€ 0,00/h	€ 3.380,00	280 h
5	Tester		100%	100%	€ 65,00/g	€ 0,00/h	€ 910,00	152 h
6	Progettista Software		100%	100%	€ 130,00/g	€ 0,00/h	€ 4.550,00	280 h













Utilizzo risorse durante l'esecuzione del progetto

ID	Nome risorsa	Dettagli	2003						
			S	O	N	D	G	F	M
	Non assegnata	Lavoro							
	<i>Fine Progetto sistema</i>	Lavoro							
	<i>Presentazione della prima versione al cliente</i>	Lavoro							
	<i>Consegna prodotto al Cliente</i>	Lavoro							
	<i>Presentazione progetto al cliente</i>	Lavoro							
1	Capo Progetto	Lavoro		16h	0h	36h		64h	
	<i>Discussione sulle fasi attività del RAD con i membri del team</i>	Lavoro				4h			
	<i>Mapping Software-Hardware</i>	Lavoro				8h			
	<i>Revisione finale</i>	Lavoro						64h	
	<i>Ms-Project</i>	Lavoro		16h					
	<i>Schedulazione del progetto</i>	Lavoro				16h			
	<i>Ripartizione compiti-inizio codifica</i>	Lavoro				8h			
	<i>Discussione risultati dello studio</i>	Lavoro			0h				
2	Analista	Lavoro		80h	48h	52h			
	<i>Identificazione Funzioni sistema (Use Case)</i>	Lavoro			48h				
	<i>Definizione Oggetti</i>	Lavoro				24h			
	<i>Descrizione dinamica del sistema</i>	Lavoro				24h			
	<i>Discussione sulle fasi attività del RAD con i membri del team</i>	Lavoro				4h			
	<i>Progettazione Software Object-Oriented</i>	Lavoro		0h					
	<i>Studio Analisi</i>	Lavoro		80h					
	<i>Discussione risultati dello studio</i>	Lavoro			0h				
3	Programmatore Java	Lavoro			48h	40h	148h	24h	
	<i>Implementazione delle componenti di controllo</i>	Lavoro				40h	96h		
	<i>Assemblaggio delle componenti</i>	Lavoro					20h		
	<i>Alfa test e correzioni</i>	Lavoro					28h		
	<i>Beta test e correzioni</i>	Lavoro					4h	24h	
	<i>Ripasso Java</i>	Lavoro			48h				
4	Programmatore GUI	Lavoro			48h	40h	148h	24h	

ID	Nome risorsa	Dettagli	2003								
			S	O	N	D	G	F	M	A	M
4	Programmatore GUI	Lavoro			48h	40h	148h	24h			
	<i>Implementazione delle Interfacce</i>	Lavoro				40h	96h				
	<i>Assemblaggio delle componenti</i>	Lavoro					20h				
	<i>Alfa test e correzioni</i>	Lavoro					28h				
	<i>Beta test e correzioni</i>	Lavoro					4h	24h			
	<i>Ripasso Java</i>	Lavoro			48h						
5	Tester	Lavoro					64h	48h			
	<i>Alfa test e correzioni</i>	Lavoro					56h				
	<i>Beta test e correzioni</i>	Lavoro					8h	48h			
6	Progettista Software	Lavoro		16h	88h	44h	52h	88h			
	<i>Discussione sulle fasi attività del RAD con i membri del team</i>	Lavoro				4h					
	<i>Decomposizione del sistema</i>	Lavoro				32h					
	<i>Mapping Software-Hardware</i>	Lavoro				8h					
	<i>Assemblaggio delle componenti</i>	Lavoro					20h				
	<i>Alfa test e correzioni</i>	Lavoro					28h				
	<i>Beta test e correzioni</i>	Lavoro					4h	24h			
	<i>Revisione finale</i>	Lavoro						64h			
	<i>Progettazione Software Object-Oriented</i>	Lavoro		16h	48h						
	<i>Rational Rose</i>	Lavoro			40h						
	<i>Discussione risultati dello studio</i>	Lavoro			0h						
		Lavoro									
		Lavoro									
		Lavoro									
		Lavoro									
		Lavoro									
		Lavoro									
		Lavoro									
		Lavoro									
		Lavoro									
		Lavoro									

Diagramma di Gantt

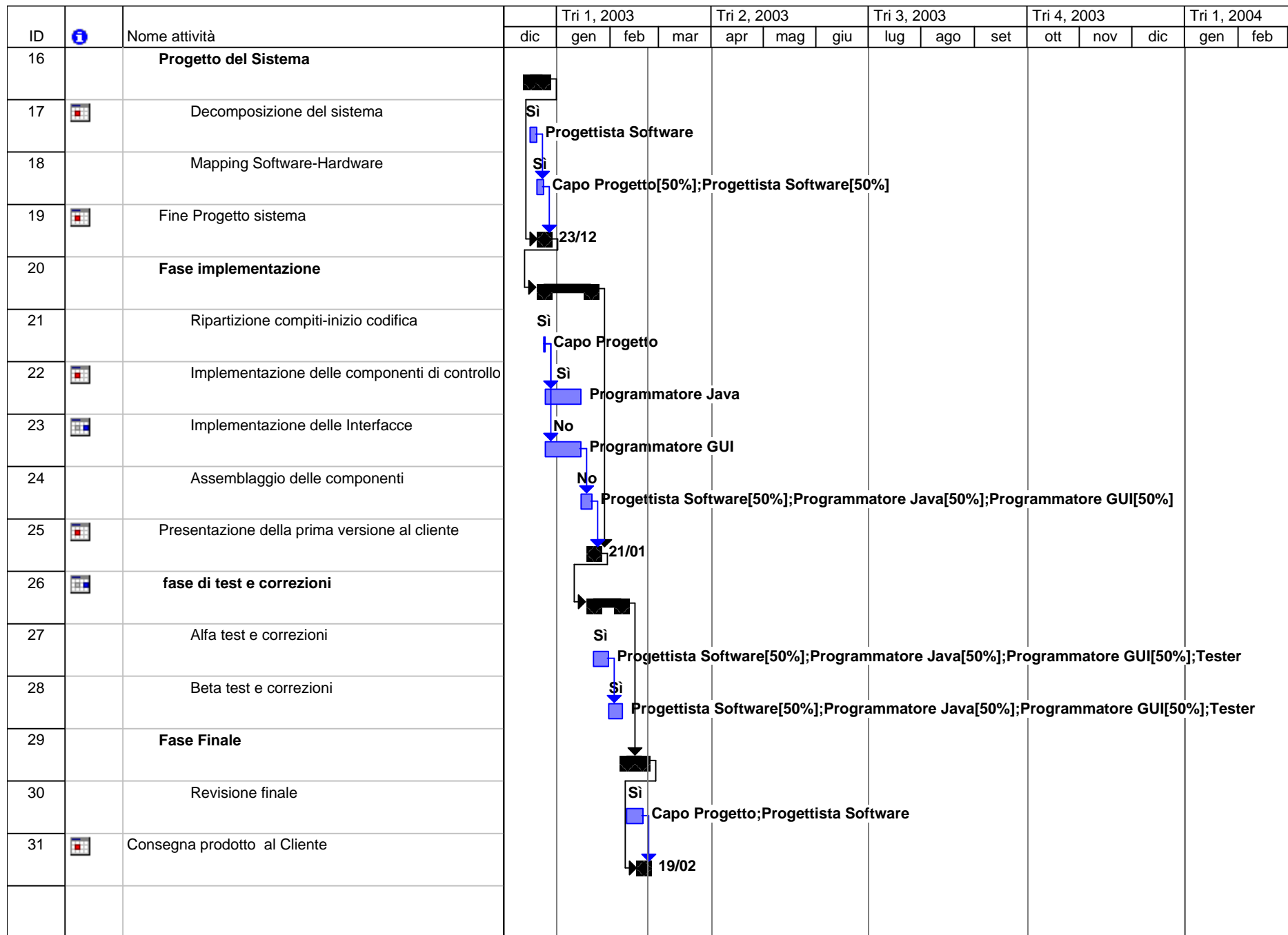
Presentiamo adesso il diagramma di Gantt che mostra l'evoluzione temporale del progetto e delle attività affrontate per costruire il sistema.

ID		Nome attività	Durata	Inizio	Fine	2003												2004											
						S	O	N	D	G	F	M	A	M	G	L	A	S	O	N	D	G	F	M	A	M			
1		Libreria G & R	102 g	mar 08/10/02	mer 19/02/03																								
2		Studi preliminari	32 g	mar 08/10/02	mer 20/11/02																								
3		Ms-Project	2 g	mar 08/10/02	mer 09/10/02	<div>No</div> <div>Capo Progetto</div>																							
4		Studio Analisi	10 g	gio 10/10/02	mer 23/10/02	<div>No</div> <div>Analista</div>																							
5		Progettazione Software Object-Oriented	8 g	gio 24/10/02	ven 08/11/02	<div>No</div> <div>Progettista Software</div>																							
6		Rational Rose	5 g	lun 11/11/02	ven 15/11/02	<div>No</div> <div>Progettista Software</div>																							
7		Ripasso Java	6 g	mer 13/11/02	mer 20/11/02	<div>No</div> <div>Programmatore Java;Programmatore GUI</div>																							
8		Discussione risultati dello studio	0 g	gio 21/11/02	gio 21/11/02	<div>Si</div> <div>21/11</div>																							
9		Analisi Dei Requisiti	16 g	ven 22/11/02	ven 13/12/02	<div>22/11</div>																							
10		Identificazione Funzioni sistema (Use Case)	6 g	ven 22/11/02	ven 29/11/02	<div>Si</div> <div>Analista</div>																							
11		Definizione Oggetti	3 g	lun 02/12/02	mer 04/12/02	<div>Si</div> <div>Analista</div>																							
12		Descrizione dinamica del sistema	3 g	gio 05/12/02	lun 09/12/02	<div>No</div> <div>Analista</div>																							
13		Discussione sulle fasi attività del RAD con i n	1 g	mar 10/12/02	mar 10/12/02	<div>No</div> <div>Capo Progetto[50%];Analista[50%];Progettista Software[50%]</div>																							
14		Schedulazione del progetto	2 g	mer 11/12/02	gio 12/12/02	<div>No</div> <div>Capo Progetto</div>																							
15		Presentazione progetto al cliente	0 g	ven 13/12/02	ven 13/12/02	<div>13/12</div>																							
16		Progetto del Sistema	6 g	lun 16/12/02	lun 23/12/02																								

ID		Nome attività	Durata	Inizio	Fine	bre					gennaio					febbraio				marzo				
						09	16	23	30	06	13	20	27	03	10	17	24	03	10	17	24			
16		Progetto del Sistema	6 g	lun 16/12/02	lun 23/12/02	<pre>graph TD Start(()) --> P16[Progetto del Sistema] P16 --> P17[Decomposizione del sistema] P17 --> P18[Mapping Software-Hardware] P18 --> P19[Fine Progetto sistema] P19 --> P20[Fase implementazione] P20 --> P21[Ripartizione compiti-inizio codifica] P21 --> P22[Implementazione delle componenti di controll] P22 --> P23[Implementazione delle Interfacce] P23 --> P24[Assemblaggio delle componenti] P24 --> P25[Presentazione della prima versione al cliente] P25 --> P26[fase di test e correzioni] P26 --> P27[Alfa test e correzioni] P27 --> P28[Beta test e correzioni] P28 --> P29[Fase Finale] P29 --> P30[Revisione finale] P30 --> P31[Consegna prodotto al Cliente] P31 --> End(()) P16 --> M1[23/12] P20 --> M2[21/01] P26 --> M3[19/02]</pre>																		
17		Decomposizione del sistema	4 g	lun 16/12/02	gio 19/12/02																			
18		Mapping Software-Hardware	2 g	ven 20/12/02	lun 23/12/02																			
19		Fine Progetto sistema	0 g	lun 23/12/02	lun 23/12/02																			
20		Fase implementazione	23 g	mar 24/12/02	lun 20/01/03																			
21		Ripartizione compiti-inizio codifica	1 g	mar 24/12/02	mar 24/12/02																			
22		Implementazione delle componenti di controll	17 g	mer 25/12/02	mar 14/01/03																			
23		Implementazione delle Interfacce	17 g	mer 25/12/02	mar 14/01/03																			
24		Assemblaggio delle componenti	5 g	mer 15/01/03	lun 20/01/03																			
25		Presentazione della prima versione al cliente	0 g	mar 21/01/03	mar 21/01/03																			
26		fase di test e correzioni	14 g	mer 22/01/03	ven 07/02/03																			
27		Alfa test e correzioni	7 g	mer 22/01/03	gio 30/01/03																			
28		Beta test e correzioni	7 g	ven 31/01/03	ven 07/02/03																			
29		Fase Finale	8 g	lun 10/02/03	mer 19/02/03																			
30		Revisione finale	8 g	lun 10/02/03	mer 19/02/03																			
31		Consegna prodotto al Cliente	0 g	mer 19/02/03	mer 19/02/03																			

Mostriamo adesso il diagramma di Gantt con l'elenco delle risorse adoperate

ID	Nome attività	Tri 4, 2002				Tri 1, 2003			Tri 2, 2003			Tri 3, 2003			Tri 4, 2003	
		set	ott	nov	dic	gen	feb	mar	apr	mag	giu	lug	ago	set	ott	nov
1	Libreria G & R															
2	Studi preliminari															
3	Ms-Project															
4	Studio Analisi															
5	Progettazione Software Object-Oriented															
6	Rational Rose															
7	Ripasso Java															
8	Discussione risultati dello studio															
9	Analisi Dei Requisiti															
10	Identificazione Funzioni sistema (Use Case)															
11	Definizione Oggetti															
12	Descrizione dinamica del sistema															
13	Discussione sulle fasi attività del RAD con i me															
14	Schedulazione del progetto															
15	Presentazione progetto al cliente															



Costi del progetto

Inseriamo adesso un grafico dal project che tiene conto dei costi totali sostenuti nelle varie fasi di sviluppo del prodotto

ID	Nome attività	Durata	Inizio	Fine	Costo	Lavoro
1	Libreria G & R	101 g	mar 08/10/02	mar 18/02/03	€18.655,00	1.200 h
2	Studi preliminari	32 g	mar 08/10/02	mer 20/11/02	€4.758,00	296 h
3	Ms-Project	2 g	mar 08/10/02	mer 09/10/02	€ 520,00	16 h
	Capo Progetto		mar 08/10/02	mer 09/10/02	€ 520,00	16 h
4	Studio Analisi	10 g	gio 10/10/02	mer 23/10/02	€ 1.300,00	80 h
	Analista		gio 10/10/02	mer 23/10/02	€ 1.300,00	80 h
5	Progettazione Software Object-Oriented	8 g	gio 24/10/02	ven 08/11/02	€ 1.040,00	64 h
	Progettista Software		mer 30/10/02	ven 08/11/02	€ 1.040,00	64 h
6	Rational Rose	5 g	lun 11/11/02	ven 15/11/02	€ 650,00	40 h
	Progettista Software		lun 11/11/02	ven 15/11/02	€ 650,00	40 h
7	Ripasso Java	6 g	mer 13/11/02	mer 20/11/02	€ 1.248,00	96 h
	Programmatore Java		mer 13/11/02	mer 20/11/02	€ 624,00	48 h
	Programmatore GUI		mer 13/11/02	mer 20/11/02	€ 624,00	48 h
8	Discussione risultati dello studio	0 g	gio 21/11/02	gio 21/11/02	€ 0,00	0 h
	Capo Progetto		gio 21/11/02	gio 21/11/02	€ 0,00	0 h
	Analista		gio 21/11/02	gio 21/11/02	€ 0,00	0 h
	Progettista Software		gio 21/11/02	gio 21/11/02	€ 0,00	0 h
9	Analisi Dei Requisiti	16 g	ven 22/11/02	ven 13/12/02	€2.340,00	124 h
10	Identificazione Funzioni sistema (Use Case)	6 g	ven 22/11/02	ven 29/11/02	€ 780,00	48 h
	Analista		ven 22/11/02	ven 29/11/02	€ 780,00	48 h
11	Definizione Oggetti	3 g	lun 02/12/02	mer 04/12/02	€ 390,00	24 h
	Analista		lun 02/12/02	mer 04/12/02	€ 390,00	24 h
12	Descrizione dinamica del sistema	3 g	gio 05/12/02	lun 09/12/02	€ 390,00	24 h
	Analista		gio 05/12/02	lun 09/12/02	€ 390,00	24 h
13	Discussione sulle fasi attività del RAD con i membri del tea	1 g	mar 10/12/02	mar 10/12/02	€ 260,00	12 h
	Capo Progetto		mar 10/12/02	mar 10/12/02	€ 130,00	4 h
	Analista		mar 10/12/02	mar 10/12/02	€ 65,00	4 h
	Progettista Software		mar 10/12/02	mar 10/12/02	€ 65,00	4 h
14	Schedulazione del progetto	2 g	mer 11/12/02	gio 12/12/02	€ 520,00	16 h

ID	Nome attività	Durata	Inizio	Fine	Costo	Lavoro
14	Schedulazione del progetto	2 g	mer 11/12/02	gio 12/12/02	€ 520,00	16 h
	<i>Capo Progetto</i>		mer 11/12/02	gio 12/12/02	€ 520,00	16 h
15	Presentazione progetto al cliente	0 g	ven 13/12/02	ven 13/12/02	€ 0,00	0 h
16	Progetto del Sistema	6 g	lun 16/12/02	lun 23/12/02	€910,00	48 h
17	Decomposizione del sistema	4 g	lun 16/12/02	gio 19/12/02	€ 520,00	32 h
	<i>Progettista Software</i>		lun 16/12/02	gio 19/12/02	€ 520,00	32 h
18	Mapping Software-Hardware	2 g	ven 20/12/02	lun 23/12/02	€ 390,00	16 h
	<i>Capo Progetto</i>		ven 20/12/02	lun 23/12/02	€ 260,00	8 h
	<i>Progettista Software</i>		ven 20/12/02	lun 23/12/02	€ 130,00	8 h
19	Fine Progetto sistema	0 g	lun 23/12/02	lun 23/12/02	€ 0,00	0 h
20	Fase implementazione	23 g	mar 24/12/02	lun 20/01/03	€4.641,00	340 h
21	Ripartizione compiti-inizio codifica	1 g	mar 24/12/02	mar 24/12/02	€ 260,00	8 h
	<i>Capo Progetto</i>		mar 24/12/02	mar 24/12/02	€ 260,00	8 h
22	Implementazione delle componenti di controllo	17 g	mer 25/12/02	mar 14/01/03	€ 1.768,00	136 h
	<i>Programmatore Java</i>		mer 25/12/02	mar 14/01/03	€ 1.768,00	136 h
23	Implementazione delle Interfacce	17 g	mer 25/12/02	mar 14/01/03	€ 1.768,00	136 h
	<i>Programmatore GUI</i>		mer 25/12/02	mar 14/01/03	€ 1.768,00	136 h
24	Assemblaggio delle componenti	5 g	mer 15/01/03	lun 20/01/03	€ 845,00	60 h
	<i>Programmatore Java</i>		mer 15/01/03	lun 20/01/03	€ 260,00	20 h
	<i>Programmatore GUI</i>		mer 15/01/03	lun 20/01/03	€ 260,00	20 h
	<i>Progettista Software</i>		mer 15/01/03	lun 20/01/03	€ 325,00	20 h
25	Presentazione della prima versione al cliente	0 g	mar 21/01/03	mar 21/01/03	€ 0,00	0 h
26	fase di test e correzioni	14 g	mer 22/01/03	ven 07/02/03	€3.276,00	280 h
27	Alfa test e correzioni	7 g	mer 22/01/03	gio 30/01/03	€ 1.638,00	140 h
	<i>Programmatore Java</i>		mer 22/01/03	gio 30/01/03	€ 364,00	28 h
	<i>Programmatore GUI</i>		mer 22/01/03	gio 30/01/03	€ 364,00	28 h
	<i>Tester</i>		mer 22/01/03	gio 30/01/03	€ 455,00	56 h
	<i>Progettista Software</i>		mer 22/01/03	gio 30/01/03	€ 455,00	28 h
28	Beta test e correzioni	7 g	ven 31/01/03	ven 07/02/03	€ 1.638,00	140 h

Calendario delle Attività del progetto

Mesi di **Ottobre-Novembre**

lunedì	martedì	mercoledì	giovedì	venerdì	sabato	domenica
30	01	02	03	04	05	06
07	08	09	10	11	12	13
	Ms-Project; 2 g		Studio Analisi; 10 g			
14	15	16	17	18	19	20
			Studio Analisi; 10 g			
21	22	23	24	25	26	27
	Studio Analisi; 10 g			Progettazione Software Object-Oriented; 8 g		
28	29	30	31	01	02	03
		Progettazione Software Object-Oriented; 8 g				
04	05	06	07	08	09	10
	Progettazione Software Object-Oriented; 8 g					
11	12	13	14	15	16	17
	Rational Rose; 5 g					
		Ripasso Java; 6 g				
18	19	20	21	22	23	24
				Identificazione Funzioni sistema (Use Case); 6 g		
	Ripasso Java; 6 g					
25	26	27	28	29	30	01
	Identificazione Funzioni sistema (Use Case); 6 g					

Mesi di **Novembre(fine)-Dicembre –**
Gennaio

lunedì	martedì	mercoledì	giovedì	venerdì	sabato	domenica
25	26	27	28	29	30	01
Identificazione Funzioni sistema (Use Case); 6 g						
02	03	04	05	06	07	08
Definizione Oggetti; 3 g		Descrizione dinamica del sistema; 3 g				
09	10	11	12	13	14	15
Descrizione dinamica del	Discussione sulle fasi att	Schedulazione del progetto; 2 g				
16	17	18	19	20	21	22
Decomposizione del sistema; 4 g			Mapping Software-Hardware; 2 g			
23	24	25	26	27	28	29
	Ripartizione compiti-inizi	Implementazione delle componenti di controllo; 17 g				
Mapping Software-Hardw		Implementazione delle Interfacce; 17 g				
30	31	01	02	03	04	05
Implementazione delle componenti di controllo; 17 g						
Implementazione delle Interfacce; 17 g						
06	07	08	09	10	11	12
Implementazione delle componenti di controllo; 17 g						
Implementazione delle Interfacce; 17 g						
13	14	15	16	17	18	19
Implementazione delle componenti di controllo; 17 g	Assemblaggio delle componenti; 5 g					
Implementazione delle Interfacce; 17 g						
20	21	22	23	24	25	26
Assemblaggio delle comp		Alfa test e correzioni; 7 g				

Mesi di Gennaio-Febbraio

lunedì	martedì	mercoledì	giovedì	venerdì	sabato	domenica
13	14	15	16	17	18	19
Implementazione delle componenti di controllo; 17 g		Assemblaggio delle componenti; 5 g				
Implementazione delle Interfacce; 17 g						
20	21	22	23	24	25	26
Assemblaggio delle comp		Alfa test e correzioni; 7 g				
27	28	29	30	31	01	02
Alfa test e correzioni; 7 g				Beta test e correzioni; 7 g		
03	04	05	06	07	08	09
Beta test e correzioni; 7 g						
10	11	12	13	14	15	16
Revisione finale; 7 g						
17	18	19	20	21	22	23
Revisione finale; 7 g						
24	25	26	27	28	01	02
03	04	05	06	07	08	09
10	11	12	13	14	15	16

Analisi dei requisiti

(Requirements Analysis Document)

1. INTRODUZIONE

1.1 Descrizione del sistema

Il sistema che si vuole realizzare ha l'obiettivo di ottimizzare la gestione di una libreria universitaria cercando di organizzare le informazioni che essa tratta nel miglior modo possibile. Ciò viene fatto per integrare alla esperienza ed alla conoscenza umana un programma che possa ordinare e trattare i dati e che possa fornire interessanti funzionalità di elaborazione di questi ultimi per suggerire strategie da utilizzare nell'attività di gestione stessa.

Il sistema dovrà gestire una libreria universitaria interfacciandosi con gli impiegati di questa per aiutarli nello svolgimento delle loro mansioni, quali:

- Gestione del magazzino.
- Acquisizione libri.
- Vendita libri.

1.2 SCOPO DEL SISTEMA

Lo scopo principale del sistema è quello di offrire una struttura gestionale ordinata, completa e semplice nella gestione delle attività principali svolte in una libreria. Si tratta in pratica di limitare al massimo le attività che portano ad un eccessivo consumo di tempo e che sono soggette ad errori.

Il tutto naturalmente si rifletterà sull'efficienza della libreria stessa.

L'obiettivo finale è dunque quello di fornire uno strumento software che permetta di avere sempre una visione completa e coerente di tutte le attività che hanno luogo nella libreria: dalla semplice ricerca di un libro alla situazione del magazzino ,dalle vendite all'amministrazione , facendo sì che le varie attività possano essere svolte in modo ordinato e senza che si intralcino a vicenda.

2. SISTEMA CORRENTE

Attualmente soltanto le grosse librerie possiedono un sistema software di gestione, spesso ridotto.

Questo si presenta abbastanza limitato e poco integrato rispetto ai vari servizi che la libreria offre.

Generalmente oltre a ciò i sistemi presenti hanno un aspetto poco amichevole e sebbene abbiano funzioni semplici da utilizzare risultano poco intuitivi e con interfacce non grafiche.

Molte piccole librerie inoltre non hanno affatto alcun sistema di gestione computerizzato, o questo è ridottissimo. Le piccole dimensioni non devono comunque far credere che l'utilizzo di un programma di gestione non apporti vantaggi notevoli. Anzi le ridotte dimensioni sono spesso causa di un lavoro confusionario dove ruoli e compiti si intrecciano e si fondono

provocando in genere una ridotta efficienza della struttura di lavoro. Manca inoltre la possibilità di integrare i dati della libreria con le informazioni esterne che sono pur necessarie a quest'ultima come ad esempio l'interazione dinamica con il mondo universitario che rappresenta il principale cliente di una qualunque libreria universitaria.

3. SISTEMA PROPOSTO

3.1 Panoramica generale

Il sistema che viene proposto si presenta come un prodotto appositamente studiato e realizzato per gestire una libreria universitaria.

Questo prodotto si concentra sulla necessità di separare i ruoli che vengono svolti all'interno di una libreria dai vari dipendenti, cercando di fornire a ciascuno di essi soltanto l'accesso a quelle funzioni che vengono usate per manipolare le parti del sistema a cui sono stati assegnati.

Si creano cioè tante diverse viste del sistema quanti sono i ruoli principali nella libreria.

Si permette così una gestione ordinata della libreria, concentrando tutte le energie di ciascun dipendente solo nelle attività di propria competenza, ed evitando altresì che si vengano a creare delle interferenze fra le varie attività.

A questa prima caratteristica di base si è affiancata anche la realizzazione di interfacce grafiche semplici ed intuitive al fine di rendere il prodotto molto amichevole.

3.2 Requisiti funzionali

Il sistema proposto ha lo scopo di poter offrire ai vari utenti che si servono di esso differenti funzioni di gestione della libreria.

Tali attività di gestione sono di vario tipo ed interessano i vari aspetti del lavoro e della struttura di quest'ultima.

Si è deciso pertanto di separare le funzioni realizzate ed offerte dal sistema in relazione all'ambito in cui queste vengono utilizzate. Tale separazione permette di realizzare un sistema strutturato in parti, e tale struttura garantisce in più una minore sensibilità agli errori, dato che essi sono comunque localizzabili in aree delimitate e corrispondenti alle differenti attività di gestione.

Il sistema fornisce le seguenti funzioni di gestione:

- ✓ Gestione Libri .
- ✓ Gestione del Magazzino.
- ✓ Gestione delle Vendite.
- ✓ Gestione Acquisti.
- ✓ Gestione Ordini.
- ✓ Gestione Prenotazioni dei clienti.
- ✓ Gestione Manageriale.

Analizziamo adesso queste funzioni che individuano i requisiti funzionali del sistema.

3.2.1 Gestione Libri

La libreria possiede un archivio permanente dove sono memorizzati tutti i libri.

In tale archivio vengono memorizzati non solo tutti i libri presenti nella libreria e nel magazzino, ma anche tutti quelli che la libreria è in grado di fornire ad un potenziale cliente che ne faccia richiesta.

Supponiamo che i libri che non sono memorizzati nell'archivio della libreria non vengono trattati dalla libreria stessa.

Per poter utilizzare in maniera appropriata tale archivio il sistema di Gestione Libri permette all'utente di poter :

- ◆ Aggiungere un libro all'archivio.
- ◆ Modificare le informazioni di un libro già presente nell'archivio.
- ◆ Eliminare un libro dall'archivio.
- ◆ Consultare e visualizzare l'archivio.

3.2.2 Gestione del Magazzino

La funzionalità di gestione del magazzino permette di conoscere lo stato costantemente aggiornato del magazzino, fornendo informazioni sulla disponibilità di ciascun libro in esso contenuto. La visione che si deve avere del magazzino deve essere dinamica cioè va automaticamente aggiornata quando lo stato del magazzino stesso viene modificato.

Il sistema permette di agire sul magazzino fornendo le funzioni di :

- Visualizza stato magazzino : permette di avere una visione sullo stato attuale di tutti i libri presenti nel magazzino.
- Registra consegna libri : permette di aggiornare il magazzino caricando i libri in arrivo alla libreria e relativi ad un ordine che la libreria ha emesso precedentemente.

3.2.3 Gestione della vendite

Il sistema di vendite della libreria deve essenzialmente tenere conto del fatto che un libro è stato venduto in un determinato giorno e quindi esso va eliminato dal magazzino.

L'addetto alle vendite deve pertanto avere modo di comunicare all'archivio che la quantità relativa ad un certo libro in seguito alla vendita è diminuita.

Inoltre può registrare che in una certa data è stato venduto un certo libro.

Il sistema permette di fare tutto ciò in maniera semplice : basta che l'addetto alle vendite immetta il codice ISBN(o l'ID del libro) , dopodiché

attraverso la funzione di vendita il sistema aggiornerà automaticamente i dati relativi a quel libro.

3.2.4 Gestione Acquisti

Questa funzionalità offerta dal sistema ricade nell'ambito della gestione magazzino in quanto l'unico modo per poter acquistare libri è quello legato alla emissione di ordini da parte della libreria nei confronti delle case editrici. (Questa funzione in realtà come quella di registrazione di nuovi libri viene considerata nell'ambito della gestione delle transazioni associate agli ordini e quindi nell'ambito della gestione degli ordini)

3.2.5 Gestione Ordini

Il sistema permette di gestire gli ordini che vengono emessi dalla libreria.

Le funzionalità che vengono offerte in questo ambito sono relative a:

- Creazione di un ordine;
- Registrazione degli ordini in un apposito archivio.
- Visualizzazione degli ordini.
- Modifica dello stato di un ordine.

La fase di creazione di un ordine tiene conto di tutte le prenotazioni che sono state fatte dai clienti.

Un ordine viene emesso dalla libreria ed ha come destinatario una casa editrice che fornisce i libri desiderati. Al momento della creazione un ordine diventa da evadere. A questo punto l'ordine può essere soddisfatto interamente da un'unica consegna di libri, ed in tal caso passa il suo stato a evaso oppure può essere soddisfatto da più consegne in tempi diversi, ed in tal caso prima diventa pendente poi evaso.

Il sistema potrebbe in questa fase interfacciarsi con una stampante per la stampa dell'ordine stesso oppure potrebbe usufruire di un servizio telematico di spedizione dell'ordine.(Parte non affrontata).

3.2.6 Gestione Prenotazioni clienti

Quando un libro viene richiesto da un cliente ma esso non è presente nella libreria allora egli può prenotarlo.

La prenotazione avviene rilasciando al cliente un biglietto contenente i dati relativi al libro che ha prenotato, al dipendente che ha effettuato l'operazione di prenotazione ed altre informazioni quali ad esempio la data di prenotazione e la possibile data di arrivo del libro presso la libreria.

Ogni ordine si riferisce ad un solo libro, quindi se un cliente vuole prenotare più libri deve effettuare più prenotazioni (tante quanti sono i libri diversi che vuole prenotare). Questa scelta si è fatta pensando che risulta più comodo in fase di scrittura degli ordini dividere le prenotazioni se queste si

riferiscono ad un solo libro, piuttosto che ad un insieme di libri, che potrebbero riferirsi a case editrici diverse.

Il preordine associato al cliente viene quindi memorizzato in un archivio, ed anche i dati del cliente vengono memorizzati al fine eventualmente di poter tenere traccia della clientela che la libreria ha.

3.2.7 Gestione manageriale

Col termine di gestione manageriale ci riferiamo a tutte le attività collegate alla libreria. Il manager è il responsabile principale della libreria e come tale può avere accesso a tutte le funzioni di gestione di questo. Ciò implica che può esaminare tutto il lavoro svolto dai vari reparti, e può accedere a tutte le funzionalità presenti nel sistema e separatamente utilizzate dai vari dipendenti della libreria.

Oltre a queste funzioni egli dispone di funzionalità proprie a cui soltanto lui può accedere. Quindi il manager (e la parte del sistema a lui dedicata) presenta le seguenti funzioni:

- Accesso alla funzione di supporto alle decisioni.
- Gestione degli accessi al sistema mediante l'utilizzo di password.
- Accesso alle funzioni di aggiornamento della libreria.¹
- Accesso alla funzione di storia acquisto-vendite.

¹ Si suppone che la libreria possieda nell'archivio tutti i libri che essa è in grado di fornire. L'eventuale introduzione di un nuovo libro può essere fatta solo dal manager attraverso questa funzionalità.

- Gestione delle informazioni relative ai corsi universitari ed utili per la libreria.
- Accesso a tutte le funzioni relative agli ambiti di gestione precedentemente elencati (gestione magazzino, gestione ordini, gestione preordini, gestione vendite, gestione acquisti, gestione libri).

Col termine di supporto alle decisioni si intende una funzionalità del sistema che permette di visualizzare tutte le informazioni necessarie per poter effettuare delle decisioni relative all'acquisto di libri da parte della libreria in un certo periodo dell'anno.

Tale funzionalità è a disposizione soltanto del manager il quale può inoltre impostare un criterio di suggerimento(funzione da implementare solo su richiesta del cliente) che viene svolto dal sistema e che tiene conto del numero di iscritti in un certo anno di studenti in un corso universitario e del numero di copie del libro richiesto da quel corso .

Il sistema nella totalità delle funzioni che offre può essere utilizzato soltanto dal manager. Gli altri dipendenti della libreria invece presentano delle viste ridotte. In questa maniera le varie viste del sistema sono associate ai vari dipendenti i quali sono responsabili soltanto della parte che a loro compete. Per evitare quindi che possano esserci accessi non consentiti ai dati del sistema dalle postazioni dei dipendenti si associa ad

ogni dipendente una password che li identifica quando questi entrano nel sistema e permette loro di utilizzare il programma in maniera esclusiva.

3.3 REQUISITI NON FUNZIONALI

3.3.1 Interfaccia utente

Il sistema utilizza delle interfacce grafiche di facile utilizzo ed aventi un aspetto gradevole. Chi utilizza il programma deve poter avere dinanzi a se una struttura grafica semplice, completa ed essenziale. Le informazioni presentate sullo schermo sono cioè in grado di indirizzare l'utente verso le funzionalità a cui esso desidera accedere, cercando di volta in volta di isolare soltanto le informazioni necessarie per lo svolgimento della funzione richiesta. Le interfacce saranno quindi diverse per i diversi membri della libreria. Ogni interfaccia presenterà solo le operazioni che competono all'impiegato per il quale il programma è stato realizzato.

Tali interfacce utilizzano una struttura grafica in cui sono presenti bottoni, finestre di dialogo, finestre scorrevoli ed aree di immissione dati.

Per il corretto utilizzo del programma si richiede che l'utente abbia dimestichezza con i sistemi ad interfacce grafiche.

3.3.2 Documentazione

La documentazione che viene fornita è quella relativa al progetto sviluppato.

Si presentano inoltre i documenti del progetto del database, ed il codice in Java sviluppato per il programma .

3.3.3 Considerazioni hardware

L'hardware richiesto per la realizzazione del sistema è quello necessario per la realizzazione di una piccola rete locale.

Sarà necessaria la presenza di una sistema al alte prestazioni che fungerà da server e che avrà il compito di ospitare e gestire il database della libreria.

Per quanto riguarda i computer presenti nelle postazioni dalle quali invece avverrà l'iterazione diretta fra i vari impiegati della libreria, il cliente ed il sistema non vi sono particolari necessità, dato che il lavoro di gestione viene svolto dal server.

Unica restrizione è la presenza presso ogni postazione di un dispositivo di puntamento (mouse) il quale permetterà di interagire velocemente con l'ambiente grafico delle interfacce, e di una tastiera per l'immissione dei dati.

3.3.4 Prestazioni del sistema

Non sono pretese particolari prestazioni per il sistema in termini di prontezza di risposta.

3.3.5 Gestione degli errori e tolleranza ai guasti

Ogni qualvolta una operazione di interrogazione o di aggiornamento dei dati sul database presenta un insuccesso, viene proposto all'utente un messaggio che notifica tale eventualità. Un messaggio di successo invece viene visualizzato in caso contrario(tranne che l'operazione effettuata porti alla visualizzazione di dati riportati dal db, in tal caso il successo è manifestato dalla visualizzazione stessa dei dati). In questo modo è possibile ripetere eventuali operazioni non andate a buon fine.

3.3.6 Sicurezza

Per quel che riguarda la sicurezza dei dati che il sistema tratta, essa viene garantita attraverso un sistema a password.

Ciascun utente del sistema, con esclusione del cliente, accede alle funzionalità solo dopo aver inserito la propria password, che è unica per ogni utente ed è assegnata dall'amministratore. Questo evita che possano avvenire accessi alle funzionalità da parte di persone che non sono state autorizzate precedentemente.

Le password sono gestite solo dall'amministratore che volendo le può cambiare. Dato inoltre che si suppone come ambito in cui il programma verrà utilizzato quello di una libreria ed essendo la rete di collegamento fra le varie postazioni ed il server locale, non vi sono problemi di accessi all'archivio dati da punti esterni alla rete stessa.

3.4 Pseudo-Requisiti

Il linguaggio scelto per implementare il programma è il Java.

Si è scelto tale linguaggio perché esso presenta dei requisiti particolari che bene si adattano per la realizzazione del programma, infatti:

- Permette di realizzare e gestire molto facilmente ed in maniera efficiente le applicazioni client-server.
- E' un linguaggio orientato agli oggetti.
- Presenza una buona pulizia stilistica.
- E' indipendente dall'architettura hardware e software sottostante (portabilità del codice grazie alla presenza della Java Virtual Machine).
- Facile interfacciamento con i database mediante JDBC.

- E' utilizzato per la progettazione orientata alla rete ed offre peraltro la possibilità di realizzare interfacce utente personalizzabili.

3.5 Modelli del sistema

3.5.1 Scenari

Rappresentazione degli scenari più comuni che possono presentarsi all'interno della libreria.

♣ Nome scenario : *ricerca cliente*

Attori partecipanti : - Marco: Cliente
 - Database

Flusso degli eventi :

1. Marco entra nella libreria universitaria G&R per comprare il libro di analisi matematica.
2. Vede un terminale, si avvicina e controlla se il libro che cerca è presente in quella libreria.
3. Immette il titolo del libro ed avvia la ricerca.
4. Dopo qualche secondo dall'archivio dati della libreria, il database, viene generata la risposta alla ricerca e vengono riportati i dati relativi al libro selezionato.

5. Marco controlla dal terminale questi dati e vedendo che il libro è quello adottato nel suo corso decide di acquistarlo.

♣ Nome scenario : *acquisto libro*.

Attori partecipanti :- Marco: Cliente

- Database
- Monica : Addetto alle vendite.

Flusso degli eventi :

1. Marco dopo aver scelto il libro da comprare si reca presso la cassa per acquistarlo.
2. Alla cassa Monica controlla dal suo terminale i dati del libro, interrogando l'archivio della libreria ,il database.
3. Monica dice a Marco il prezzo del libro che lo paga.
4. Quando il libro viene venduto, il sistema di gestione aggiorna automaticamente il numero di copie presente in magazzino .

♣ Nome scenario : *prenotazione*

Attori partecipanti : - Marco: Cliente

- database
- Mara :Addetto alle vendite

Flusso degli eventi :

1. Marco dopo aver fatto la ricerca di un testo si reca alla cassa perché lo vuole acquistare ma Mara, addetto alle vendite, gli comunica che non vi sono più copie disponibili. Marco decide allora di prenotare il libro e quindi alla cassa usufruisce del servizio di prenotazione.
2. Marco dice a Mara di voler prenotare il libro.
3. Mara accede dal suo terminale alle funzioni di prenotazione del sistema.
4. Verifica l'assenza del libro dal magazzino della libreria e compila il modulo di prenotazione per Marco.
5. Marco lascia un acconto .
6. Mara registra la prenotazione sul sistema di archiviazione ,il database.
7. Mara rilascia a Marco un biglietto che presenta : un codice relativo alla prenotazione, il nome e l'identificativo di Mara ,il giorno in cui presentarsi per ritirare il libro, ed altre informazioni.

♣ Nome scenario : *ritiro libro*.

Attori partecipanti : - Zaira : Cliente

- Database
- Valerio : Addetto alle vendite.

Flusso degli eventi :

1. Zaira si presenta al bancone della libreria dicendo di aver prenotato qualche giorno prima un libro.
2. Valerio ritira il biglietto di prenotazione ed accede alle funzioni di gestione delle prenotazioni dal suo terminale dopo aver digitato la sua password.
3. Controlla fra i preordini ancora aperti quello di Zaira.
4. Una volta trovato, aggiorna il preordine chiudendolo, ed aggiornando il magazzino in cui una copia del libro prenotato diventa adesso libera.
5. Accede a questo punto al servizio di vendita dei libri,
6. Aggiunge all'acconto la rimanenza per il prezzo del libro e vende il libro a Zaira aggiornando l'archivio dei libri disponibili presso la libreria.

♣ Nome scenario : *arrivo libri*

Attori partecipanti : - Claudio : Addetto al magazzino

- Luigi : Addetto agli ordini

- Database

Flusso degli eventi :

1. Luigi viene avvisato da Claudio dell'arrivo di nuovi libri.
2. Luigi accede al programma di gestione dal suo terminale dopo aver digitato la sua password.

3. Controlla i libri e l'ordine ad esso relativo ed effettua una transazione con la quale chiude l'ordine e carica i libri in magazzino che sono arrivati. Claudio dopo aver sistemato i libri controlla dal suo terminale la correttezza fra libri adesso presenti in magazzino e libri registrati nell'archivio.

♣ Nome scenario : *Amministra impiegati*

Attori partecipanti : - Walter : Manager

- Database
- Monica : nuova Responsabile Ordini.

Flusso degli eventi :

1. Monica viene assunta presso la libreria.
2. Walter, manager della libreria, accede al servizio di gestione della libreria, dopo aver introdotto la sua password.
3. Sceglie le funzionalità relative alla gestione degli impiegati.
4. Aggiunge i dati del nuovo impiegato, il suo ruolo ed assegna a Monica una password personale attraverso la quale essa può accedere in maniera esclusiva all'utilizzo del terminale da cui lavora e alle funzioni di gestione degli ordini di cui è la nuova responsabile.

▲ Nome Scenario : *Supporto alle decisioni*

Attori partecipanti : - Walter : Manager

- Database

Flusso degli eventi :

1. Walter vuole fare ordinare dei libri ma non sa bene in che quantità.
2. Accede allora al programma di supporto alle decisioni dal suo terminale.
3. Inserisce i dati relativi al corso in cui il libro è utilizzato.
4. Il sistema visualizza le informazioni relative al libro quali ad esempio:
 - se il libro è richiesto oppure è consigliato ;
 - il numero medio di studenti che ha frequentato il corso che adotta quel libro negli anni passati.
 - Il numero di copie del libro presente in magazzino.
5. Una volta visualizzate queste informazioni è in grado di poter scegliere quante copie del libro acquistare.
6. Walter utilizza inoltre la funzione di suggerimento del sistema che indica il numero di libri che si potrebbero comprare utilizzando una stima calcolata a partire dai parametri impostati.

♣ Nome scenario : *Aggiungi libro*

Attori partecipanti : - Walter : Manager

- Database

Flusso degli eventi :

1. Walter, manager della libreria, accede al servizio di gestione della libreria, dopo aver introdotto la sua password.
2. Sceglie le funzionalità relative alla gestione della libreria.
3. Accede alla funzione di aggiornamento della libreria.
4. Aggiunge i dati relativi ad un nuovo libro che la libreria inizierà a vendere e preme il bottone Aggiungi.
5. Il database riceve una richiesta di aggiungi elemento ed aggiunge una nuova voce nell' elenco dei libri della libreria.

♣ Nome scenario : *Nuovo Ordine*

Attori partecipanti: - Cassandra : Addetto agli ordini.

- Database

Flusso degli eventi :

1. Cassandra, addetto agli ordini della libreria, accede al servizio di creazione nuovo ordine per emettere un ordine verso una casa editrice.

2. Cassandra nella creazione degli ordini utilizza le informazioni relative ai libri che sono stati prenotati dai clienti.
3. Crea un nuovo ordine selezionando fra i preordini quelli che presentano libri che si riferiscono alla stessa casa editrice.
4. Compila l'ordine ed preme il bottone di registrazione dell'ordine.
5. Il database riceve una richiesta di registrazione ordine ed aggiunge una nuova voce nell'elenco degli ordini.

3.5.2 Modello dei casi d'uso del sistema

Si riportano di seguito i casi d'uso principali del progetto.

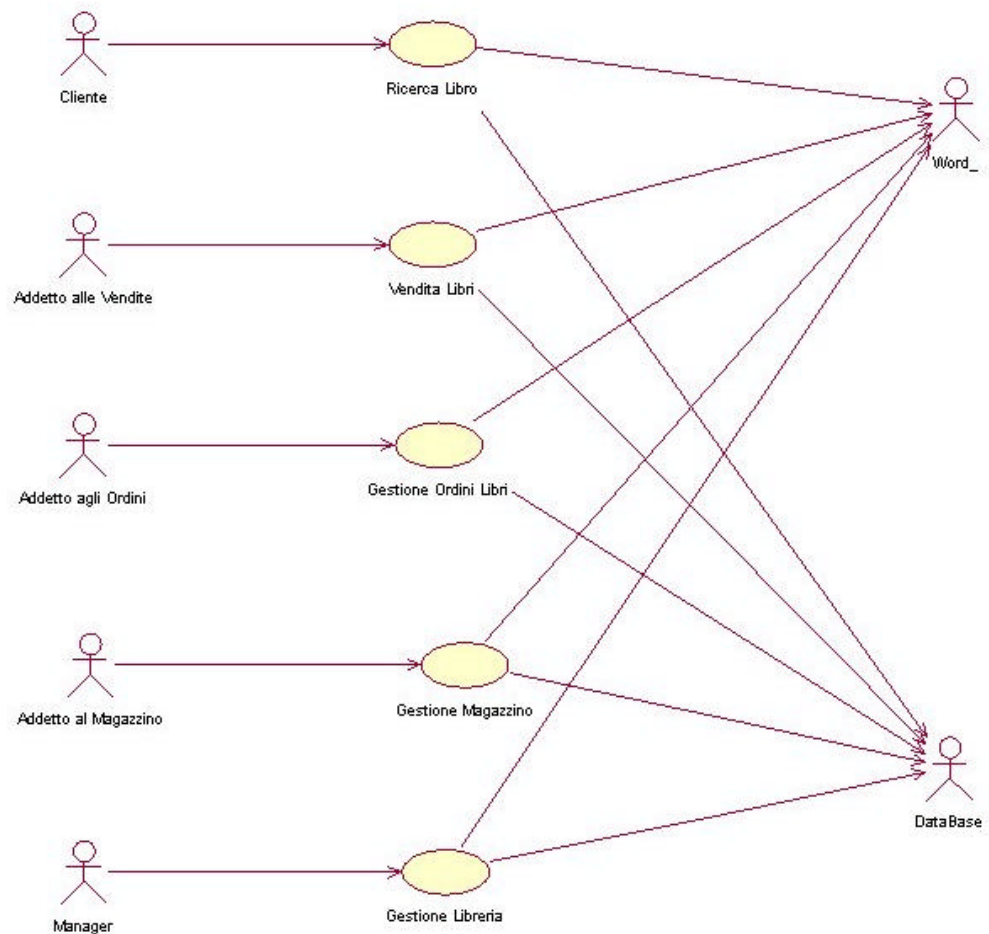
3.5.2.1 Attori

NOME	DESCRIZIONE
<i>Manager</i>	<i>Utente del sistema che ha accesso a tutte le funzionalità del sistema di gestione della libreria universitaria. E' il responsabile di tutta la libreria.</i>
<i>Addetto al magazzino</i>	<i>Utente del sistema che ha il compito di gestire il magazzino e che si occupa prevalentemente del controllo dello stato del magazzino</i>
<i>Addetto alle vendite</i>	<i>Utente del sistema che si occupa della vendita dei libri ,ma anche della compilazione di eventuali prenotazioni per il cliente.</i>
<i>Cliente</i>	<i>Cliente della libreria il cui solo accesso al sistema è quello per la consultazione dei libri che si trovano nella libreria.</i>
<i>Responsabile Ordini</i>	<i>Utente del sistema che ha il compito di creare e seguire gli ordini che la libreria spicca nei confronti di un fornitore.</i>
<i>Database</i>	<i>Questo attore interagisce col sistema dato che è sempre interrogato da questo per avere informazioni sui vari ambiti di gestione della libreria stessa.</i>
<i>Word</i>	<i>Questo attore viene chiamato dal sistema quando viene generato il file di log² che tiene conto di tutte le operazioni di collegamento ed interrogazione del database.</i>

² Si tratta di una funzione del sistema che permette di generare un file di testo in formato dat leggibile mediante word e che indica tutte le operazioni di accesso al database mediante la richiesta al server della libreria.

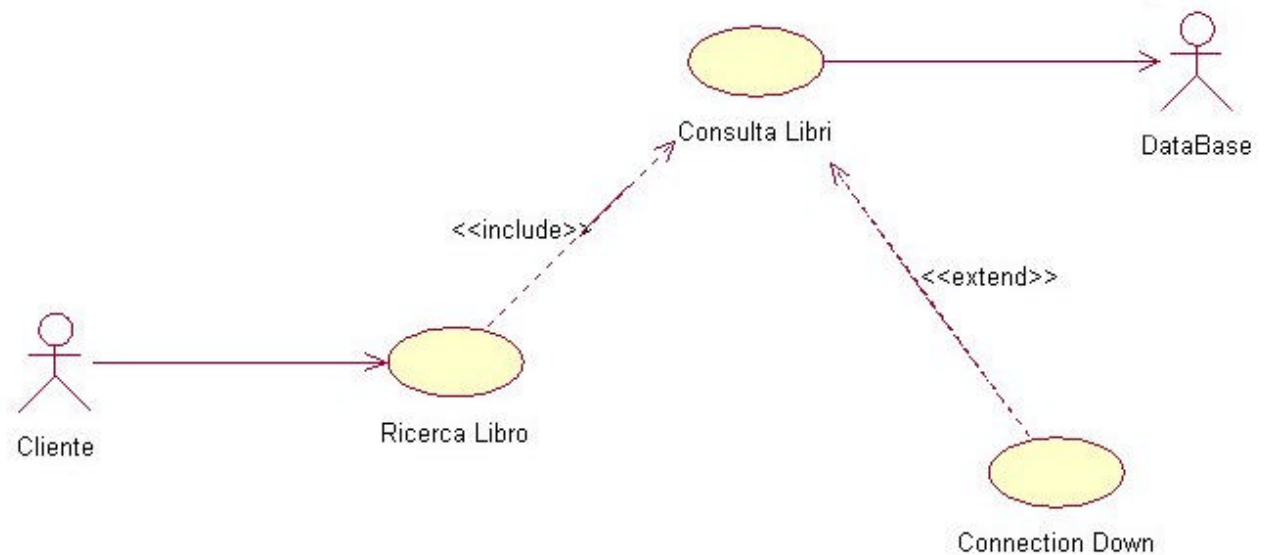
CASI D'USO DEL SISTEMA

3.5.2.2 Diagramma principale dei casi d'uso.



Il diagramma precedente fornisce una vista globale delle funzionalità che il sistema offre ai vari attori e di come questi interagiscano con il sistema.

3.5.2.3 Diagramma delle funzionalità offerte dal sistema al Cliente

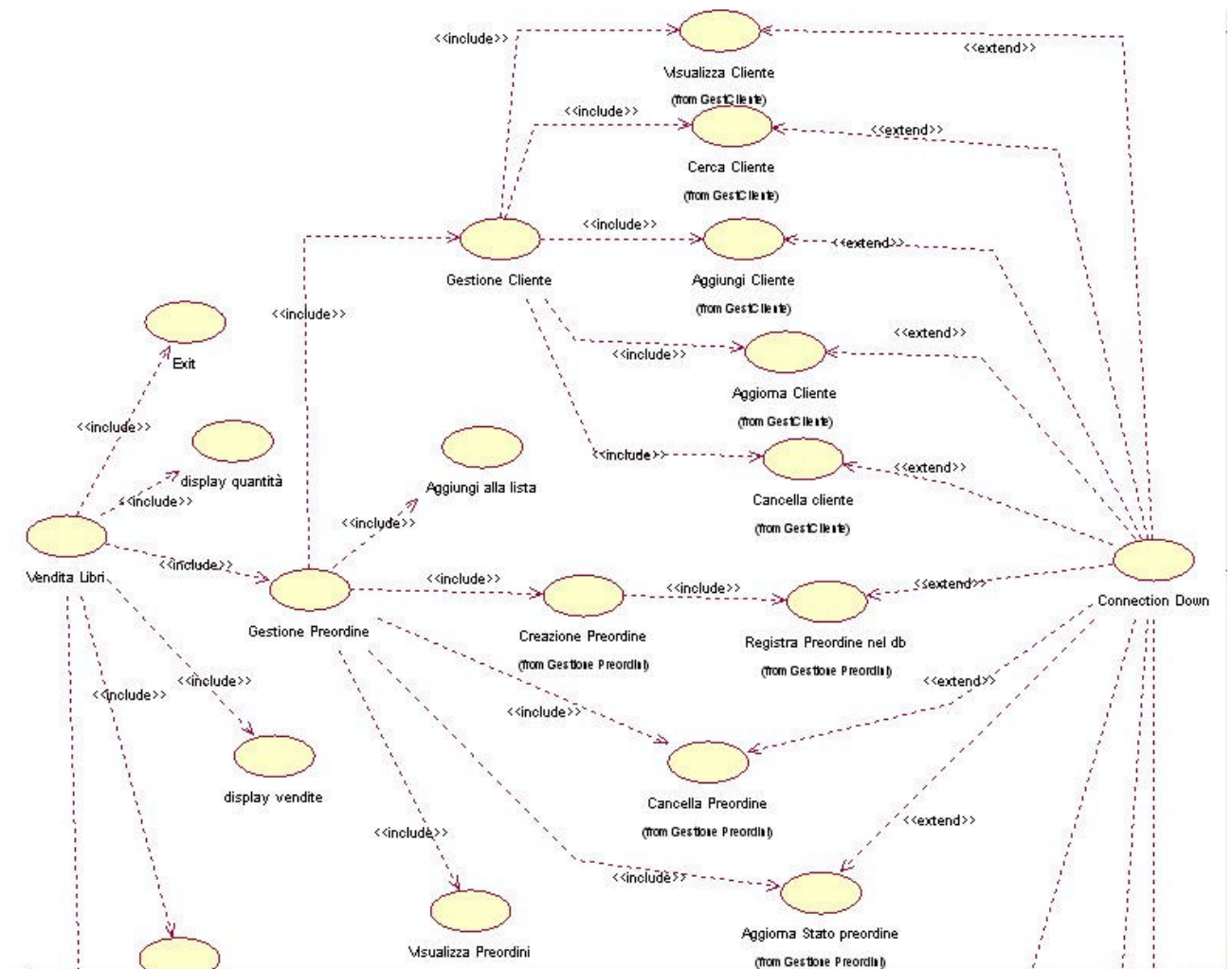


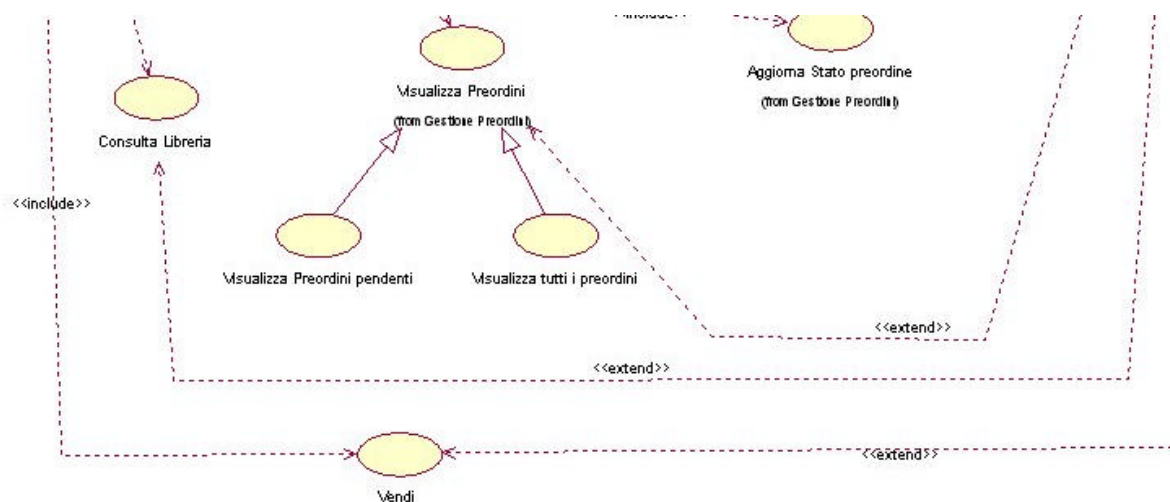
Nome del caso d'uso	Ricerca Libro
Attori coinvolti	Invocato da cliente, Comunica con Database.
Condizione di ingresso (entry condition)	Il cliente seleziona l'operazione ricerca dal terminale riservato al cliente.
Flusso degli eventi	1)- Il sistema risponde presentando una finestra che contiene tutti i tipi di ricerca che possono essere fatti. 2)- Il cliente sceglie il tipo di ricerca da effettuare. 3)- Il sistema visualizza un form contenete i campi in cui immettere i dati per effettuare la ricerca. 4)- Il cliente immette i dati ed avvia la ricerca.

	3)- Il sistema avvia la ricerca interrogando il database. 5)-Il database restituisce un elenco di libri che soddisfano ai dati usati per la ricerca.
Condizione di uscita (exit condition)	Il sistema visualizza la lista dei libri richiesti.

3.5.2.4 Diagramma delle funzionalità offerte dal sistema all'Addetto alle vendite

Il diagramma seguente mostra l'insieme dell funzionalità che vengono offerte all'addetto alle vendite.





Descriviamo di seguito le principali funzioni che l'addetto alle vendite può usare.

Nome del caso d'uso	Vendita Libri
Attori coinvolti	Invocato da Addetto alle Vendite, comunica con Database.
Condizione di ingresso (entry condition)	L'addetto alle vendite seleziona fra le funzioni a sua disposizione quella di Vendita libri.
Flusso degli eventi	<ol style="list-style-type: none"> 1)- Il sistema risponde presentando una un form da riempire contenete i dati del libro che viene venduto. 2)- L'addetto alle vendite riempie il form inserendo codice ID del libro e controlla il numero di copie di quel libro presente in magazzino mediante Visualizza quantità. 3)-Completa poi il form relativo alla vendita del libro inserendo oltre al ID libro, il suo ID anche la quantità venduta. Poi pressa il bottone Vendi.

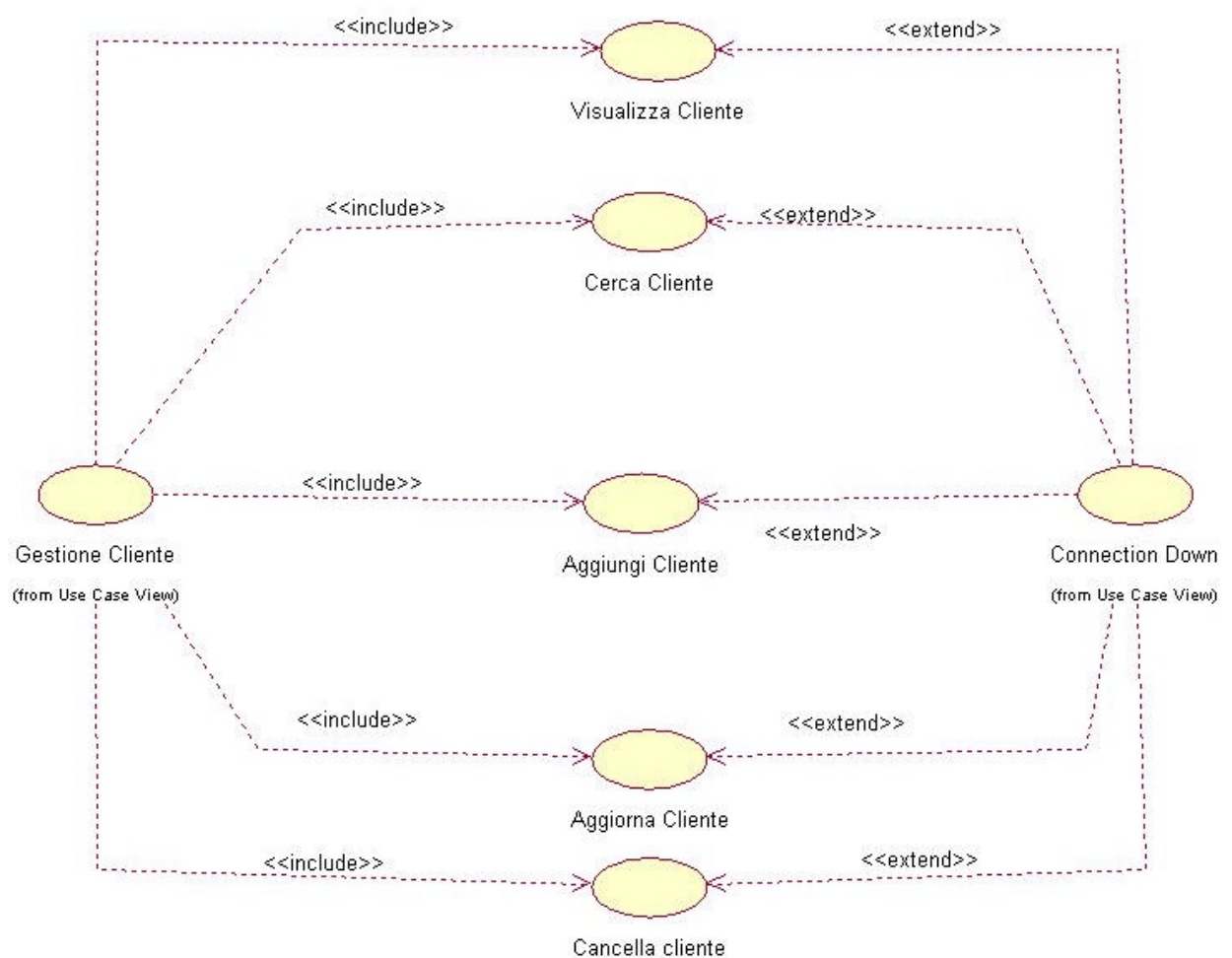
	4)- Il sistema invia una richiesta di aggiornamento al database, che decrementa il numero di copie di quel libro presente in magazzino.
Exit condition	Il sistema ritorna alla finestra principale relativa all'addetto alle vendite.

Nome del caso d'uso	Preordine
Attori coinvolti	Invocato da Addetto alle Vendite, comunica con Database.
Condizione di ingresso (entry condition)	L'addetto alle vendite seleziona fra le funzioni a sua disposizione quella di Preordine per registrare il fatto che un cliente vuole prenotare un libro presso la libreria non attualmente presente.
Flusso degli eventi	<p>1)- Il sistema risponde presentando una un form da riempire contenete i dati del libro che deve essere prenotato ed un form relativo ai dati del cliente che fa la prenotazione .</p> <p>2)- L'addetto alle vendite riempie il form Dati Cliente inserendo i vari dati (se si tratta di un nuovo cliente altrimenti accede ai dati del cliente consultando il database mediante Visualizza Dati Cliente) .</p> <p>3)Preme il bottone di Aggiungi al Database.</p> <p>4)-Completa poi il form relativo al Preordine. Poi pressa il bottone Aggiungi al</p>

	Database. 5)- Il sistema invia una richiesta di inserimento al database, che inserisce i nuovi dati . 6)- Il sistema visualizza una finestra che indica la corretta esecuzione delle operazioni sul db.
Exit condition	Il sistema ritorna alla finestra principale relativa all'addetto alle vendite.

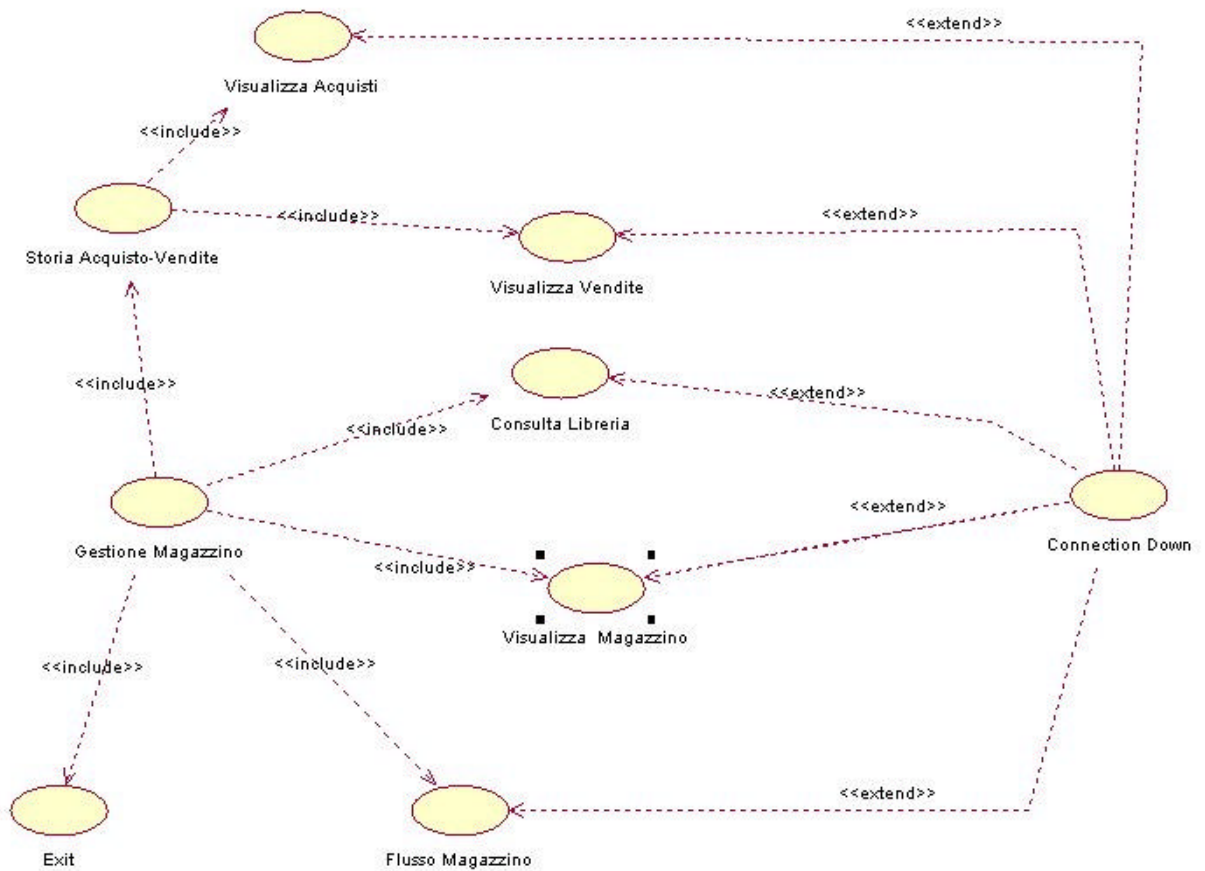
Gli altri use case inclusi fra le funzionalità a disposizione vengono utilizzati in maniera simile a quelli precedenti presentati (Vendita libri e Preordine).

Visualizziamo con più dettaglio le funzionalità di Gestione del Cliente.



Nome del caso d'uso	Connection down
Attori coinvolti	Indica una situazione eccezionale di caduta del collegamento fra le varie postazioni della libreria e il server
Condizione di ingresso (entry condition)	Si verifica una caduta di collegamento fra server ed altre postazioni.
Flusso degli eventi	<ol style="list-style-type: none"> 1)- Il sistema presenta una finestra che avvisa sul'impossibilità di poter effettuare operazioni sul db 2)- L'utente(uno fra i vari attori) legge il messaggio e preme ok. 3)-Si cerca poi di risolvere il problema.

3.5.2.5 Diagramma delle funzionalità offerte dal sistema all'Addetto al Magazzino



Nome del caso d'uso	Visualizza Magazzino
Attori coinvolti	Invocato da Addetto al Magazzino, comunica con Database.
Condizione di ingresso (entry condition)	L'addetto al magazzino sceglie la funzione di aggiornamento del magazzino.(Visualizza Magazzino) inclusa nella funzione Gestione Magazzino.

Flusso degli eventi	<p>1)-Il sistema invia una richiesta al database</p> <p>2)- Il sistema risponde presentando una finestra in cui vengono fornite tutte le informazioni relative ai libri presenti in magazzino</p> <p>2)- L'addetto al magazzino preme torna al menù principale.</p>
Exit condition (condizione di uscita)	Viene visualizzata la finestra principale..

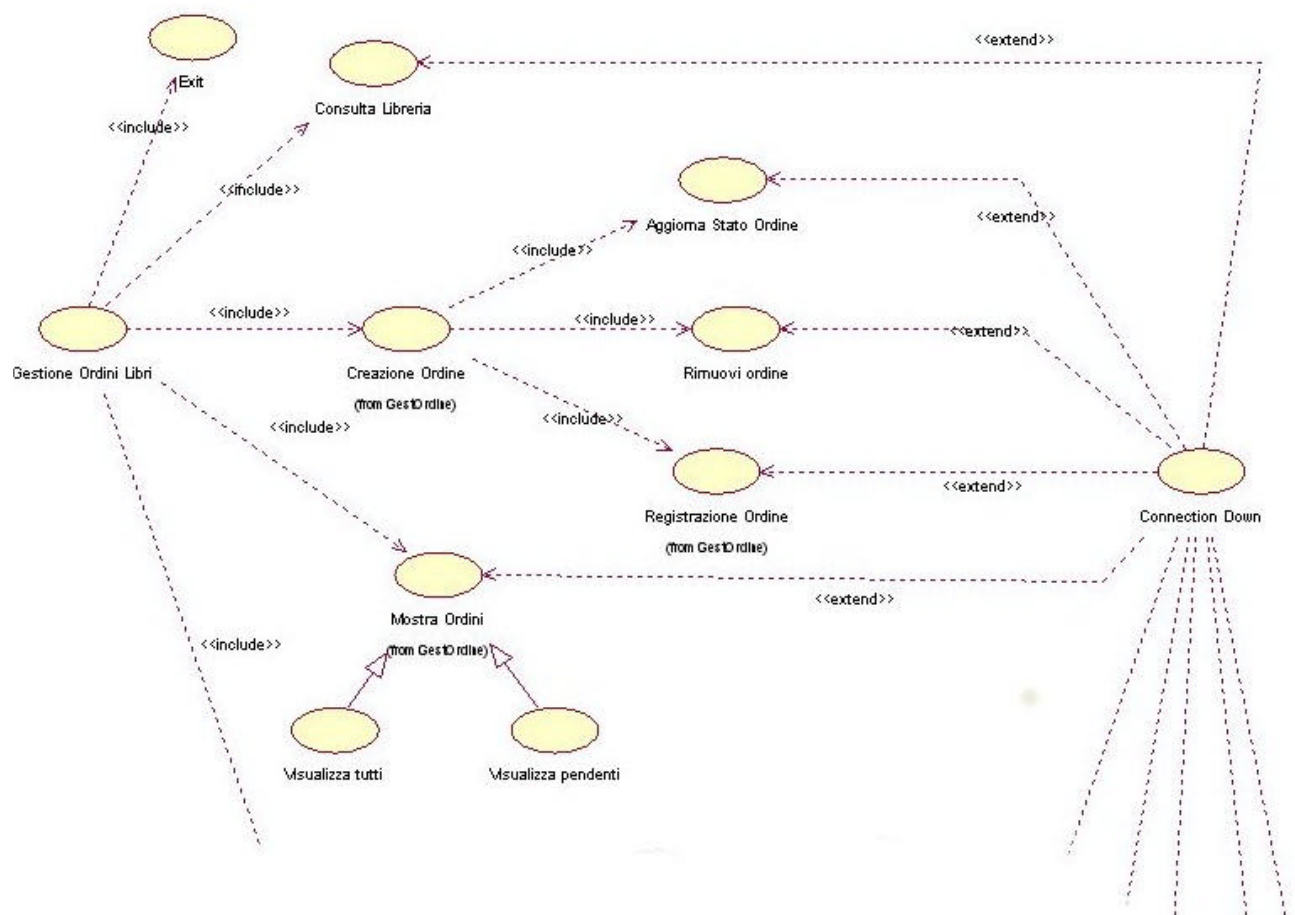
Consulta Libreria: uguale a Ricerca Libro.

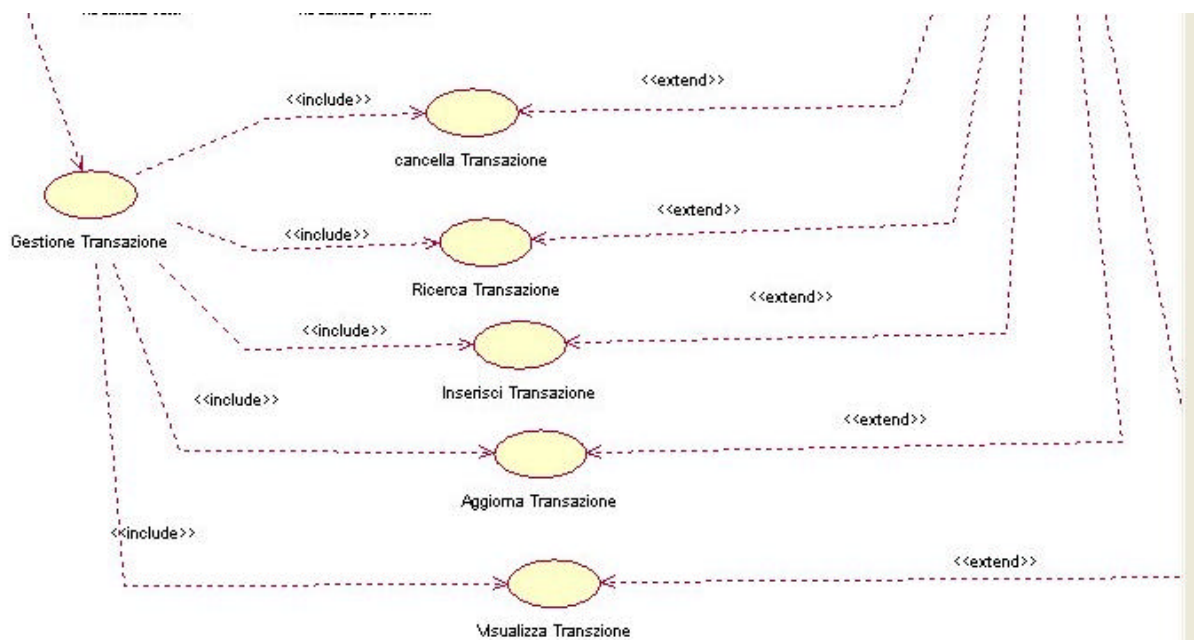
Nome del caso d'uso	Storia Acquisto Vendite
Attori coinvolti	Invocato da Addetto al Magazzino, comunica con Database.
Condizione di ingresso (entry condition)	L'addetto al magazzino sceglie la avere informazioni circa l'arrivo e la vendita di un libro in magazzino dalla funzione generale Gestione Magazzino.
Flusso degli eventi	<p>1)- Il sistema risponde presentando un form dove inserire i dati relativi al libro di cui si vuole visualizzare la storia.</p> <p>2)- L'addetto al magazzino riempie il form e poi accede alla funzione di storia(delle vendite o degli acquisti)</p> <p>3)-Il sistema invia una richiesta al database.</p> <p>4)- Il sistema visualizza i dati che ha recuperato sul</p>

	Database.
Exit condition (condizione di uscita)	L'addetto al magazzino pressa il bottone torna al menu principale e ritorna alla finestra principale.

Use case Flusso Magazzino : permette di avere una visione relativa agli arrivi in magazzino di un certo libro. Il suo funzionamento è simile agli use case precedenti.

3.5.2.6 .Diagramma delle funzionalità offerte dal sistema all'Addetto agli ordini



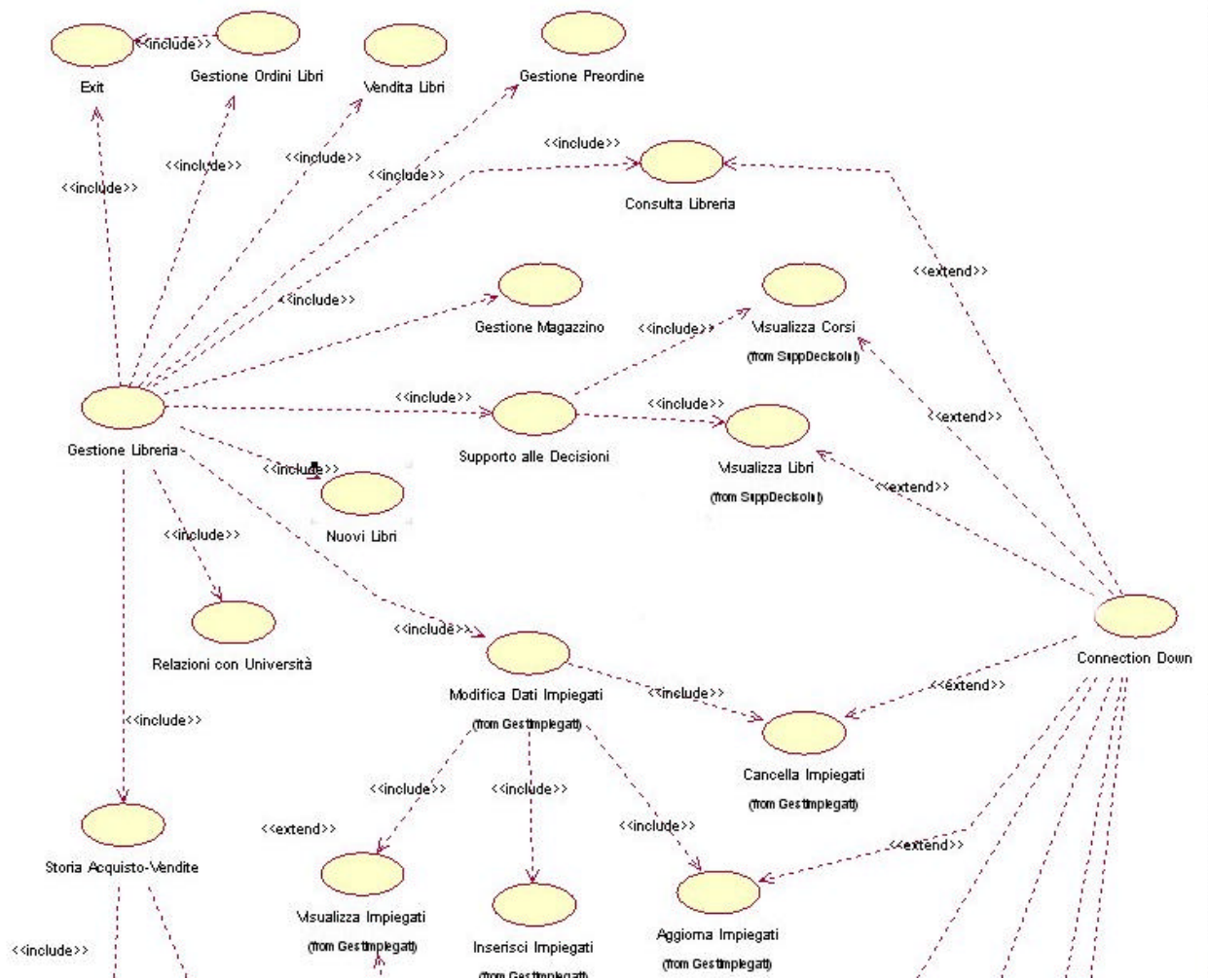


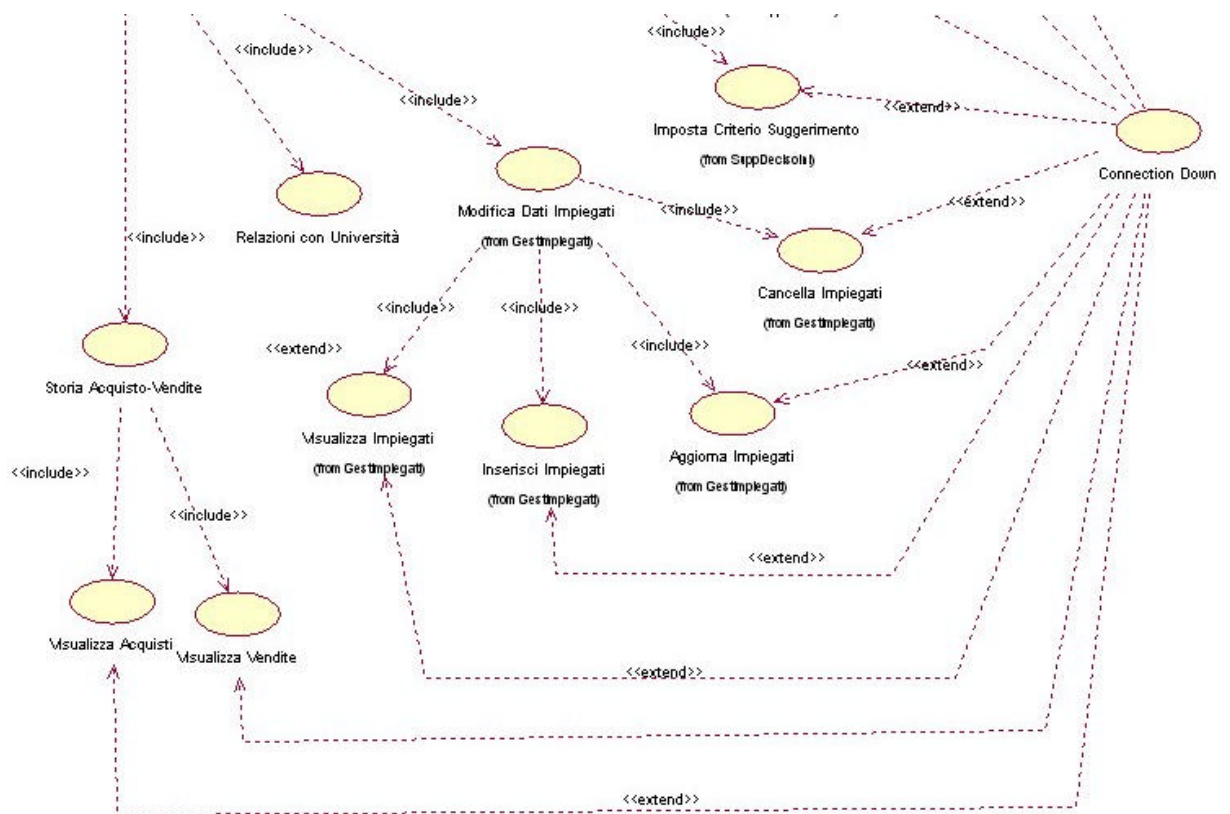
Nome del caso d'uso	Creazione ordine
Attori coinvolti	Invocato da Addetto agli ordini, comunica con Database.
Condizione di ingresso (entry condition)	L'addetto agli ordini usa la funzione Ordina.
Flusso degli eventi	<ol style="list-style-type: none"> 1)- Il sistema risponde presentando una finestra contenente tutte le informazioni e le funzioni necessarie per gestire gli ordini. 2)- L'addetto agli ordini inserisce i dati relativi all'ordine che vuole creare. 3)- Poi sceglie la funzione di creazione di un ordine. 4)-Il sistema invia l'ordine al Database che lo registra. 5)-Il sistema visualizza una finestra indicante il buon esito

	<p>dell'operazione.</p> <p>6) L'addetto agli ordini chiude questa finestra .</p> <p>7) L'addetto agli ordini poi preme il tasto di Ritorna al menù principale.</p>
Exit condition	Il sistema ritorna alla finestra principale.

Nome del caso d'uso	Visualizza transazione
Attori coinvolti	Invocato da Addetto agli ordini, comunica con Database.
Condizione di ingresso (entry condition)	L'addetto agli ordini usa la funzione Transazioni.
Flusso degli eventi	<ol style="list-style-type: none"> 1)- Il sistema risponde presentando una finestra contenente tutte le informazioni e le funzioni necessarie per gestire le transazioni. 2)- L'addetto agli ordini usa la funzionalità visualizza transazioni. 3)- Il sistema invia una richiesta al db. 4)-Il sistema visualizza le informazioni che ha trovato sul db. 5) L'addetto agli ordini poi preme il tasto di Ritorna al menù principale.
Exit condition	Il sistema ritorna alla finestra principale.

3.5.2.7 Diagramma delle funzionalità offerte dal sistema all'Amministratore





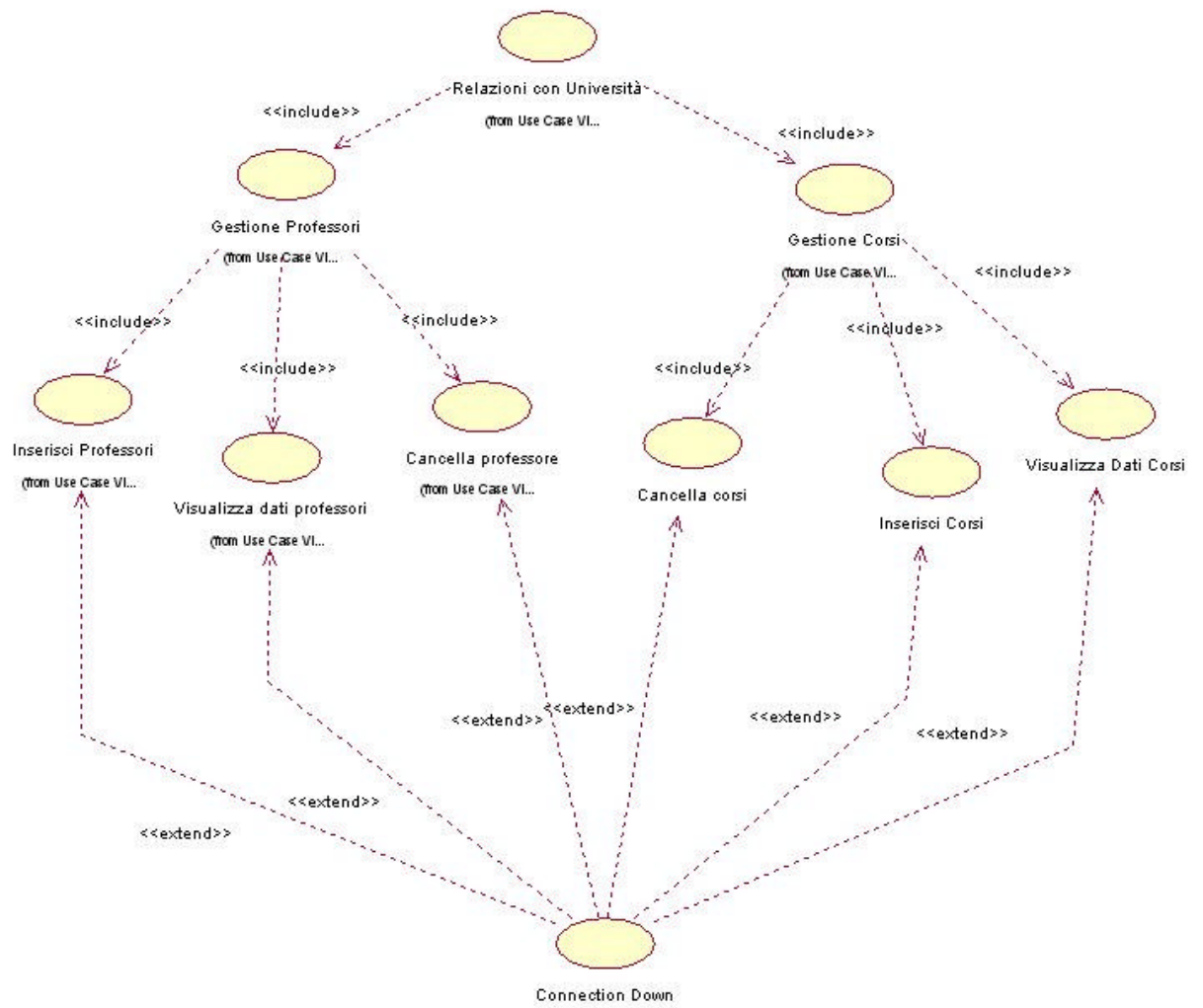
Nome del caso d'uso	Supporto alla decisioni
Attori coinvolti	Invocato dal Manager comunica con Database.
Condizione di ingresso (entry condition)	L'addetto al Magazzino seleziona fra le funzioni a sua disposizione quella di supporto alle decisioni
Flusso degli eventi	1)- Il sistema presenta un form dove immettere i dati. 2)- Il Manager immette i dati e poi preme il tasto di suggerimento. 3)-Il sistema interroga il database ed elabora i dati. 4)-Il sistema mostra i risultati al manager. 5) Il manager dopo aver osservato i risultati preme il bottone per tornare al menu principale.

Exit condition	Il sistema ritorna alla finestra principale .
----------------	---

Nome del caso d'uso	Nuovo libro
Attori coinvolti	Invocato da Manager , comunica con Database.
Condizione di ingresso (entry condition)	Il Manager accede alla finestra principale delle funzioni offerte dal sistema e specifiche per lui.
Flusso degli eventi	<p>1)- Il sistema presenta una finestra in cui scegliere le varie funzioni di gestione della libreria.</p> <p>2)- Il Manager sceglie l'operazione che intendere effettuare dalla finestra.</p> <p>3)-Il sistema visualizza la finestra relativa alla funzione di gestione scelta dal manager.(inserimento di un nuovo libro nella libreria).</p> <p>4)-Il manager seleziona l'operazione da effettuare.</p> <p>5)- Il sistema comunica col database e lo aggiorna</p> <p>6)- Il sistema visualizza un messaggio col quali indica il buon esito dell'operazione svolta.</p> <p>7)-Il manager osserva i risultati riportati e pressa il bottone Torna Indietro.</p>
Exit condition	Il sistema ritorna alla finestra principale, dove sono visualizzate le funzioni principali.

Nome del caso d'uso	Aggiorna dati impiegati
Attori coinvolti	Invocato Manager comunica con Database.
Condizione di ingresso (entry condition)	Il manager accede dal suo menù principale alla funzione di modifica dati impiegati.
Flusso degli eventi	<p>1)- Il sistema visualizza le operazioni di gestione dei dati degli impiegati (Visualizza Impiegati, Aggiorna Impiegati, Modifica Impiegati), insieme ad un form dove inserire i dati.</p> <p>2)-Il manager inserisce i dati relativi all'impiegato di cui vuole cambiare i dati e poi preme aggiorna.</p> <p>3)- Il sistema accede al database ed effettua la modifica.</p> <p>7)- Il sistema visualizza il buon esito dell'operazione.</p> <p>8)- L'amministratore preme il bottone di ok.</p>
Exit condition	Il sistema ritorna alla finestra principale relativa alla modifica dati impiegati.

Mostriamo di seguito l'insieme delle funzionalità che vengono offerte dal sistema all'Amministratore in relazione alla gestione dei dati universitari utili alla libreria.

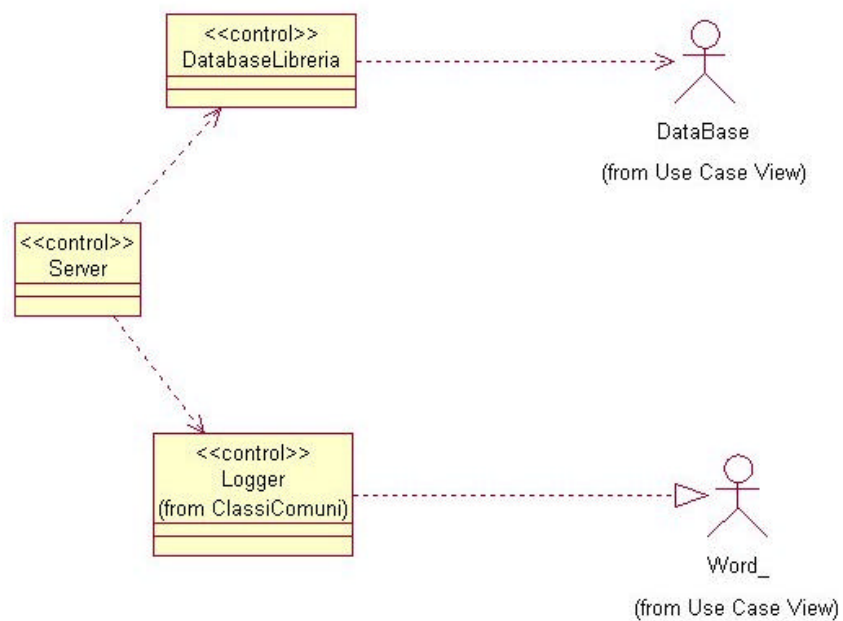


3.5.3 Modello Oggetto

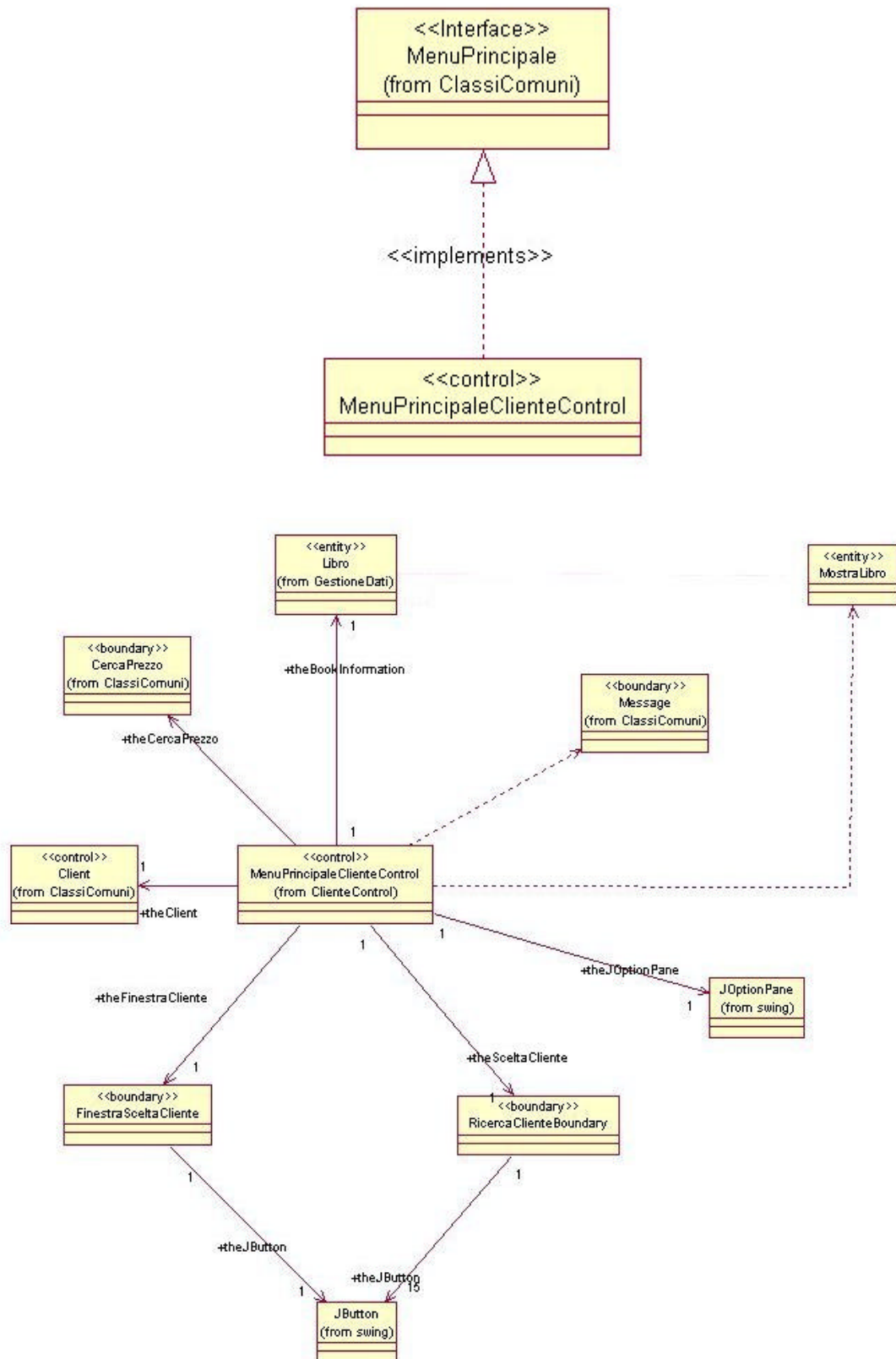
Presentiamo di seguito i diagrammi delle classi del sistema

3.5.3.2 Class diagram

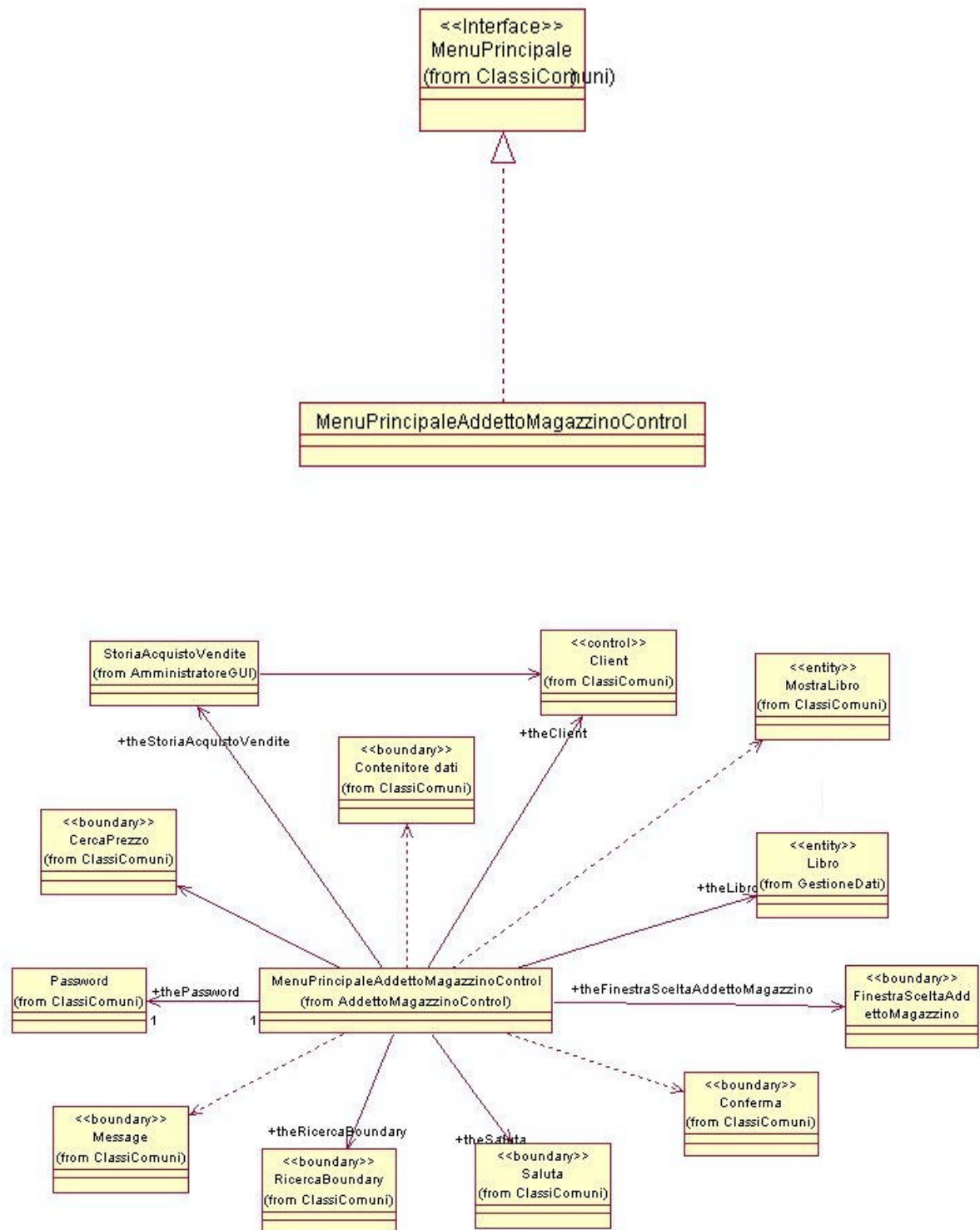
3.5.3.2.1 Diagramma delle classi residenti nel server



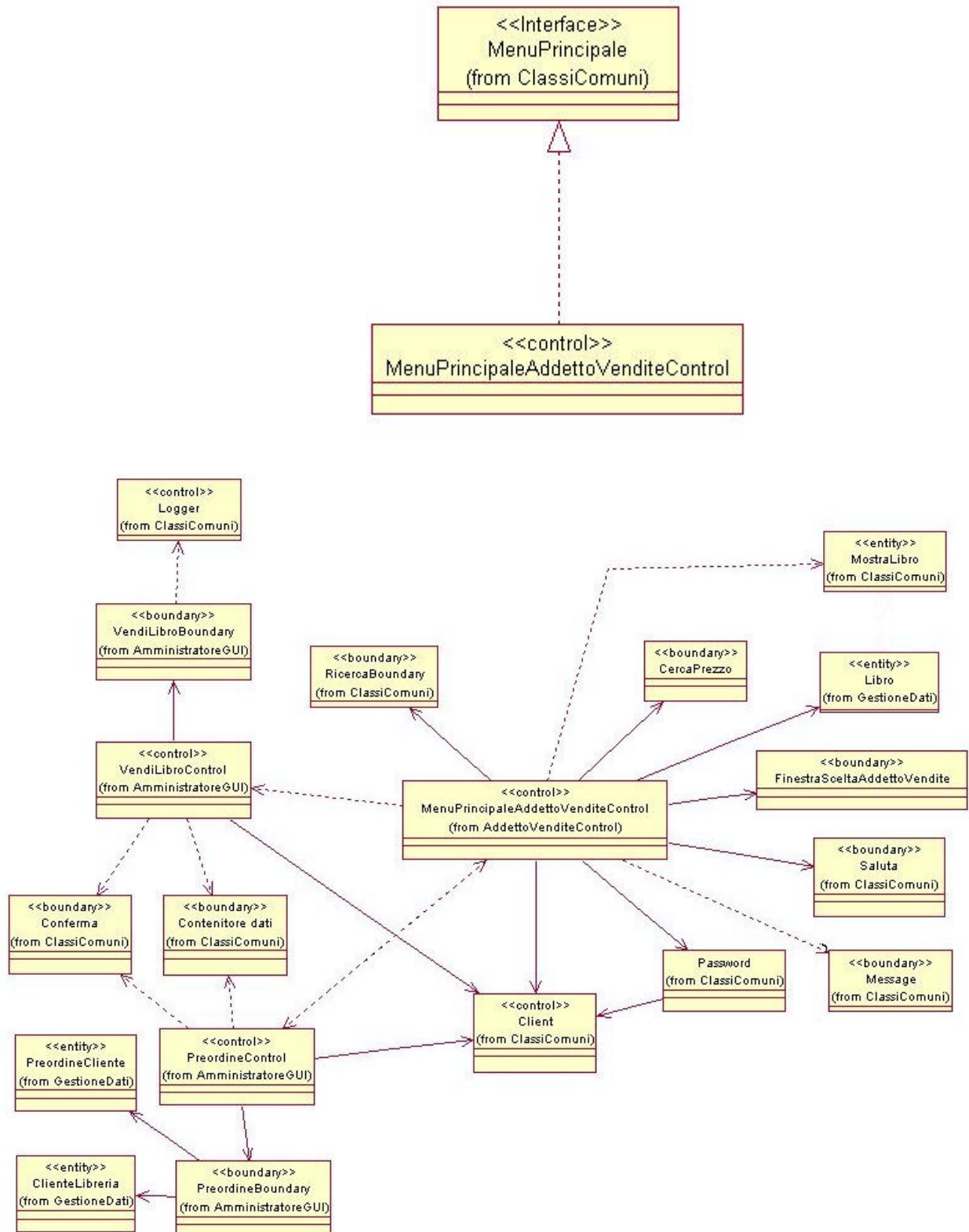
3.5.3.2.2 Diagramma delle classi che implementano le funzionalità accessibili dal cliente



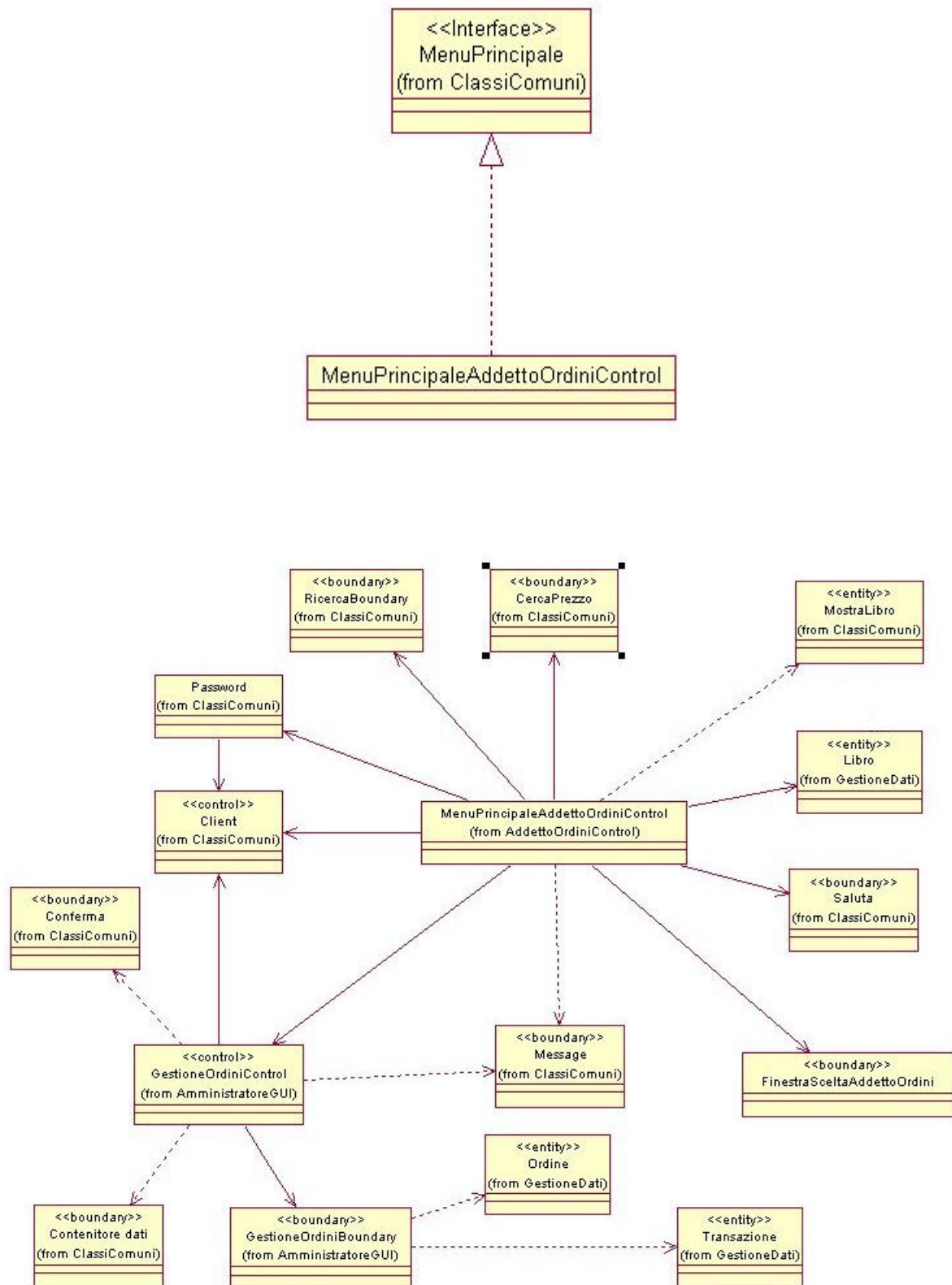
3.5.3.2.3 Diagramma delle classi che implementano le funzionalità accessibili dall'Addetto al magazzino



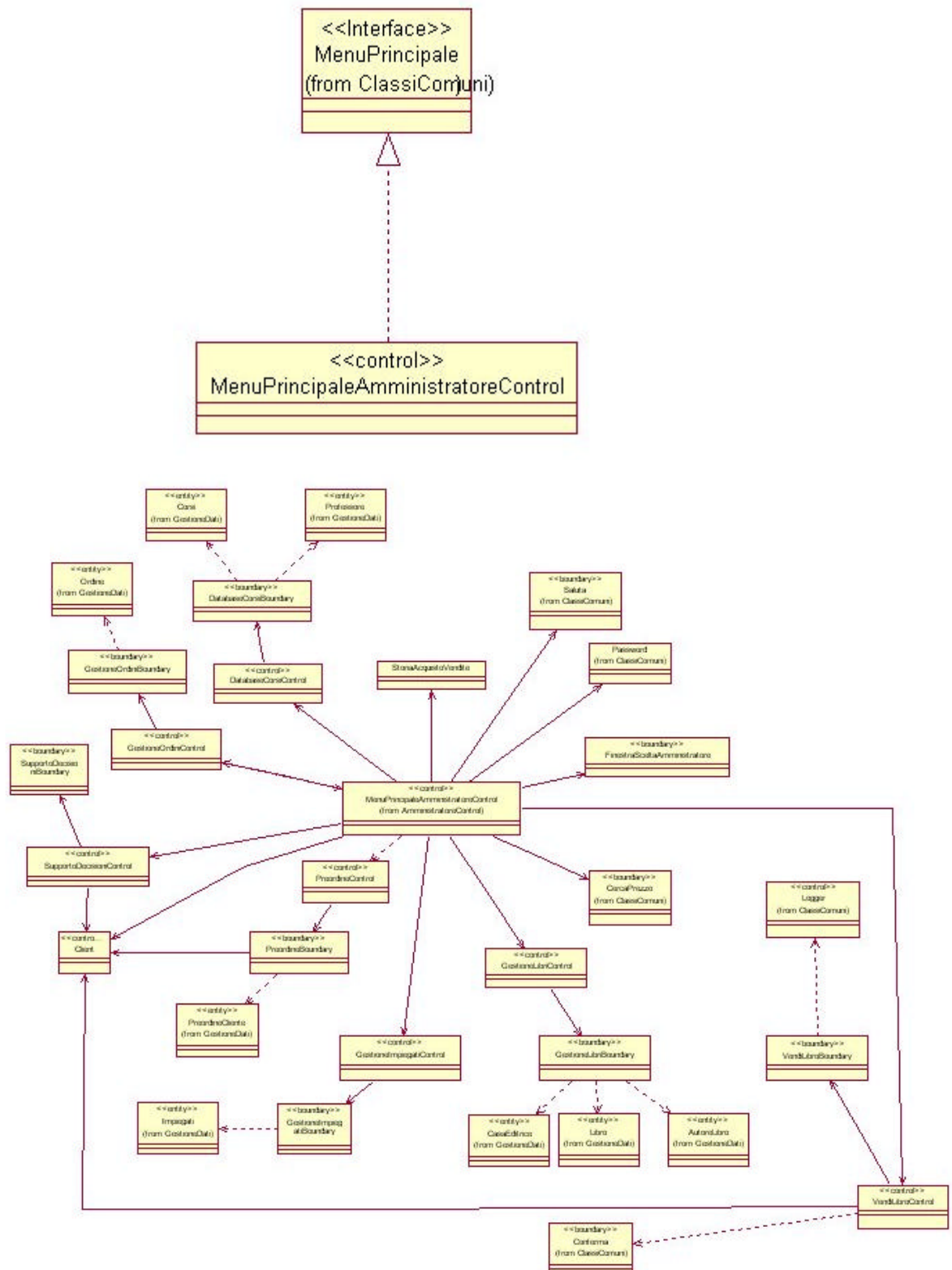
3.5.3.2.4 Diagramma delle classi che implementano le funzionalità accessibili dall'Addetto alle vendite

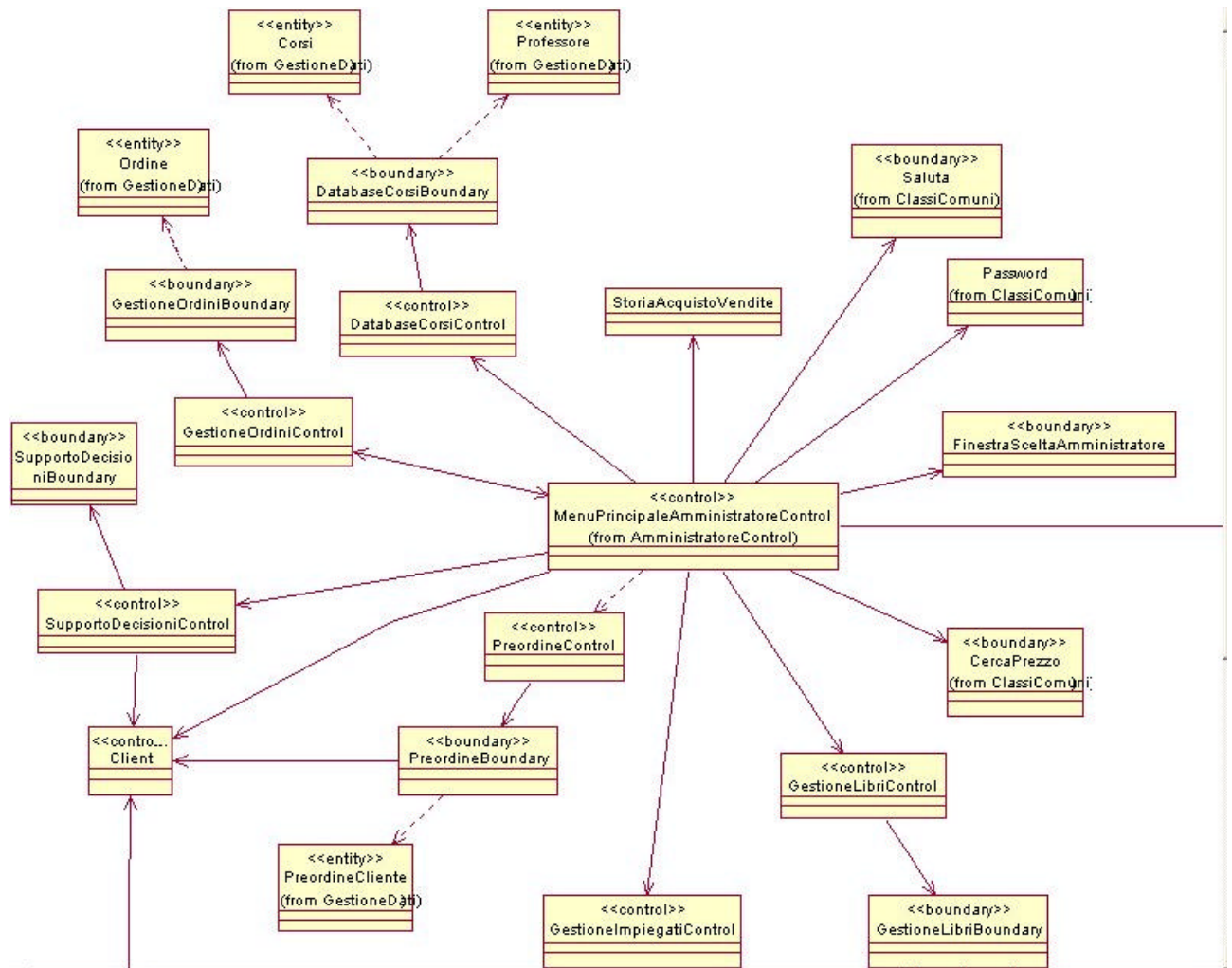


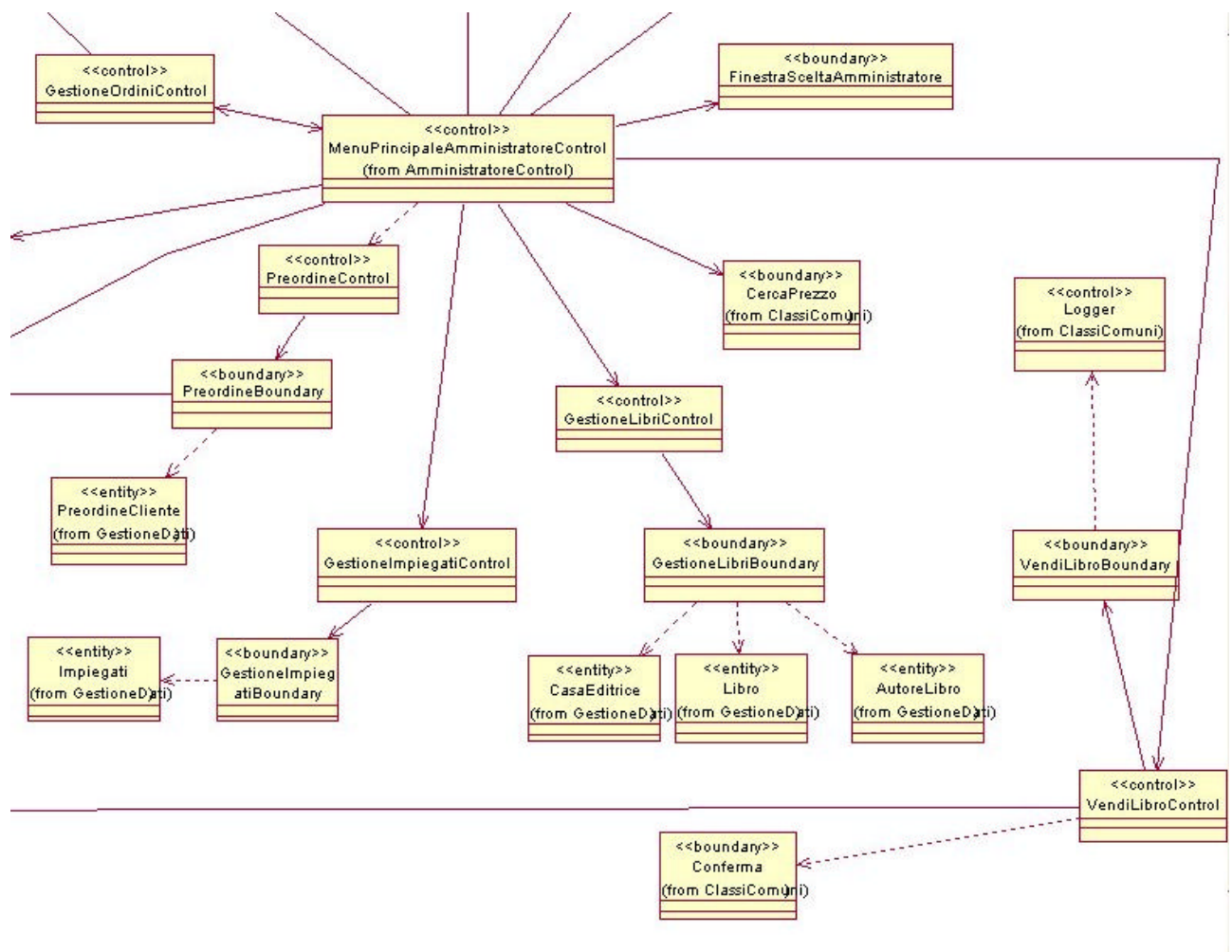
3.5.3.2.5 Diagramma delle classi che implementano le funzionalità accessibili dall'Addetto agli ordini



3.5.3.2.6 Diagramma delle classi che implementano le funzionalità accessibili dall'Amministratore







Nota: i diagrammi delle classi sono stati visualizzati per comodità senza mostrare gli attributi e le operazioni che implementano.

Tali informazioni sono tuttavia presenti come si può osservare consultando il file :

NuovaLibreriaUniversitariaG&R2.mdl

3.5.3.3 Progetto concettuale del Database

La libreria per essere gestita necessita di una struttura dati dove vengono archiviate tutte le informazioni.

Tale archivio deve contenere le informazioni relative a :

- libri;
- case editrici che pubblicano e trattano questi libri;
- corsi che utilizzano questi libri;
- professori che tengono corsi nei quali vengono usati i libri trattati dalla libreria universitaria;
- ordini che vengono emessi dalla libreria;
- prenotazioni che vengono fatte dai clienti;
- dipendenti che lavorano all'interno della libreria.

Tutte queste informazioni devono essere opportunamente archiviate e collegate fra loro.

Riportiamo di seguito il progetto concettuale ,rimandando al documento SDD per quel che riguarda il progetto logico e al documento ODD per la descrizione delle classi che permettono l'accesso ai dati. Il modello dei dati che è stato utilizzato è di tipo relazionale.

Identificazione delle entità e delle relazioni

In questa fase procediamo alla descrizione dell'archivio ed identifichiamo i principali elementi che in esso si relazionano.

(utilizziamo la seguente notazione per individuare le entità, gli attributi di una entità e le relazioni fra entità: **entità** ,**attributo**, **verbo che rappresenta una relazione fra entità**).

In una libreria universitaria l'elemento fondamentale che viene trattato sono i libri.

Un libro è caratterizzato da un **titolo** e da un codice identificativo (**codice ISBN**).

Il libro inoltre è pubblicato in un certo anno(**anno pubblicazione**),e di un libro ci possono essere un numero **vario di edizioni** (tascabile, economica ecc..) come pure il **numero delle edizioni** pubblicate.

Ogni libro ha un proprio **numero di pagine**, un **prezzo**, uno **sconto** ed una **collocazione** all'interno della libreria. Inoltre per la gestione è importante conoscere il **numero di copie** presenti in magazzino (numero **totale**, di copie **prenotate,libere** , **vendute** ed **acquistate**).

Ogni libro infine **appartiene** ad una specifica **categoria**.

Ogni libro tratta una certa **categoria** di argomenti che **viene trattata** in un certa **facoltà**.(ad esempio un libro tratta elettronica, categoria di informazione che viene affrontata nella facoltà di ingegneria elettronica).

Ogni libro è **scritto** da uno o più **autori**, ed un autore in genere scrive più libri.

Ogni autore ha un **nome**, un **cognome** e si possono avere delle **note** relative a questo autore.

Ogni libro viene **pubblicato** da una **casa editrice**.(una casa editrice può pubblicare più libri ed un libro può venire pubblicato da più case editrici).

Ogni casa editrice ha un **nome**, un **indirizzo**,la **città** dove si trova,la **provincia**,il **C.A.P.**,il **paese** (nazione) dove opera ,un **numero di telefono**, un **numero di fax**, un indirizzo di posta elettronica (**e-mail**).

Al fine inoltre di poter avere un archivio che tenga conto della storia degli acquisti e delle vendite di un libro dobbiamo tener presente che :

- ogni libro è **venduto** in un certo **giorno**;
- ogni libro è **acquistato** in un certo **giorno**;

La libreria tratta inoltre i libri che vengono utilizzati nei corsi universitari.

Ogni libro è **usato** in uno o più **corsi universitari** e più libri possono essere usati da un corso universitario. Ogni **corso** viene identificato da un **nome**,viene svolto in un certo **semestre** ed è relativo ad un certo **anno universitario**(I,II,III,...anno) e si **tiene** in un certo **anno**.

Ad ogni anno possiamo associare il **numero di studenti** che si sono iscritti per quell'anno a quel corso.

Ogni corso è **tenuto** da un **professore** e un professore può tenere più corsi.

La libreria potrebbe avere dei contatti con i professori per avere sui libri che adotta o che vuole adottare in un certo anno. Per tenere conto di queste informazioni osserviamo che:

ogni professore ha un **nome**, un **cognome**, un **numero di telefono** ed un indirizzo di posta elettronica (**e-mail**).

Quando un libro viene prenotato inoltre avremo che :

un libro è **associato** ad un **preordine**(una prenotazione del cliente);

un preordine ha informazioni quali : il **numero di copie** che viene **richiesto** per quel libro, il **prezzo** di quel libro,il **subtotale**(cioè il prezzo complessivo di tutte le copie prenotate), **l'acconto** lasciato dal cliente, la differenza fra acconto e prezzo complessivo(**rimanenza**), la **data di prenotazione**, lo **stato** della **prenotazione** stessa (da evadere oppure evaso).

Un preordine si **riferisce** ad un **cliente** ed ogni preordine è **creato** da un **impiegato** della libreria.

Ogni cliente ha un **nome**, **cognome**,**indirizzo**, un riferimento alla **città** in cui vive, un **CAP**,un **numero di telefono** ed una **e-mail**.

Ogni impiegato invece ha : un **nome**, un **cognome**,un **ruolo** all'interno della libreria(manager ,addetto alle vendite,addetto al magazzino,addetto agli ordini), un **reparto** in cui lavora, un **numero di interno** a cui poterlo trovare,una **password** che gli permette di utilizzare il programma.

Ugualmente quando si fanno gli ordini :

un libro fa **parte** di una **transazione** che è **relativa** ad un **ordine** che viene **spiccato** nei confronti di una casa editrice.

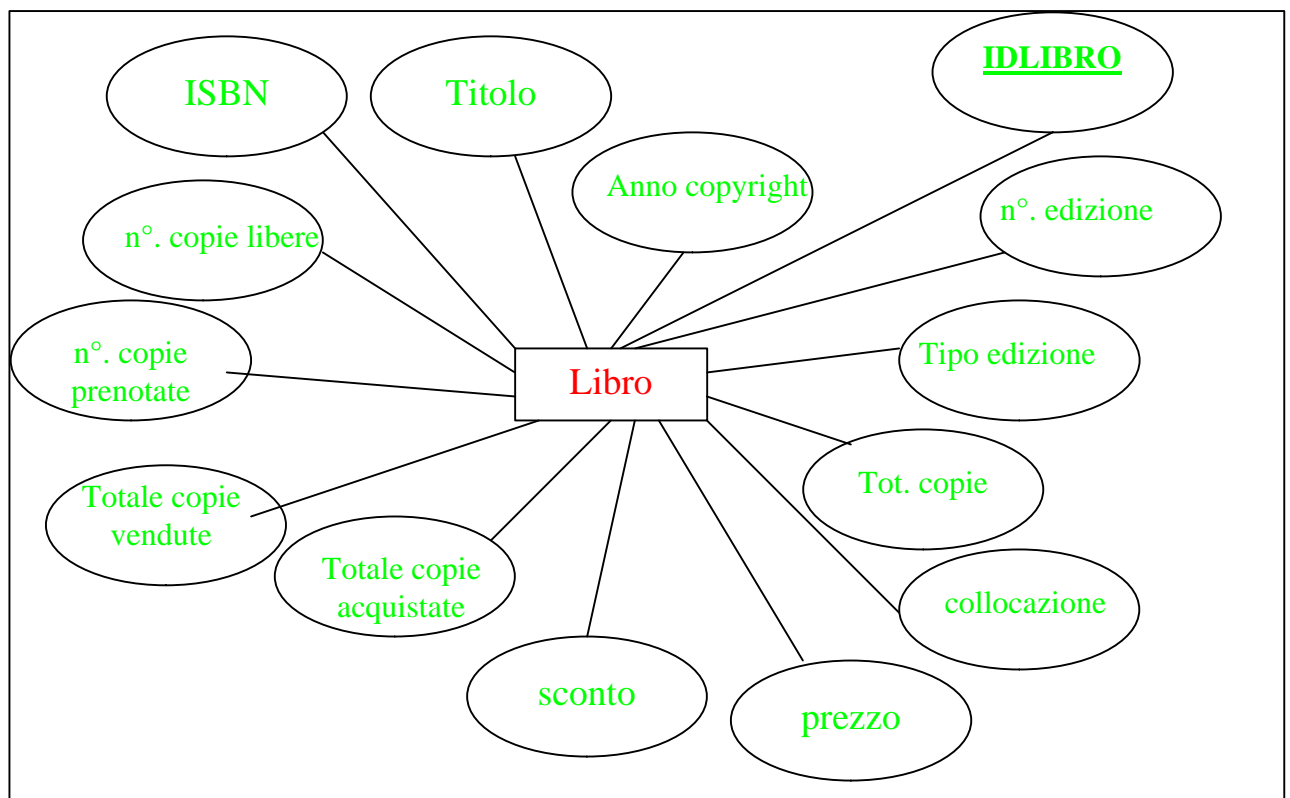
Ogni transazione ha una **data**, presenta un riferimento alla **quantità richiesta** per un certo libro, a quella **arrivata** ed a quella che ancora deve arrivare (**rimanente**).

Ogni ordine presenta una **data** relativa a quando esso viene **creato**, una **data** relativa a quando viene **spedito**, un **numero identificativo** per quell'ordine ed uno **stato** relativo al fatto che esso sia da evadere, pendente o evaso

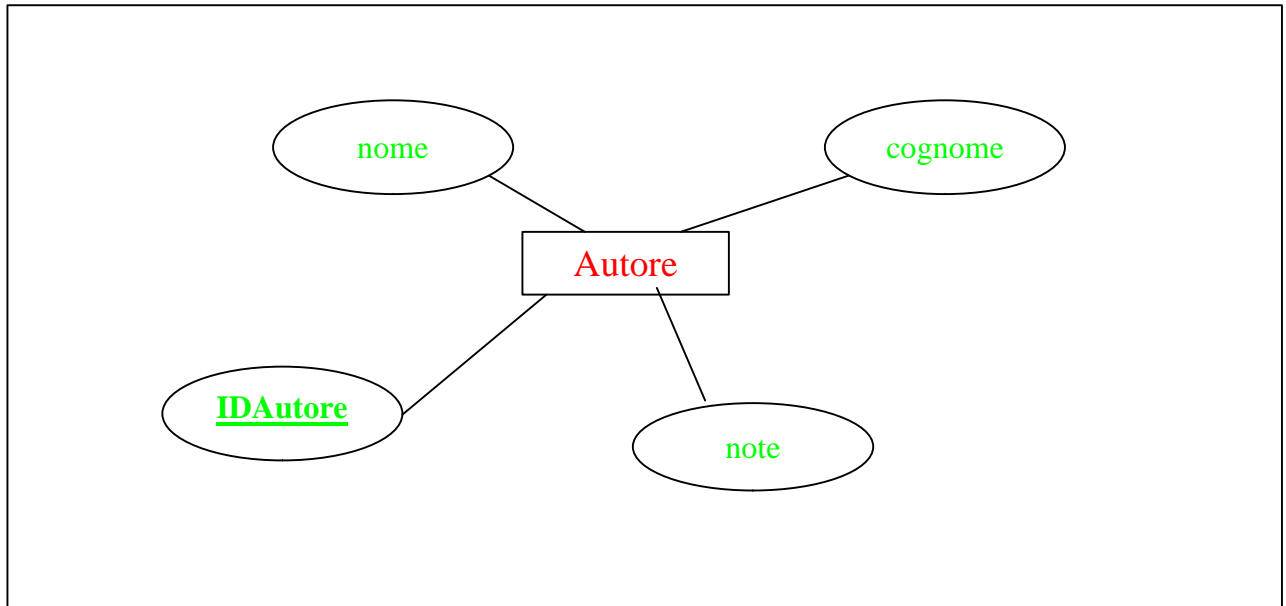
Diagrammi Entità Relazioni (ERD)

Passiamo adesso alla definizione astratta e generica dei dati e delle loro relazioni mediante l'utilizzo dei diagrammi ERD.

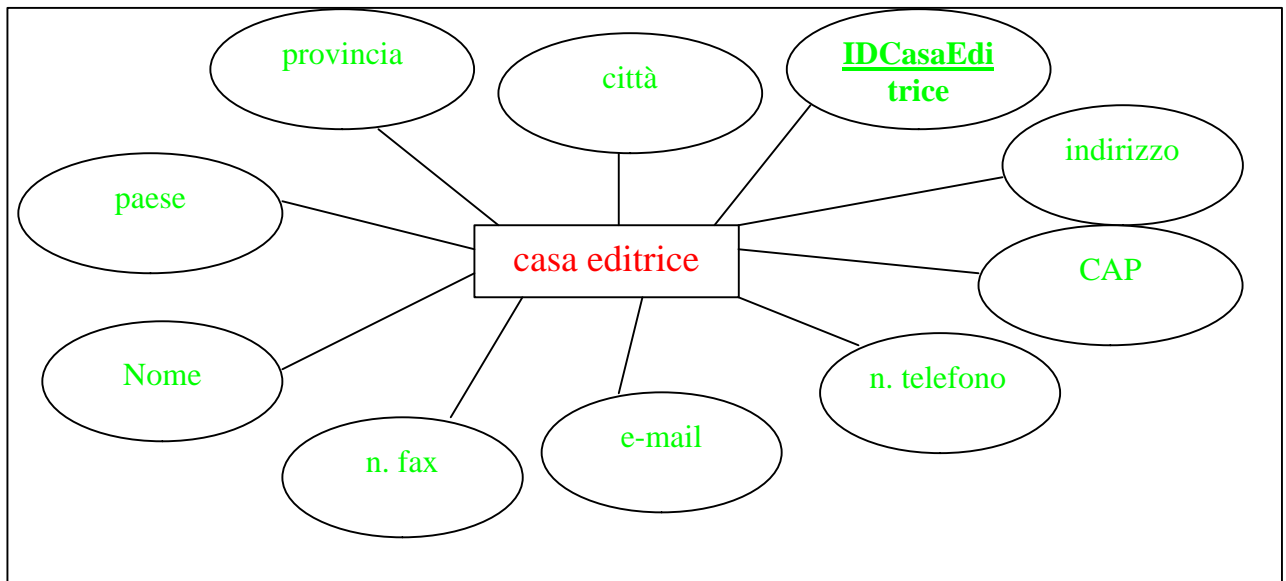
Entità *Libro* con attributi:



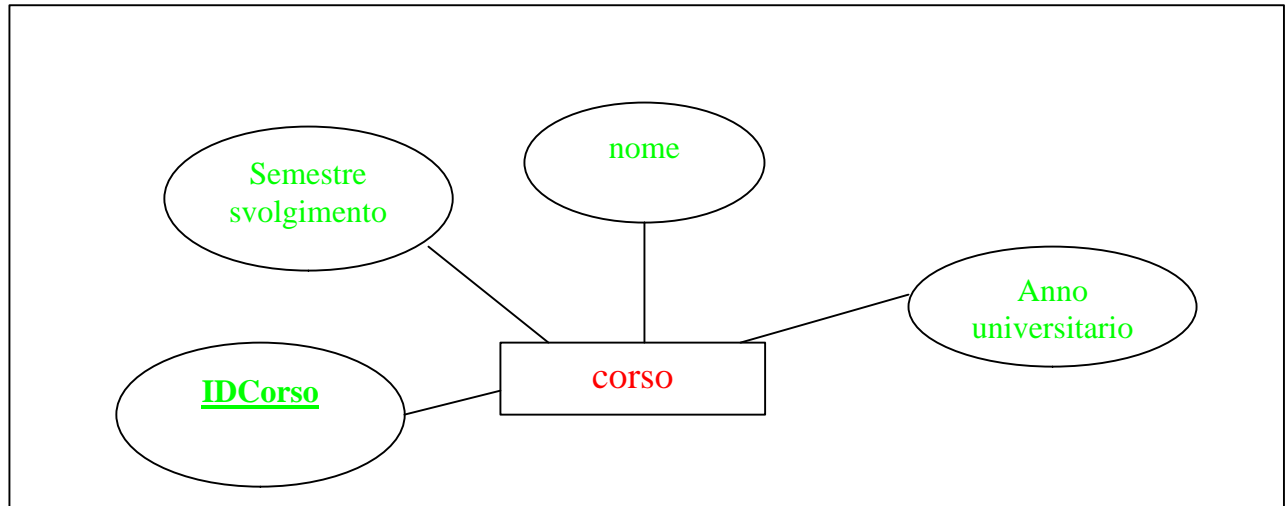
Entità **autore** con attributi:



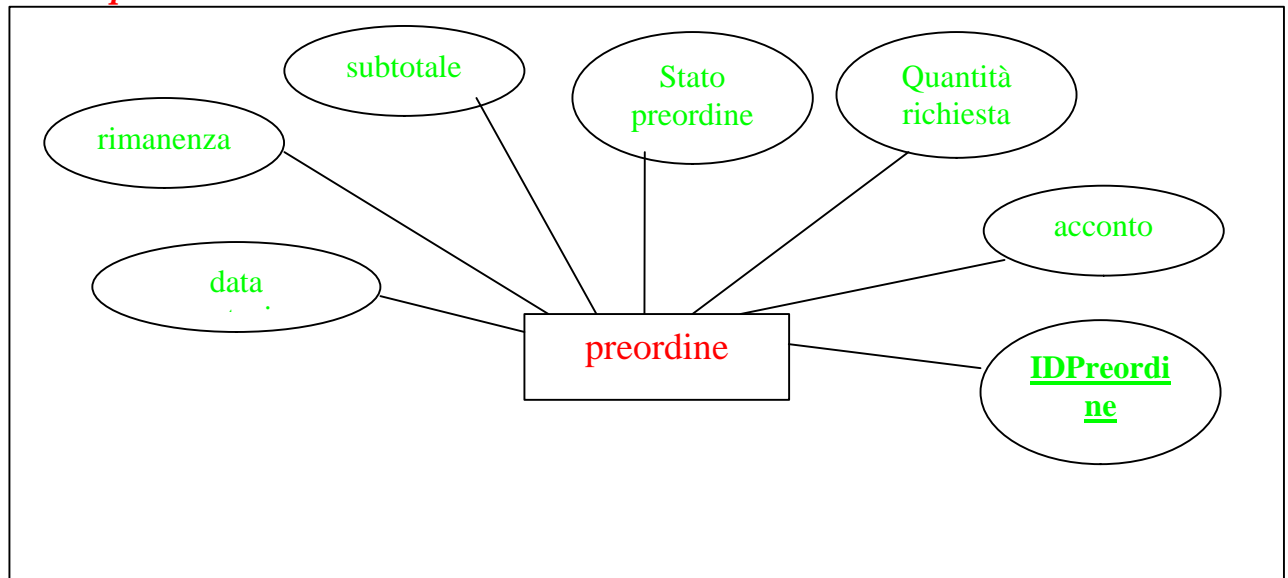
Entità **casa editrice** con attributi:



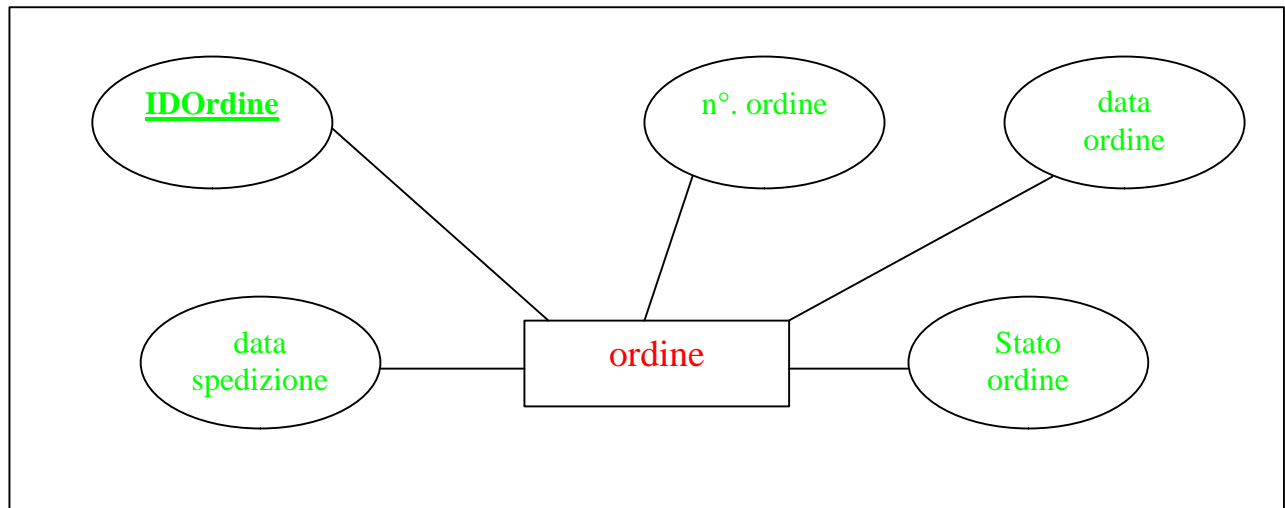
Entità **corso** con **attributi**:



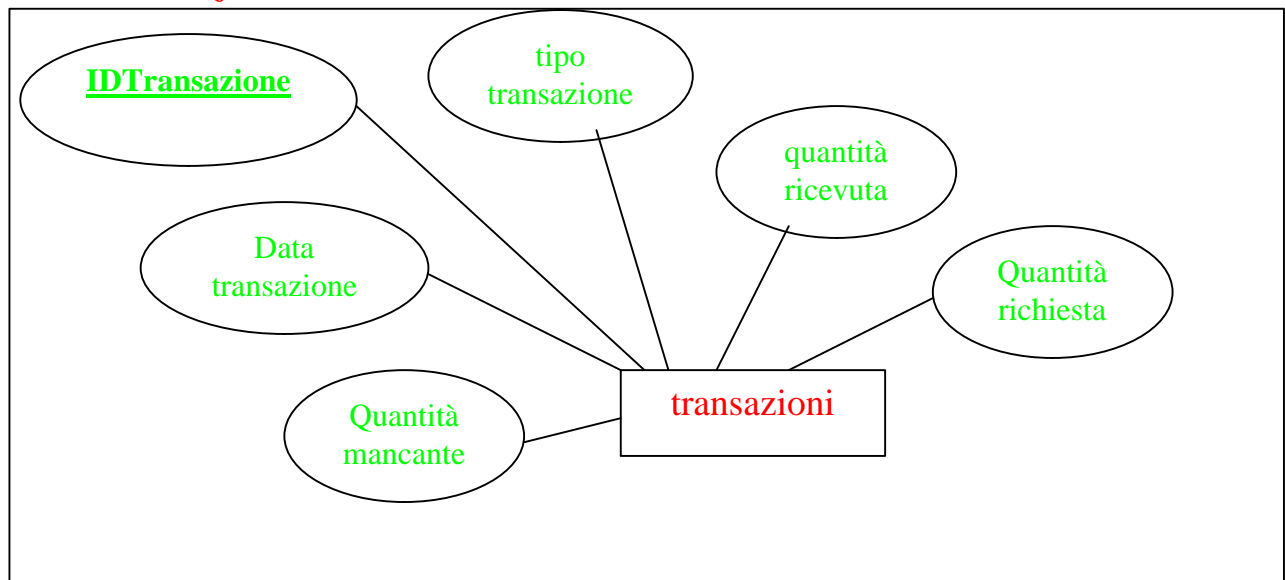
Entità **preordine** con **attributi**:



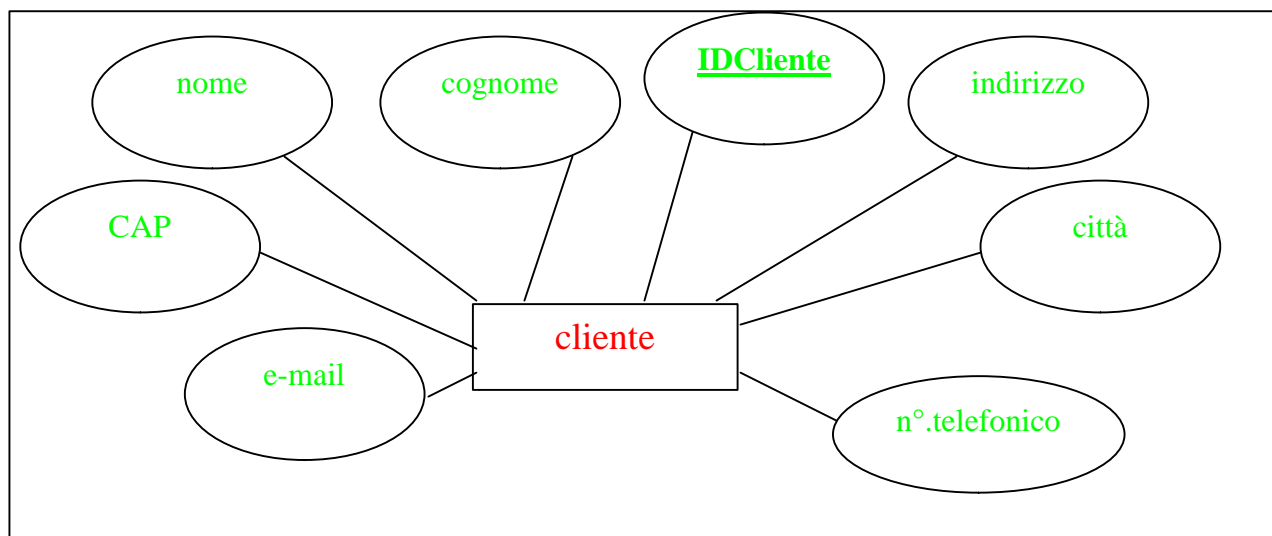
Entità *Ordine* con attributi:



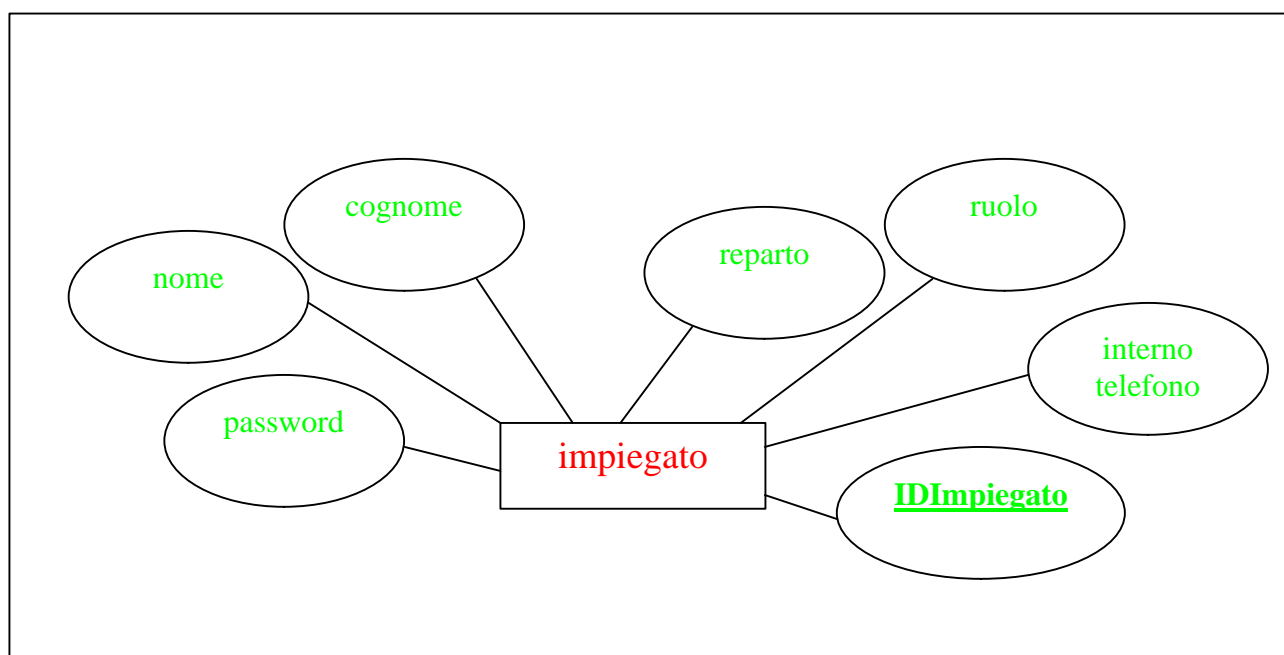
Entità *transazioni* con attributi:



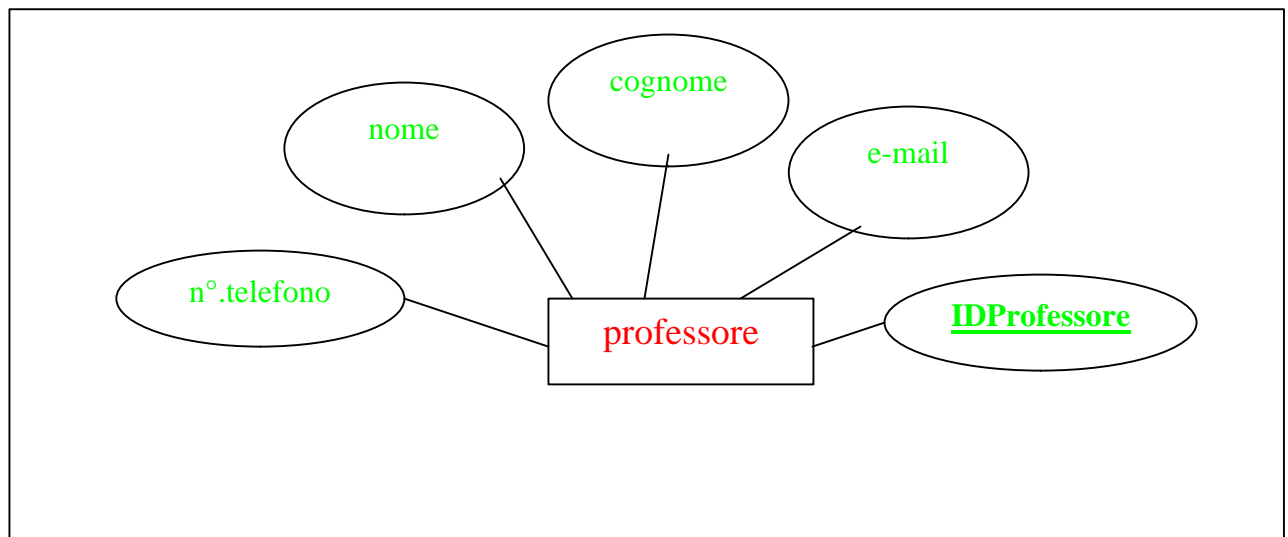
Entità **cliente** con attributi:



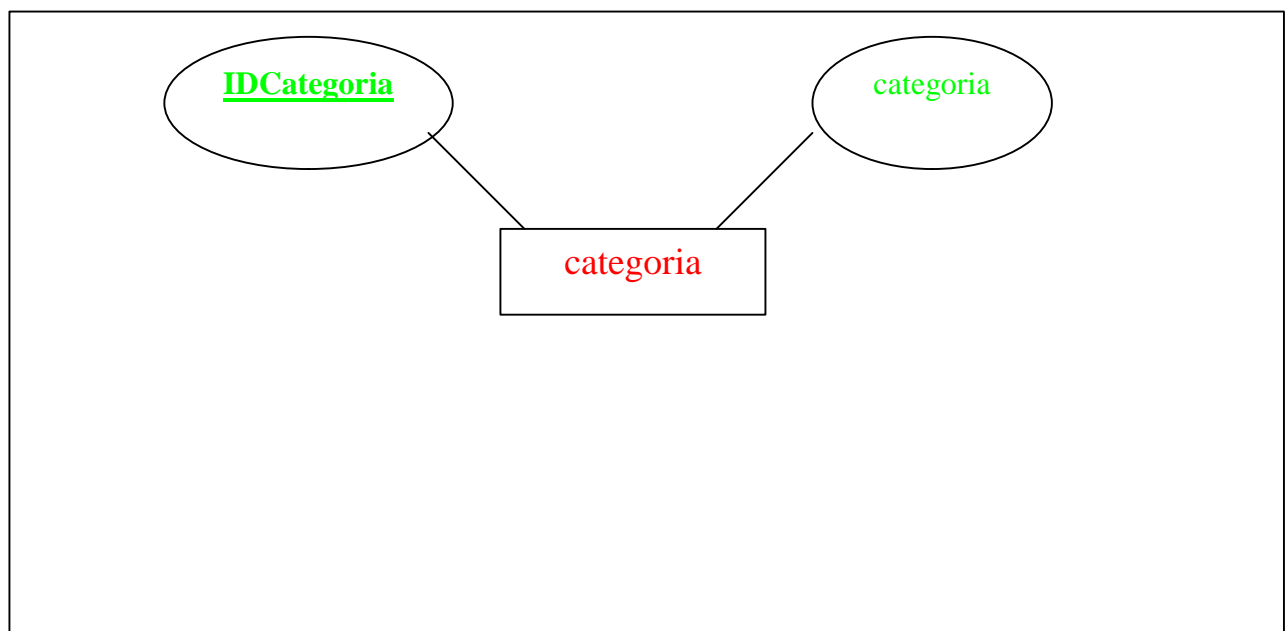
Entità **impiegato** con attributi:



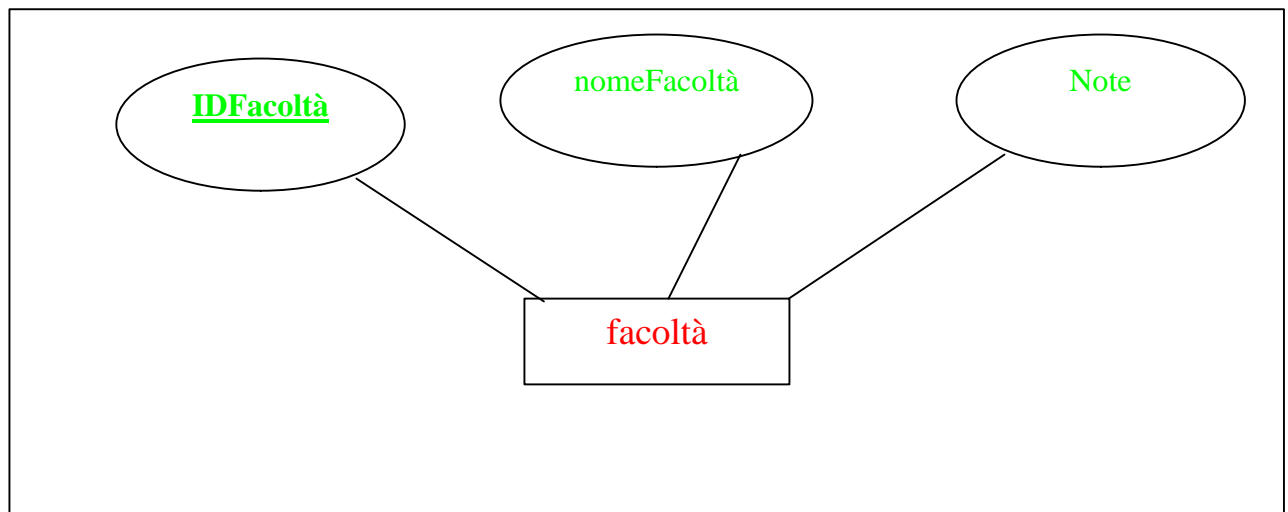
Entità **professore** con **attributi**:



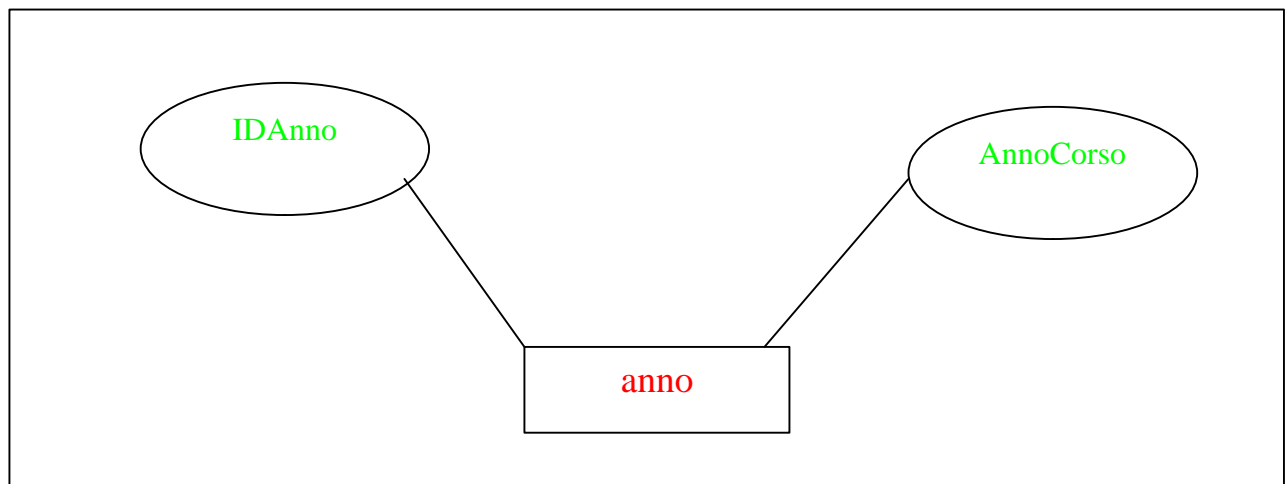
Entità **categoria** con **attributi**:



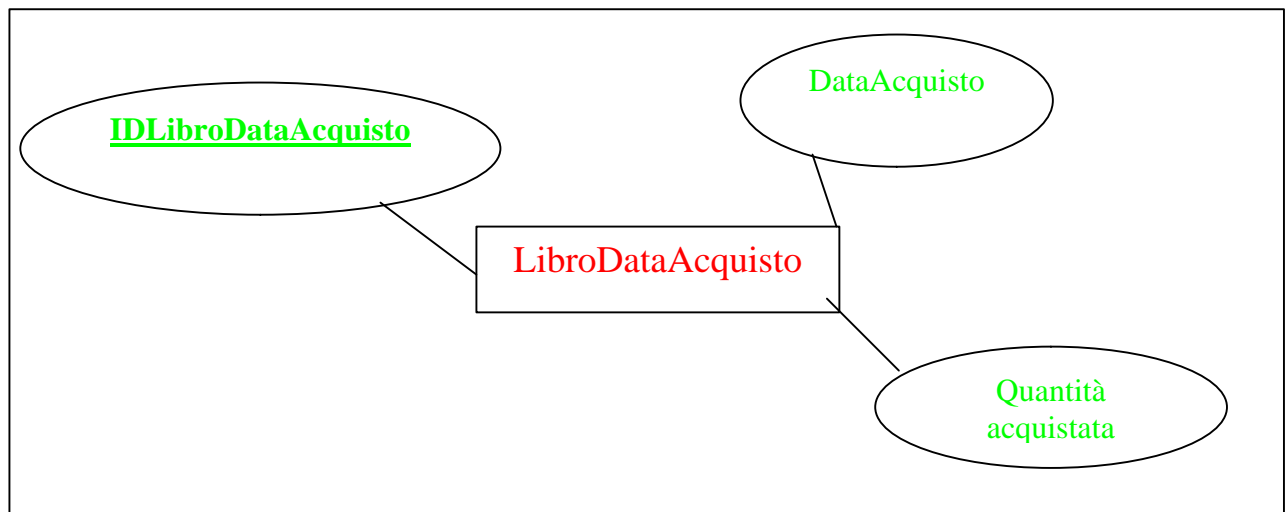
Entità *facoltà* con *attributi*:



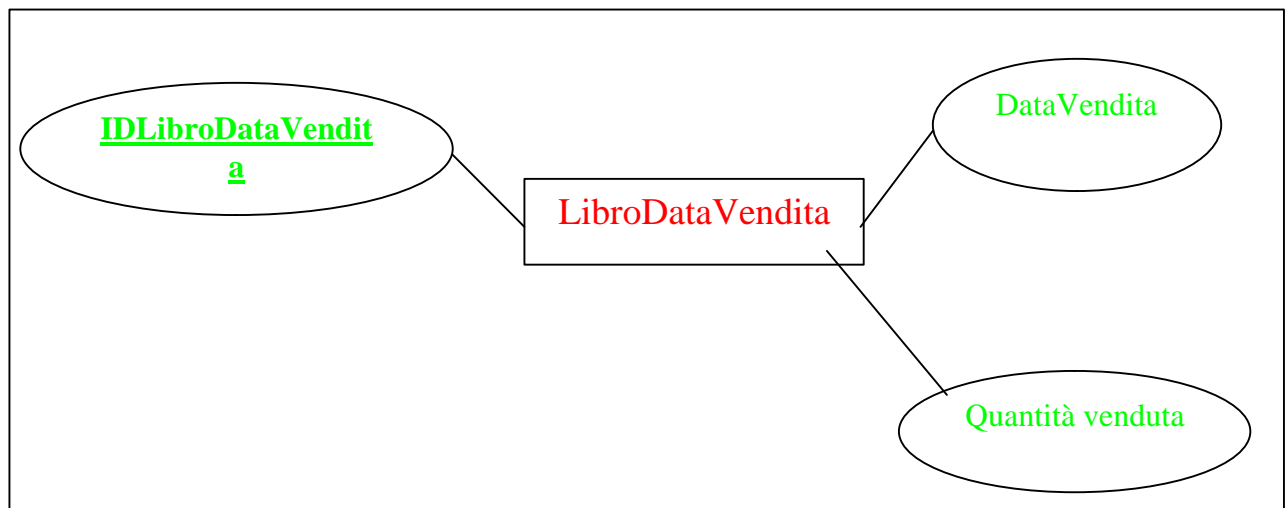
Entità *anno* con *attributi*:



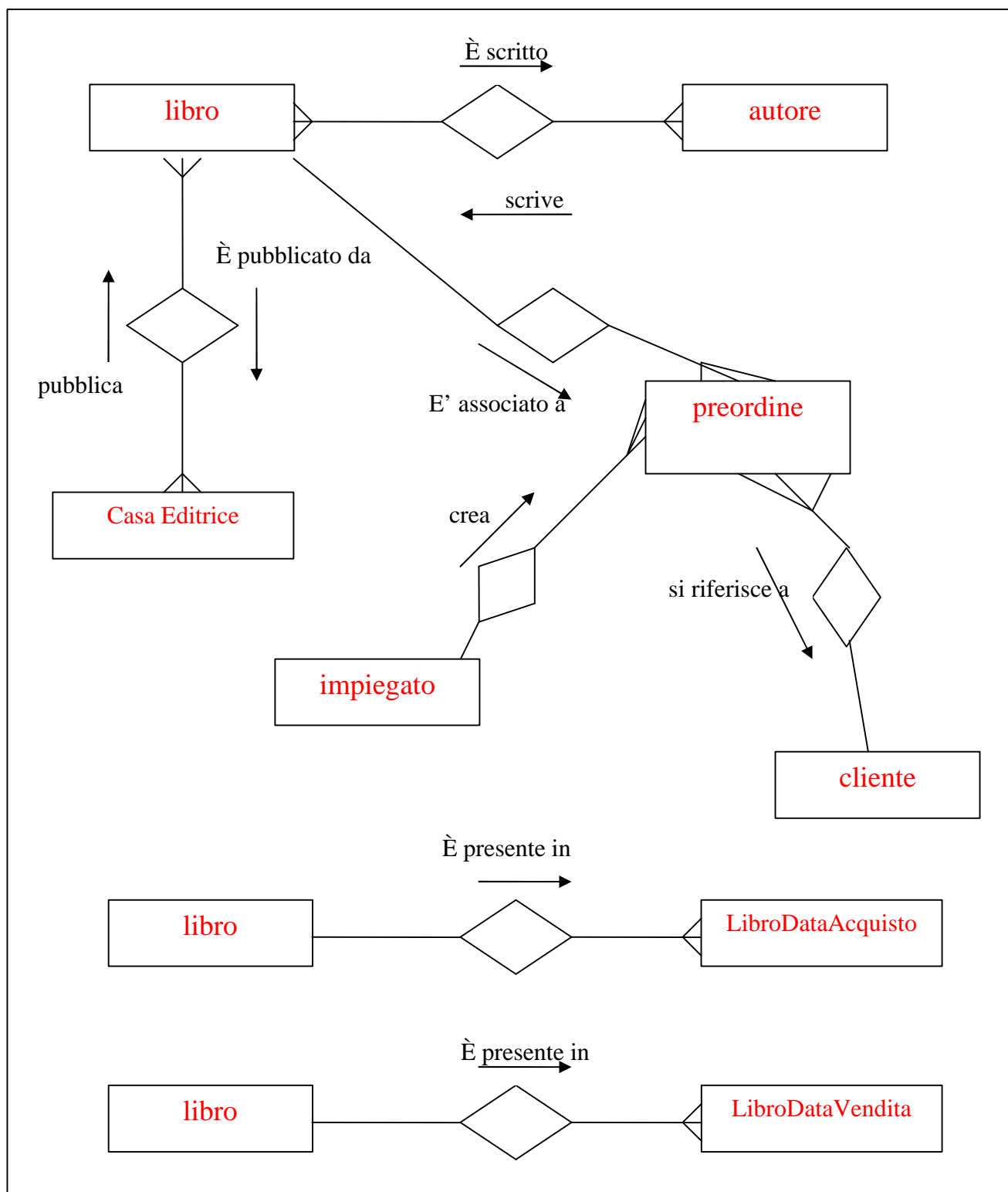
Entità **LibroDataaAcquisti** con **attributi**:

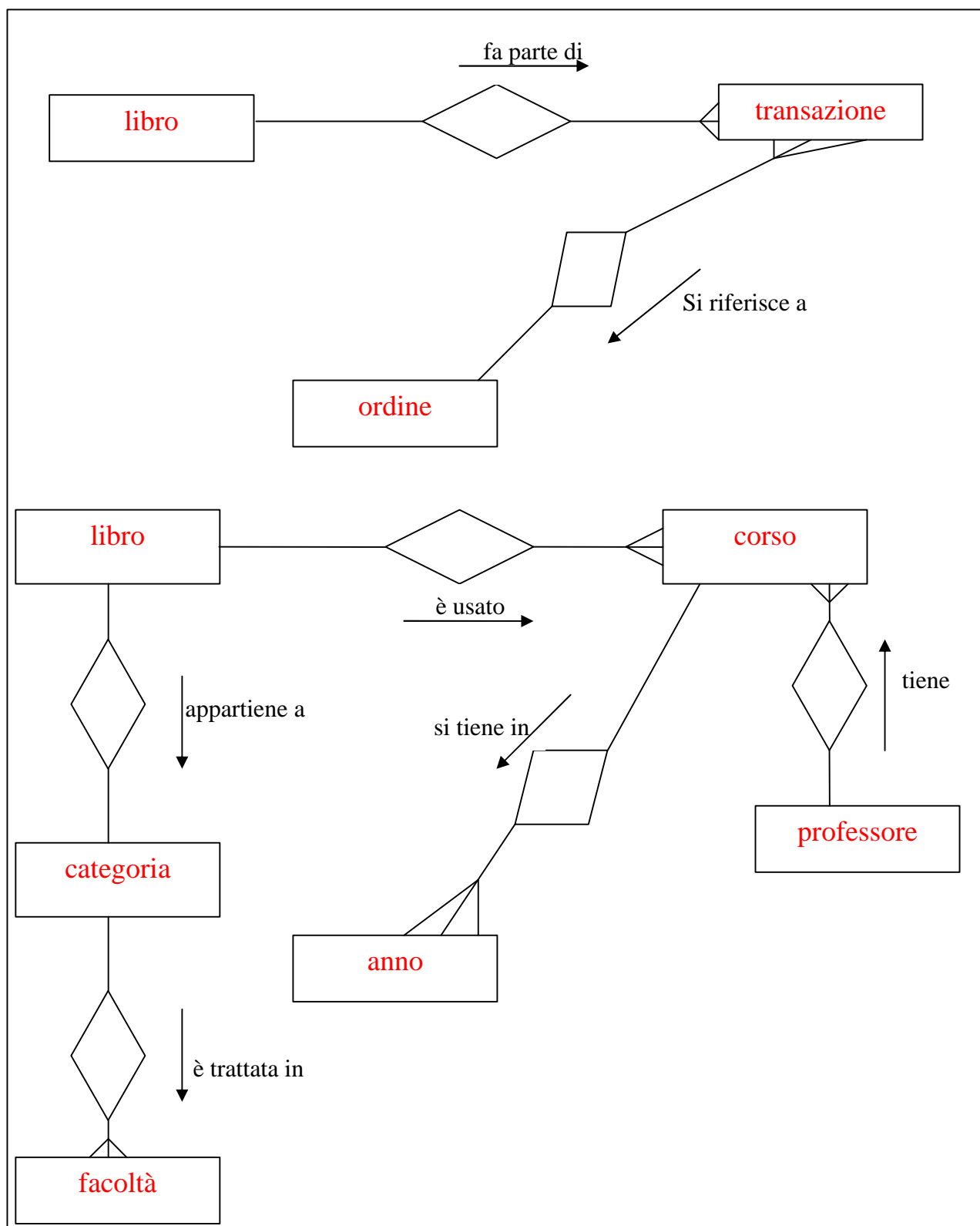


Entità **LibroDataVendite** con **attributi**:



Riportiamo adesso i seguenti ERD che mostrano come le varie entità siano collegate(relazionate) fra loro:





3.5.4 Modello dinamico del sistema

Riportiamo di seguito i diagrammi di sequenza del sistema più importanti.

Non vengono riportati tutti i diagrammi per le funzionalità possedute dal sistema in quanto tutte le operazioni che vengono effettuate comportano un accesso alle informazioni del database in sola lettura o anche in scrittura (aggiornamento del database). Quindi l'insieme delle azioni che vengono effettuate in moltissime operazioni è molto simile a quello dei diagrammi di sequenza di seguito riportati.

3.5.4.1 Sequence diagram

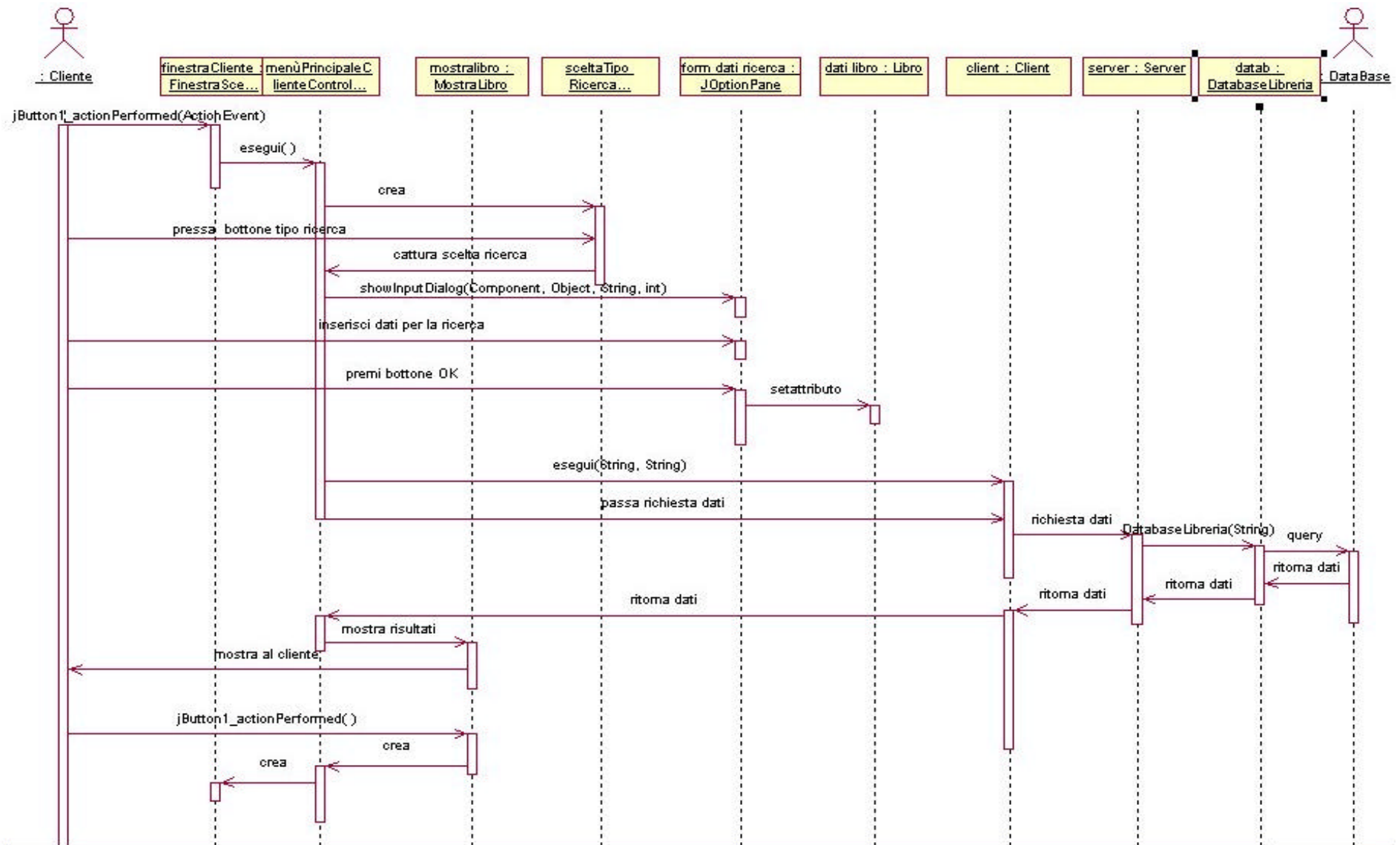
Mostriamo adesso i sequence diagram del sistema che si riferiscono alle principali funzionalità che il sistema presenta.

Dobbiamo notare che alcuni di essi pur descrivendo il sistema in relazione alla funzionalità proposta sono rappresentativi di altre funzionalità di accesso e modifica dei dati dell'archivio e che pertanto non sono stati presentati.

✦ Ricerca libro

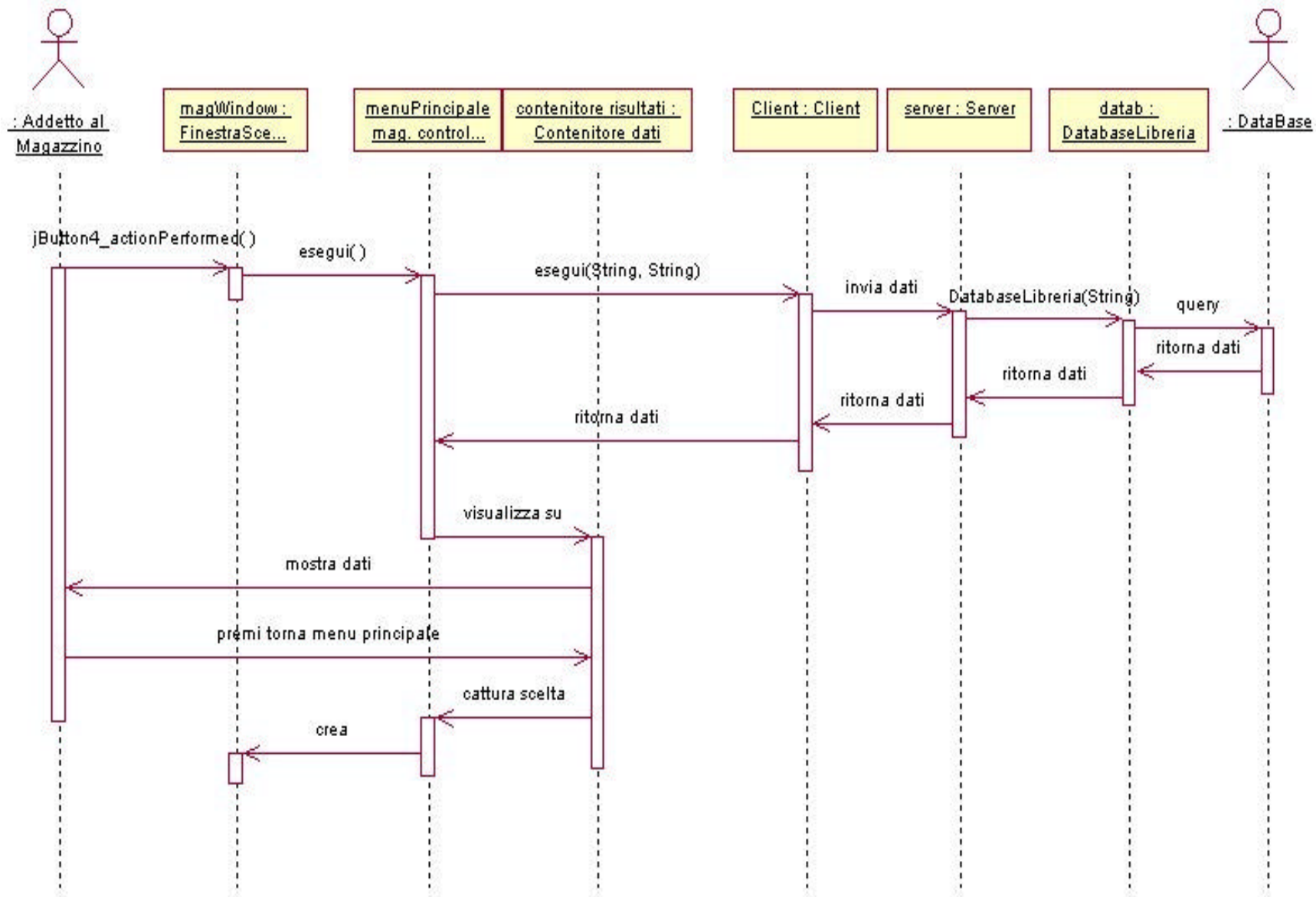
Questo sequence diagram descrive la ricerca di un libro da parte del cliente.

Tale sequenza è identica per tutti gli attori della libreria che usano la funzionalità di ricerca del sistema.



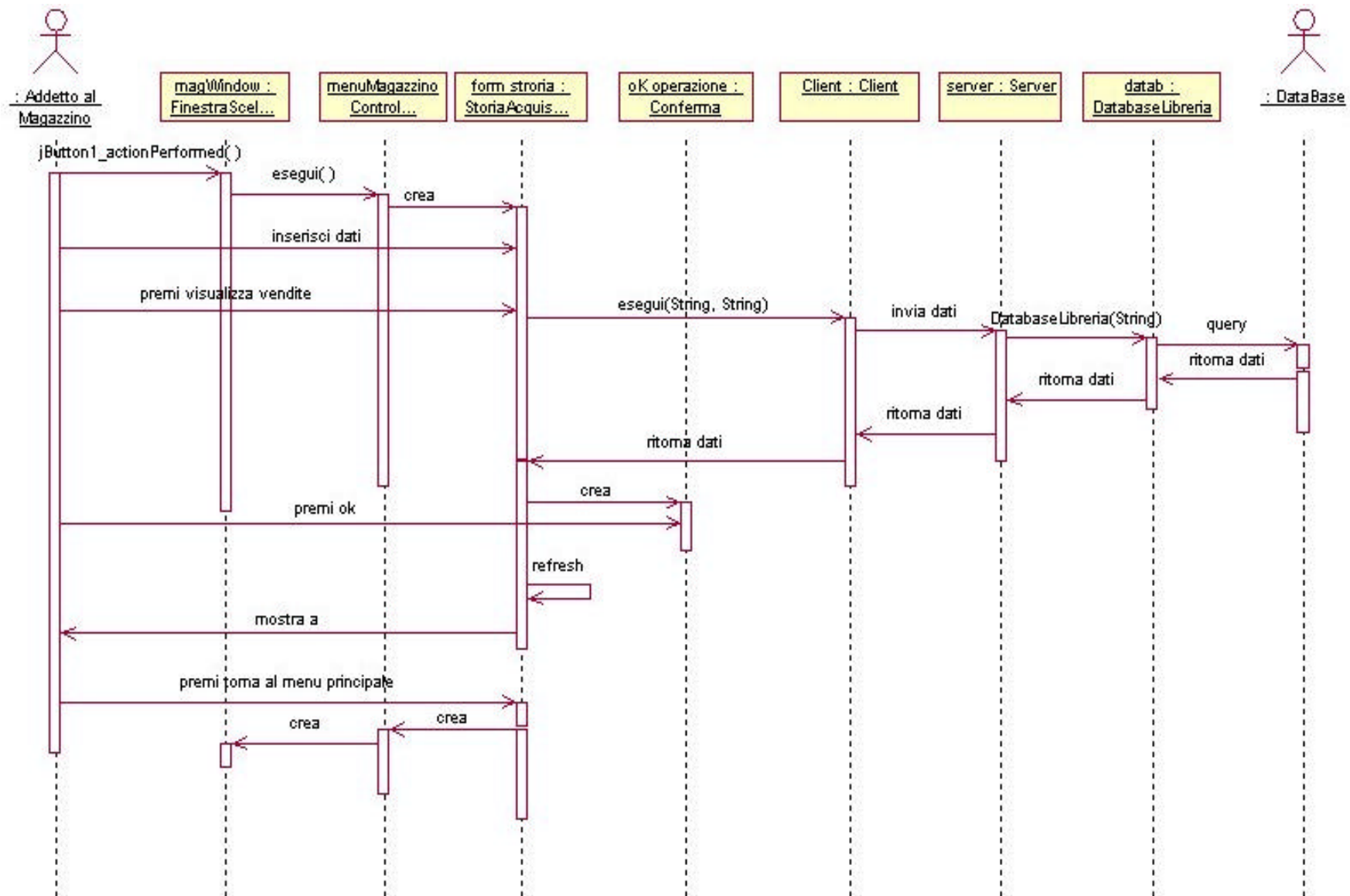
✦ Visualizza magazzino

Questo sequence diagram è relativo alla funzionalità di visualizzazione delle informazioni del magazzino della libreria . Esso può essere considerato abbastanza generale per tutte le funzionalità offerte dal sistema e relative al recupero ed alla visualizzazione di informazioni relative al magazzino della libreria.



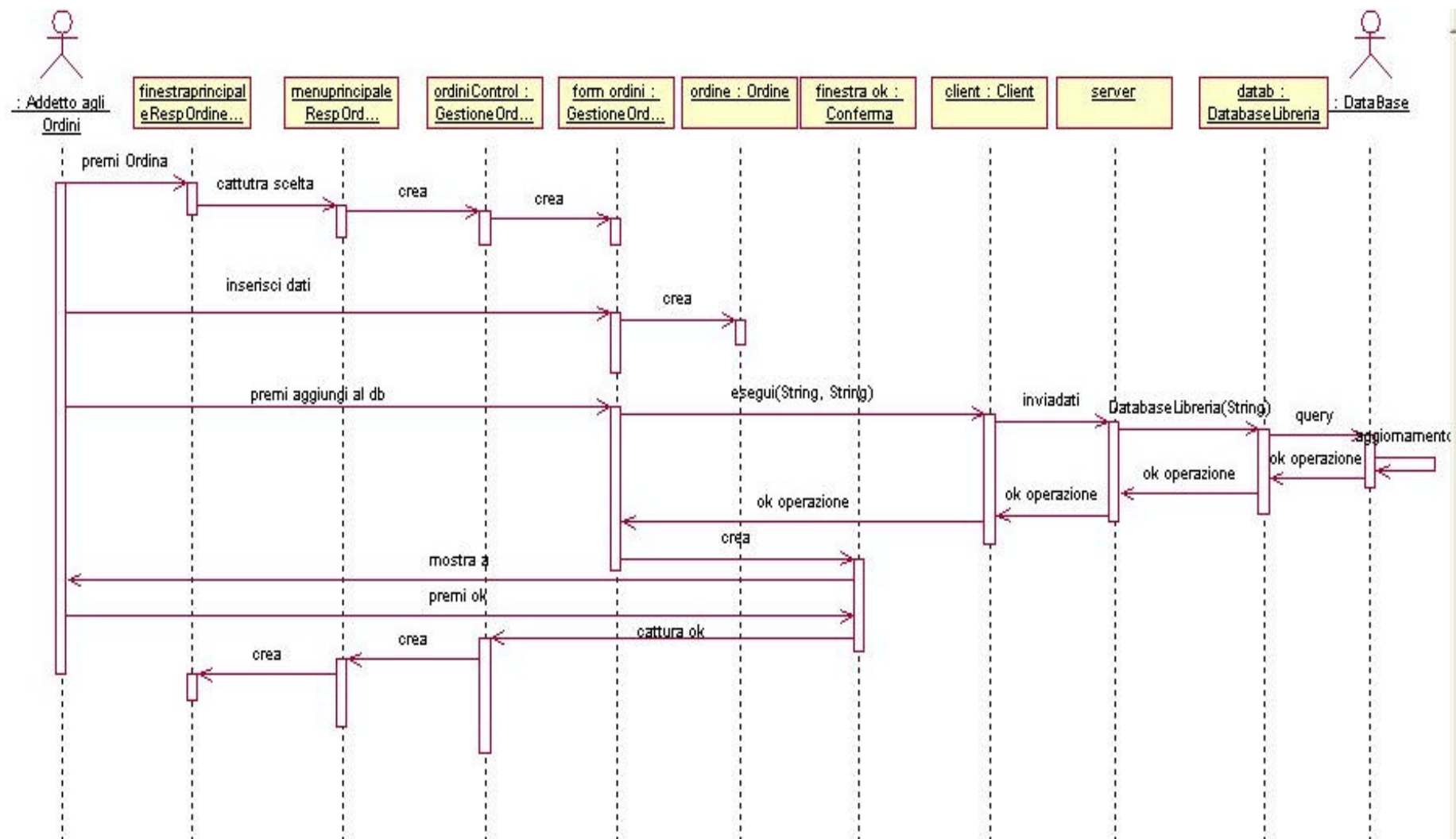
✦ Storia Acquisti vendite

Questo sequence diagram è relativo alla funzione offerta dal sistema e che mi consente di ottenere informazioni sulla storia di un libro in magazzino. Tale storia si riferisce all'acquisto (dalla casa editrice mediante ordine) ed alla vendita di un libro che la libreria tratta.



✦ Creazione Nuovo Ordine

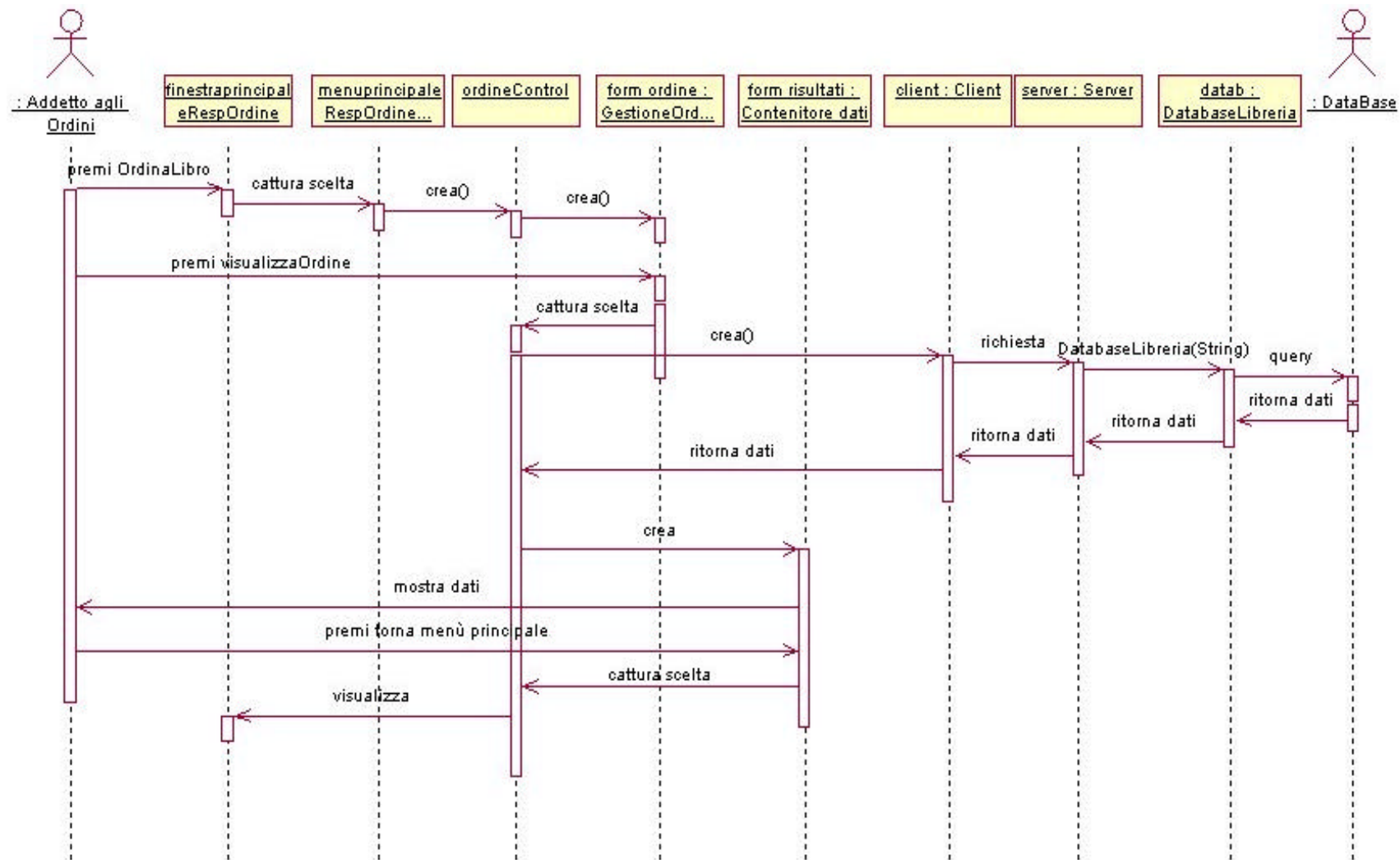
Questo sequence diagram si riferisce alla funzionalità offerta dal sistema per la gestione degli ordini ed in particolare alla creazione di un nuovo ordine relativo ai libri che la libreria vuole ricevere da una casa editrice e che i clienti hanno precedentemente prenotato.



✦ Visualizza Ordini

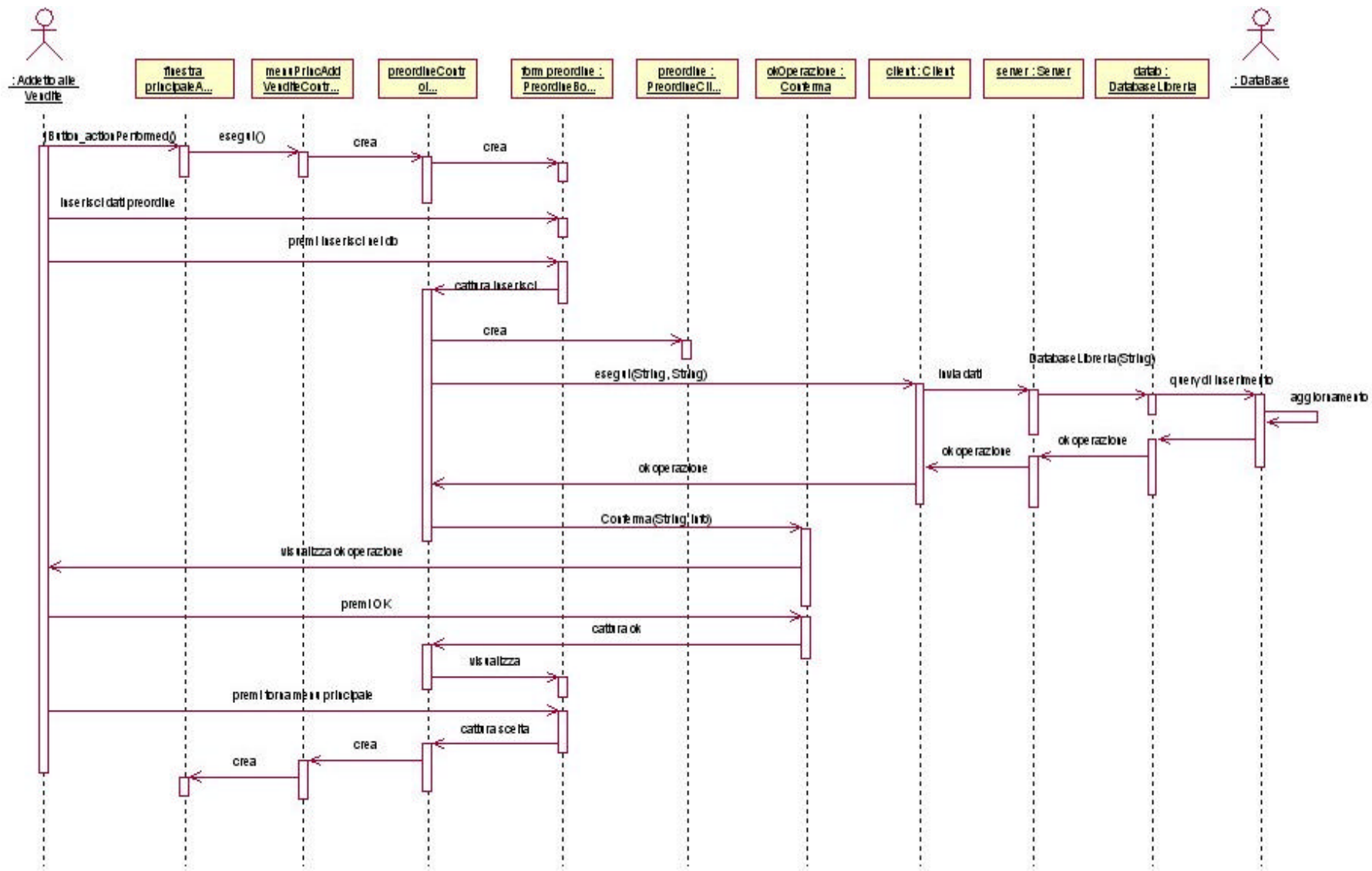
Questo sequence diagram si riferisce alla funzione offerta dal sistema relativa alla visualizzazione degli ordini emessi dalla libreria.

Tale sequence diagram è indicativo anche della funzione di visualizzazione dei preordini della libreria relativi alle prenotazioni dei libri fatte dai clienti



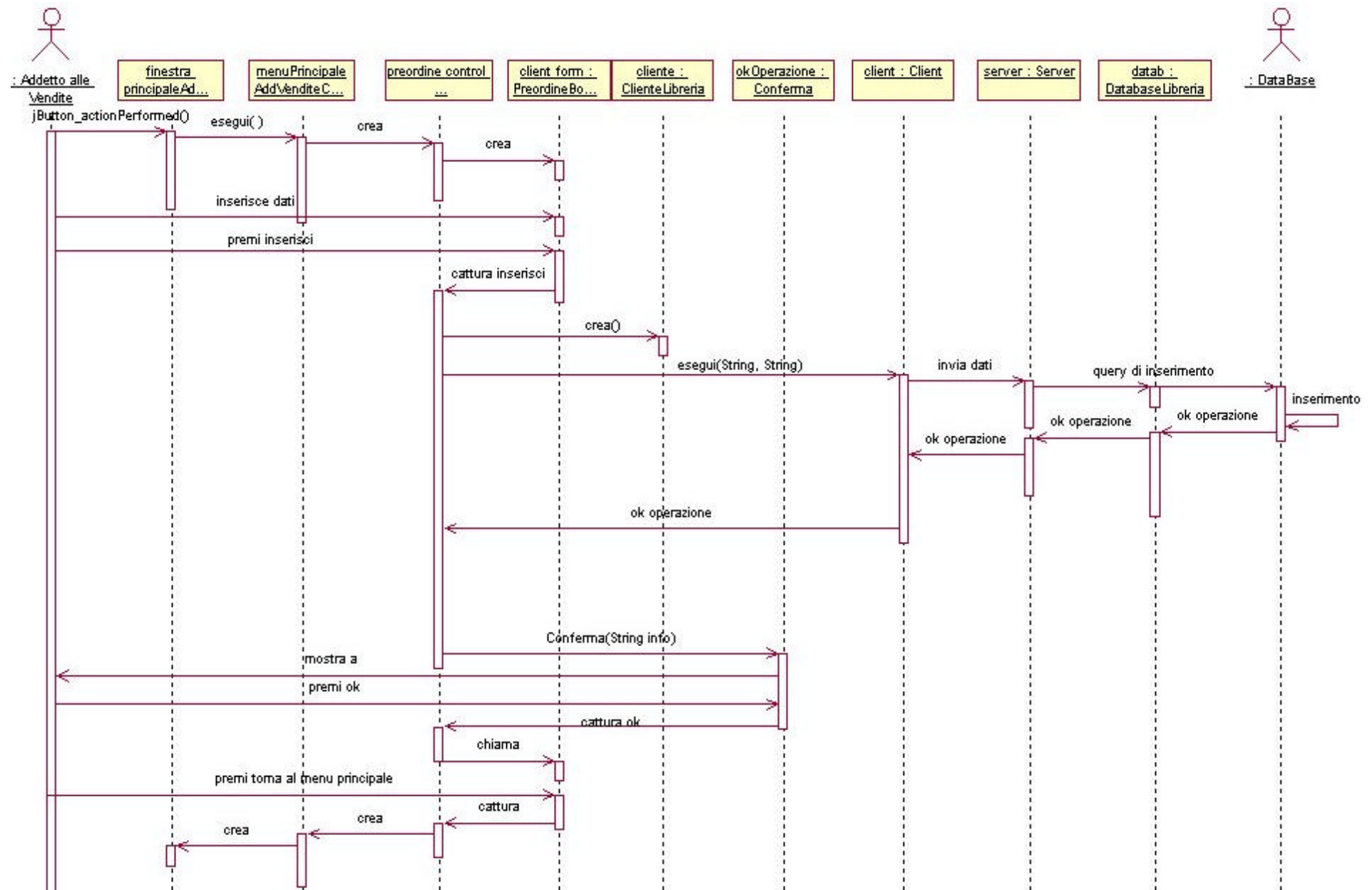
✦ Crea Preordine

Questo sequence diagram si riferisce alla funzione offerta dal sistema relativa alla creazione di un preordine relativo alla prenotazione di un cliente presso la libreria per un libro trattato dalla libreria ma non al momento disponibile presso la stessa.



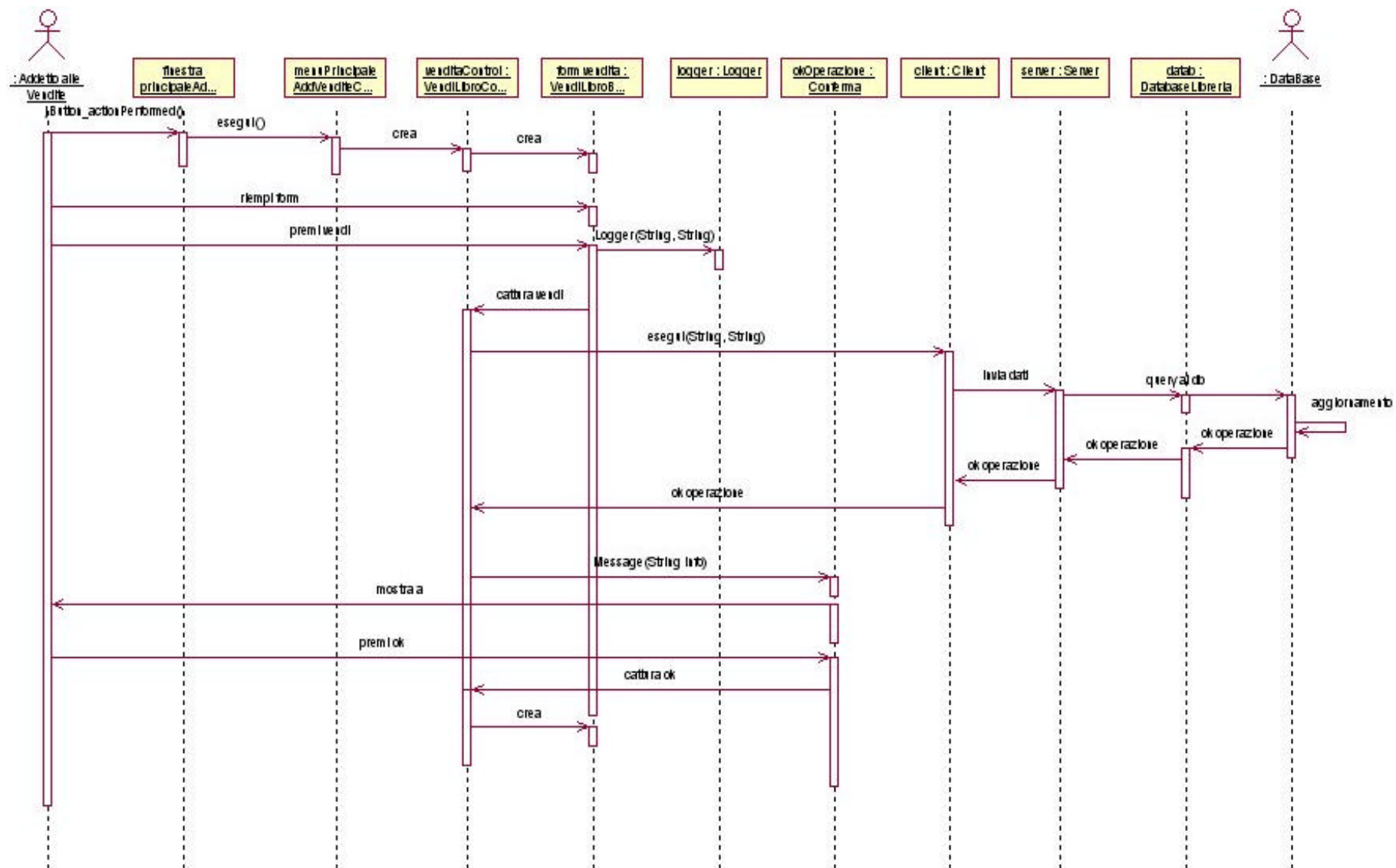
✦ Nuovo Cliente

Questo sequence diagram si riferisce alla funzione offerta dal sistema relativa alla possibilità durante la fase di prenotazione di un libro di memorizzare i dati relativi al cliente ed associarli al preordine stesso.

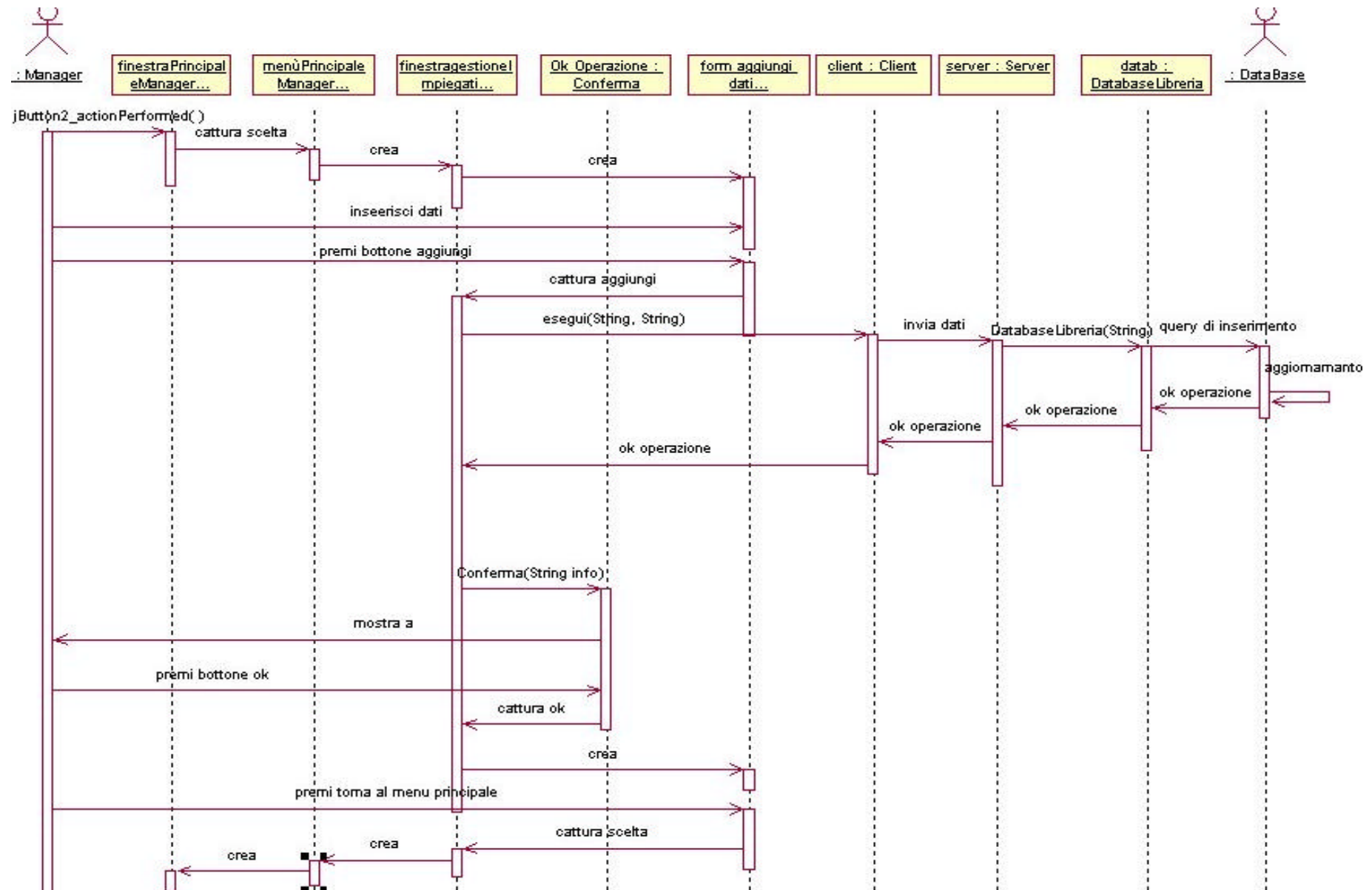


✦ Vendita Libro

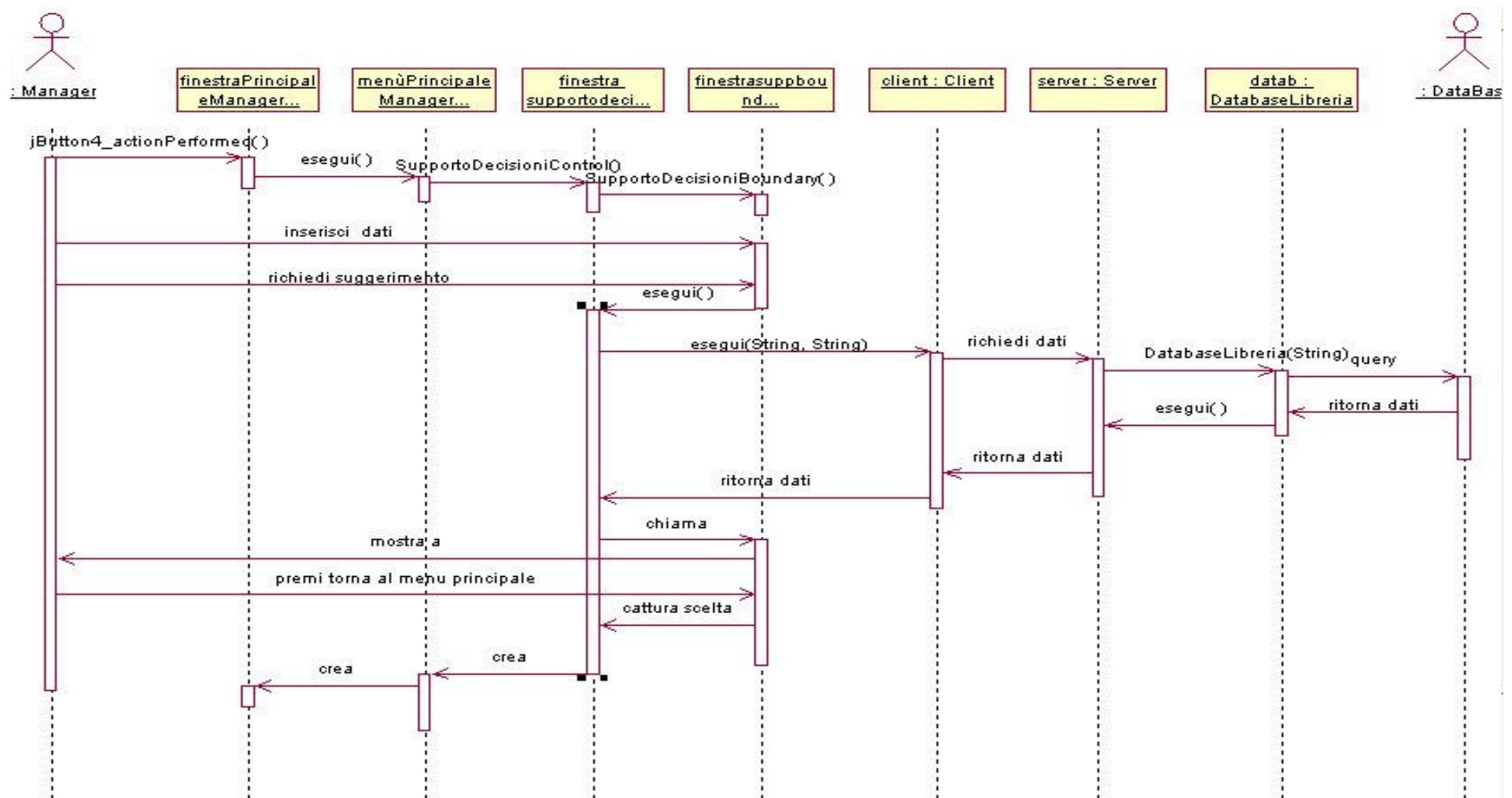
Questo sequence diagram si riferisce alla funzionalità di vendita che il sistema offre all'addetto alle vendite.



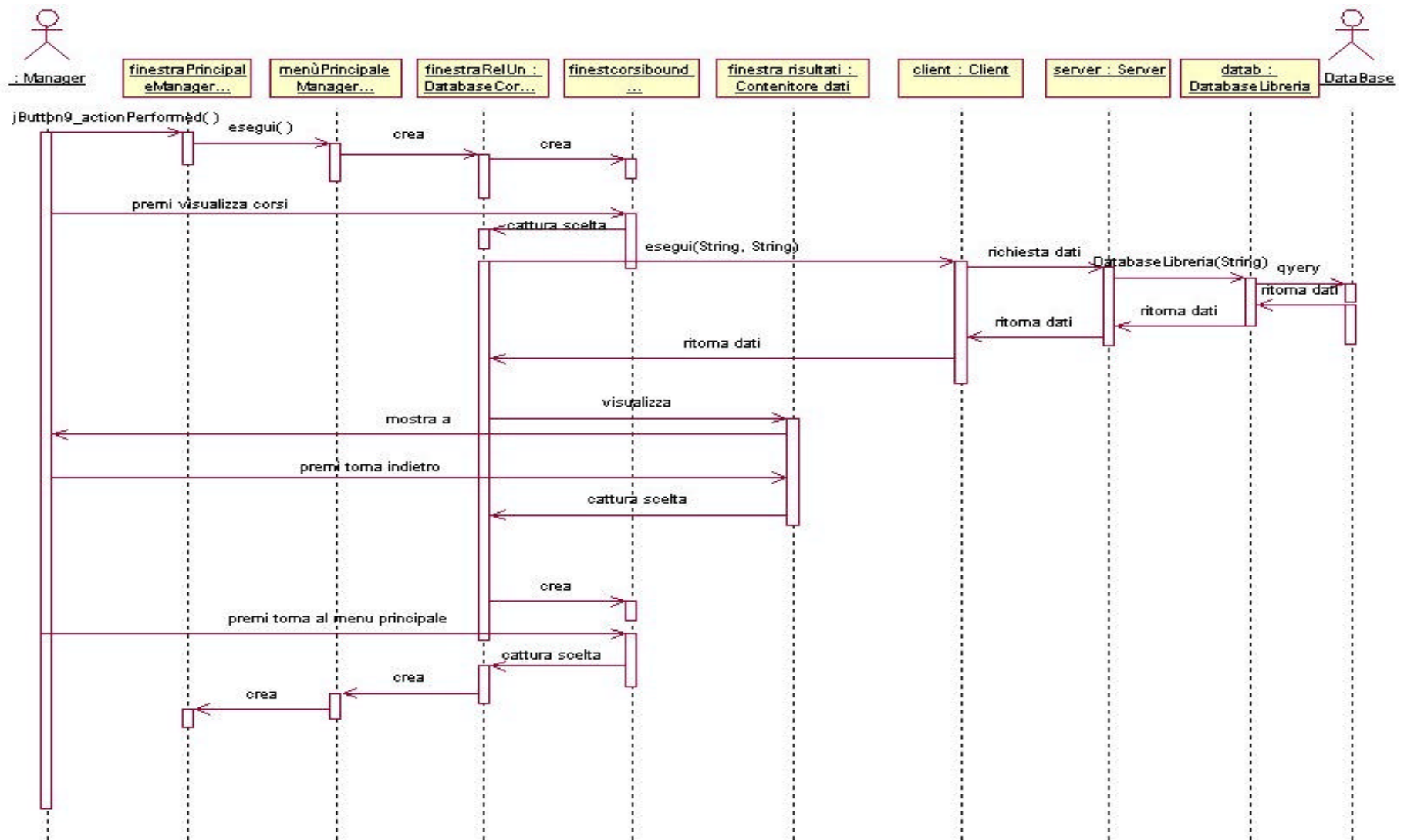
✦ Aggiungi Impiegato



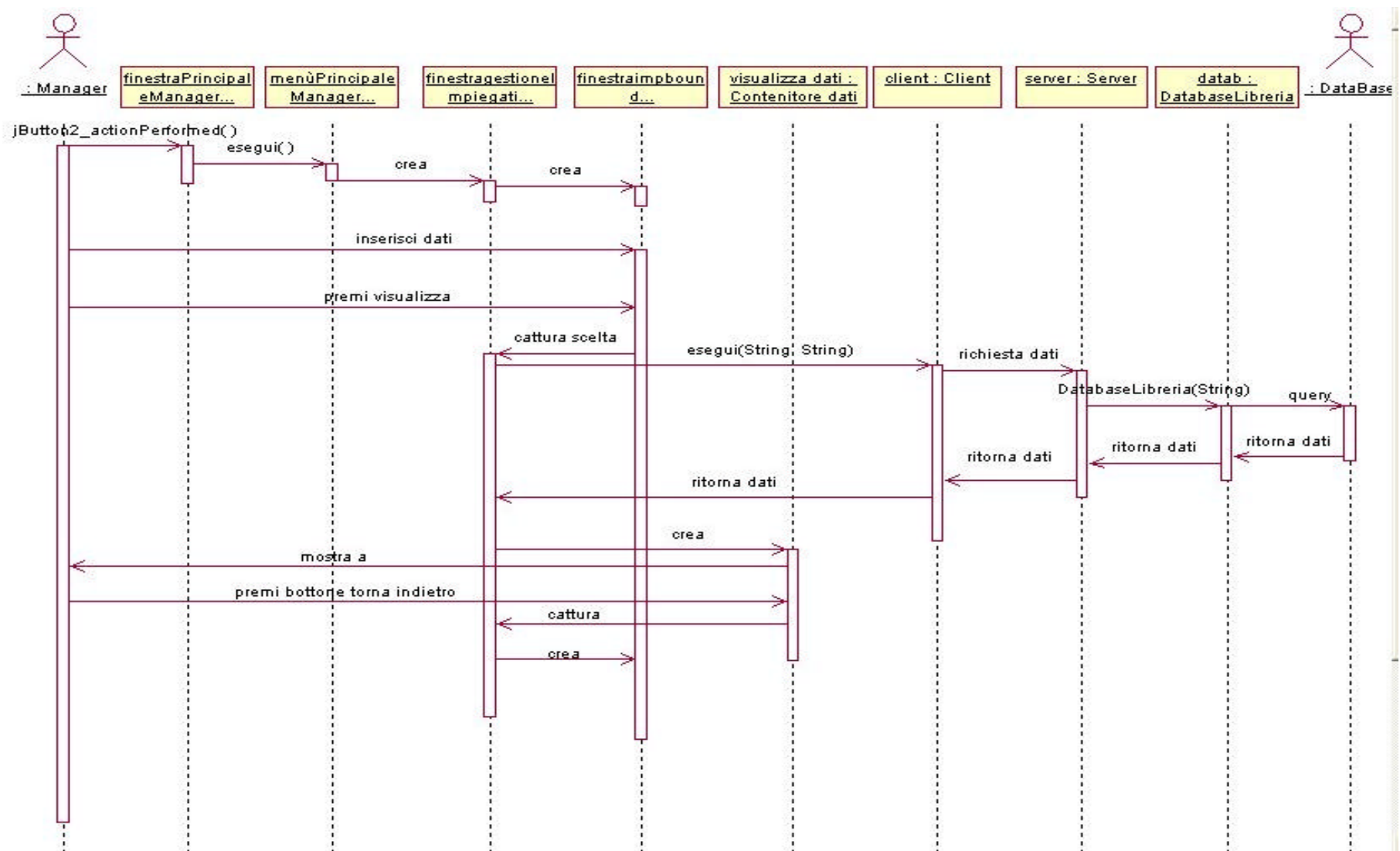
✦ Supporto alle decisioni



✦ Visualizza info corso



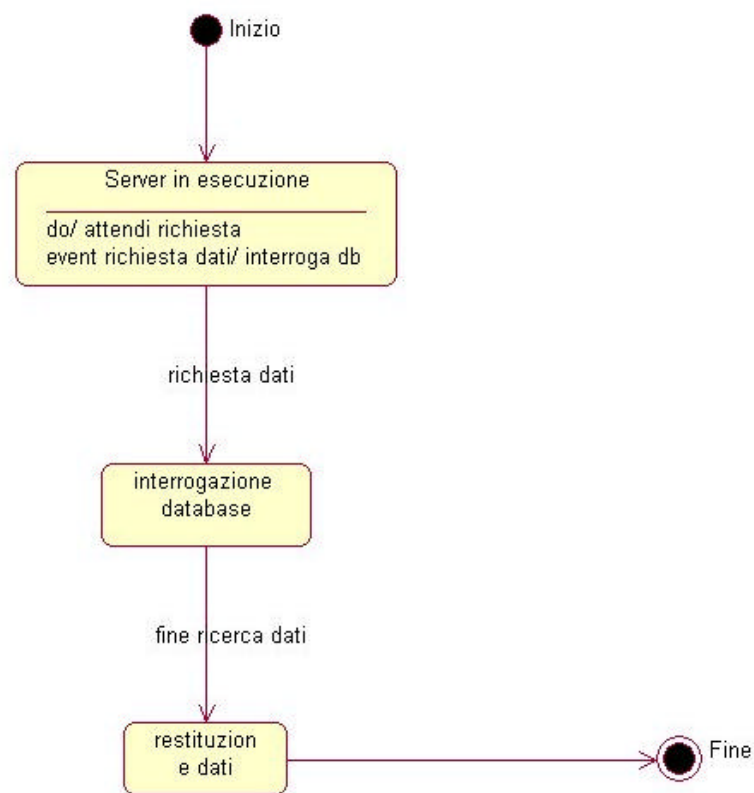
✦ Visualizza informazioni impiegato



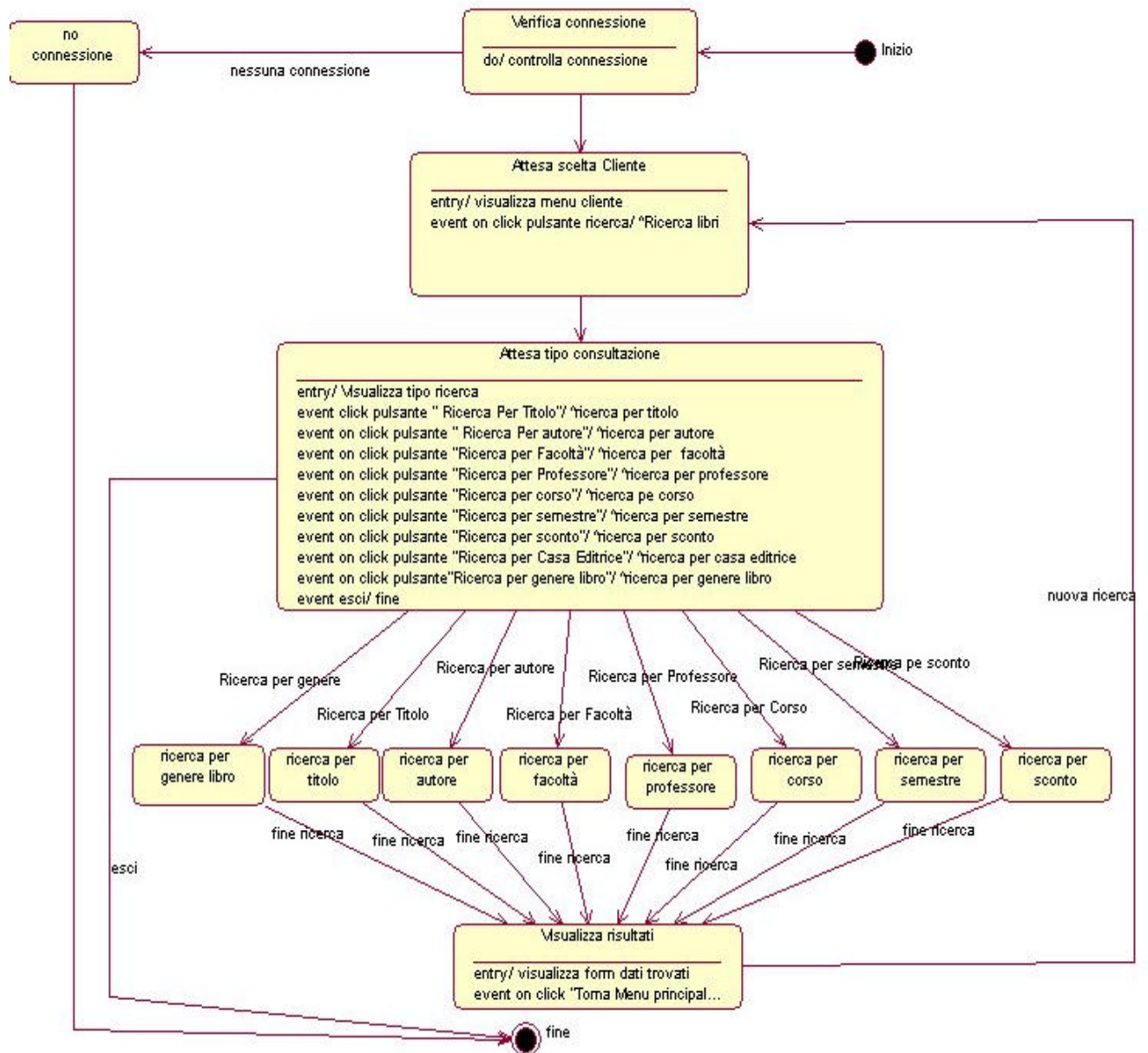
3.5.4.2 Diagrammi di stato

Vengono riportati di seguito i diagrammi di stato del sistema. Essi hanno lo scopo di mostrare ad alto livello gli stati in cui si vengono a trovare le applicazioni in esecuzione sui vari terminali della libreria.

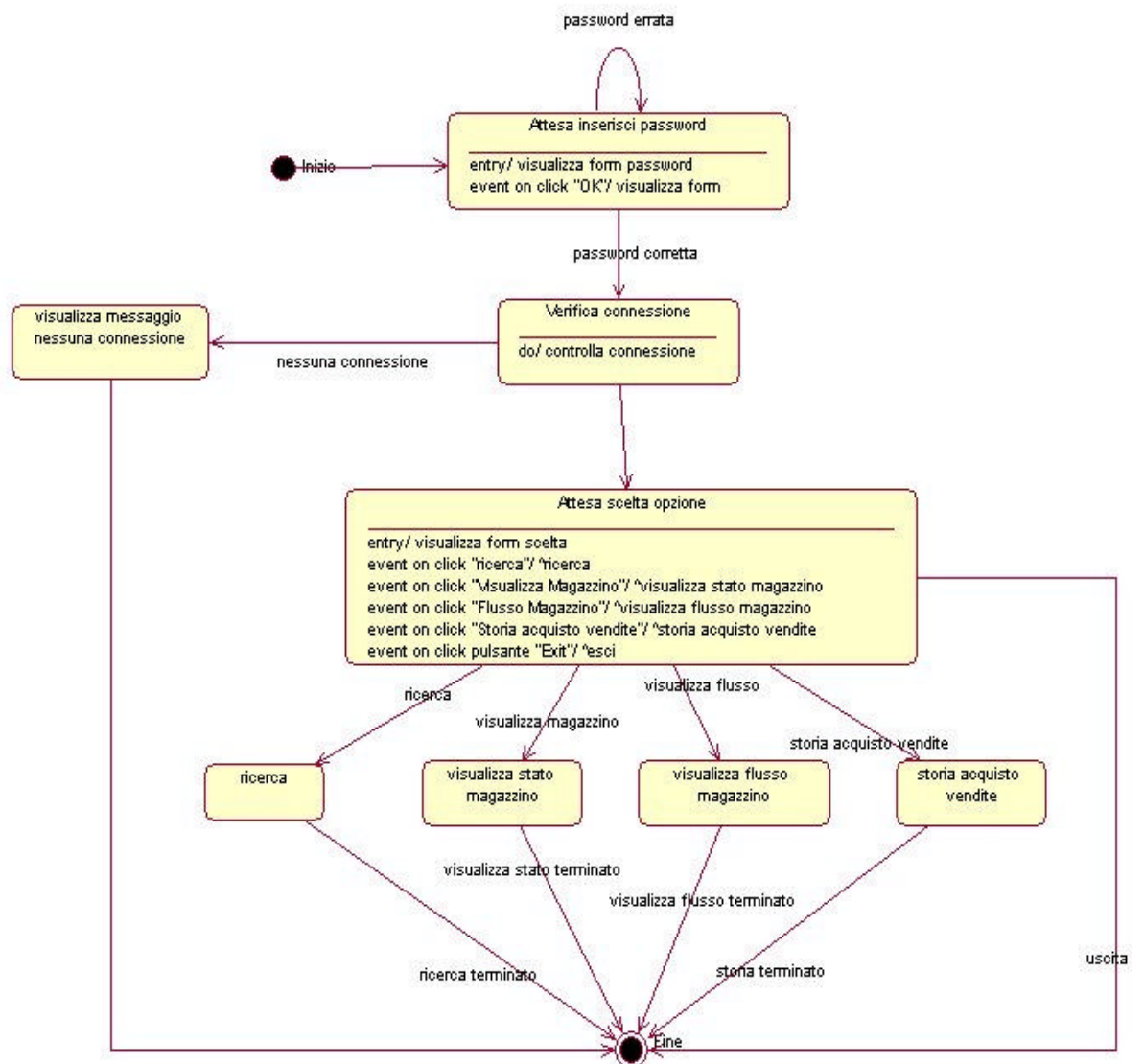
✖ Server



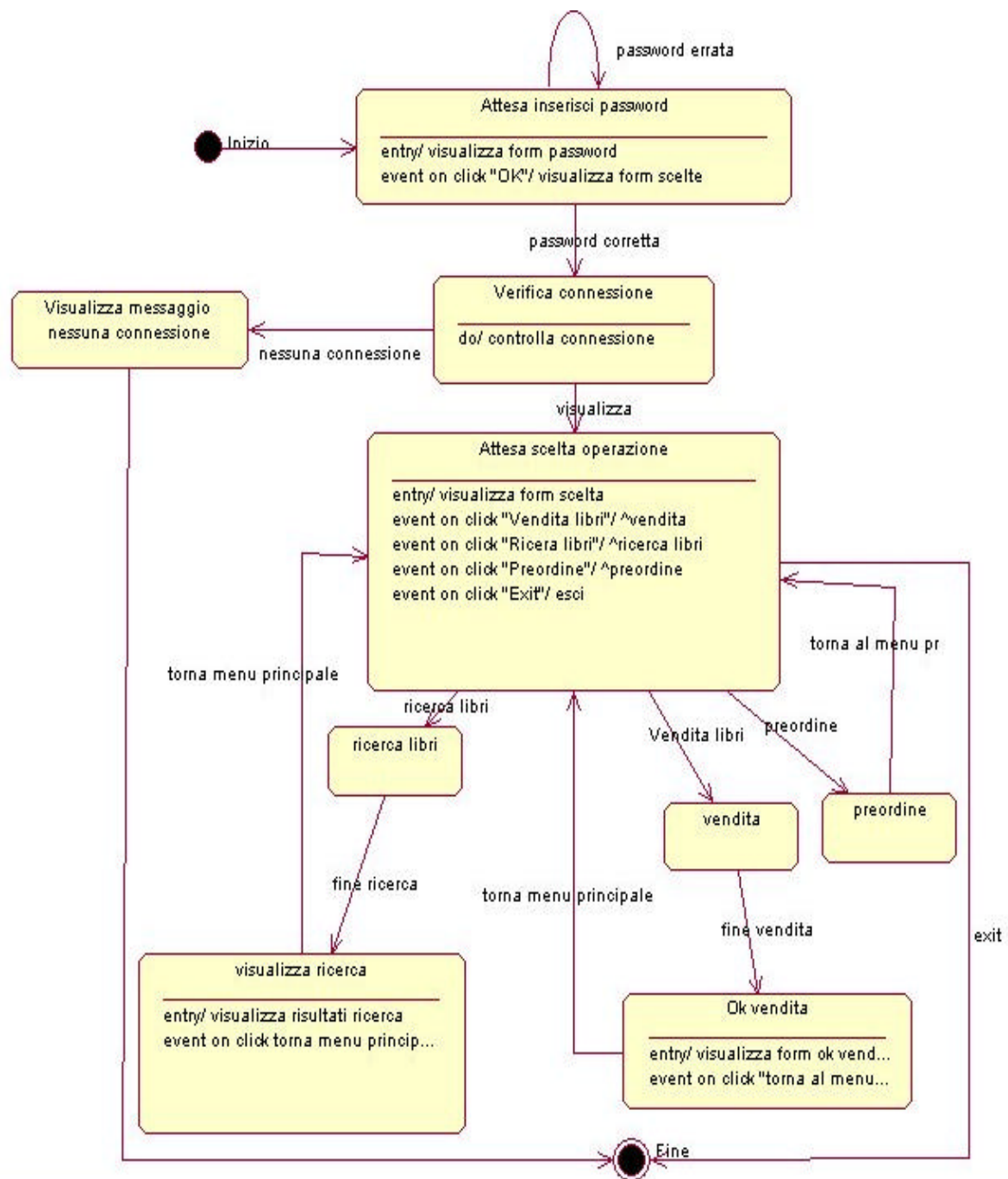
✖Postazione cliente



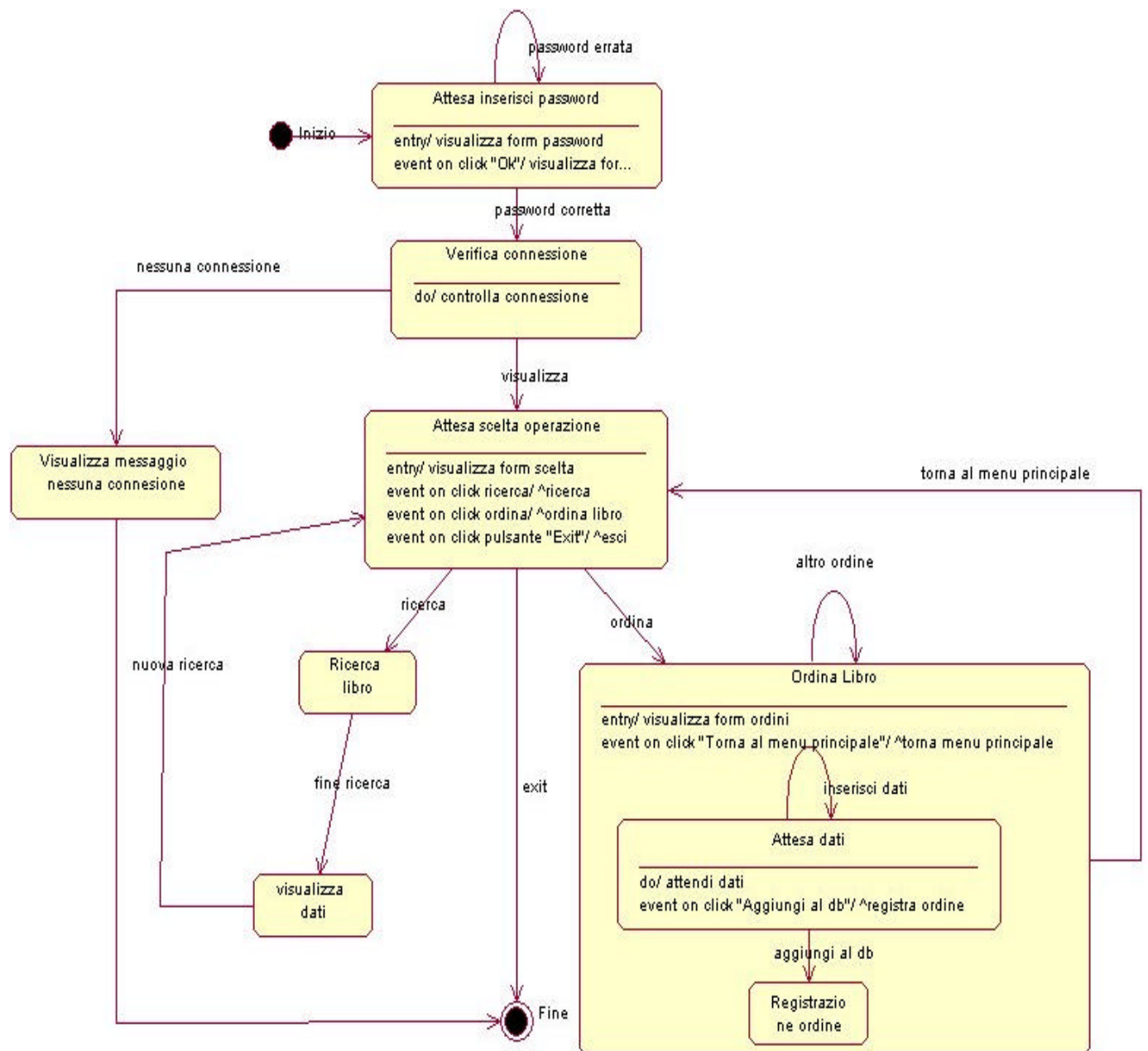
✕Magazzino



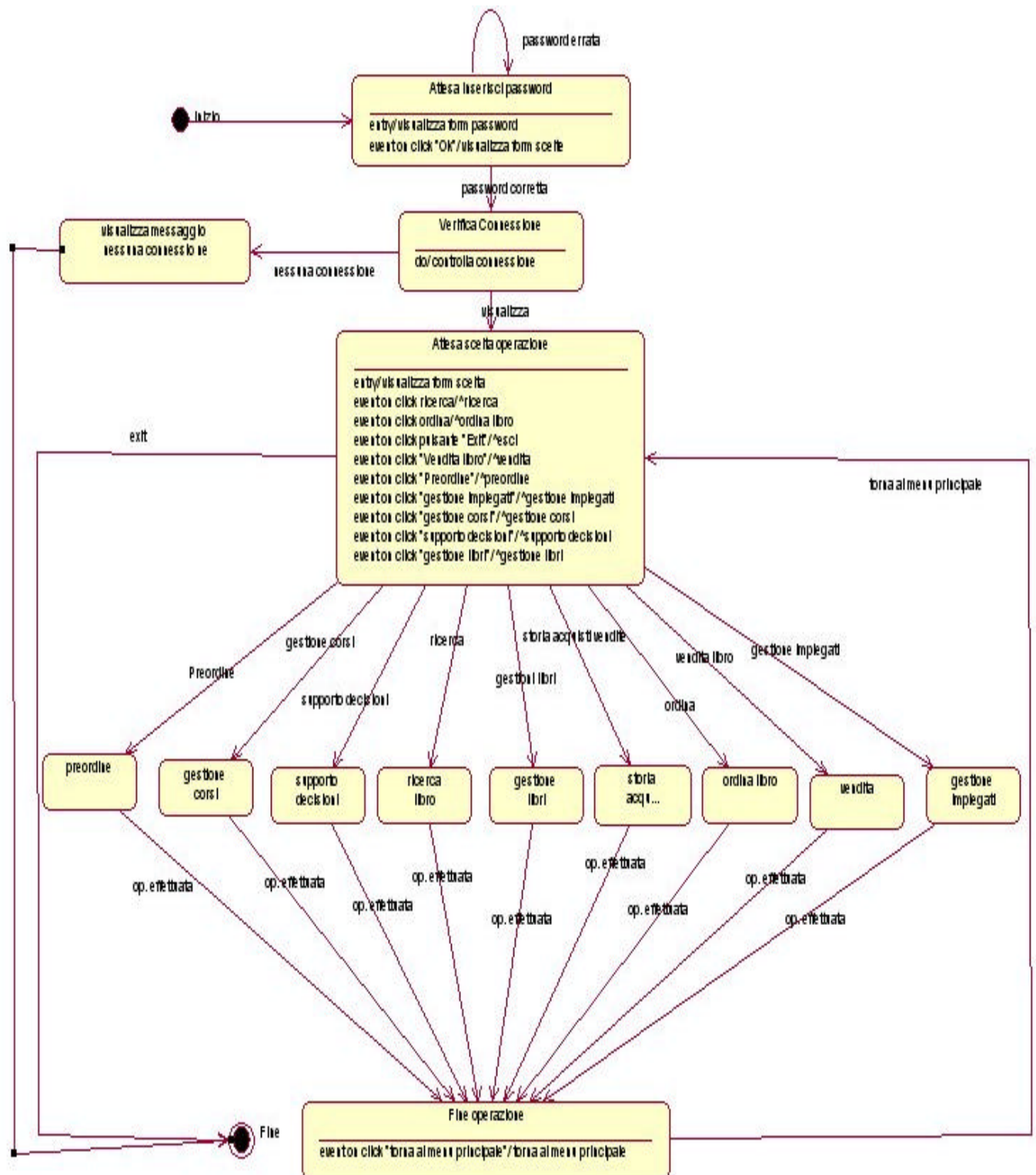
× Postazione Vendita



×Postazione Ordine



× Postazione amministratore



3.5.5 Interfacce Utente

Riportiamo di seguito alcune delle interfacce grafiche per le varie funzionalità offerte dal sistema ai vari “attori”.

GUI Cliente Principale



GUI Cliente Ricerca



Proponiamo adesso l'interfaccia che visualizza i risultati della ricerca

Libreria G&R - Libri presenti in magazzino

Ecco tutte le informazioni trovate nel nostro magazzino!

IDLibro	Titolo	Casa Editri...	Anno Copyr...	Tipo Edizio...	Numero Ed...	Numero pa...	Prezzo	Sconto	NomeCors
11	Sistemi Op...	Addison W...	1998	Universitaria	4	866	39.1500	10%	Sistemi Op

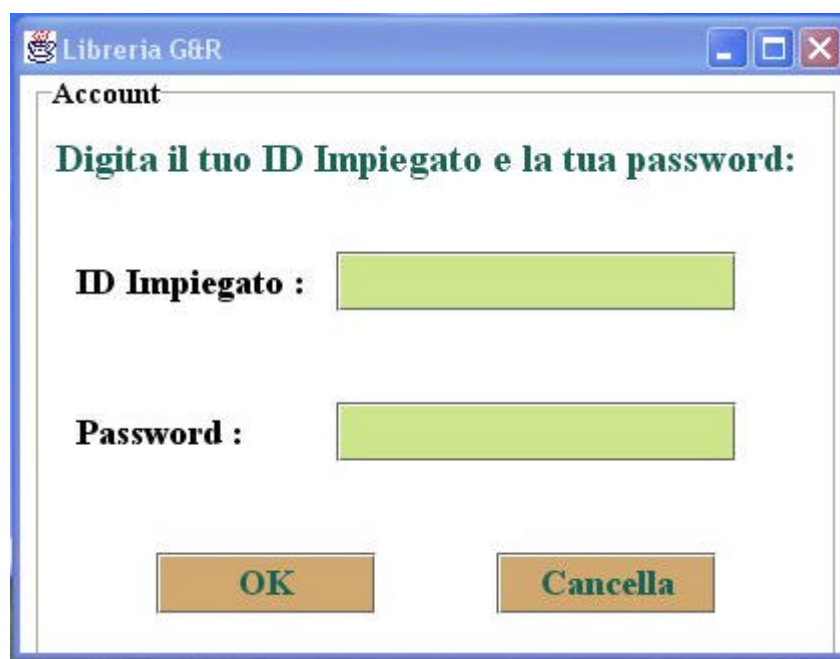
Trova autore/i libro

Inserisci ID Libro

Risultato ricerca: Nella nostra libreria è stato trovato 1 titolo di libro che soddisfa la richiesta dell'utente.

Torna al menu principale

Finestra che si presenta all'inizio del programma per accedere alle proprie funzionalità (visualizzata da Amministratore, Addetto Magazzino, Addetto Vendite, Addetto Ordini).



The image shows a Windows-style dialog box titled "Libreria G&R". Inside the dialog, the text "Account" is displayed at the top. Below it, a bold instruction reads "Digita il tuo ID Impiegato e la tua password:". There are two input fields: the first is labeled "ID Impiegato :" and the second is labeled "Password :". Both fields are currently empty. At the bottom of the dialog, there are two buttons: "OK" and "Cancella".

Mostriamo di seguito la finestra principale delle funzionalità offerte all'amministratore:



La finestra seguente mostra l'aspetto tipico dei form che vengono visualizzati quando si effettua una specifica operazione al sistema.

The screenshot shows a software window titled "Libreria G&R - Gestione Ordini". It has two tabs: "Gestione Ordine" (selected) and "Transazione Ordine". The main area contains several forms:

- Ordine :**
 - Data Ordine :
 - ID Ordine :
 - Data Spedizione :
 - Numero Ordine :
 - Stato Ordine :
- Trova l'ID associato alla Casa Editrice:**
 - Inserisci nome Casa Editr.
 - ID Casa editrice trovato:
- Elimina ordine dal database**
 - Inserisci l'ID ordine:
 -
- Crea lista giornaliera ordini**
 -
 -
-
-
-
-
-

La finestra seguente mostra l'aspetto della generica finestra che riporta i risultati di una ricerca.

Libreria G&R - Database Information

Ecco tutte le informazioni trovate nel Database!

IDImpiegato	Nome	Cognome	Ruolo	Reparto	Interno Telefono	Password
1	Luigi	Rossi	Addetto agli ordini	1	1001	uiw123
2	Mario	Bianchi	Addetto alle vendite	2	1000	vend100
3	Monica	Bellucci	Addetto agli ordini	10	1006	monbell6
4	Eva	Erzigova	Addetto al magazzino	99	1009	everz9
5	Alessandro	Corradi	Addetto alle vendite	1	1255	magica
8	Roberto	Pellitteri	Addetto agli ordini	11	1556	abcde
9	Enza	Di Fazio	Addetto alle vendite	9	6225	edf
10	Hernan	Crespo	Amministratore	51	1299	inter
11	Cristiano	Zanetti	Addetto agli ordini	36	3655	crizan
12	Pino	Lino	Addetto alle vendite	4	12	pinolino4
13	Marialisa	Di Maggio	Addetto agli ordini	44	1589	mdm
16	Christian	Vieri	Addetto alle vendite	55	9988	cvinter
17	Sebastiano	Rossi	Addetto agli ordini	45	1254	seby

Risultato ricerca: Nel nostro database sono stati trovati 13 record che soddisfano la richiesta dell'utente.

[Torna indietro](#)

Presentiamo adesso le interfacce dei vari attori:

GUI Addetto Magazzino



GUI Addetto Ordini



GUI Addetto Vendite



Architettura del sistema

(System Design Document)

1.Obiettivi del sistema

Lo scopo del sistema è quello di incrementare la produttività della libreria mediante meccanismi di gestione che si basano sull'utilizzo di un archivio centrale dove sono organizzate le informazioni.

In tal maniera il sistema renderà più veloci ed organizzate tutte quelle attività che richiedono un elevato costo in termini di tempo e che attualmente vengono svolte solo manualmente. Vantaggio fondamentale di tutto ciò sarà oltre all'incremento dell'efficienza della libreria stessa la possibilità di fornire anche nuovi servizi che potranno aggiungersi a quelli che il sistema proposto attualmente realizza.

2.Architettura software proposta

2.1.Overview

Il tipo di architettura software che si è scelta è legata (e vincolata) all'analisi del sistema. Infatti il sistema che si vuole realizzare è un sistema distribuito nel quale esiste un elemento centrale contenente l'archivio dati della libreria a cui i vari attori dalle loro postazioni vogliono accedere.

Quindi esistono dei moduli client che richiedono dati ad un modulo centrale che rappresenta il server. Fra loro i vari moduli non comunicano: l'unica comunicazione è rivolta da e verso il server.

Naturalmente tali moduli risiedono in macchine differenti.

L'architettura scelta è pertanto quella client-server.

Il server a sua volta fornisce la comunicazione dei dati con il database.

La comunicazione tra client e server avviene utilizzando l'architettura Three-Tier, evoluzione dell'architettura client/server.

In riferimento a tali sistemi infatti possiamo osservare che esse bene si adattano al nostro sistema di accesso ai dati.

La necessità di accesso da parte del personale della libreria al sistema informativo sta alla base dell'organizzazione n-tier. Un'architettura *n-tier* è costituita da n sistemi che contribuiscono in successione, a portare a termine una computazione.

In **Figura 1** è schematizzata questa struttura.

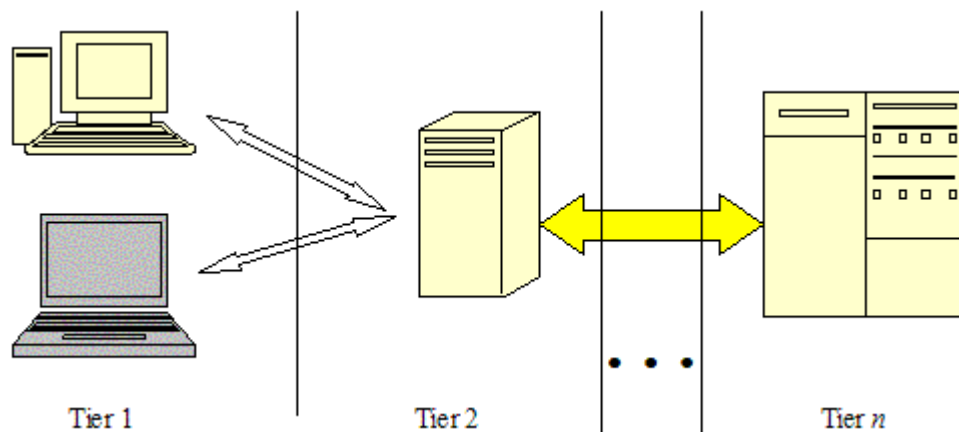


Figura 1

Tra ogni coppia di livelli esiste una relazione *client-server* e l'interazione viene propagata dal primo all'ultimo livello dell'architettura a viceversa. In ogni passo sono coinvolti programmi che si occupano di rispondere alle richieste eventualmente appoggiandosi ai livelli successivi. In ogni *tier* possono essere coinvolti più calcolatori. Le richieste vengono elaborate dai vari *tier* fino al loro completamento.

Il modello *n-tier* si è affermato proprio perché riesce a identificare "isole" di calcolo che possono essere associate alle reti su cui risiedono le risorse su cui un programma opera.

Le architetture *n-tier* riguardano la struttura di un sistema che è composto da elementi che nel loro piccolo sono del tutto standard. La ricchezza che deriva dalla struttura è principalmente una migliore ingegnerizzazione del sistema che porta ad una sua maggiore manutenibilità. Inoltre la conoscenza delle interfacce

tra un livello e l'altro aiuta a rendere più flessibile il sistema, ad esempio accedendo più DB invece che uno solo per distribuire il carico senza che il livello intermedio si accorga del cambiamento

Le applicazioni *three-tier* (n=3) prevedono un primo livello costituito dai clients, questi si connettono al secondo livello in cui vengono elaborate le richieste. Nel livello intermedio si gestisce l'elaborazione della richiesta accedendo a dati che risiedono sul terzo ed ultimo livello.

Client + Front-end + Back-end = three-tier

È importante osservare come l'interfaccia tra i livelli sia ben definita: questo aiuta a effettuare verifiche di sicurezza sulle comunicazioni. Inoltre è possibile separare il livello dei clients dai restanti due utilizzando un *firewall*.

In questo progetto abbiamo realizzato una struttura 3-tier in cui c'è un database, un client ed un server che interagisce sia con il database che con il client e che svolge il compito di modulo intermedio. Tale modulo intermedio, (rappresentato per noi dalla classe Server e dai driver JDBC-ODBC di Java), si occupa di inoltrare le richieste dell'utente al livello archivio dati e di trasmettere i relativi risultati.

Il principale vantaggio di tale scelta è quello di rendere il client del tutto indipendente dalle problematiche di accesso ai dati, rendendo il sistema

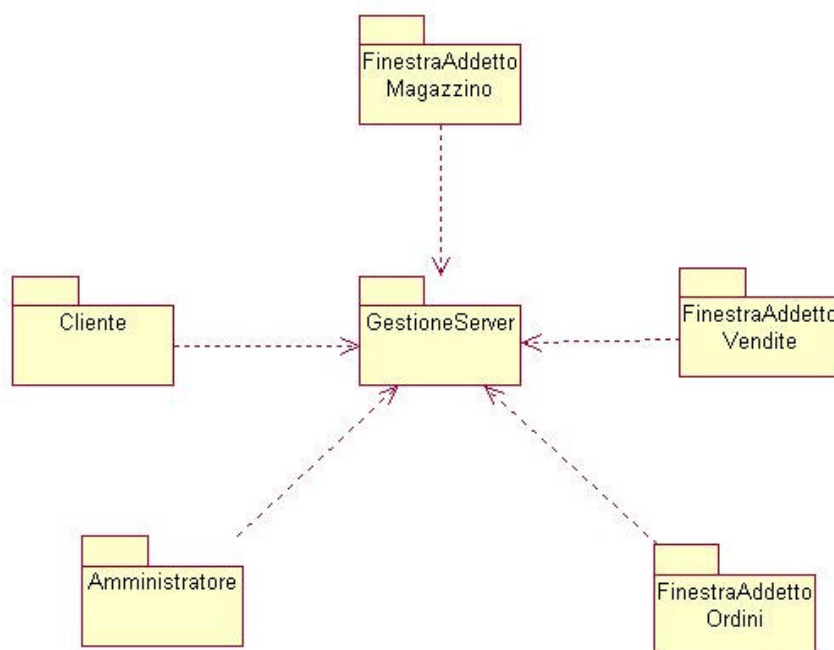
aggiornabile e mantenibile soprattutto per quel che riguarda la parte del sistema che si occupa dell'accesso all'archivio dei dati.

Resta da definire il tipo di protocollo utilizzato per la comunicazione client-Server. Si è scelto l'uso del protocollo UDP. Questo protocollo offre un servizio di comunicazione fra client e server senza connessione evitando così la fase di handshaking.

2.2 Decomposizione del sistema

Presentiamo adesso la decomposizione del sistema.

Il sistema è composto da cinque componenti principali. Riportiamo il diagramma dei package principali del sistema ed una breve descrizione di ogni package.



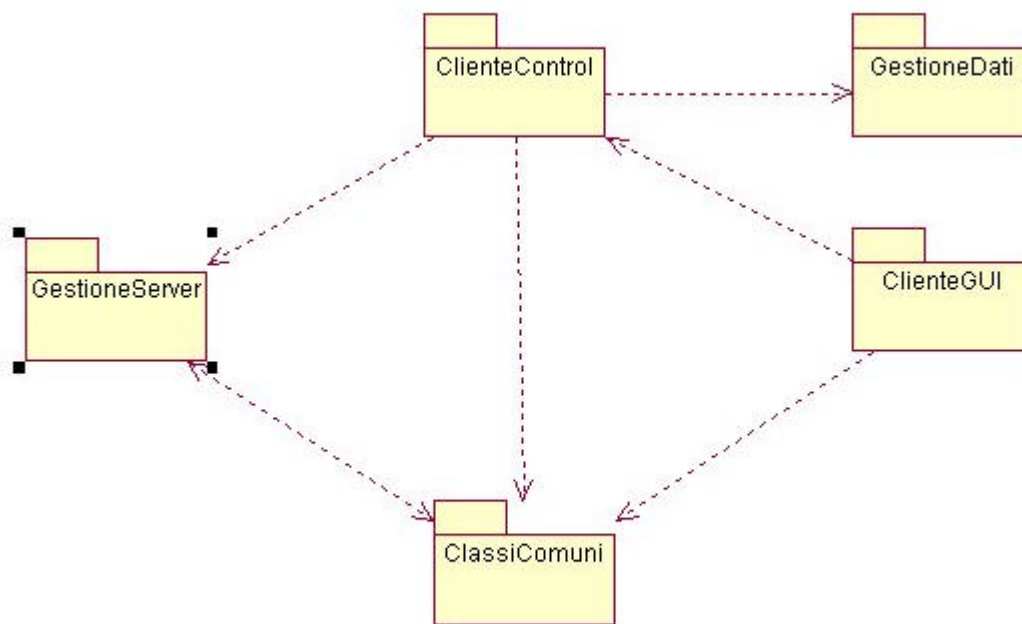
I sottosistemi indicati sono in esecuzione su macchine diverse e comunicano (tutti col sottosistema server) in maniera remota.

Il sistema è stato partizionato nei sottosistemi che sono in esecuzione su (nodi) computer diversi. Ognuno di questo sottosistema indica i package di classi java dedicate alla gestione delle funzioni che vengono offerte.

2.2.1 Package cliente

Il package Cliente è composto dai package ClienteControl e ClienteGUI.

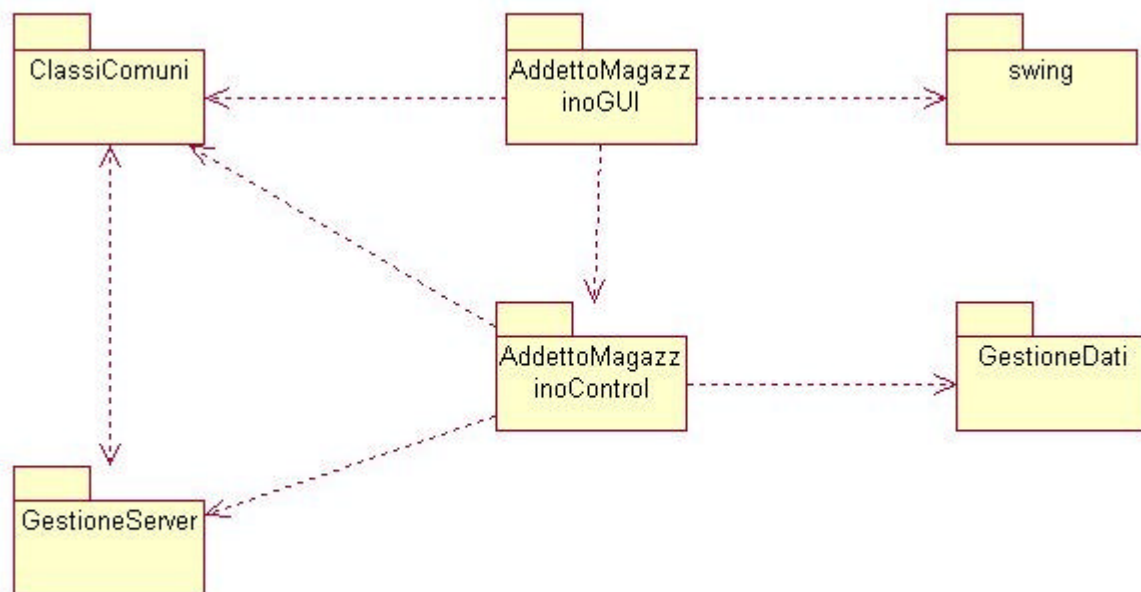
Riportiamo di seguito il diagramma che fornisce una vista dettagliata di questo package e le relazioni con gli altri package del sistema.



2.2.2 Package FinestraAddettoMagazzino

Il package FinestraAddettoMagazzino è composto dai package AddettoMagazzinoControl e AddettoMagazzinoGUI.

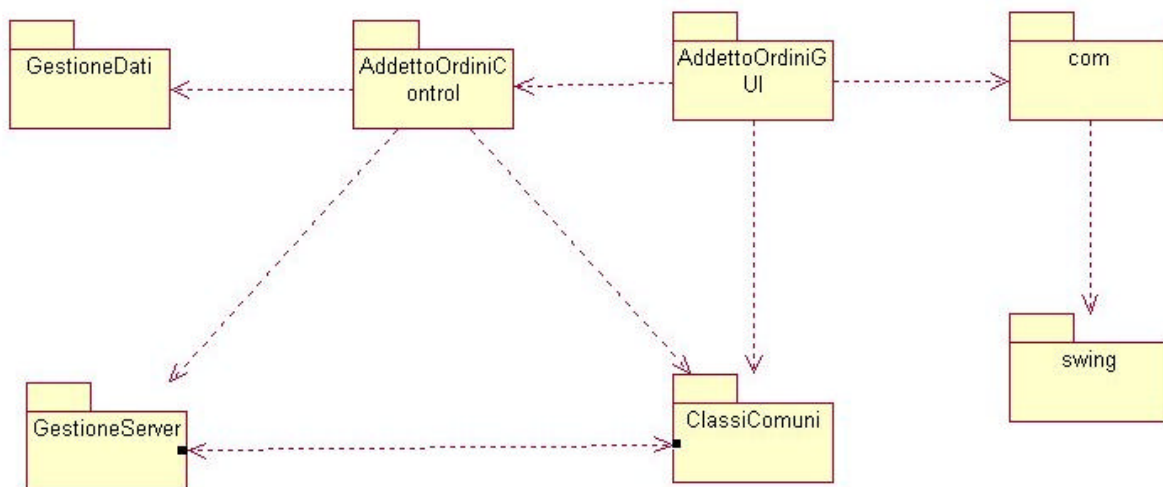
Riportiamo di seguito il diagramma che fornisce una vista dettagliata di questo package e le relazioni con gli altri package del sistema.



2.2.3 Package FinestraAddettoOrdini

Il package FinestraAddettoOrdini è composto dai package AddettoOrdiniControl e AddettoOrdiniGUI. Tali package ,insieme ad altri, permettono di realizzare tutte le funzionalità messe a disposizione dell'Addetto agli ordini.

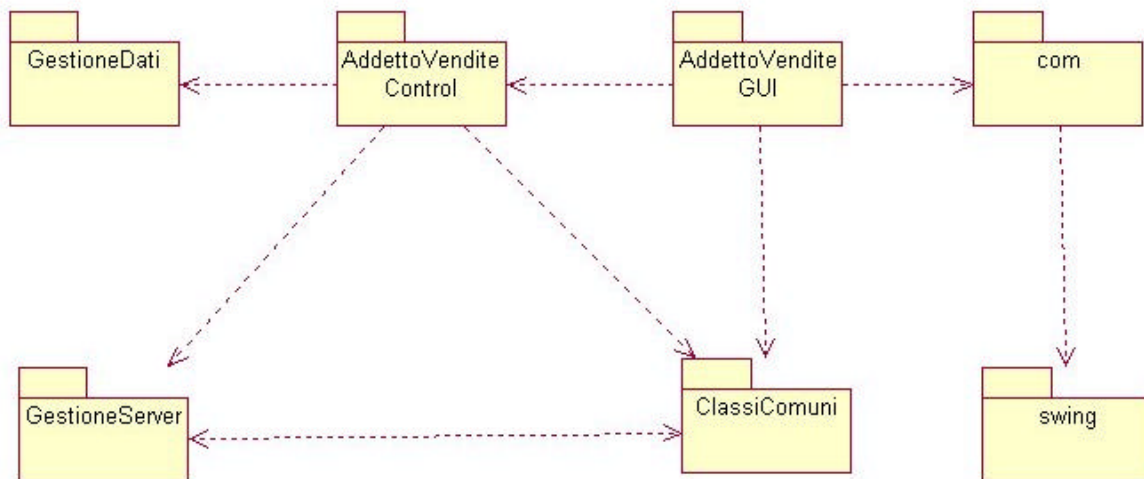
Riportiamo di seguito il diagramma che fornisce una vista dettagliata di questo package e le relazioni con gli altri package del sistema:



2.2.4 Package FinestraAddettoVendite

Il package FinestraAddettoVendite è composto dai package AddettoVenditeControl e AddettoVenditeGUI. Tali package, insieme ad altri, permettono di realizzare tutte le funzionalità che vengono presentate all'addetto alle vendite.

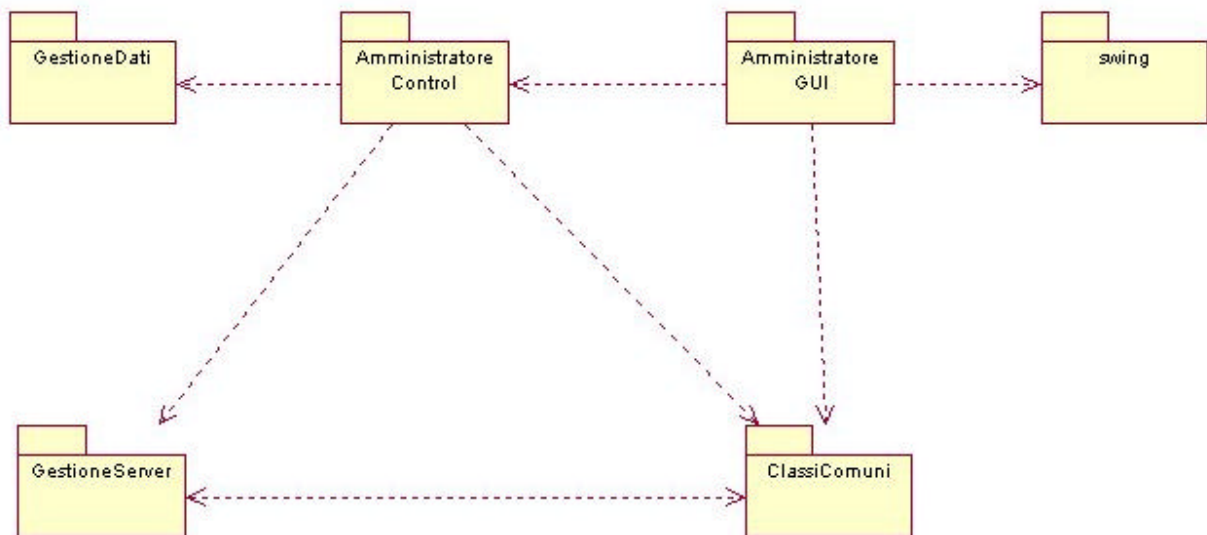
Riportiamo di seguito il diagramma che fornisce una vista dettagliata di questo package e le relazioni con gli altri package del sistema.



2.2.5 Package Amministratore

Il package Amministratore è composto dai package AmministratoreControl e AmministratoreGUI. Tali package, insieme ad altri, permettono di realizzare tutte le funzionalità che vengono presentate all'addetto all'amministratore.

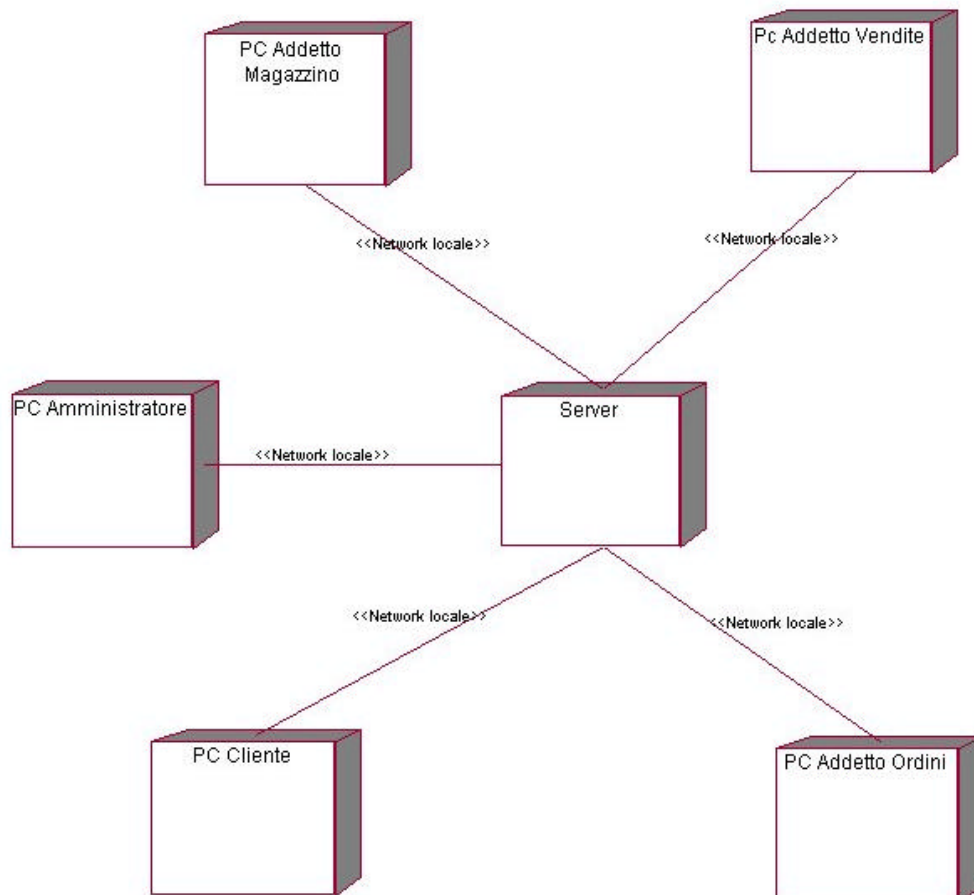
Riportiamo di seguito il diagramma che fornisce una vista dettagliata di questo package e le relazioni con gli altri package del sistema.



2.3 Hardware-Software mapping

L'intero sistema risulta essere composto da sei componenti software principali. Tali componenti non risiedono sulla stessa macchina ma su macchine differenti. I componenti sono inoltre indipendenti gli uni dagli altri tranne che per il server che è l'unico componente che comunica con gli altri. A parte questa comunicazione gli altri componenti sono fra loro isolati. Mostriamo di seguito come tale struttura venga modellata mediante l'utilizzo del diagramma di deployment.

2.3.1 Diagramma di Deployment del sistema.



2.3.2 Diagramma delle componenti edl sistema

Riportiamo adesso il diagramma delle componenti del sistema.

Indichiamo con componente “un elemento del sistema che è una entità autocontenuta che fornisce servizi ad altri componenti od attori”.³In pratica moduli software che interagiscono fra loro richiedendo ed fornendo servizi.⁴

Il diagramma contiene solo alcune delle componenti che sono state associate ad ogni nodo hardware del nostro sistema e che ci sembra siano rappresentative sia dell'esecuzione del sistema nelle sue parti allocate sui vari nodi sia delle dipendenze a tempo di esecuzione delle componenti stesse e quindi delle varie parti del sistema.

Nel realizzare tale diagramma abbiamo anche tenuto conto delle indicazioni che vengono date dalla guida del Rationale Rose in relazione al progetto di sistemi attraverso l'utilizzo del linguaggio di programmazione Java. In effetti viene detto che una componente viene indicata da un file java.⁵ Pertanto tale diagramma rappresenta le dipendenze fra esse intendendo tali relazioni di dipendenza col significato che :

le classi contenute nel componente client

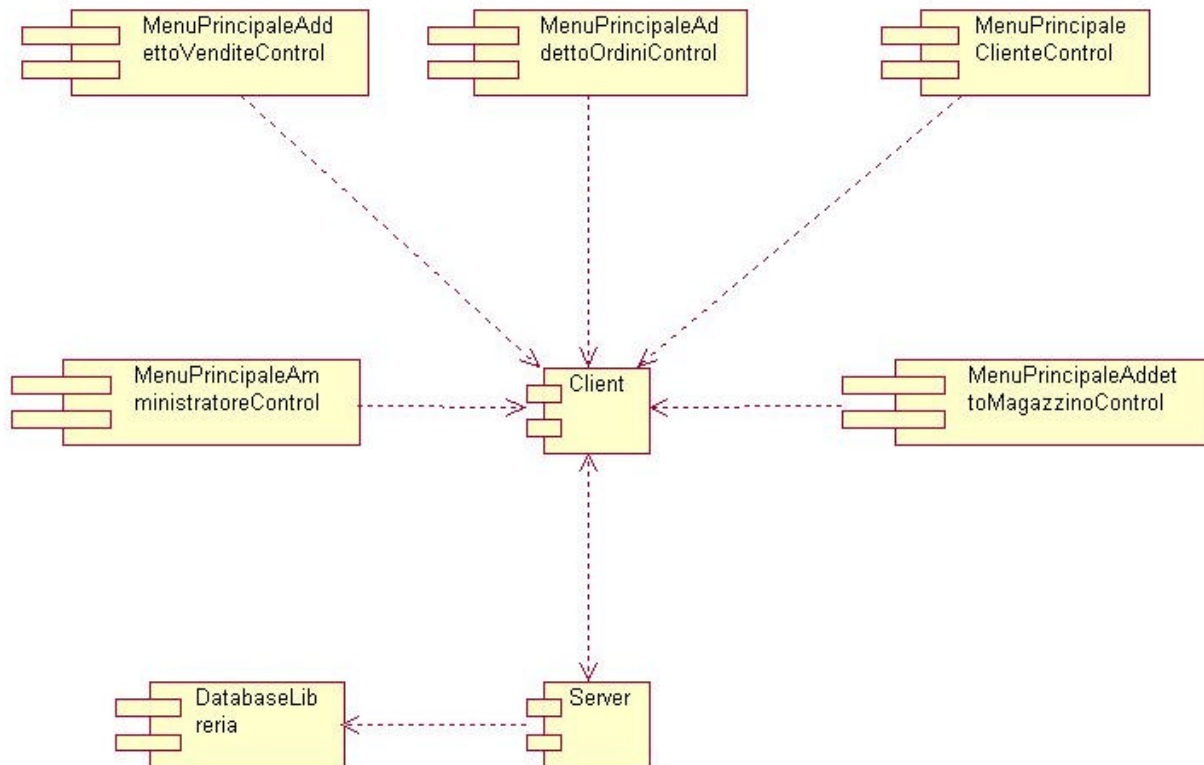
- ereditano da
- contengono istanze di
- usano

³ pag. 188, par. 6.3.6 , libro: Object-Oriented Software Engineering

⁴ riportiamo dalla guida del Rational Rose :”A component represents a software module (source code, binary code, executable, DLL, etc.) with a well-defined interface”.

⁵ riportiamo dalla guida di Rational Rose : “Rational Rose J models *.java files as Rose components in a model's Component View.”

le classi che si trovano sul componente che fornisce il servizio.⁶



In questo diagramma Client è un componente che si trova su ogni nodo del nostro sistema. Lo si è voluto mettere in evidenza per mostrare come esso fornisca un servizio ai vari componenti e a sua volta ne richieda uno al

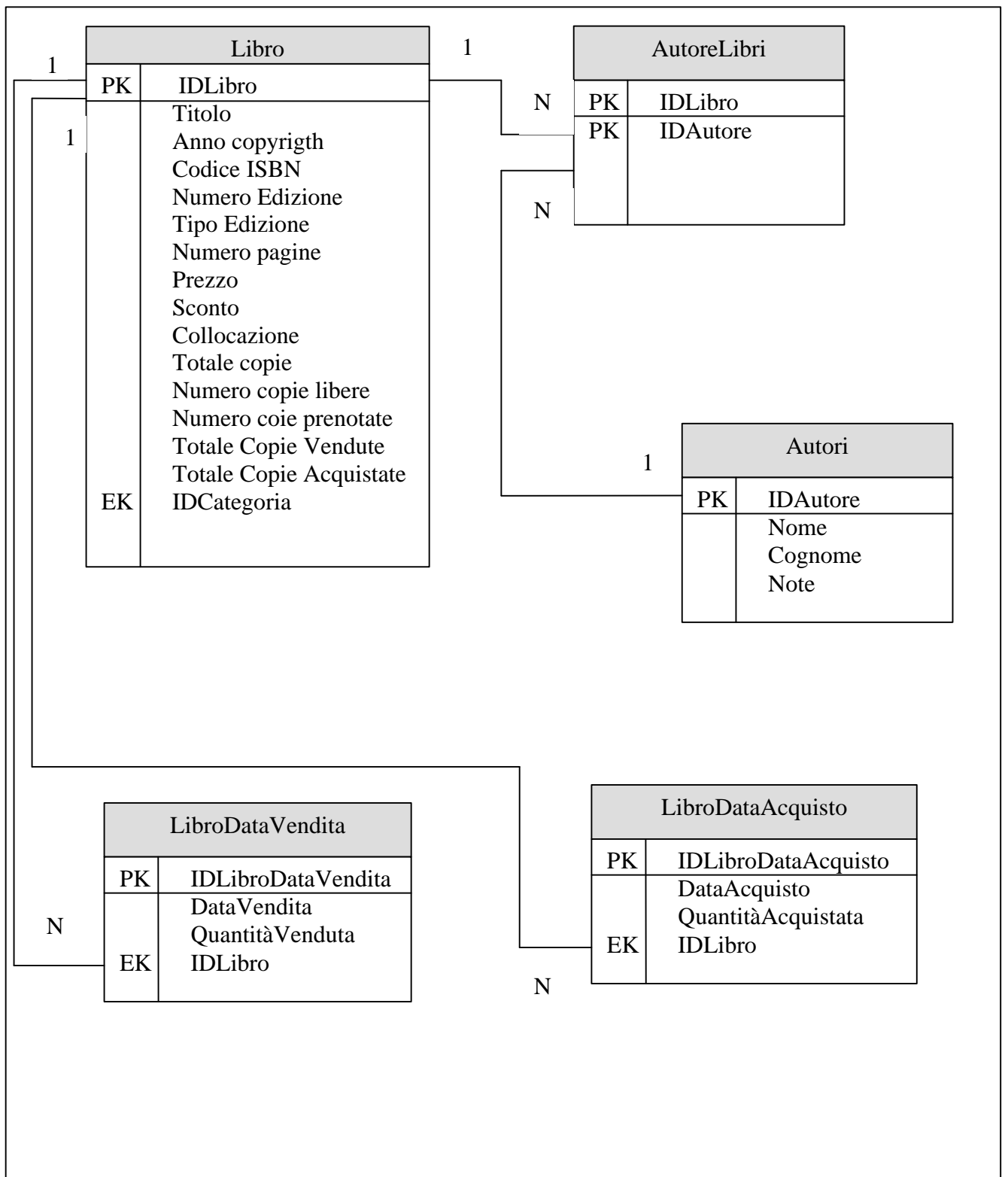
⁶ Riportiamo dalla guida del Rational Rose : "dependency relationship means that the classes contained in the client component can inherit from, contain instances of, use, and otherwise depend on classes that are exported from the supplier component. "

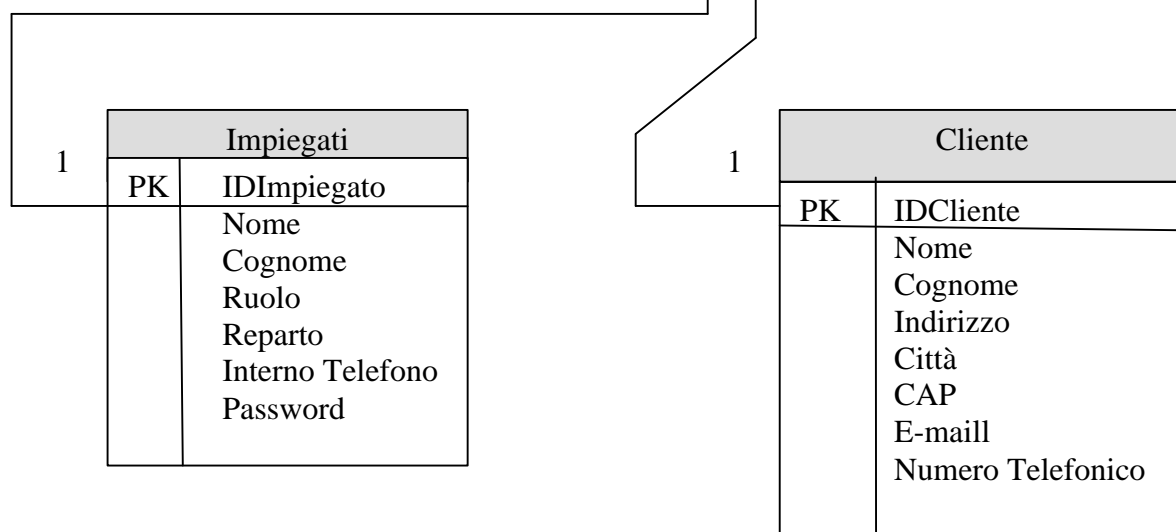
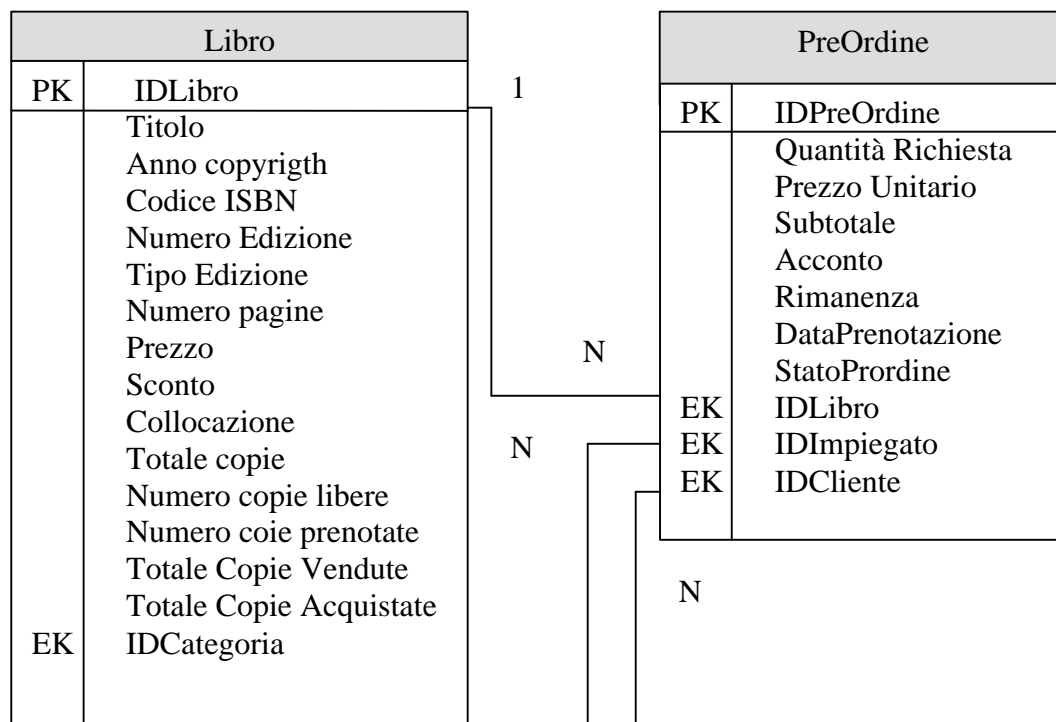
componente Server. Le componenti Server e DatabaseLibreria sono invece in esecuzione sulla postazione server.

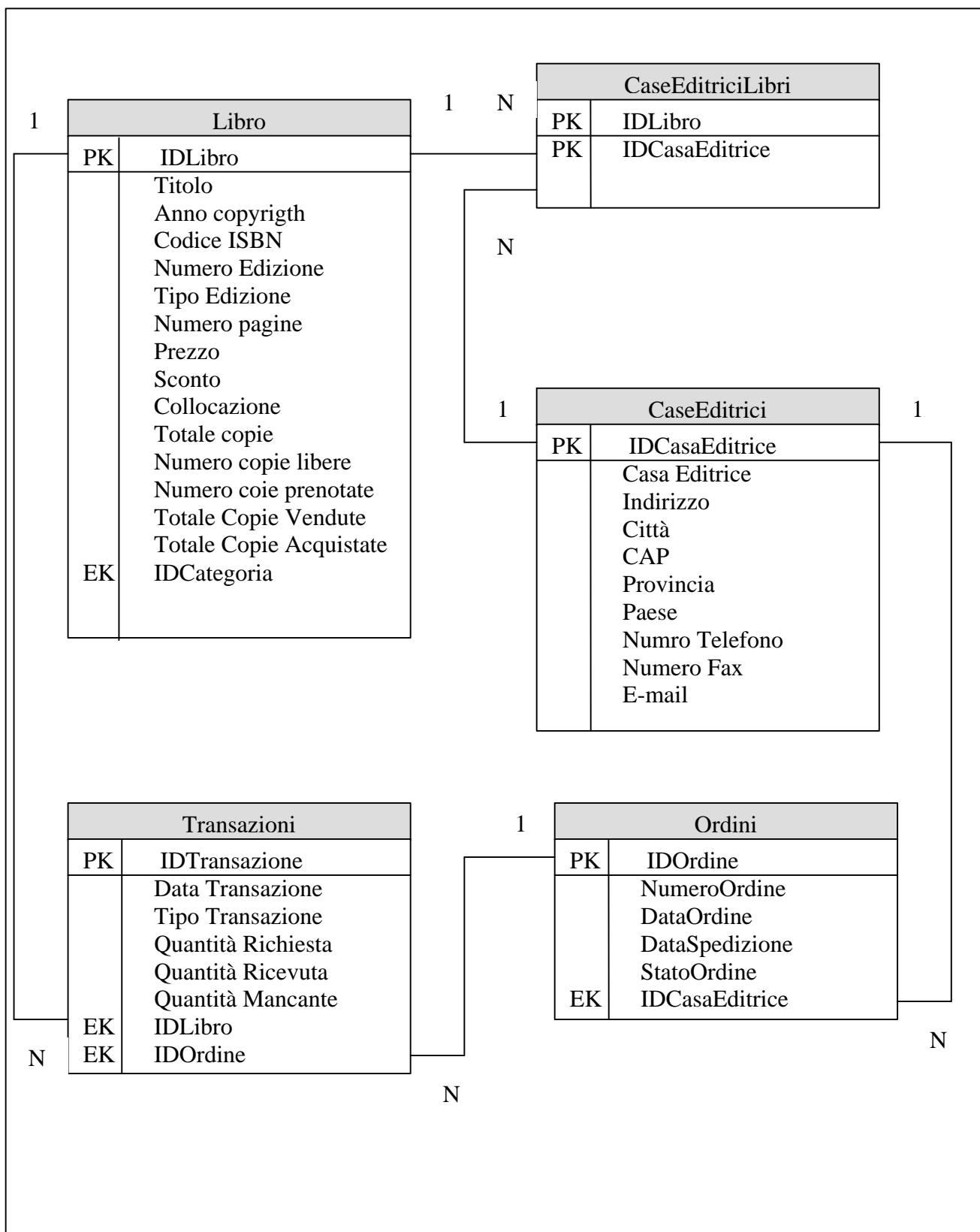
2.4 Data Management

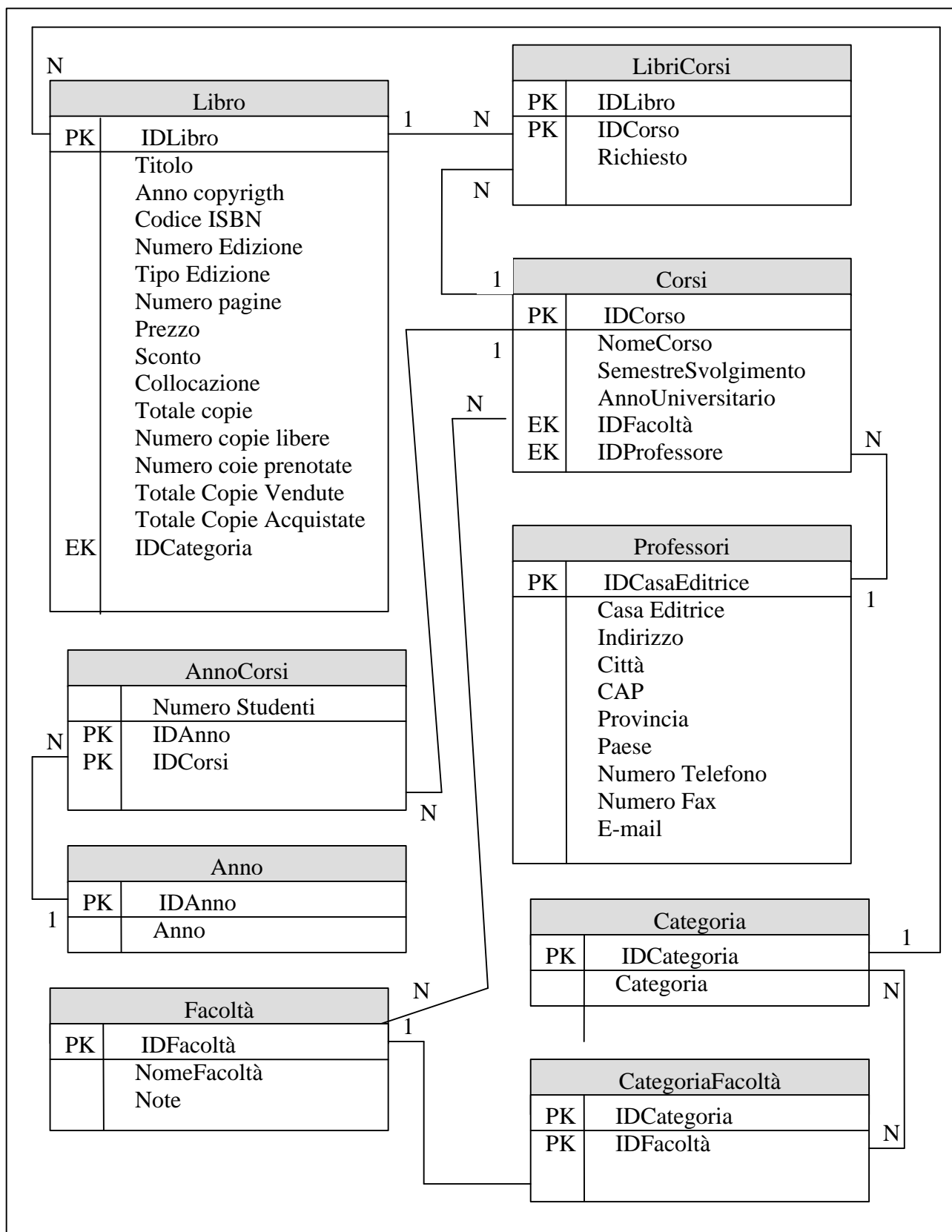
2.4.1 Progetto logico del Database

L'archivio dati risulta essere la struttura centrale più importante per il sistema di gestione della libreria. Infatti tutte le operazioni e le funzioni che vengono fornite dal sistema necessitano della ricerca. Si è scelto come modello dati un modello relazionale basato sulle relazioni esistenti fra varie tabelle che raggruppano informazioni. Il database nella sua struttura a tabelle ha la seguente costruzione in cui tutti gli elementi sono ottenuti dall'analisi fatta nel documento di progettazione concettuale. Tale sezione si trova nel RAD. Dagli ERD presentati in tale parte otteniamo le seguenti tabelle con le seguenti relazioni:









La struttura del Database è stata poi realizzata con l'ausilio del programma Microsoft Access.

Riportiamo qui di seguito la struttura delle tabelle che sono state usate nel nostro database:

Libri

Campo	Tipo
<u>IDLibro(PK)</u>	Contatore
Titolo	Testo(70)
Anno Copyright	Numerico(intero)
Codice ISBN	Testo(25)
Numero Edizione	Numerico(Byte)
Tipo Edizione	Testo(50)
Numero Pagine	Numerico(Intero)
Prezzo	Valuta[in euro]
Sconto	Testo(4)
Collocazione	Testo(30)
Totale copie	Numerico(Intero)
Numero copie libere	Numerico(Intero)
Numero copie prenotate	Numerico(Intero)

Totale Copie Vendute	Numerico(Intero lungo)
Totale Copie Acquistate	Numerico(Intero lungo)
IDCategoria	Numerico(Intero lungo)

Autori

Campo	Tipo
<u>IDAutore</u> (PK)	Contatore
Nome	Testo(50)
Cognome	Testo(50)
Note	Testo(70)

AutoriLibri

Campo	Tipo
IDAutore	Numerico(intero lungo)
IDLibro	Numerico(intero lungo)

La tabella precedente realizza la relazione scritto (fra autore e libro)

Categoria

Campo	Tipo
<u>IDCategoria</u> (PK)	Contatore
Categoria	Testo(50)

Facoltà

Campo	Tipo
<u>IDFacoltà</u> (PK)	Contatore
NomeFacoltà	Testo(50)
Note	Testo(70)

CategoriaFacoltà

Campo	Tipo
IDCategoria	Numerico(intero lungo)
IDFacoltà	Numerico(intero lungo)

La tabella precedente realizza la relazione trattata(fra Categoria e Facoltà).

PreOrdine

Campo	Tipo
<u>IDPreOrdine</u> (PK)	Contatore
Quantità Richiesta	Numerico(Intero)

Prezzo Unitario	Valuta
Subtotale	Valuta
Acconto	Valuta
Rimanenza	Valuta
DataPrenotazione	Data/ora
StatoPreordine	Testo(20)
IDLibro	Numerico(intero lungo)
IDImpiegato	Numerico(intero lungo)
IDCliente	Numerico(intero lungo)

Clienti

Campo	Tipo
<u>IDCliente</u>	Contatore
Nome	Testo(30)
Cognome	Testo(30)
Indirizzo	Testo(50)
Città	Testo(30)
CAP	Testo(10)
E-mail	Testo(50)
Numero Telefonico	Testo(30)

Impiegati

Campo	Tipo
<u>IDImpiegato</u>	Contatore
Nome	Testo(30)
Cognome	Testo(30)
Ruolo	Testo(50)
Reparto	Testo(30)
Interno Telefono	Testo(20)
Password	Testo(10)

LibroDataAcquisto

Campo	Tipo
<u>IDLibroDataAcquisto</u>	Contatore
DataAcquisto	Data/ora
QuantitàAcquistata	Numerico(intero)
IDLibro	Numerico(intero lungo)

LibroDataVendita

Campo	Tipo
<u>IDLibroDataVendita</u>	Contatore
DataVendita	Data/ora

QuantitàVenduta	Numerico(intero)
IDLibro	Numerico(intero lungo)

CaseEditrici

Campo	Tipo
<u>IDCasaEditrice</u>	Contatore
Casa Editrice	Testo(70)
Indirizzo	Testo(50)
Città	Testo(30)
CAP	Testo(10)
Provincia	Testo(30)
Paese	Testo(50)
Numero Telefono	Testo(30)
Numero Fax	Testo(30)
E-mail	Testo(50)

CaseEditriciLibri

Campo	Tipo
IDLibro	Numerico(Intero Lungo)
IDCasaEditrice	Numerico(Intero Lungo)

La tabella precedente realizza la relazione pubblicato (fra Libro e Casa Editrice).

Ordini

Campo	Tipo
<u>IDOrdine</u>	Contatore
NumeroOrdine	Testo(50)
DataOrdine	Data/ora
DataSpedizione	Data/ora
StatoOrdine	Testo(20)
IDCasaEditrice	Numerico(intero lungo)

Transazioni

Campo	Tipo
<u>IDTransazione</u>	Contatore
Data Transazione	Data/ora
Tipo Transazione	Testo(50)

Quantità Richiesta	Numerico(intero)
Quantità Ricevuta	Numerico(intero)
Quantità Mancante	Numerico(intero)
IDLibro	Numerico(intero lungo)
IDOrdine	Numerico(intero lungo)

Anno

Campo	Tipo
<u>IDAnno</u>	Contatore
Anno Corso	Numerico(Intero lungo)

Corsi

Campo	Tipo
<u>IDCorso</u>	Contatore
NomeCorso	Testo(50)
SemestreSvolgimento	Numerico(1/2)
AnnoUniversitario	Testo(20)
IDFacoltà	Numerico(intero lungo)
IDProfessore	Numerico(intero lungo)

AnnoCorsi

Campo	Tipo
IDanno	Numerico(intero lungo)
IDCorsi	Numerico(intero lungo)
IDAnno	Numerico(intero lungo)
Numero Studenti	Numerico(Intero)

La tabella precedente realizza la relazione si tiene(fra Corso e Anno)

LbriCorsi

Campo	Tipo
IDLibro	Numerico(intero lungo)
IDCorso	Numerico(intero lungo)
Richiesto	Testo(15)[si/no]

La tabella precedente realizza la relazione adottato(fra Libro e Corso)

Professori

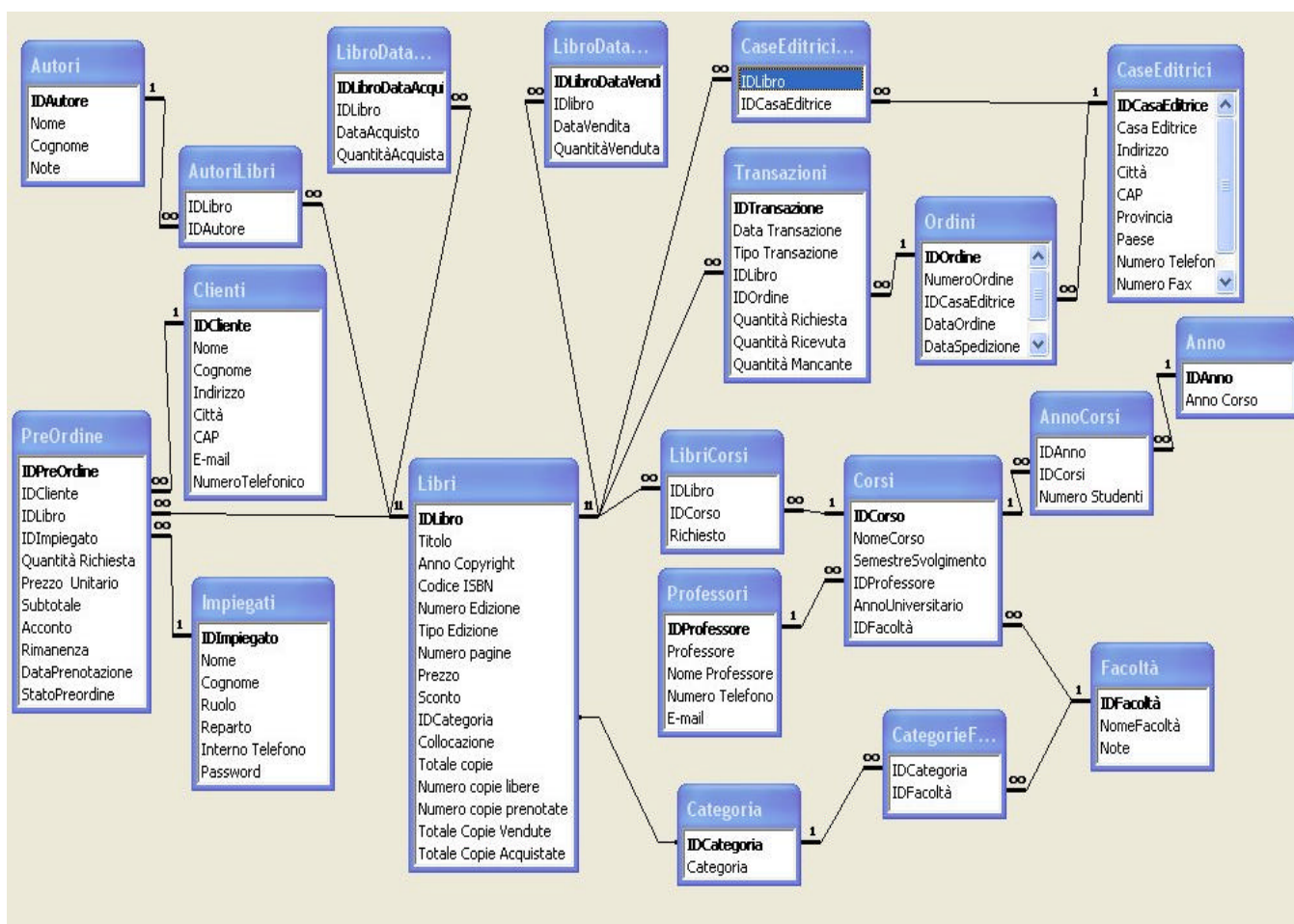
Campo	Tipo
<u>IDProfessore</u>	Contatore
Nome	Testo(30)
Cognome	Testo(30)
Numero Telefonico	Testo(30)
E-mail	Testo(50)

2.4.2 Progetto fisico del database

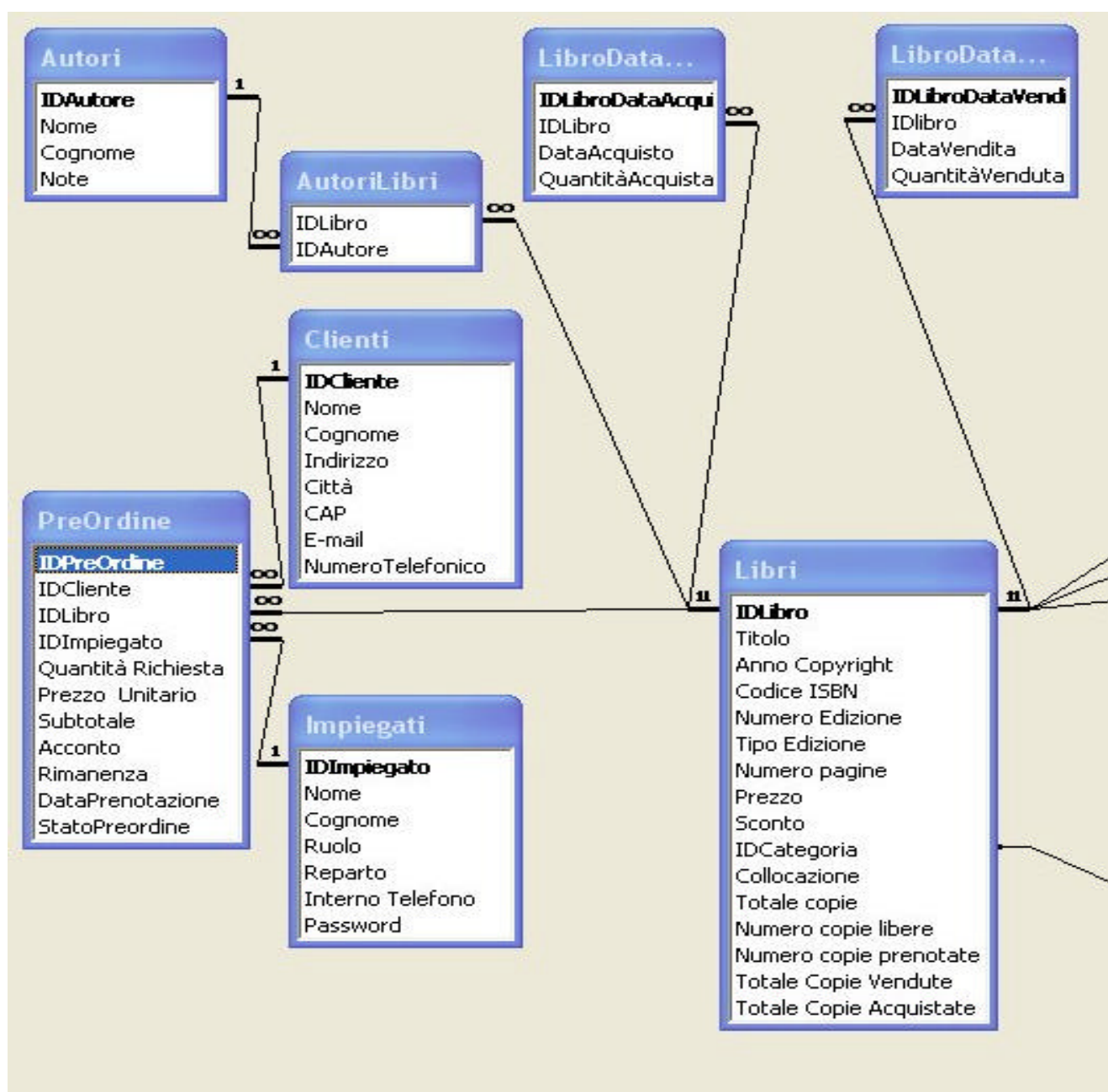
L'archivio dei dati che viene utilizzato dal programma è stato realizzato con Microsoft Access.

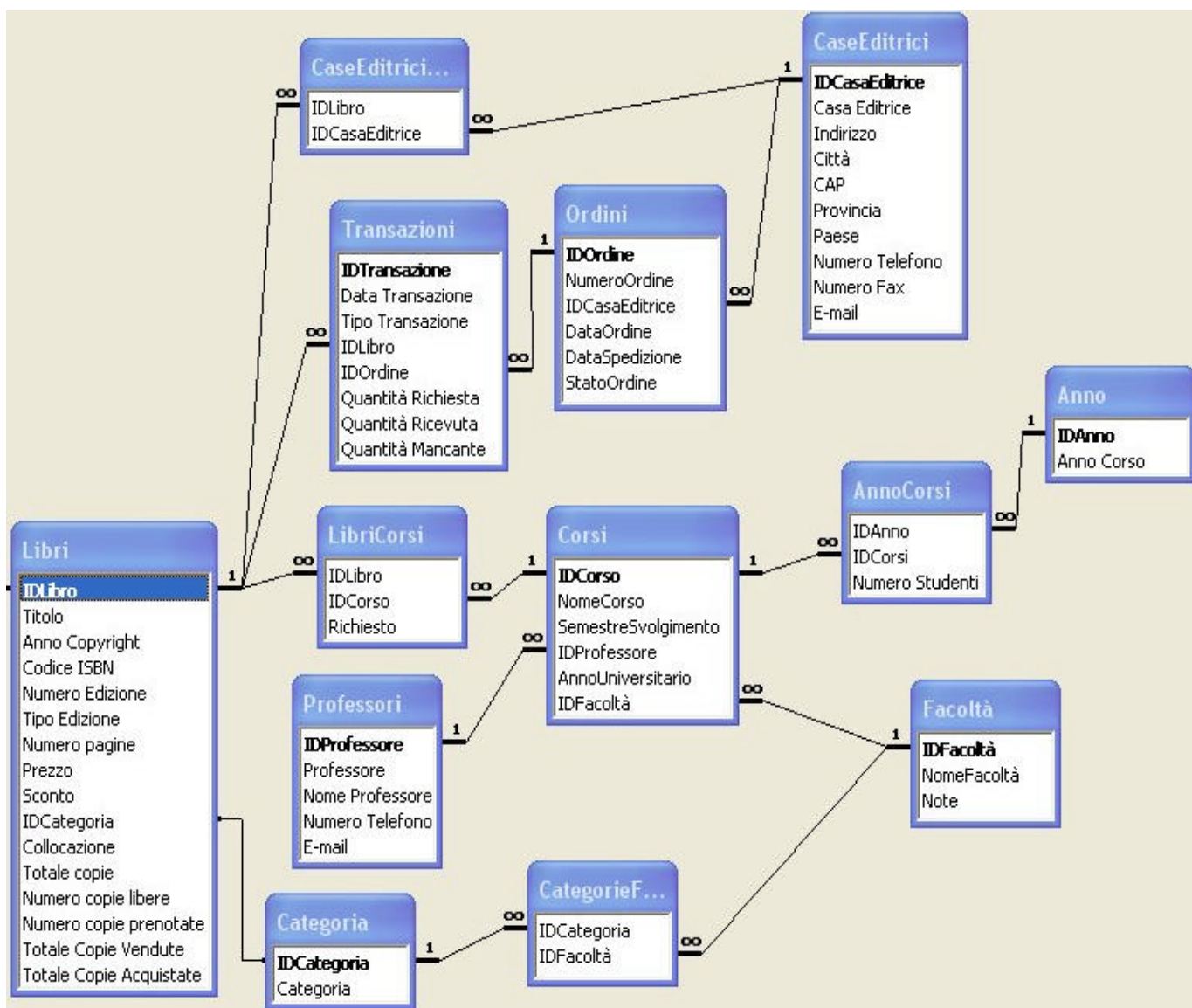
Lo scambio dei dati fra la struttura dati costruita ed il sistema realizzato e la manipolazione dei dati stessi dell'archivio mediante il linguaggio di programmazione Java è stato possibile grazie all'utilizzo del JDBC. In particolare modo è stato utilizzato il driver della sun : "sun.jdbc.odbc.JdbcOdbcDriver".

Riportiamo qui di seguito la rappresentazione sotto Access delle relazioni che esistono fra le varie tabelle che sono state realizzate.



In seguito vengono mostrate più in dettaglio le varie parti del database.





Il file contenente il database che viene gestito dal programma si trova nella cartella:

Dati\Libreria.mdb.

2.5 Controllo degli accessi al sistema e sicurezza

Dato che i compiti svolti dai vari “attori” nel nostro sistema sono differenti ed accedono ad informazioni diverse dell’archivio al fine di evitare :

- la visualizzazione di dati specifici per un certo compito e non utilizzati per altre mansioni;
- la modifica di dati fondamentali per la gestione della libreria stessa;
- e che le attività svolte dagli impiegati della libreria non possano essere ricondotte ad una specifica persona;

si è pensato di utilizzare più meccanismi di protezione dei dati e delle attività eseguite.

Il primo livello di protezione è un filtro che si è realizzato nell'accesso e nella modifica dei dati e che viene posto a livello delle funzionalità offerte ad ogni impiegato dal sistema in esecuzione sul proprio terminale, secondo il seguente schema:

- ◆ Cliente : può accedere solo in lettura ai dati relativi ai libri.
- ◆ Addetto Magazzino : può accedere ai dati del magazzino per visualizzare tutti i dati relativi ai libri agli ordini emessi ,all'insieme di tutte le transazioni ed a i libri ricevuti. Si è scelto di dare al magazziniere delle funzioni di sola consultazione per isolare l'insieme delle sue attività da quelle di modifica dei dati dei libri. Infatti abbiamo supposto che la fase di registrazione di libri in arrivo nel magazzino è collegata all'emissione di un ordine che deve essere a nostro avviso totalmente gestita dall'attore responsabile degli ordini. In questo modo le due attività sono fra loro isolate.

- ◆ Addetto agli Ordini : può accedere ai dati dell'archivio sia per consultazione che per modifica. La modifica dei dati è legata soltanto alle informazioni relative alla gestione di un ordine.
- ◆ Addetto alle vendite : può accedere all'archivio dati sia per consultazione che per modifica. Tale accessi permettono di manipolare solo i dati relativi alla vendita ed alla prenotazione di un libro.
- ◆ Amministratore : può accedere ad ogni informazione dell'archivio dati per consultazione e per modifiche.

Un secondo livello di protezione viene garantito durante la fase di accesso al sistema dalla necessità di dover inserire una password ed il proprio ID.

Questa operazione preliminare deve essere svolta da tutti gli utilizzatori del sistema ad eccezione del cliente.

L'utilizzo delle password insieme all'ID garantisce inoltre la possibilità di poter ricondurre ogni operazione di modifica dei dati ad un impiegato della libreria.

La gestione delle password inoltre viene effettuata solamente dall'Amministratore.

Esiste infine un terzo livello di protezione. In realtà non si tratta di un vero meccanismo di protezione ma piuttosto di un meccanismo di controllo che permette di visualizzare tutte le attività di comunicazione fra i vari terminali ed il server identificando l'impiegato che effettua l'operazione attraverso il suo ID.

Tale meccanismo viene realizzato attraverso l'utilizzo di un file di dati (Filelog.dat) che registra tutte le informazioni di ogni accesso al server e quindi al database che può essere consultato dall'amministratore mediante un programma di videoscrittura(Word).

2.6 Global Software Control

2.6.1 External control flow (fra sottosistemi)

Ogni sottosistema che si trova collocato sui vari nodi del nostro sistema è indipendente dagli altri. Dato che l'architettura del sistema è di tipo client/server il controllo del flusso nei vari client e nel server è indipendenti dagli altri, eccetto che per la comunicazione fra client e server stesso⁷. Infatti in tal caso si richiede la sincronizzazione per poter gestire le richieste e la ricezione dei dati.

Il funzionamento del server è piuttosto semplice : esso è sempre attivo ed aspetta la richiesta di un servizio da parte di un client. Se più client fanno una richiesta allora le richieste vengono soddisfatte dal server una alla volta e vengono gestite in modo sequenziale (chi prima arriva prima viene servito) .

⁷ dal libro Object-Oriented Software Engineering pag 186 par. Client/server architecture.

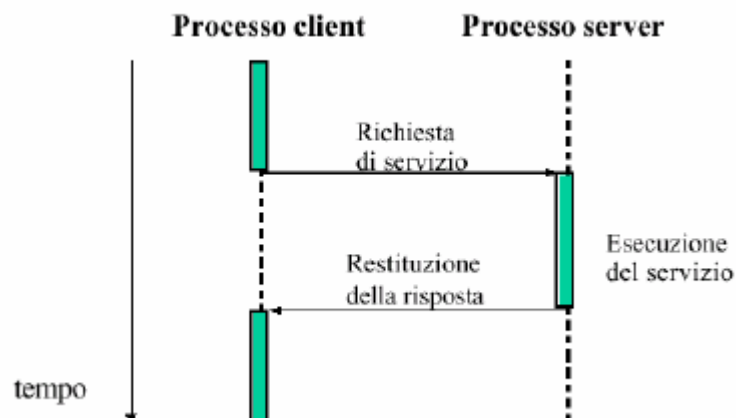
2.6.2 Controllo interno (per un singolo sottosistema)

All'interno dei vari sottosistemi il meccanismo di flow control che viene utilizzato è di tipo event-driven. Ossia abbiamo un ciclo principale che attende eventi esterni che vengono generati dai vari attori.

2.6.3 Concurrent control

Come abbiamo già detto ogni sottosistema è indipendente dagli altri ed è in esecuzione su una singola macchina. Tutti quindi possono essere in esecuzione in nello stesso tempo. L'unica risorsa che essi condividono è rappresentata dall'archivio dati e dal server che deve soddisfare le loro richieste . L'accesso separato alle risorse del database è garantito dal server stesso che, come detto in precedenza, soddisfa le una alla volta in modo sequenziale (chi prima arriva prima viene servito) .

Questa immagine mostra la relazione fra processo client e processo server in relazione al tempo.



2.7 Boundary Condition

Data la natura distribuita del sistema proposto l'avvio e l'esecuzione delle sue varie parti ossia dei sottosistemi che forniscono le varie funzionalità ai vari "attori" può avvenire indipendentemente. Inoltre per il corretto funzionamento di un sottosistema non è richiesto che gli altri siano attivi, eccetto che per il server.

E' richiesto infatti che sia prima avviato il server affinché una qualunque postazione client possa usufruire dei servizi offerti da esso. Sebbene questa sia una condizione indispensabile per il corretto funzionamento del sistema(nella sua interezza o in parte) se si verifica l'avvio di un sottosistema senza che il server sia attivo si avrà comunque una gestione di tale situazione da parte del sottosistema stesso.

Per quel che invece riguarda la termination phase del sistema , essa si realizza soltanto quando tutte i sottosistemi sono stati disattivati. Il sistema resta attivo comunque se due sottosistemi (di cui uno è il server) sono ancora in esecuzione.

La disattivazione di un singolo sottosistema avviene mediante la funzione di exit presente su tutte le varie postazioni. Questa funzionalità è però nascosta sulla postazione dedicata al cliente. Infatti per terminare l'esecuzione del sottosistema ivi presente si utilizza una procedura particolare consistente nell'immissione di una password nella finestra di ricerca per titolo.

Object Design Document (ODD)

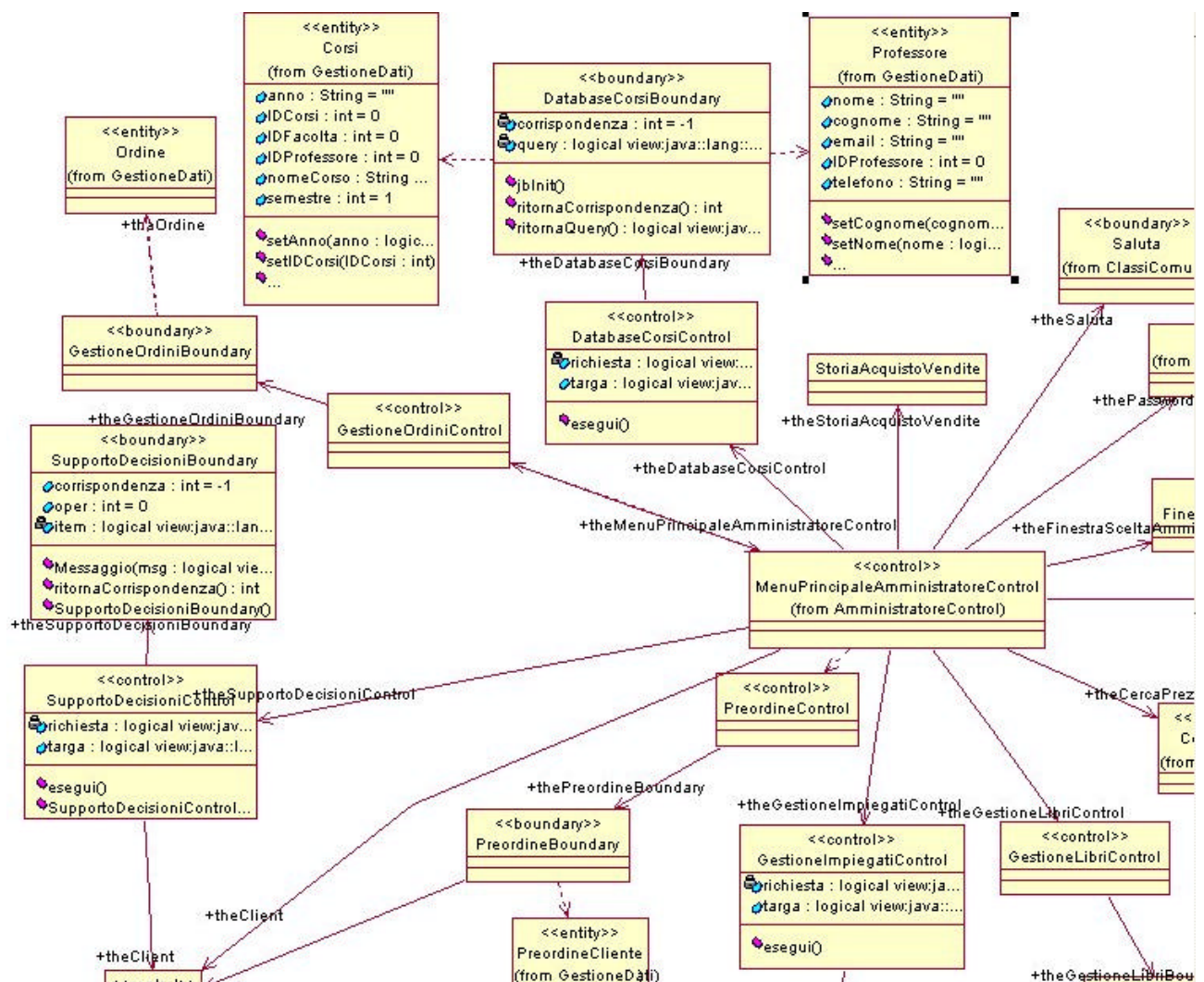
Diagramma delle classi del sistema (Class Diagram).

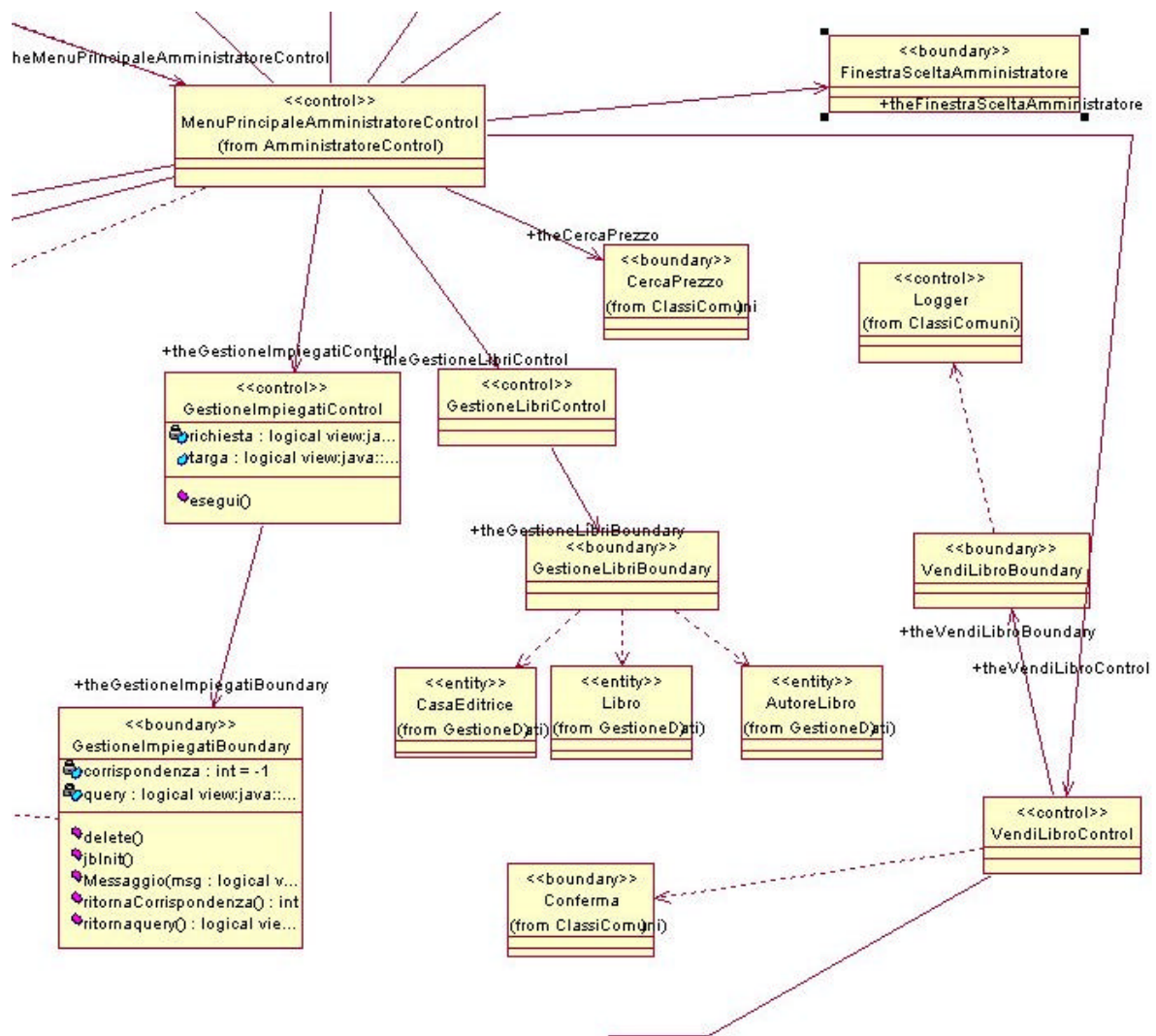
Presentiamo adesso i diagrammi delle classi che sono state utilizzate per la creazione del sistema di gestione della libreria universitaria.

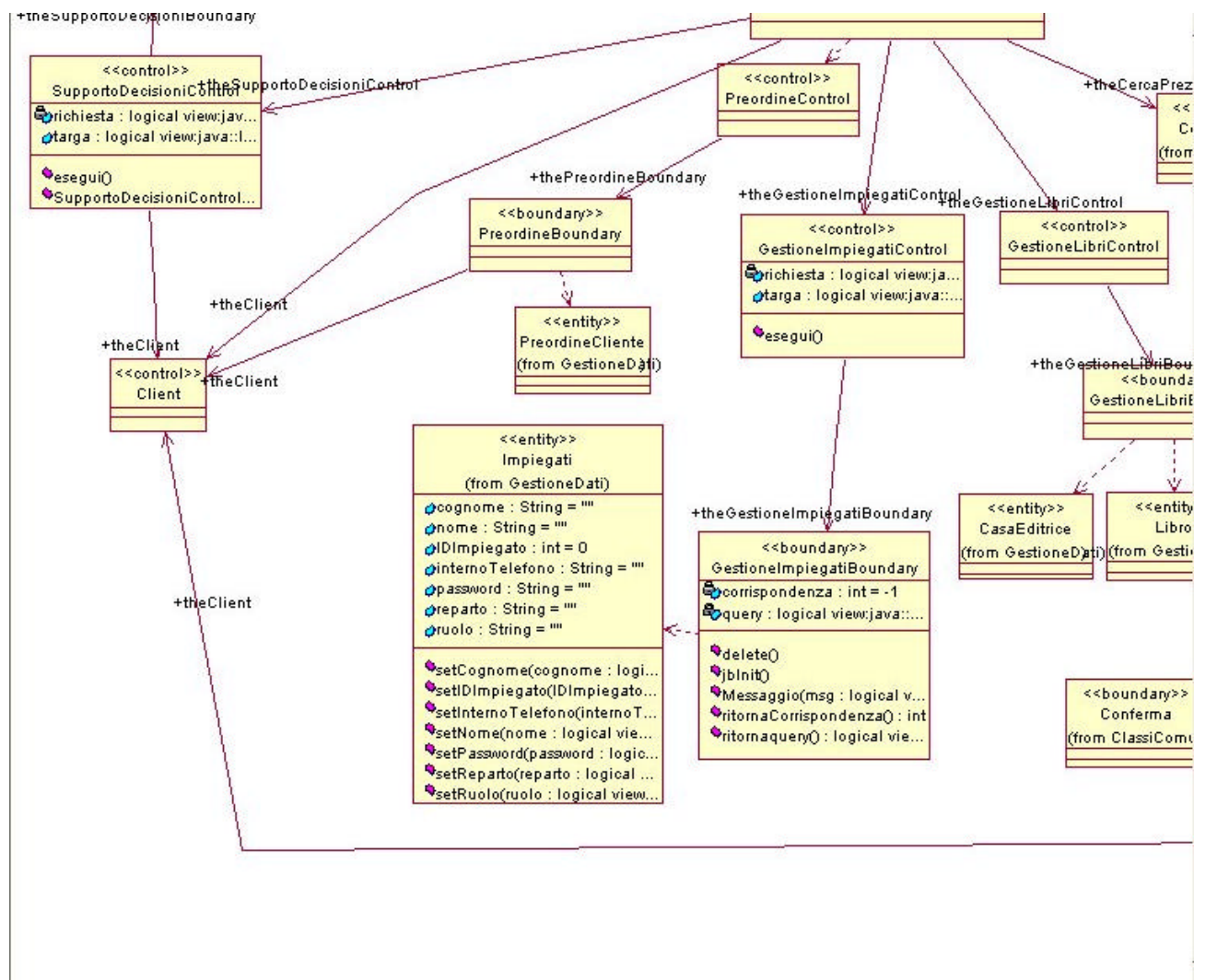
In tali diagrammi per motivi di visualizzazione alcune classi sono state raffigurate senza la presenza di attributi ed operazioni. Una descrizione completa di esse è però presentata successivamente.

Per una visione totale (grafica) di tutte le classi con i rispettivi attributi e metodi si rimanda al file NuovaLibreriaG&R2.mdl fornito nel CD allegato alla presente documentazione.

Diagramma delle classi relative al package Amministratore







Presentiamo adesso una vista più dettagliata delle classi che sono coinvolte per la funzione di gestione Libri.

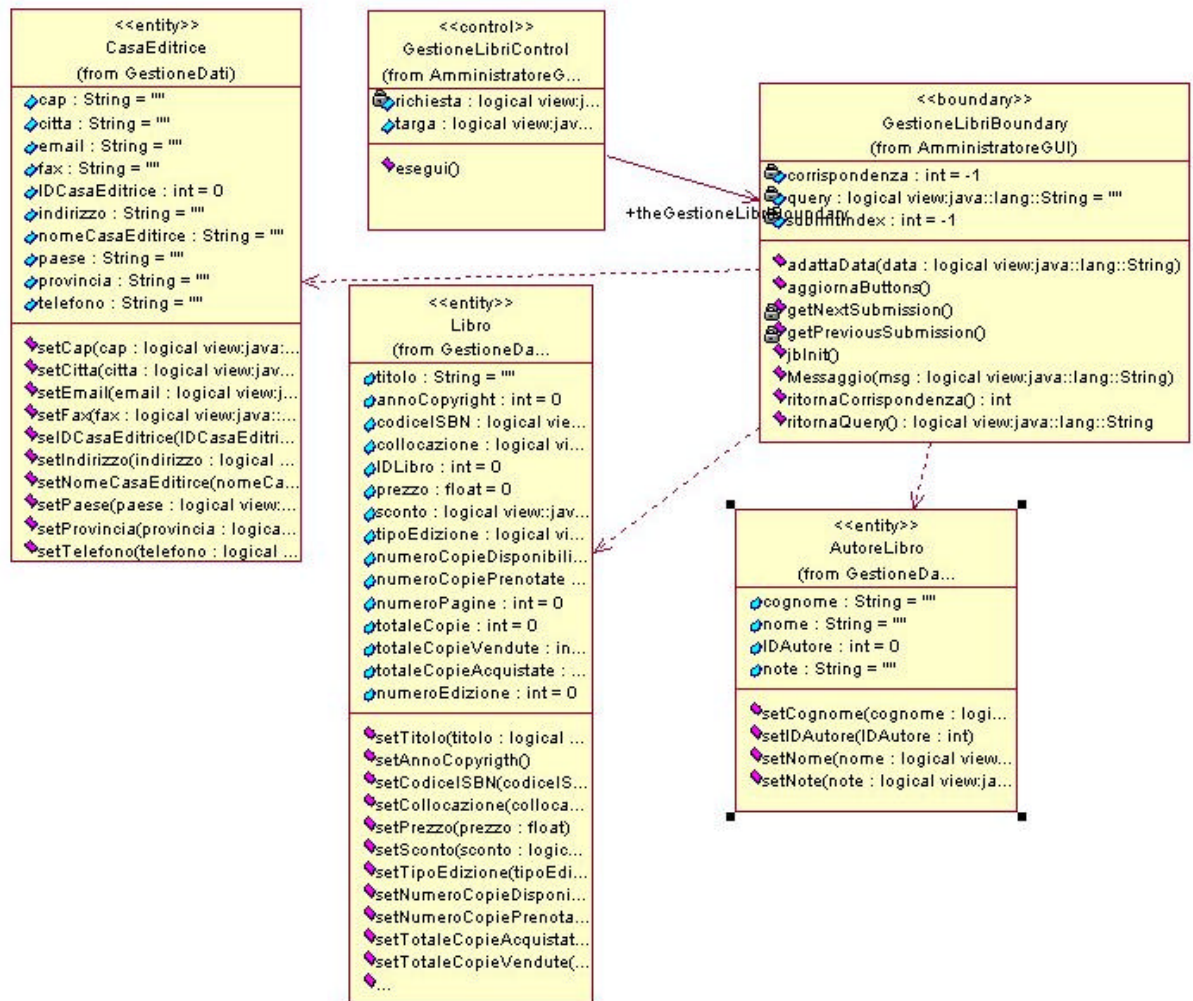


Diagramma delle classi relative al package Cliente

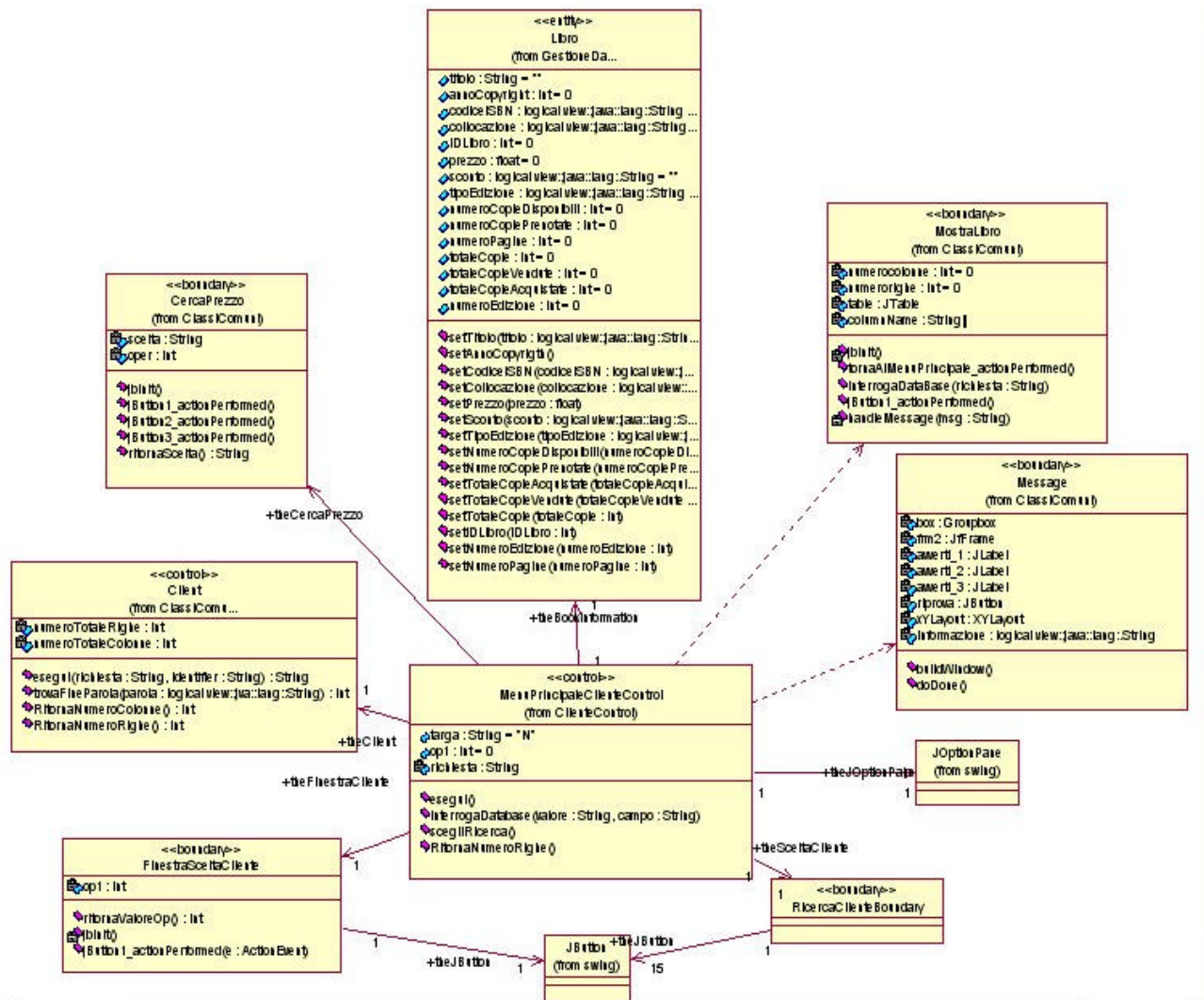


Diagramma delle classi relative al package FinestraAddettoMagazzino

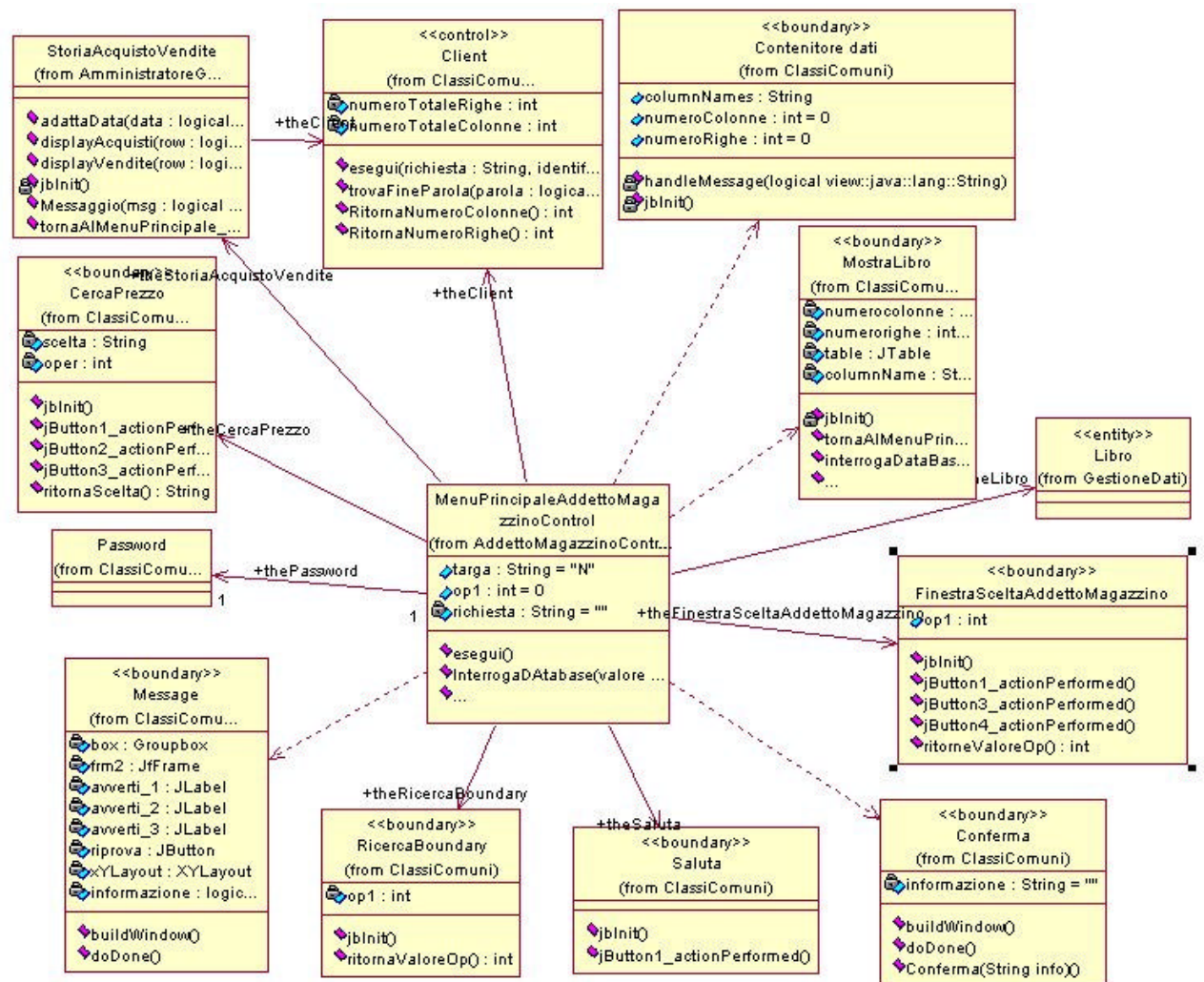
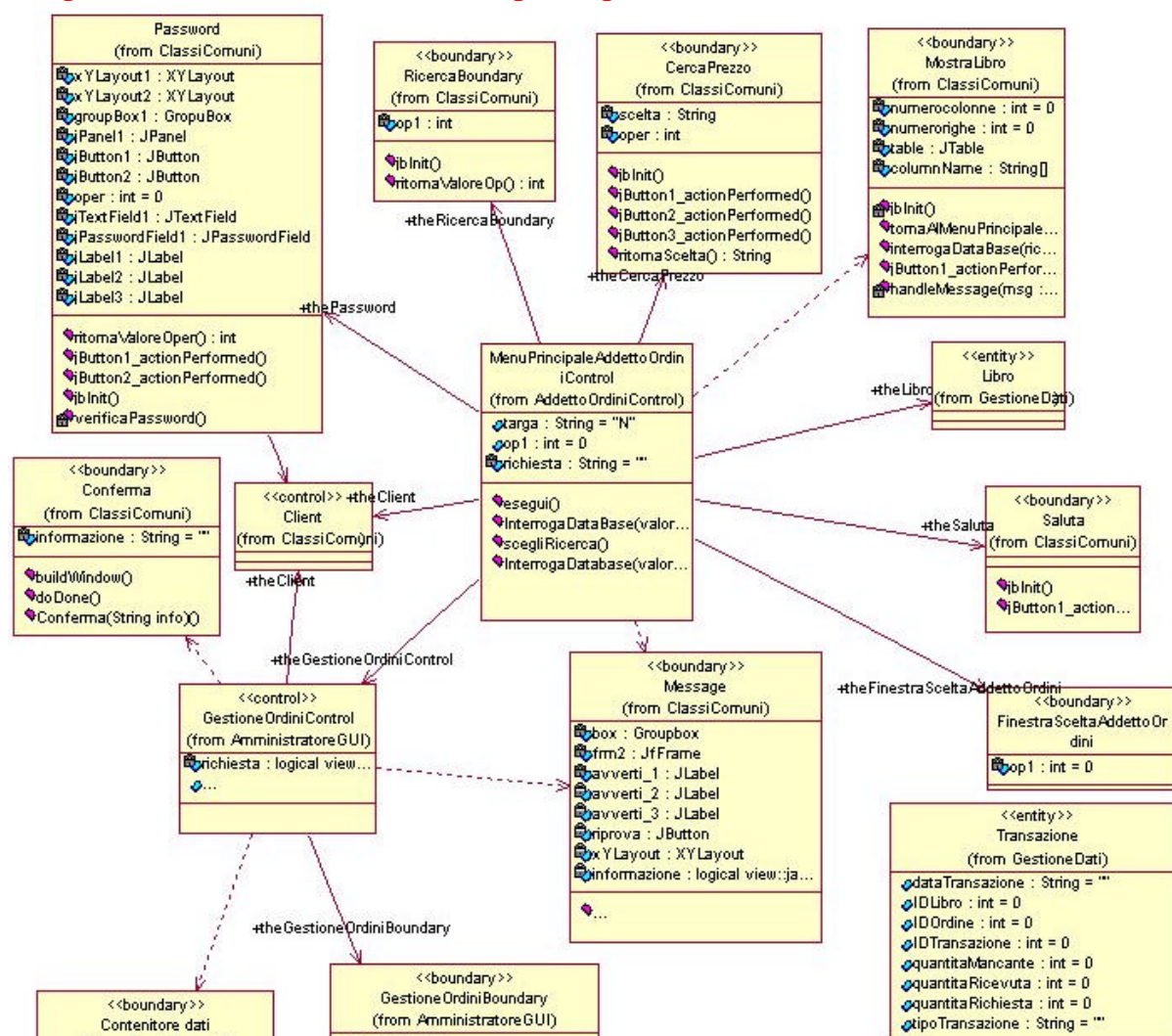


Diagramma delle classi relative al package FinestraAddettoOrdini



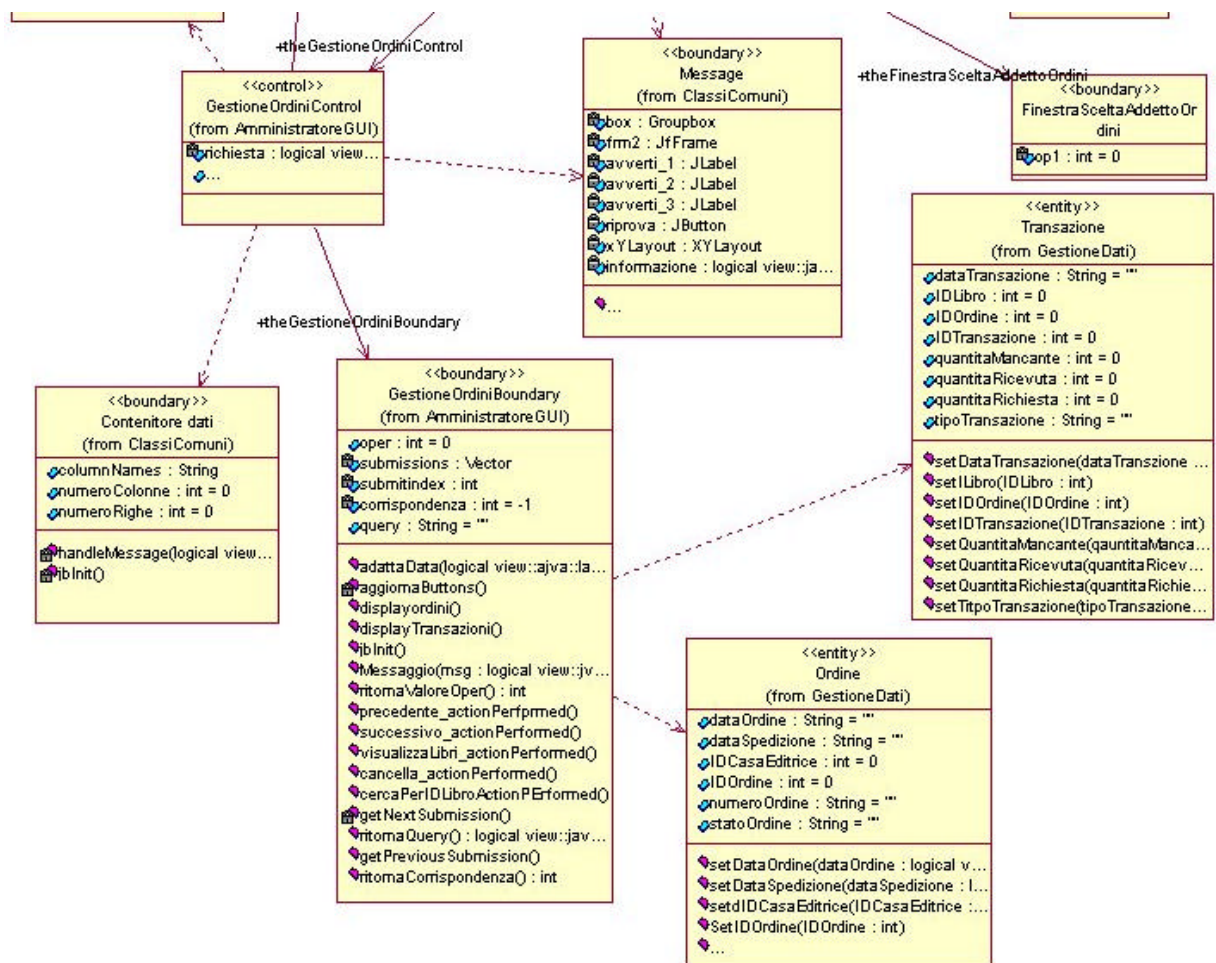
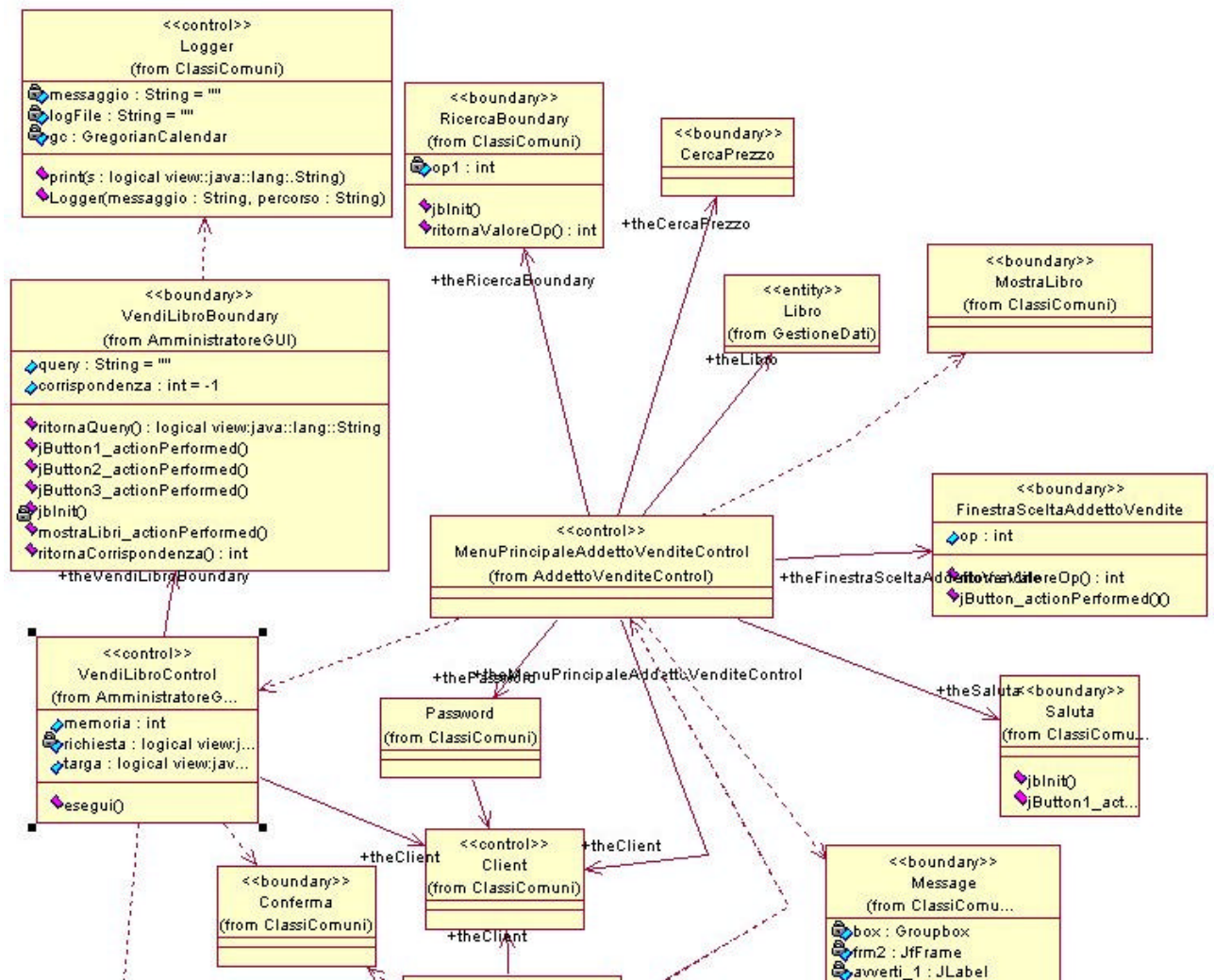
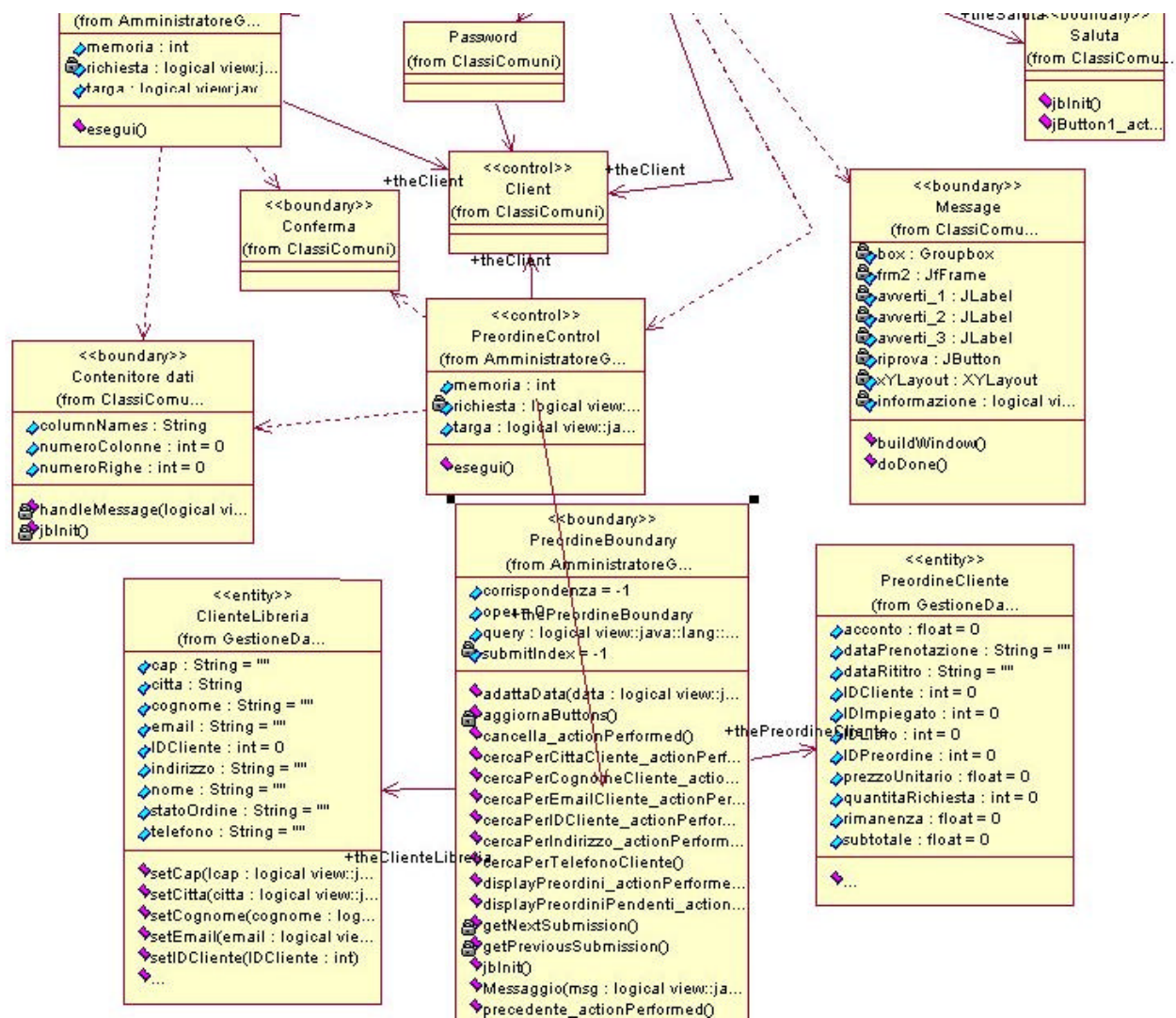


Diagramma delle classi relative al package FinestraAddettoVendite





4. Classi del sistema

4.1 Package Amministratore

4.1.1 AmministratoreControl.

MenuPrincipaleAmministratoreControl

E' la classe principale del package **Amministratore**. Si tratta di un classe di controllo che serve a chiamare ed a gestire tutte le altre classi di controllo facenti parte del package *Amministratore*. Questa classe all'avvio (nel metodo *main*) istanzia un oggetto di tipo *ClassiComuni.Password* che serve ad autenticare l'utente e ad abilitarlo, in caso di riscontro positivo, all'uso del sistema. In questo caso viene istanziato un oggetto di tipo, appunto, *MenuPrincipaleAmministratoreControl*. Invece, nel caso di riscontro negativo, cioè nel caso in cui l'utente non viene riconosciuto dal sistema, viene istanziato un oggetto di tipo *ClassiComuni.Message*, la cui funzione è quella di comunicare all'utente che i dati inseriti sono errati e di invitarlo a riprovare.

Questa classe estende l'interfaccia *ClassiComuni.MenuPrincipale*.

4.1.1.1 Attributi della classe

public static int op1

A questa variabile viene assegnato il valore ritornato dal metodo *ritornaValoreOp()* della classe *FinestraSceltaAmministratore*. Questo valore

corrisponde alla scelta fatta dall'utente del sistema nell'interfaccia relativa al menu principale che gli si presenta dopo che il processo di autenticazione è andato a buon fine. A seconda del valore della variabile *op1*, viene istanziata un'opportuna classe di controllo, la cui funzione è quella di svolgere il compito richiesto dall'utente.

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore "S", allora la query da effettuare sul database è di tipo *select*, quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all'insieme dei record trovati nel database. Se targa assume valore "N", la query da effettuare sul database non sarà del tipo *select*, ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle assegnato un valore, questa variabile viene passata al metodo *InterrogaDatabase(String)*, che a sua volta la passa come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

4.1.1.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *FinestraSceltaAmministratore*, ed in base al valore intero restituito dal metodo *ritornaValoreOp()* di questa classe, viene chiamata la classe di controllo corrispondente alla funzionalità richiesta nel menu principale dall'utente del sistema.

public void scegliRicerca()

Anche questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *ClassiComuni.RicercaBoundary*, ed in base al valore intero restituito dal metodo *ritornaValoreOp()* di questa classe, viene costruita una stringa rappresentante la query da inviare al database. Questa stringa viene poi passata al metodo *InterrogaDatabase(String)* che si occuperà dell'avvio del client.

public void InterrogaDatabase(String query)

Questo metodo viene chiamato all'interno del metodo *scegliRicerca()*. Inizialmente viene istanziata la classe *ClassiComuni.Client*. Il metodo *esegui* di quest'ultima classe ritornerà alla classe chiamante (cioè *MenuPrincipaleAmministratoreControl*) una matrice corrispondente all'insieme

dei record trovati nel database. Viene quindi istanziata la classe *ClassiComuni.MostraLibro* che si occuperà di mostrare in una tabella i dati ricevuti dal server. Nel caso in cui si verificano dei problemi relativi alla connessione tra client e server, viene visualizzato un messaggio di errore utilizzando la classe *ClassiComuni.Message*.

public static void main (String[] arg)

Nel metodo main viene istanziata la classe *ClassiComuni.Password* che serve ad autenticare l'utente e ad autorizzarlo all'uso del sistema.

Se l'utente inserisce i dati esatti, viene creato un oggetto *MenuPrincipaleAmministratoreControl* che avvia il menu principale utilizzabile dall'utente, nella fattispecie l'Amministratore della libreria.

4.1.2 AmministratoreControl.DatabaseCorsiControl

Si tratta di una classe di controllo che serve a gestire il database relativo ai corsi ed ai relativi professori. Questa classe viene chiamata esclusivamente dalla classe *MenuPrincipaleAmministratoreControl*, dato che questa funzionalità è accessibile solamente all'Amministratore della libreria.

4.1.2.1 Attributi della classe

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore “S”, allora la query da effettuare sul database è di tipo *select* , quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all’insieme dei record trovati nel database. Se targa assume valore “N”, la query da effettuare sul database non sarà del tipo *select* , ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle assegnato un valore, questa variabile viene passata come parametro al metodo *esegui* dell’oggetto istanziato della classe *ClassiComuni.Client*.

4.1.2.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un’istruzione switch. Inizialmente viene istanziata la classe *DatabaseCorsiBoundary*, ed in base al valore intero restituito dal metodo *ritornaCorrispondenza()* di questa classe, viene chiamata la classe di controllo *MenuPrincipaleAmministratoreControl* nel caso si voglia ritornare al menu principale, oppure viene inviata una opportuna query al

database corrispondente alla funzionalità richiesta nella finestra *DatabaseCorsiBoundary* dall'utente del sistema (nella fattispecie l'Amministratore della libreria).

4.1.3 AmministratoreGUI.DatabaseCorsiBoundary

Si tratta di una classe boundary che implementa l'interfaccia utente utilizzata per la gestione dei dati relativi ai corsi ed ai professori. Questa interfaccia utente è utilizzata dall'Amministratore della libreria. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.1.3.1 Attributi della classe

public static int corrispondenza

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *DatabaseCorsiControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Amministratore della libreria) nell'interfaccia utente.

String query

Questa variabile rappresenta la query da inoltrare al database. Essa sarà passata alla classe *DatabaseCorsiControl* attraverso il metodo *ritornaQuery()*.

4.1.3.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton4_actionPerformed(ActionEvent e)

void jButton6_actionPerformed(ActionEvent e)

void jButton7_actionPerformed(ActionEvent e)

void jButton10_actionPerformed(ActionEvent e)

void jButton11_actionPerformed(ActionEvent e)

void jButton12_actionPerformed(ActionEvent e)

void jButton13_actionPerformed(ActionEvent e)

void jButton14_actionPerformed(ActionEvent e)

void jButton111_actionPerformed(ActionEvent e)

void libriInUnCorso_actionPerformed(ActionEvent e)

void visualizzaDatiCorsi_actionPerformed(ActionEvent e)

void visualizzaDatiProfessori_actionPerformed(ActionEvent e)

void menu_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente. Essi essenzialmente

assegnano alla variabile *corrispondenza* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *DatabaseCorsiControl*.

void jRadioButton1_actionPerformed(ActionEvent e)

void jRadioButton2_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo componente grafico del tipo *JRadioButton* dell'interfaccia utente. Essi essenzialmente assegnano alla variabile *corrispondenza* un valore intero corrispondente al componente selezionato dall'utente, e successivamente passano il controllo alla classe *DatabaseCorsiControl*.

public int ritornaCorrispondenza()

Questo metodo restituisce il valore associato alla variabile *corrispondenza*.

public String ritornaQuery()

Questo metodo restituisce il valore associato alla variabile *query*.

4.1.4 AmministratoreControl. GestioneImpiegatiControl

Si tratta di una classe di controllo che serve a gestire il database relativo agli impiegati della libreria, permettendo, ad esempio, all'Amministratore della libreria di aggiungere, aggiornare, cancellare i dati degli impiegati (Nome, Cognome, Password, ecc.). Questa classe viene chiamata esclusivamente dalla classe *MenuPrincipaleAmministratoreControl*, dato che questa funzionalità è accessibile solamente all'Amministratore della libreria.

4.1.4.1 Attributi della classe

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore "S", allora la query da effettuare sul database è di tipo *select*, quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all'insieme dei record trovati nel database. Se targa assume valore "N", la query da effettuare sul database non sarà del tipo *select*, ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle

assegnato un valore, questa variabile viene passata come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

4.1.4.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *GestioneImpiegatiBoundary*, ed in base al valore intero restituito dal metodo *ritornaCorrispondenza()* di questa classe, viene chiamata la classe di controllo *MenuPrincipaleAmministratoreControl* nel caso si voglia ritornare al menu principale, oppure viene inviata una opportuna query al database corrispondente alla funzionalità richiesta nella finestra *GestioneImpiegatiBoundary* dall'utente del sistema (nella fattispecie l'Amministratore della libreria).

4.1.5 AmministratoreGUI.GestioneImpiegatiBoundary

Si tratta di una classe boundary che implementa l'interfaccia utente utilizzata per la gestione dei dati relativi agli impiegati della libreria. Questa interfaccia utente è utilizzata dall'Amministratore della libreria. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.1.5.1 Attributi della classe

public static int corrispondenza

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *GestioneImpiegatiControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Amministratore della libreria) nell'interfaccia utente.

String query

Questa variabile rappresenta la query da inoltrare al database. Essa sarà passata alla classe *GestioneImpiegatiControl* attraverso il metodo *ritornaQuery()*.

4.1.5.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void Button1_actionPerformed(ActionEvent e)

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton4_actionPerformed(ActionEvent e)

void jButton5_actionPerformed(ActionEvent e)

void jButton6_actionPerformed(ActionEvent e)

void jButton7_actionPerformed(ActionEvent e)

void jButton8_actionPerformed(ActionEvent e)

void jButton9_actionPerformed(ActionEvent e)

void jButton10_actionPerformed(ActionEvent e)

void jButton11_actionPerformed(ActionEvent e)

void jButton12_actionPerformed(ActionEvent e)

void jButton13_actionPerformed(ActionEvent e)

void jButton14_actionPerformed(ActionEvent e)

void jButton15_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente. Essi essenzialmente

assegnano alla variabile *corrispondenza* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *GestioneImpiegatiControl*.

public int ritornaCorrispondenza()

Questo metodo restituisce il valore associato alla variabile *corrispondenza*.

public String ritornaQuery()

Questo metodo restituisce il valore associato alla variabile *query*.

public void Messaggio(String msg)

Questo metodo visualizza un messaggio di errore (contenuto nel parametro *String msg*) facendo uso di un componente grafico del tipo *com.sun.java.swing.JOptionPane*.

void delete()

Questo metodo viene utilizzato per cancellare i dati digitati dall'utente nei componenti grafici del tipo *com.sun.java.swing.JTextField*.

4.1.6 AmministratoreControl. GestioneLibriControl

Si tratta di una classe di controllo che serve a gestire il database relativo ai libri della libreria, permettendo, ad esempio, all'Amministratore della libreria di aggiungere, aggiornare, cancellare i dati dei libri (Titolo, Codice ISBN, Tipo Edizione, Numero Edizione, ecc.), dei rispettivi autori e case editrici. Questa classe viene chiamata esclusivamente dalla classe *MenuPrincipaleAmministratoreControl*, dato che questa funzionalità è accessibile solamente all'Amministratore della libreria.

4.1.6.1 Attributi della classe

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore "S", allora la query da effettuare sul database è di tipo *select*, quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all'insieme dei record trovati nel database. Se targa assume valore "N", la query da effettuare sul database non sarà del tipo *select*, ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle

assegnato un valore, questa variabile viene passata come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

4.1.6.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *GestioneLibriBoundary*, ed in base al valore intero restituito dal metodo *ritornaCorrispondenza()* di questa classe, viene chiamata la classe di controllo *MenuPrincipaleAmministratoreControl* nel caso si voglia ritornare al menu principale, oppure viene inviata una opportuna query al database corrispondente alla funzionalità richiesta nella finestra *GestioneLibriBoundary* dall'utente del sistema (nella fattispecie l'Amministratore della libreria).

4.1.7 AmministratoreGUI.GestioneLibriBoundary

Si tratta di una classe boundary che implementa l'interfaccia utente utilizzata per la gestione dei dati relativi ai libri, agli autori ed alle case editrici. Questa interfaccia utente è utilizzata dall'Amministratore della libreria. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.1.7.1 Attributi della classe

public static int corrispondenza

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *GestioneLibriControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Amministratore della libreria) nell'interfaccia utente.

String query

Questa variabile rappresenta la query da inoltrare al database. Essa sarà passata alla classe *GestioneLibriControl* attraverso il metodo *ritornaQuery()*.

4.1.7.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void aggiorna_actionPerformed(ActionEvent e)

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton4_actionPerformed(ActionEvent e)

void jButton5_actionPerformed(ActionEvent e)

void jButton6_actionPerformed(ActionEvent e)

void jButton7_actionPerformed(ActionEvent e)

void jButton8_actionPerformed(ActionEvent e)

void jButton9_actionPerformed(ActionEvent e)

void jButton10_actionPerformed(ActionEvent e)

void jButton11_actionPerformed(ActionEvent e)

void jButton17_actionPerformed(ActionEvent e)

void jButton20_actionPerformed(ActionEvent e)

void menu_actionPerformed(ActionEvent e)

void visualizzaAutori_actionPerformed(ActionEvent e)

void visualizzaCaseEditrici_actionPerformed(ActionEvent e)

void visualizzaCaseEditrici1_actionPerformed(ActionEvent e)

void visualizzaCaseEditrici2_actionPerformed(ActionEvent e)

void visualizzaCaseEditrici4_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente. Essi essenzialmente assegnano alla variabile *corrispondenza* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *GestioneLibriControl*.

public int ritornaCorrispondenza()

Questo metodo restituisce il valore associato alla variabile *corrispondenza*.

public String ritornaQuery()

Questo metodo restituisce il valore associato alla variabile *query*.

void cancella_actionPerformed()

void delete_actionPerformed()

Questo metodo viene utilizzato per cancellare i dati digitati dall'utente nei componenti grafici del tipo *com.sun.java.swing.JTextField*.

4.1.8 AmministratoreControl. SupportoDecisioniControl

Si tratta di una classe di controllo che serve a gestire la funzionalità di supporto all'acquisto dei libri offerto dal sistema all'Amministratore. Questa classe viene chiamata esclusivamente dalla classe *MenuPrincipaleAmministratoreControl*, dato che questa funzionalità è accessibile solamente all'Amministratore della libreria.

4.1.8.1 Attributi della classe

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore "S", allora la query da effettuare sul database è di tipo *select*, quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all'insieme dei record trovati nel database. Se targa assume valore "N", la query da effettuare sul database non sarà del tipo *select*, ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle

assegnato un valore, questa variabile viene passata come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

4.1.8.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *SupportoDecisioniBoundary*, ed in base al valore intero restituito dal metodo *ritornaCorrispondenza()* di questa classe, viene chiamata la classe di controllo *MenuPrincipaleAmministratoreControl* nel caso si voglia ritornare al menu principale, oppure viene inviata una opportuna query al database corrispondente alla funzionalità richiesta nella finestra *SupportoDecisioniBoundary* dall'utente del sistema (nella fattispecie l'Amministratore della libreria).

4.1.9 AmministratoreGUI.SupportoDecisioniBoundary

Si tratta di una classe boundary che implementa l'interfaccia utente utilizzata per il supporto alle decisioni relative all'acquisto di un determinato libro da parte della libreria. Questa interfaccia utente è utilizzata dall'Amministratore della libreria. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.1.9.1 Attributi della classe

public static int corrispondenza

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *SupportoDecisioniControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Amministratore della libreria) nell'interfaccia utente.

String query

Questa variabile rappresenta la query da inoltrare al database. Essa sarà passata alla classe *SupportoDecisioniControl* attraverso il metodo *ritornaQuery()*.

4.1.9.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void annoAcc_1_actionPerformed(ActionEvent e)

void annoAcc_2_actionPerformed(ActionEvent e)

void annoAcc_3_actionPerformed(ActionEvent e)

void tornaAlMenuPrincipale_actionPerformed(ActionEvent e)

void visualizza_actionPerformed(ActionEvent e)

void visualizzaLibri1_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente. Essi essenzialmente assegnano alla variabile *corrispondenza* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *SupportoDecisioniControl*.

public int ritornaCorrispondenza()

Questo metodo restituisce il valore associato alla variabile *corrispondenza*.

public void Messaggio(String msg)

Questo metodo visualizza un messaggio di errore (contenuto nel parametro *String msg*) facendo uso di un componente grafico del tipo *com.sun.java.swing.JOptionPane*.

4.1.10 AmministratoreControl. GestioneOrdiniControl

Si tratta di una classe di controllo che serve a gestire gli ordini spiccati dalla libreria nei confronti del suo fornitore (Casa Editrice), ad esempio per gestire lo stato di un ordine (che può essere Aperto, Pendente o Chiuso) . Questa classe viene chiamata sia dalla classe *MenuPrincipaleAmministratoreControl*, sia dalla classe *MenuPrincipaleAddettoOrdiniControl*, dato che la funzionalità “Ordina Libri” è accessibile sia all’Amministratore della libreria, sia all’Addetto agli Ordini.

4.1.10.1 Attributi della classe

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore “S”, allora la query da effettuare sul database è di tipo *select* , quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all’insieme dei record trovati nel database. Se targa assume valore “N”, la query da effettuare sul database non sarà del tipo *select* , ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

int memoria

Questa variabile viene inizializzata dal costruttore della classe. Quando la classe *GestioneOrdiniControl* viene chiamata, le viene passato un parametro che le

serve a distinguere quale è la classe chiamante (*MenuPrincipaleAmministratoreControl* oppure *MenuPrincipaleAddettoOrdiniControl*).

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle assegnato un valore, questa variabile viene passata come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

4.1.10.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *GestioneOrdiniBoundary*, ed in base al valore intero restituito dal metodo *ritornaCorrispondenza()* di questa classe, viene chiamata la classe di controllo *MenuPrincipaleAmministratoreControl* (oppure *MenuPrincipaleAddettoOrdiniControl*) nel caso si voglia ritornare al menu principale, oppure viene inviata una opportuna query al database corrispondente alla funzionalità richiesta nella finestra *GestioneOrdiniBoundary* dall'utente del

sistema (nella fattispecie l'Amministratore della libreria oppure l'Addetto agli Ordini).

4.1.11 AmministratoreGUI.GestioneOrdiniBoundary

Si tratta di una classe boundary che implementa l'interfaccia utente utilizzata per la gestione degli ordini spiccati dalla libreria verso i fornitori (case editrici). Questa interfaccia utente è utilizzata dall'Amministratore della libreria e dall'addetto agli ordini. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.1.11.1 Attributi della classe

public static int corrispondenza

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *GestioneOrdiniControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Amministratore della libreria oppure l'Addetto agli Ordini) nell'interfaccia utente.

String query

Questa variabile rappresenta la query da inoltrare al database. Essa sarà passata alla classe *GestioneOrdiniControl* attraverso il metodo *ritornaQuery()*.

4.1.11.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void cercaPerDataTransazione_actionPerformed(ActionEvent e)

void cercaPerIDLibro_actionPerformed(ActionEvent e)

void cercaPerIDOrdine_actionPerformed(ActionEvent e)

void cercaPerIDTransazione_actionPerformed(ActionEvent e)

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton4_actionPerformed(ActionEvent e)

void jButton5_actionPerformed(ActionEvent e)

void jButton6_actionPerformed(ActionEvent e)

void jButton7_actionPerformed(ActionEvent e)

void jButton8_actionPerformed(ActionEvent e)

void jButton9_actionPerformed(ActionEvent e)

void jButton13_actionPerformed(ActionEvent e)

void jButton20_actionPerformed(ActionEvent e)

void jButton21_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente. Essi essenzialmente assegnano alla variabile *corrispondenza* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *SupportoDecisioniiControl*.

public int ritornaCorrispondenza()

Questo metodo restituisce il valore associato alla variabile *corrispondenza*.

public void Messaggio(String msg)

Questo metodo visualizza un messaggio di errore (contenuto nel parametro *String msg*) facendo uso di un componente grafico del tipo *com.sun.java.swing.JOptionPane*.

String adattaData(String data)

Questo metodo è utilizzato per modificare una data ricevuta come parametro in una forma diversa a causa di problemi di visualizzazione della data stessa nell'interfaccia.

void cancella_actionPerformed()

Questo metodo viene utilizzato per cancellare i dati digitati dall'utente nei componenti grafici del tipo *com.sun.java.swing.JTextField*.

4.1.12 AmministratoreControl. PreordineControl

Si tratta di una classe di controllo che serve a gestire i preordini fatti dall'Amministratore oppure dall'Addetto alle Vendite, quando un Cliente vuole ordinare un libro non presente in libreria. Questa classe serve anche a inserire nel database o a recuperare da esso dati relativi ai Clienti della libreria, come ad esempio l'acconto lasciato all'atto della prenotazione e la rimanenza. Questa classe viene chiamata sia dalla classe *MenuPrincipaleAmministratoreControl*, sia dalla classe *MenuPrincipaleAddettoVenditeControl*, dato che la funzionalità "Preordina Libri" è accessibile sia all'Amministratore della libreria, sia all'Addetto alle Vendite.

4.1.12.1 Attributi della classe

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore "S", allora la query da effettuare sul database è di tipo *select*, quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all'insieme dei record trovati nel database. Se targa assume valore "N", la query da effettuare sul database non sarà del tipo *select*, ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle assegnato un valore, questa variabile viene passata come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

int memoria

Questa variabile viene inizializzata dal costruttore della classe. Quando la classe *PreordineControl* viene chiamata, le viene passato un parametro che le serve a distinguere quale è la classe chiamante (*MenuPrincipaleAmministratoreControl* oppure *MenuPrincipaleAddettoVenditeControl*).

4.1.12.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *PreordineBoundary*, ed in base al valore intero restituito dal metodo *ritornaCorrispondenza()* di questa classe, viene chiamata la classe di controllo *MenuPrincipaleAmministratoreControl* (oppure *MenuPrincipaleAddettoVenditeControl*) nel caso si voglia ritornare al menu principale, oppure viene inviata una opportuna query al database corrispondente alla funzionalità richiesta nella finestra *PreordineBoundary* dall'utente del

sistema (nella fattispecie l'Addetto alle Vendite oppure l'Amministratore della libreria).

4.1.13 AmministratoreGUI. PreordineBoundary

Si tratta di una classe boundary che implementa l'interfaccia utente utilizzata per la gestione degli preordini relativi ai libri richiesti dai clienti e che non sono presenti momentaneamente in libreria. Questa interfaccia utente è utilizzata dall'Amministratore della libreria e dall'Addetto alle Vendite. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.1.13.1 Attributi della classe

public static int corrispondenza

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *PreordineControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Amministratore della libreria oppure l'Addetto agli Ordini) nell'interfaccia utente.

String query

Questa variabile rappresenta la query da inoltrare al database. Essa sarà passata alla classe *PreordineControl* attraverso il metodo *ritornaQuery()*.

4.1.13.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void cercaPerCittaCliente_actionPerformed(ActionEvent e)

void cercaPerCognomeCliente_actionPerformed(ActionEvent e)

void cercaPerEmailCliente_actionPerformed(ActionEvent e)

void cercaPerIDCliente_actionPerformed(ActionEvent e)

void cercaPerIndirizzoCliente_actionPerformed(ActionEvent e)

void cercaPerTelefonoCliente_actionPerformed(ActionEvent e)

void displayPreordini_actionPerformed(ActionEvent e)

void displayPreordiniPendenti_actionPerformed(ActionEvent e)

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton4_actionPerformed(ActionEvent e)

void jButton5_actionPerformed(ActionEvent e)

void jButton7_actionPerformed(ActionEvent e)

void jButton8_actionPerformed(ActionEvent e)

void jButton9_actionPerformed(ActionEvent e)

void jButton15_actionPerformed(ActionEvent e)

void jButton20_actionPerformed(ActionEvent e)

void jButton21_actionPerformed(ActionEvent e)

void jButton22_actionPerformed(ActionEvent e)

void visualizzaLibri_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente. Essi essenzialmente assegnano alla variabile *corrispondenza* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *PreordineControl*.

public int ritornaCorrispondenza()

Questo metodo restituisce il valore associato alla variabile *corrispondenza*.

public int ritornaQuery()

Questo metodo restituisce il valore associato alla variabile *query*.

public void Messaggio(String msg)

Questo metodo visualizza un messaggio di errore (contenuto nel parametro *String msg*) facendo uso di un componente grafico del tipo *com.sun.java.swing.JOptionPane*.

String adattaData(String data)

Questo metodo è utilizzato per modificare una data ricevuta come parametro in una forma diversa a causa di problemi di visualizzazione della data stessa nell'interfaccia.

void cancella_actionPerformed()

Questo metodo viene utilizzato per cancellare i dati digitati dall'utente nei componenti grafici del tipo *com.sun.java.swing.JTextField*.

4.1.14 AmministratoreControl.VendiLibroControl

Si tratta di una classe di controllo che serve a gestire la vendita dei libri ai Clienti della libreria. Questa classe viene chiamata sia dalla classe *MenuPrincipaleAmministratoreControl*, sia dalla classe *MenuPrincipaleAddettoVenditeControl*, dato che la funzionalità “Vendi Libri” è accessibile sia all’Addetto alle Vendite, sia all’Amministratore della libreria.

4.1.14.1 Attributi della classe

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore “S”, allora la query da effettuare sul database è di tipo *select*, quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all’insieme dei record trovati nel database. Se targa assume valore “N”, la query da effettuare sul database non sarà del tipo *select*, ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

int memoria

Questa variabile viene inizializzata dal costruttore della classe. Quando la classe *VendiLibroControl* viene chiamata, le viene passato un parametro che le serve a

distinguere quale è la classe chiamante (*MenuPrincipaleAddettoVenditeControl* oppure *MenuPrincipaleAmministratoreControl*).

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle assegnato un valore, questa variabile viene passata come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

4.1.14.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *VendiLibroBoundary*, ed in base al valore intero restituito dal metodo *ritornaCorrispondenza()* di questa classe, viene chiamata la classe di controllo *MenuPrincipaleAddettoVenditeControl* (oppure *MenuPrincipaleAmministratoreControl*) nel caso si voglia ritornare al menu principale, oppure viene inviata una opportuna query al database corrispondente alla funzionalità richiesta nella finestra *VendiLibroBoundary* dall'utente del sistema (nella fattispecie l'Amministratore della libreria oppure l'Addetto alle Vendite).

4.1.15 AmministratoreGUI.VendiLibroBoundary

Si tratta di una classe boundary che implementa l'interfaccia utente utilizzata per la gestione delle vendite dei libri al cliente. Questa interfaccia utente è utilizzata dall'Amministratore della libreria e dall'Addetto alle Vendite. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.1.15.1 Attributi della classe

public static int corrispondenza

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *VendiLibroControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Amministratore della libreria oppure l'Addetto agli Ordini) nell'interfaccia utente.

String query

Questa variabile rappresenta la query da inoltrare al database. Essa sarà passata alla classe *VendiLibroControl* attraverso il metodo *ritornaQuery()*.

4.1.15.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void mostraLibri_actionPerformed(ActionEvent e)

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton4_actionPerformed(ActionEvent e)

void jButton5_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente. Essi essenzialmente assegnano alla variabile *corrispondenza* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *VendiLibroControl*.

public int ritornaCorrispondenza()

Questo metodo restituisce il valore associato alla variabile *corrispondenza*.

public int ritornaQuery()

Questo metodo restituisce il valore associato alla variabile *query*.

4.1.16 AmministratoreGUI.FinestraSceltaAmministratore

Si tratta di una classe boundary che implementa l'interfaccia utente rappresentante il menu principale a disposizione dell'Amministratore della libreria. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.1.16.1 Attributi della classe

public static int op1

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *MenuPrincipaleAmministratoreControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Amministratore della libreria) nel menu principale.

4.1.16.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

```
void jButton3_actionPerformed(ActionEvent e)  
void jButton4_actionPerformed(ActionEvent e)  
void jButton5_actionPerformed(ActionEvent e)  
void jButton6_actionPerformed(ActionEvent e)  
void jButton7_actionPerformed(ActionEvent e)  
void jButton8_actionPerformed(ActionEvent e)  
void jButton9_actionPerformed(ActionEvent e)  
void jButton10_actionPerformed(ActionEvent e)
```

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente (il menu principale). Essi essenzialmente assegnano alla variabile *op1* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *MenuPrincipaleAmministratoreControl*.

```
public int ritornaValoreOp()
```

Questo metodo restituisce il valore associato alla variabile *op1*.

4.1.17 AmministratoreGUI.StoriaAcquistiVendite

Si tratta di una classe che implementa l'interfaccia utente utilizzata per la visualizzazione delle date relative alle vendite ed agli acquisti dei libri da parte della libreria. Questa interfaccia utente è utilizzata dall'Amministratore della libreria e dall'Addetto al Magazzino. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.1.17.1 Attributi della classe

Assenti.

4.1.17.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void tornaAlMenuPrincipale_2_actionPerformed(ActionEvent e)

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente.

public void Messaggio(String msg)

Questo metodo visualizza un messaggio di errore (contenuto nel parametro *String msg*) facendo uso di un componente grafico del tipo *com.sun.java.swing.JOptionPane*.

void displayAcquisti(String[][] row)

Questo metodo serve a visualizzare i dati trovati nel database relativi alle quantità ed agli acquisti di un determinato libro. Questi dati sono passati al metodo attraverso il parametro *String[][] row*. Questo metodo è invocato all'interno del metodo *void jButton2_actionPerformed(ActionEvent e)*.

void displayVendite(String[][] row)

Questo metodo serve a visualizzare i dati trovati nel database relativi alle quantità ed alle vendite di un determinato libro. Questi dati sono passati al metodo attraverso il parametro *String[][] row*. Questo metodo è invocato all'interno del metodo *void jButton1_actionPerformed(ActionEvent e)*.

String adattaData(String data)

Questo metodo è utilizzato per modificare una data ricevuta come parametro in una forma diversa a causa di problemi di visualizzazione della data stessa nell'interfaccia.

4.2 Package FinestraAddettoMagazzino

4.2.1 AddettoMagazzinoControl.

MenuPrincipaleAddettoMagazzinoControl

E' la classe principale del package **FinestraAddettoMagazzino**. Si tratta di un classe di controllo che serve a chiamare ed a gestire altre classi di controllo facenti parte del package *Amministratore*. Questa classe all'avvio (nel metodo *main*) istanzia un oggetto di tipo *ClassiComuni.Password* che serve ad autenticare l'utente e ad abilitarlo, in caso di riscontro positivo, all'uso del sistema. In questo caso viene istanziato un oggetto di tipo, appunto, *MenuPrincipaleAddettoMagazzinoControl*. Invece, nel caso di riscontro negativo, cioè nel caso in cui l'utente non viene riconosciuto dal sistema, viene istanziato un oggetto di tipo *ClassiComuni.Message*, la cui funzione è quella di comunicare all'utente che i dati inseriti sono errati e di invitarlo a riprovare. Questa classe estende l'interfaccia *ClassiComuni.MenuPrincipale*.

4.2.1.1 Attributi della classe

public static int op1

A questa variabile viene assegnato il valore ritornato dal metodo *ritornaValoreOp()* della classe *FinestraSceltaAddettoMagazzino*. Questo valore corrisponde alla scelta fatta dall'utente del sistema nell'interfaccia relativa al menu principale che gli si presenta dopo che il processo di autenticazione è andato a buon fine. A seconda del valore della variabile *op1*, viene istanziata un'opportuna classe di controllo, la cui funzione è quella di svolgere il compito richiesto dall'utente.

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore "S", allora la query da effettuare sul database è di tipo *select*, quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all'insieme dei record trovati nel database. Se targa assume valore "N", la query da effettuare sul database non sarà del tipo *select*, ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle assegnato un valore, questa variabile viene passata al metodo

InterrogaDatabase(String), che a sua volta la passa come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

4.2.1.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *FinestraSceltaAddettoMagazzino*, ed in base al valore intero restituito dal metodo *ritornaValoreOp()* di questa classe, viene chiamata la classe di controllo corrispondente alla funzionalità richiesta nel menu principale dall'utente del sistema (nella fattispecie l'Addetto al Magazzino).

public void scegliRicerca()

Anche questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *ClassiComuni.RicercaBoundary*, ed in base al valore intero restituito dal metodo *ritornaValoreOp()* di questa classe, viene costruita una stringa rappresentante la query da inviare al database. Questa stringa viene poi passata al metodo *InterrogaDatabase(String)* che si occuperà dell'avvio del client.

public void InterrogaDatabase(String query)

Questo metodo viene chiamato all'interno del metodo *scegliRicerca()*. Inizialmente viene istanziata la classe *ClassiComuni.Client*. Il metodo *esegui* di quest'ultima classe ritornerà alla classe chiamante (cioè *MenuPrincipaleAddettoMagazzinoControl*) una matrice corrispondente all'insieme dei record trovati nel database. Viene quindi istanziata la classe *ClassiComuni.MostraLibro* che si occuperà di mostrare in una tabella i dati ricevuti dal server. Nel caso in cui si verificano dei problemi relativi alla connessione tra client e server, viene visualizzato un messaggio di errore utilizzando la classe *ClassiComuni.Message*.

public static void main (String[] arg)

Nel metodo main viene istanziata la classe *ClassiComuni.Password* che serve ad autenticare l'utente e ad autorizzarlo all'uso del sistema.

Se l'utente inserisce i dati esatti, viene creato un oggetto *MenuPrincipaleAddettoMagazzinoControl* che avvia il menu principale utilizzabile dall'utente, nella fattispecie l'Addetto al Magazzino.

4.2.2 AddettoMagazzinoGUI.

FinestraSceltaAddettoMagazzino

Si tratta di una classe boundary che implementa l'interfaccia utente rappresentante il menu principale a disposizione dell'Addetto alla gestione del Magazzino della libreria. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.2.2.1 Attributi della classe

public static int op1

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *MenuPrincipaleAmministratoreControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Addetto al Magazzino della libreria) nel menu principale.

4.2.2.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton4_actionPerformed(ActionEvent e)

void jButton5_actionPerformed(ActionEvent e)

void jButton6_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente (il menu principale). Essi essenzialmente assegnano alla variabile *op1* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *MenuPrincipaleAddettoMagazzinoControl*.

public int ritornaValoreOp()

Questo metodo restituisce il valore associato alla variabile *op1*.

4.3 Package FinestraAddettoOrdini

4.3.1 AddettoOrdiniControl.

MenuPrincipaleAddettoOrdiniControl

E' la classe principale del package **FinestraAddettoOrdini**. Si tratta di un classe di controllo che serve a chiamare ed a gestire altre classi di controllo facenti parte del package *Amministratore*. Questa classe all'avvio (nel metodo *main*) istanzia un oggetto di tipo *ClassiComuni.Password* che serve ad autenticare l'utente e ad abilitarlo, in caso di riscontro positivo, all'uso del sistema. In questo caso viene istanziato un oggetto di tipo, appunto, *MenuPrincipaleAddettoOrdiniControl*. Invece, nel caso di riscontro negativo, cioè nel caso in cui l'utente non viene riconosciuto dal sistema, viene istanziato un oggetto di tipo *ClassiComuni.Message*, la cui funzione è quella di comunicare all'utente che i dati inseriti sono errati e di invitarlo a riprovare. Questa classe estende l'interfaccia *ClassiComuni.MenuPrincipale*.

4.3.1.1 Attributi della classe

public static int op1

A questa variabile viene assegnato il valore ritornato dal metodo *ritornaValoreOp()* della classe *FinestraSceltaAddettoOrdini*. Questo valore corrisponde alla scelta fatta dall'utente del sistema nell'interfaccia relativa al

menu principale che gli si presenta dopo che il processo di autenticazione è andato a buon fine. A seconda del valore della variabile *op1*, viene istanziata un'opportuna classe di controllo, la cui funzione è quella di svolgere il compito richiesto dall'utente (in questo caso l'Addetto agli Ordini).

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore "S", allora la query da effettuare sul database è di tipo *select*, quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all'insieme dei record trovati nel database. Se targa assume valore "N", la query da effettuare sul database non sarà del tipo *select*, ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle assegnato un valore, questa variabile viene passata al metodo *InterrogaDatabase(String)*, che a sua volta la passa come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

4.3.1.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *FinestraSceltaAddettoOrdini*, ed in base al valore intero restituito dal metodo *ritornaValoreOp()* di questa classe, viene chiamata la classe di controllo corrispondente alla funzionalità richiesta nel menu principale dall'utente del sistema.

public void scegliRicerca()

Anche questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *ClassiComuni.RicercaBoundary*, ed in base al valore intero restituito dal metodo *ritornaValoreOp()* di questa classe, viene costruita una stringa rappresentante la query da inviare al database. Questa stringa viene poi passata al metodo *InterrogaDatabase(String)* che si occuperà dell'avvio del client.

public void InterrogaDatabase(String query)

Questo metodo viene chiamato all'interno del metodo *scegliRicerca()*. Inizialmente viene istanziata la classe *ClassiComuni.Client*. Il metodo *esegui* di quest'ultima classe ritornerà alla classe chiamante (cioè *MenuPrincipaleAddettoOrdiniControl*) una matrice corrispondente all'insieme dei record trovati nel database. Viene quindi istanziata la classe

ClassiComuni.MostraLibro che si occuperà di mostrare in una tabella i dati ricevuti dal server. Nel caso in cui si verificano dei problemi relativi alla connessione tra client e server, viene visualizzato un messaggio di errore utilizzando la classe *ClassiComuni.Message*.

public static void main (String[] arg)

Nel metodo main viene istanziata la classe *ClassiComuni.Password* che serve ad autenticare l'utente e ad autorizzarlo all'uso del sistema.

Se l'utente inserisce i dati esatti, viene creato un oggetto *MenuPrincipaleAddettoOrdiniControl* che avvia il menu principale utilizzabile dall'utente, nella fattispecie l'Addetto agli Ordini della libreria.

4.3.2 AddettoOrdiniGUI.

FinestraSceltaAddettoOrdini

Si tratta di una classe boundary che implementa l'interfaccia utente rappresentante il menu principale a disposizione dell'Addetto agli Ordini. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.3.2.1 Attributi della classe

public static int op1

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *MenuPrincipaleAddettoOrdiniControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Addetto agli Ordini) nel menu principale.

4.3.2.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton7_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente (il menu principale). Essi essenzialmente assegnano alla variabile *op1* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *MenuPrincipaleAddettoOrdiniControl*.

public int ritornaValoreOp()

Questo metodo restituisce il valore associato alla variabile *op1*.

4.4 Package FinestraAddettoVendite

4.4.1 AddettoVenditeControl.

MenuPrincipaleAddettoVenditeControl

E' la classe principale del package **FinestraAddettoVendite**. Si tratta di un classe di controllo che serve a chiamare ed a gestire altre classi di controllo facenti parte del package *Amministratore*. Questa classe all'avvio (nel metodo *main*) istanzia un oggetto di tipo *ClassiComuni.Password* che serve ad autenticare l'utente e ad abilitarlo, in caso di riscontro positivo, all'uso del sistema. In questo caso viene istanziato un oggetto di tipo, appunto, *MenuPrincipaleAddettoVenditeControl*. Invece, nel caso di riscontro negativo, cioè nel caso in cui l'utente non viene riconosciuto dal sistema, viene istanziato un oggetto di tipo *ClassiComuni.Message*, la cui funzione è quella di comunicare all'utente che i dati inseriti sono errati e di invitarlo a riprovare. Questa classe estende l'interfaccia *ClassiComuni.MenuPrincipale*.

4.4.1.1 Attributi della classe

public static int op1

A questa variabile viene assegnato il valore ritornato dal metodo *ritornaValoreOp()* della classe *FinestraSceltaAddettoVendite*. Questo valore corrisponde alla scelta fatta dall'utente del sistema nell'interfaccia relativa al menu principale che gli si presenta dopo che il processo di autenticazione è

andato a buon fine. A seconda del valore della variabile *opl*, viene istanziata un'opportuna classe di controllo, la cui funzione è quella di svolgere il compito richiesto dall'utente.

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se targa assume il valore "S", allora la query da effettuare sul database è di tipo *select*, quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all'insieme dei record trovati nel database. Se targa assume valore "N", la query da effettuare sul database non sarà del tipo *select*, ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle assegnato un valore, questa variabile viene passata al metodo *InterrogaDatabase(String)*, che a sua volta la passa come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

4.4.1.2 Metodi della classe

public void esegui() throws Exception

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *FinestraSceltaAddettoVendite*, ed in base al valore intero restituito dal metodo *ritornaValoreOp()* di questa classe, viene chiamata la classe di controllo corrispondente alla funzionalità richiesta nel menu principale dall'utente del sistema.

public void scegliRicerca()

Anche questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *ClassiComuni.RicercaBoundary*, ed in base al valore intero restituito dal metodo *ritornaValoreOp()* di questa classe, viene costruita una stringa rappresentante la query da inviare al database. Questa stringa viene poi passata al metodo *InterrogaDatabase(String)* che si occuperà dell'avvio del client.

public void InterrogaDatabase(String query)

Questo metodo viene chiamato all'interno del metodo *scegliRicerca()*. Inizialmente viene istanziata la classe *ClassiComuni.Client*. Il metodo *esegui* di quest'ultima classe ritornerà alla classe chiamante (cioè *MenuPrincipaleAddettoVenditeControl*) una matrice corrispondente all'insieme dei record trovati nel database. Viene quindi istanziata la classe

ClassiComuni.MostraLibro che si occuperà di mostrare in una tabella i dati ricevuti dal server. Nel caso in cui si verificano dei problemi relativi alla connessione tra client e server, viene visualizzato un messaggio di errore utilizzando la classe *ClassiComuni.Message*.

public static void main (String[] arg)

Nel metodo main viene istanziata la classe *ClassiComuni.Password* che serve ad autenticare l'utente e ad autorizzarlo all'uso del sistema.

Se l'utente inserisce i dati esatti, viene creato un oggetto *MenuPrincipaleAddettoVenditeControl* che avvia il menu principale utilizzabile dall'utente, nella fattispecie l'Addetto alle Vendite della libreria.

4.4.2 AddettoVenditeGUI.

FinestraSceltaAddettoVendite

Si tratta di una classe boundary che implementa l'interfaccia utente rappresentante il menu principale a disposizione dell'Addetto alle Vendite. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.4.2.1 Attributi della classe

public static int op1

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un

valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *MenuPrincipaleAddettoVenditeControl* quale bottone è stato pigiato dall'utente (nella fattispecie l'Addetto alle Vendite) nel menu principale.

4.4.2.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton5_actionPerformed(ActionEvent e)

void jButton6_actionPerformed(ActionEvent e)

Tutti questi metodi servono ad implementare la funzione che deve svolgere ogni singolo bottone (*JButton*) dell'interfaccia utente (il menu principale). Essi essenzialmente assegnano alla variabile *op1* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *MenuPrincipaleAddettoVenditeControl*.

public int ritornaValoreOp()

Questo metodo restituisce il valore associato alla variabile *op1*.

4.5 Package Cliente

4.5.1 ClienteControl.MenuPrincipaleClienteControl

E' la classe principale del package **Cliente**. Si tratta di un classe di controllo che serve a gestire le interfacce utente che si troveranno sul terminale dedicato al cliente della libreria. Questa classe estende l'interfaccia *ClassiComuni.MenuPrincipale*.

4.5.1.1 Attributi della classe

public static int op1

A questa variabile viene assegnato il valore ritornato dal metodo *ritornaValoreOp()* della classe *RicercaClienteBoundary*. Questo valore corrisponde alla scelta fatta dall'utente del sistema nell'interfaccia relativa al tipo di ricerca da effettuare, che gli si presenta subito dopo il menu principale. A seconda del valore della variabile *op1* viene creata un'opportuna query, con la quale successivamente verrà interrogato il database.

String targa

Questa variabile viene passata come parametro alla classe *ClassiComuni.Client* (Quando essa viene istanziata). Essa serve per comunicare al server il tipo di query da effettuare sul database. Se *targa* assume il valore "S", allora la query da effettuare sul database è di tipo *select*, quindi si presuppone che il server debba

ritornare dei dati al client corrispondenti all'insieme dei record trovati nel database. Dato che il cliente della libreria può solamente consultare il database, la query effettuata sarà di tipo *select*, e quindi targa assumerà valore "S".

private String richiesta

Questa variabile viene utilizzata per immagazzinare temporaneamente la query da inviare al server e quindi, successivamente, al database. Dopo averle assegnato un valore, questa variabile viene passata al metodo *InterrogaDatabase(String)*, che a sua volta la passa come parametro al metodo *esegui* dell'oggetto istanziato della classe *ClassiComuni.Client*.

4.5.1.2 Metodi della classe

public void esegui() throws Exception

Questo metodo istanzia la classe *Cliente.ClienteGUI.FinestraSceltaCliente*, e successivamente chiama il metodo *scegliRicerca()*.

public void scegliRicerca()

Questo metodo è costituito essenzialmente da un'istruzione switch. Inizialmente viene istanziata la classe *RicercaClienteBoundary*, ed in base al valore intero restituito dal metodo *ritornaValoreOp()* di questa classe, viene costruita una stringa rappresentante la query da inviare al database. Questa stringa viene poi

passata al metodo *InterrogaDatabase(String)* che si occuperà dell'avvio del client.

public void InterrogaDatabase(String query)

Questo metodo viene chiamato all'interno del metodo *scegliRicerca()*. Inizialmente viene istanziata la classe *ClassiComuni.Client*. Il metodo *esegui* di quest'ultima classe ritornerà alla classe chiamante (cioè *MenuPrincipaleClienteControl*) una matrice corrispondente all'insieme dei record trovati nel database. Viene quindi istanziata la classe *ClassiComuni.MostraLibro* che si occuperà di mostrare in una tabella i dati ricevuti dal server. Nel caso in cui si verificano dei problemi relativi alla connessione tra client e server, viene visualizzato un messaggio di errore utilizzando la classe *ClassiComuni.Message*.

public static void main (String[] arg)

Nel metodo main viene creato un oggetto di tipo *MenuPrincipaleClienteControl* che avvia il menu principale utilizzabile dall'utente, nella fattispecie il Cliente della libreria.

4.5.2 ClienteGUI.FinestraSceltaCliente

Si tratta di una classe boundary che implementa l'interfaccia utente rappresentante il menu principale a disposizione del Cliente della libreria. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.5.2.1 Attributi della classe

public static int op1

Questa variabile viene utilizzata dal metodo della classe *void jButton1_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *MenuPrincipaleClienteControl* quale bottone è stato pigiato dall'utente (nella fattispecie il Cliente della libreria) nel menu principale.

4.5.2.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

Questo metodo serve ad implementare la funzione che deve svolgere l'unico bottone (*JButton*) dell'interfaccia utente (il menu principale). Esso

essenzialmente assegna alla variabile *op1* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passa il controllo alla classe *MenuPrincipaleClienteControl*.

4.5.3 ClienteGUI.RicercaClienteBoundary

Si tratta di una classe boundary che implementa l'interfaccia utente che mette a disposizione del Cliente della libreria varie opzioni di ricerca dei libri (per titolo, per Casa Editrice, per Corso, per Professore, ecc.). Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.5.3.1 Attributi della classe

public static int op1

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alla classe *MenuPrincipaleClienteControl* quale bottone è stato pigiato dall'utente (nella fattispecie il Cliente della libreria) nella finestra dedicata alla scelta delle opzioni di ricerca.

4.5.3.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton4_actionPerformed(ActionEvent e)

void jButton5_actionPerformed(ActionEvent e)

void jButton6_actionPerformed(ActionEvent e)

void jButton7_actionPerformed(ActionEvent e)

void jButton8_actionPerformed(ActionEvent e)

void jButton9_actionPerformed(ActionEvent e)

void jButton10_actionPerformed(ActionEvent e)

void jButton11_actionPerformed(ActionEvent e)

void jButton12_actionPerformed(ActionEvent e)

void jButton13_actionPerformed(ActionEvent e)

void jButton14_actionPerformed(ActionEvent e)

void jButton15_actionPerformed(ActionEvent e)

Questi metodi servono ad implementare la funzione che deve essere svolta da ciascun bottone (*JButton*) dell'interfaccia utente (la finestra di ricerca). Questi metodi essenzialmente assegnano alla variabile *op1* un valore intero corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alla classe *MenuPrincipaleClienteControl*.

public int ritornaValoreOp()

Questo metodo restituisce il valore associato alla variabile *op1*.

4.6 Package ClassiComuni

4.6.1 ClassiComuni.RicercaBoundary

Si tratta di una classe boundary che implementa l'interfaccia utente che mette a disposizione di tutto il personale della libreria varie opzioni di ricerca dei libri (per titolo, per Casa Editrice, per Codice ISBN, per Professore, ecc.). Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.6.1.1 Attributi della classe

public static int op1

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alle varie classi del tipo **.MenuPrincipale*Control* (ad eccezione della classe *Cliente.ClienteControl.MenuPrincipaleClienteControl*) quale bottone è stato pigiato dall'utente (nella fattispecie l'Amministratore, l'Addetto agli Ordini, l'Addetto alle Vendite oppure l'Addetto al Magazzino) nella finestra dedicata alla scelta delle opzioni di ricerca.

4.6.1.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

void jButton4_actionPerformed(ActionEvent e)

void jButton5_actionPerformed(ActionEvent e)

void jButton6_actionPerformed(ActionEvent e)

void jButton7_actionPerformed(ActionEvent e)

void jButton8_actionPerformed(ActionEvent e)

void jButton9_actionPerformed(ActionEvent e)

void jButton10_actionPerformed(ActionEvent e)

void jButton11_actionPerformed(ActionEvent e)

void jButton12_actionPerformed(ActionEvent e)

void jButton13_actionPerformed(ActionEvent e)

void jButton14_actionPerformed(ActionEvent e)

void jButton15_actionPerformed(ActionEvent e)

Questi metodi servono ad implementare la funzione che deve essere svolta da ciascun bottone (*JButton*) dell'interfaccia utente (la finestra di ricerca). Questi metodi essenzialmente assegnano alla variabile *op1* un valore intero

corrispondente al bottone pigiato dall'utente, e successivamente passano il controllo alle classi del tipo **.MenuPrincipale*Control* (ad eccezione della classe *Cliente.ClienteControl.MenuPrincipaleClienteControl*).

public int ritornaValoreOp()

Questo metodo restituisce il valore associato alla variabile *op1*.

4.6.2 Classi Comuni.Message

Si tratta di una classe di tipo boundary che serve a visualizzare un messaggio di errore. Il particolare messaggio di errore da visualizzare nell'area di testo (*JTextArea*), presente nella finestra, è costituito da un oggetto di tipo *String* passato come parametro al costruttore di questa classe al momento della sua chiamata.

4.6.2.1 Attributi della classe

String informazione

Questa variabile viene inizializzata dal costruttore della classe. Quando la classe *Message* viene chiamata, le viene passato un oggetto di tipo *String* (il messaggio di errore da visualizzare) che essa assegnerà alla variabile *informazione*.

4.6.2.2 Metodi della classe

public final void buildWindow()

Questo metodo è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, *JTextArea*, ecc.).

public final void doDone()

Questo metodo serve ad implementare la funzione che deve essere svolta dall'unico bottone (*JButton*) presente in questa finestra. Questo metodo essenzialmente chiude la finestra che visualizza il messaggio.

4.6.3 Classi Comuni.Conferma

Si tratta di una classe di tipo boundary che serve a visualizzare un messaggio di conferma relativa all'operazione precedentemente compiuta. Il particolare messaggio di conferma da visualizzare nell'area di testo (*JTextArea*), presente nella finestra, è costituito da un oggetto di tipo String passato come parametro al costruttore di questa classe al momento della sua chiamata.

4.6.3.1 Attributi della classe

String informazione

Questa variabile viene inizializzata dal costruttore della classe. Quando la classe *Conferma* viene chiamata, le viene passato un oggetto di tipo `String` (il messaggio di conferma da visualizzare) che essa assegnerà alla variabile *informazione*.

4.6.3.2 Metodi della classe

public final void buildWindow()

Questo metodo è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JTextArea*, ecc.).

public final void doDone()

Questo metodo serve ad implementare la funzione che deve essere svolta dall'unico bottone (*JButton*) presente in questa finestra. Questo metodo essenzialmente chiude la finestra che visualizza il messaggio.

9.6.4 ClassiComuni.Saluta

Si tratta di una classe di tipo boundary che serve a visualizzare un messaggio di saluto all'uscita dal sistema. Il messaggio di saluto viene visualizzato utilizzando delle *Jlabel*. Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.6.4.1 Attributi della classe

Assenti.

4.6.4.2 Metodi della classe


private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

Questo metodo serve ad implementare la funzione che deve svolgere l'unico bottone presente (*JButton*) in questa interfaccia utente. Esso serve a chiudere la finestra e ad uscire definitivamente dal sistema.

protected void processWindowEvent(WindowEvent e)

Questo metodo svolge la stessa funzione del metodo precedente, consente infatti di uscire dal sistema pigiando il bottone  della finestra.

4.6.5 ClassiComuni.CercaPrezzo

Si tratta di una classe di tipo boundary che serve a visualizzare una finestra in cui l'utente può scegliere il tipo di ricerca per prezzo che vuol fare sul database della libreria (ad esempio l'utente vuol cercare tutti i libri il cui prezzo è inferiore, uguale oppure superiore ad un determinato prezzo). Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.6.5.1 Attributi della classe

public String[] scelta

Si tratta di un vettore di oggetti di tipo String. Il primo elemento di questo vettore conterrà una stringa corrispondente al bottone pigiato dall'utente (“<=”, “=” oppure “>=”). Il secondo elemento di questo vettore conterrà, invece, una stringa corrispondente al prezzo digitato dall'utente nel campo di testo (*TextField*) presente nella finestra.

int oper

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore diverso a seconda del metodo in cui viene utilizzata, e serve per comunicare alle classi del tipo **.MenuPrincipale*Control* quale bottone è stato pigiato dall'utente nella finestra.

4.6.5.2 Metodi della classe

public String[] ritornaScelta()

Questo metodo restituisce il valore associato alla variabile *scelta*.

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JTextField*, *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

void jButton2_actionPerformed(ActionEvent e)

void jButton3_actionPerformed(ActionEvent e)

Questi metodi servono ad implementare la funzione che deve essere svolta da ciascun bottone (*JButton*) dell'interfaccia che offre la funzionalità “Cerca per Prezzo”. Questi metodi essenzialmente assegnano alla variabile *oper* un vettore di oggetti di tipo *String* corrispondente alle scelte fatte dall'utente (bottone pigiato + prezzo digitato), e successivamente passano il controllo alle classi del tipo **.MenuPrincipale*Control*.

4.6.6 ClassiComuni.Password

Si tratta di una classe che serve a gestire l'autenticazione, tramite password e ID Impiegato, dell'utente all'avvio dell'applicativo. Nella finestra che si presenta l'utente è invitato ad inserire i dati che gli sono stati assegnati precedentemente dall'Amministratore della libreria. Successivamente viene inviata una richiesta al server, e successivamente al database per la verifica dei dati inseriti. Nel caso in cui i dati siano corretti viene visualizzato un messaggio di conferma tramite la classe *ClassiComuni.Conferma*. Invece, nel caso in cui i dati siano errati viene visualizzato un messaggio di errore tramite la classe *ClassiComuni.Message*, e l'utente viene invitato a reinserire i dati. La classe *Password* estende la classe *com.sun.java.swing.JFrame*. Questa classe viene istanziata da tutte le classi del tipo **.MenuPrincipale*Control* ad eccezione della classe *Cliente.ClienteControl.MenuPrincipaleClienteControl*.

4.6.6.1 Attributi della classe

int oper

Questa variabile viene utilizzata dai metodi della classe del tipo *void jButton_actionPerformed(ActionEvent e)*. A questa variabile viene assegnato un valore pari a -5 se i dati inseriti dall'utente (ID Impiegato e password) sono corretti, un valore pari a 0 (valore di default) altrimenti.

4.6.6.2 Metodi della classe

public int ritornaValoreOper()

Questo metodo restituisce il valore associato alla variabile *oper*.

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JTextField*, *JButton*, *JLabel*, ecc.).


void jButton1_actionPerformed(ActionEvent e)

Questo metodo serve ad implementare la funzione che deve essere svolta dal bottone (*JButton jButton1*) dell'interfaccia. Esso essenzialmente invoca il metodo *verificaPassword()* della classe *Password*.

void jButton2_actionPerformed(ActionEvent e)

Questo metodo serve ad implementare la funzione che deve essere svolta dal bottone (*JButton jButton2*) dell'interfaccia. Esso viene utilizzato per cancellare i dati digitati dall'utente nei due campi di testo *JTextField* e *JPasswordField*.

protected void processWindowEvent(WindowEvent e)

Questo metodo serve a chiudere la finestra e consente di uscire definitivamente dal sistema pigiando il bottone  della finestra.

private void verificaPassword()

In questo metodo viene creata ed inviata una query al server istanziando un oggetto del tipo *ClassiComuni.Client()*, per verificare l'autenticità dei dati inseriti dall'utente. Nel caso in cui i dati siano corretti viene visualizzato un messaggio di benvenuto tramite un oggetto di tipo *com.sun.java.swing.JOptionPane*. Se, invece, i dati non sono corretti, oppure non è possibile instaurare una connessione con il server, viene visualizzato un messaggio di errore tramite un oggetto di tipo *ClassiComuni.Message*.

4.6.7 ClassiComuni.MostraLibro

Si tratta di una classe di tipo boundary, che serve a visualizzare una Tabella (di tipo *com.sun.java.swing.JTable*) in cui vengono mostrati i risultati della ricerca dei libri fatta da uno degli utenti del sistema. In un componente del tipo *TextArea* viene mostrato, inoltre, il risultato della ricerca (cioè il numero di libri trovati nel database della libreria). Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.6.7.1 Attributi della classe

Object[][] data

Questa matrice sarà utilizzata per contenere l'insieme dei dati trovati nel database.

String[] columnNames

Si tratta di un vettore di oggetti di tipo String. Esso conterrà il nome dei campi delle tabelle del database in cui è stata fatta la query di ricerca.

int numeroRighe

A questa variabile intera sarà assegnato un valore pari al numero di righe della tabella da visualizzare.

int numeroColonne

A questa variabile intera sarà assegnato un valore pari al numero di colonne della tabella da visualizzare.

4.6.7.2 Metodi della classe

private void InterrogaDatabase(String richiesta)

Questo metodo viene invocato dal metodo *private void InterrogaDatabase(String richiesta)*, ed in esso viene fatta una interrogazione al database (dopo aver istanziato la classe *ClassiComuni.Client*) affinché siano trovati tutti gli autori associati al libro selezionato. Il parametro *String richiesta* rappresenta la query da fare al database.

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JScrollPane*, *JTable*, *JButton*, *JLabel*, ecc.).

void jButton1_actionPerformed(ActionEvent e)

Questo metodo serve ad implementare la funzione che deve essere svolta dal bottone (*JButton jButton1*) dell'interfaccia. Esso viene utilizzato per leggere

l'ID Libro inserito dall'utente nel campo di testo *textField1*, e viene utilizzato per avviare la ricerca nel database degli autori che hanno scritto quel determinato libro.

void tornaAlMenuPrincipale_actionPerformed(ActionEvent e)

Questo metodo serve ad implementare la funzione che deve essere svolta dal bottone (*JButton tornaAlMenuPrincipale*) dell'interfaccia. Questo metodo essenzialmente passa il controllo alle classi del tipo **.MenuPrincipale*Control*.

4.6.7.3 Costruttore della classe

public MostraLibro(String[][] query, int contatoreRighe, int contatoreColonne)

Questo costruttore ha tre parametri che servono ad inizializzare rispettivamente le variabili *Object[][] data*, *int numeroRighe* e *int numeroColonne*. In questo costruttore viene chiamato il metodo *jbInit()*.

4.6.8 ClassiComuni.ContenitoreDati

Si tratta di una classe di tipo boundary, che serve a visualizzare una Tabella (di tipo *com.sun.java.swing.JTable*) in cui vengono mostrati i risultati delle ricerche di qualsiasi tipo fatte sul database della libreria da uno degli utenti del sistema. In un componente del tipo *TextArea* viene mostrato, inoltre, il risultato della

ricerca (cioè il numero informazioni trovate nel database della libreria). Questa classe estende la classe *com.sun.java.swing.JFrame*.

4.6.8.1 Attributi della classe

Object[][] data

Questa matrice sarà utilizzata per contenere l'insieme dei dati trovati nel database.

String[] columnNames

Si tratta di un vettore di oggetti di tipo String. Esso conterrà il nome dei campi delle tabelle del database in cui è stata fatta la query di ricerca.

int numeroColonne

A questa variabile intera sarà assegnato un valore pari al numero di colonne della tabella da visualizzare.

int numeroRighe

A questa variabile intera sarà assegnato un valore pari al numero di righe della tabella da visualizzare.

4.6.8.2 Metodi della classe

private void jbInit() throws Exception

Questo metodo è ereditato dalla classe *com.sun.java.swing.JFrame*. Esso è chiamato dal costruttore della classe, e serve per assemblare ed ordinare tutti gli elementi grafici costituenti l'interfaccia (ad esempio *JScrollPane*, *JTable*, *JButton*, *JLabel*, ecc.).

void tornaAlMenuPrincipale_actionPerformed(ActionEvent e)

Questo metodo serve ad implementare la funzione che deve essere svolta dal bottone (*JButton tornaAlMenuPrincipale*) dell'interfaccia. Questo metodo essenzialmente passa il controllo alle classi del tipo **MenuPrincipale*Control*.

4.6.8.3 Costruttore della classe

public ContenitoreDati(String[][] query, int contatoreRighe, int contatoreColonne)

Questo costruttore ha tre parametri che servono ad inizializzare rispettivamente le variabili *Object[][] data*, *int numeroRighe* e *int numeroColonne*. In questo costruttore viene chiamato il metodo *jbInit()*.

4.6.9 ClassiComuni.Logger

Questa classe viene utilizzata per immagazzinare in un documento Microsoft Word tutte le informazioni relative alle operazioni compiute dal server (ad esempio, ora, tipo di interrogazione fatta sul database, informazione trovate in esso, ecc.). Questa classe viene anche istanziata dalla classe *VendiLibroBoundary*, per registrare le informazioni relative alla vendita di un libro (quantità venduta, ID del libro venduto, ID dell'impiegato che ha portato a termine l'operazione di vendita, data e ora in cui è stata effettuata l'operazione).

4.6.9.1 Attributi della classe

String logfile

Questo oggetto di tipo *String* rappresenta il percorso del file Word in cui immagazzinare le informazioni suddette.

String messaggio

Questo oggetto di tipo *String* rappresenta l'informazione da registrare nel file Word.

4.6.9.2 Metodi della classe

public void print(String s)

In questo metodo viene aperto uno stream di output per poter scrivere la stringa passatagli come parametro (*String s*) nel documento Word.

4.6.9.3 Costruttore della classe

public Logger(String comunicazione,String fileLog)

Questo costruttore ha due parametri che servono ad inizializzare rispettivamente le variabili *String messaggio* e *String logfile*. In questo costruttore viene chiamato il metodo *public void print(String s)*.

4.6.10 ClassiComuni.Client

Questa classe implementa il client che serve per comunicare con il server. Si tratta di un client UDP, quindi di un servizio senza connessione in cui non c'è una fase di handshaking iniziale tra il processo client ed il processo server. Quando un processo di questo tipo vuole spedire un batch di bytes ad un'altro processo, esso deve attaccare l'indirizzo del processo destinazione a ciascun batch di bytes inviato. L'insieme dell'indirizzo destinazione, del numero di porta e del batch di bytes di informazione inviato costituisce un pacchetto. Dopo aver creato un pacchetto, il client inserisce il pacchetto nella rete attraverso un socket. UDP non fornisce un servizio di trasporto affidabile ad i suoi processi comunicanti, esso, infatti, non garantisce che un datagram raggiungerà la sua destinazione finale. Per gli scopi del nostro progetto, trattandosi di una rete locale interna alla libreria, non sono necessari particolari requisiti di affidabilità.

4.6.10.1 Attributi della classe

static int numeroTotaleRighe

A questa variabile sarà assegnato un valore pari al numero totale di righe della matrice, rappresentante le informazioni trovate nel database, che il processo client riceve dal processo server.

static int numeroTotaleColonne

A questa variabile sarà assegnato un valore pari al numero totale di colonne della matrice, rappresentante le informazioni trovate nel database, che il processo client riceve dal processo server.

4.6.10.2 Metodi della classe

public static String[][] esegui(String richiesta, String identifier) throws Exception

Questo metodo implementa le funzioni svolte dal processo client. Esso ha due parametri che costituiscono rispettivamente la query da effettuare sul database ed un parametro che serve per comunicare al server il tipo di query da effettuare. Se quest'ultima variabile assume il valore "S", allora la query da effettuare sul database è di tipo *select* , quindi si presuppone che il server debba ritornare dei dati al client corrispondenti all'insieme dei record trovati nel database. Se targa assume valore "N", la query da effettuare sul database non sarà del tipo *select* ,

ma del tipo *delete*, *update* oppure *insert*, non presupponendo quindi un invio di dati dal server al client.

In questo metodo è presente una istruzione *if* che metterà il client in attesa di dati da parte del server nel caso in cui il parametro *identifier* assume valore “S”.

In questo metodo vengono costruiti degli stream e dei socket. Il socket è stato chiamato *clientSocket* ed è di tipo *DatagramSocket*. Lo stream *inFromUser* è uno stream di input al programma. UDP invia ogni pacchetto individualmente attraverso l’oggetto *DatagramSocket*.

Questo metodo ritorna una matrice contenente l’insieme delle informazioni trovate nel database.

public int RitornaNumeroRighe()

Questo metodo ritorna il valore associato alla variabile *numeroTotaleRighe*.

public int RitornaNumeroColonne()

Questo metodo ritorna il valore associato alla variabile *numeroTotaleColonne*.

public static int trovaFineParola(String parola)

Questo metodo viene utilizzato per trovare la fine di una stringa valida. Esso è stato implementato per risolvere alcuni problemi di visualizzazione dei dati trasmessi tra il processo client ed il processo server. Esso ritorna un valore intero pari all’ultimo indice valido della stringa passata come parametro al metodo.

4.6.11 ClassiComuni.MenuPrincipale

Si tratta di una interfaccia (*interface*) Java che è implementata da tutte le classi del tipo **MenuPrincipale*Control*.

4.6.11.1 Attributi della classe

Assenti.

4.6.11.2 Metodi della classe

public void esegui() throws Exception;

public void scegliRicerca();

public void InterrogaDatabase(String query);

4.7 GestioneServer.Server

Questa classe implementa il server che serve ad accettare ed a soddisfare le richieste del processo client. Si tratta di un server UDP, quindi di un servizio senza connessione in cui non c'è una fase di handshaking iniziale tra il processo server ed il processo client. Dopo aver creato un pacchetto, il server inserisce il pacchetto nella rete attraverso un socket e lo invia al client. UDP non fornisce un servizio di trasporto affidabile ad i suoi processi comunicanti, esso, infatti, non garantisce che un datagram raggiungerà la sua destinazione finale. Per gli scopi del nostro progetto, trattandosi di una rete locale interna alla libreria, non sono necessari particolari requisiti di affidabilità.

4.7.1.1 Attributi della classe

Assenti.

4.7.1.2 Metodi della classe

public static void esegui() throws Exception

Questo metodo implementa le funzioni svolte dal processo server.

In questo metodo è presente una istruzione *while* infinito che metterà il server in attesa di dati da parte del client. In questo metodo vengono costruiti degli stream e dei socket. Il socket è stato chiamato *serverSocket* ed è di tipo *DatagramSocket*. UDP invia e riceve ogni pacchetto individualmente attraverso l'oggetto *DatagramSocket*. I pacchetti che arrivano dal client vengono

spacchettati. I dati, quindi, vengono estratti dal pacchetto e assegnati ad una variabile locale *String sentence*. Questa istruzione ha un equivalente nel codice relativo al client. Successivamente vengono estratti l'indirizzo IP ed il numero di porta del processo client chiamante. Questo numero di porta è scelto dal client ed è diverso dal numero di porta del server che abbiamo scelto pari a 9876. E' necessario per il server ottenere sia l'indirizzo IP che il numero di porta del client per poter rispondere successivamente al client giusto, e non ad uno qualsiasi.

public static int trovaFineParola(String parola)

Questo metodo viene utilizzato per trovare la fine di una stringa valida. Esso è stato implementato per risolvere alcuni problemi di visualizzazione dei dati trasmessi tra il processo client ed il processo server. Esso ritorna un valore intero pari all'ultimo indice valido della stringa passata come parametro al metodo.

4.7.2 GestioneServer.DatabaseLibreria

Si tratta di una classe di controllo che serve a gestire le relazioni con il database Microsoft Access. Questa classe si occupa dell'interrogazione del database e di ritornare i dati trovati al server nel caso in cui la query in questione sia di tipo *select*.

4.7.2.1 Attributi della classe

private static final String SUB_DATA = "Libreria";

Essa rappresenta il nome da noi dato al database Microsoft Access che abbiamo creato.

private static final String DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver";

Rappresenta il nome del driver della Sun Microsystems da noi utilizzato. Si tratta di un bridge jdbc/odbc che utilizza l'interfaccia ODBC per connettersi alla base dei dati. Questo tipo di driver dipende dalle piattaforme Microsoft. I driver JDBC vengono caricati dall'applicazione in modo dinamico mediante un'istruzione del tipo *Class.forName(package.Class)*. Una volta che il driver (classe java) viene caricato è possibile connettersi al database tramite driver manager utilizzando il metodo *getConnection()*. Quest'ultimo metodo richiede in input la URL della base di dati ed una serie di informazioni necessarie ad effettuare la connessione. Il formato con cui devono essere passate queste informazioni dipende dal produttore dei driver.

private static final String PROTOCOL = "jdbc";

Rappresenta il nome del protocollo utilizzato per connettersi alla base di dati.

private static final String SUBPROTOCOL = "odbc";

Rappresenta il nome del sottoprotocollo utilizzato per connettersi alla base di dati.

public String[][] result

A questa variabile saranno assegnati i dati trovati nel database nel caso in cui la query effettuata sia del tipo *select*.

public String url

Rappresenta l'URL della base di dati.

int cols

A questa variabile intera sarà assegnato un valore pari al numero di colonne della tabella rappresentante i dati trovati nel database.

int rows

A questa variabile intera sarà assegnato un valore pari al numero di righe della tabella rappresentante i dati trovati nel database.

private static String query_1

Questa variabile rappresenta la query da inoltrare al database.

4.7.2.2 Metodi della classe

private void closeDB()

Questo metodo viene invocato quando si deve chiudere la connessione con il database. In questo caso l'oggetto di tipo *java.sql.Connection*, se è diverso da *null*, viene chiuso utilizzando il metodo *close()* di quest'ultimo oggetto.

private void closeStatement()

Questo metodo viene invocato per chiudere l'oggetto di tipo *java.sql.Statement*, se è diverso da *null*, utilizzando il metodo *close()* di quest'ultimo oggetto.

public int colonne()

Questo metodo ritorna il valore associato alla variabile *int cols*.

public int righe()

Questo metodo ritorna il valore associato alla variabile *int rows* a cui viene sommato un valore pari a 2 per aggiungere una riga all'inizio, contenente i nomi dei campi delle tabelle del database su cui è stata fatta la query, e una riga alla fine, che è stata posta per delimitare la fine dei dati trovati .

public boolean connectToDB()

Questo metodo viene utilizzato per stabilire una connessione con il database.

private void displayResultSet(String cmd)

Questo metodo viene utilizzato per eseguire la query (rappresentata dal parametro *String cmd*) sul database. Nel caso in cui la query in questione sia di tipo *select*, i dati trovati nel database saranno raccolti (assegnati) alla variabile globale *String[][] result*.

public void executeSQL()

Questo metodo inizialmente controlla la sintassi della query da effettuare sul database per vedere se rispetta le regole imposte dal linguaggio SQL. In caso positivo, questo metodo invoca il metodo *displayResultSet(String)* se la query in questione è di tipo *select*. Se la query in questione, invece, è di tipo *insert*, *update* oppure *delete*, è il metodo *executeSQL()* che si occupa di effettuare la query al database.

public String[][] ritornaDati()

Questo metodo ritorna il valore associato alla variabile *String[][] result*.

4.8 GestioneDati

4.8.1 GestioneDati.AutoreLibro

Si tratta di una classe entità che rappresenta l'autore del libro.

4.8.1.1 Attributi della classe

public int IDAutore

Questo attributo rappresenta l' ID Autore, cioè un parametro che rappresenta in modo univoco un autore di un libro all'interno del database.

public String nome

Questo attributo rappresenta il nome dell'autore.

public String cognome

Questo attributo rappresenta il cognome dell'autore.

public String note

Questo attributo rappresenta un'informazione associata ad un autore di un libro (ad esempio autore di libri dedicati al linguaggio di programmazione Java).

4.8.1.2 Metodi della classe

public void setCognome(String cognome)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String cognome*) all'attributo *cognome* dell'autore.

public void setIDAutore(String IDAutore)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDAutore*) all'attributo *IDAutore*.

public void setNome(String nome)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String nome*) all'attributo *nome* dell'autore.

public void setNote(String note)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String note*) all'attributo *note* associato all'autore di un libro.

4.8.1.3 Costruttore della classe

public AutoreLibro(int IDAutore, String nome, String cognome, String note)

Questo costruttore ha quattro parametri che servono ad inizializzare rispettivamente le variabili *int IDAutore*, *String nome*, *String cognome* e *String note*.

4.8.2 GestioneDati.CasaEditrice

Si tratta di una classe entità che rappresenta la casa editrice associata ad un libro.

4.8.2.1 Attributi della classe

public int IDCasaEditrice

Questo attributo rappresenta l' ID Casa Editrice, cioè un parametro che rappresenta in modo univoco una casa editrice associata ad un libro all'interno del database.

public String nomeCasaEditrice

Questo attributo rappresenta il nome della casa editrice.

public String indirizzo

Questo attributo rappresenta l'indirizzo della casa editrice.

public String citta

Questo attributo rappresenta il nome della città in cui ha sede la casa editrice.

public String cap

Questo attributo rappresenta il codice di avviamento postale della città in cui ha sede la casa editrice.

public String provincia

Questo attributo rappresenta la provincia della città in cui ha sede la casa editrice.

public String paese

Questo attributo rappresenta il nome dello Stato in cui ha sede la casa editrice.

public String telefono

Questo attributo rappresenta il numero di telefono della casa editrice.

public String fax

Questo attributo rappresenta il numero di fax della casa editrice.

public String email

Questo attributo rappresenta l'indirizzo di posta elettronica della casa editrice.

4.8.2.2 Metodi della classe

public void setIDCasaEditrice(int IDCasaEditrice)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDCasaEditrice*) all'attributo *IDCasaEditrice*.

public void setNomeCasaEditrice(String nomeCasaEditrice)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String nomeCasaEditrice*) all'attributo *nomeCasaEditrice*.

public void setCap(String cap)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String cap*) all'attributo *cap*.

public void setIndirizzo(String indirizzo)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String indirizzo*) all'attributo *indirizzo* associato alla casa editrice di un libro.

public void setCitta(String citta)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String citta*) all'attributo *citta* associato alla casa editrice di un libro.

public void setProvincia(String provincia)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String provincia*) all'attributo *provincia* associato alla città della casa editrice di un libro.

public void setPaese(String paese)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String paese*) all'attributo *paese* associato alla casa editrice di un libro.

public void setTelefono(String telefono)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String telefono*) all'attributo *telefono* associato alla casa editrice di un libro.

public void setFax(String fax)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String fax*) all'attributo *fax* associato alla casa editrice di un libro.

public void setEmail(String email)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String email*) all'attributo *email* associato alla casa editrice di un libro.

4.8.2.3 Costruttore della classe

```
public CasaEditrice(int IDCasaEditrice, String nomeCasaEditrice,  
String indirizzo, String citta, String cap, String provincia, String paese,  
String telefono, String fax, String email)
```

Questo costruttore ha dieci parametri che servono ad inizializzare rispettivamente le variabili *int IDCasaEditrice*, *String nomeCasaEditrice*, *String indirizzo*, *String citta*, *String cap*, *String provincia*, *String paese*, *String telefono*, *String fax*, *String email*.

4.8.3 GestioneDati.AutoreLibro

Si tratta di una classe entità che rappresenta l'autore del libro.

4.8.3.1 Attributi della classe

public int IDProfessore

Questo attributo rappresenta l' ID Professore, cioè un parametro che rappresenta in modo univoco un professore di un corso universitario all'interno del database.

public String nome

Questo attributo rappresenta il nome del professore di un corso.

public String cognome

Questo attributo rappresenta il cognome di un professore.

public String telefono

Questo attributo rappresenta il numero di telefono di un professore di un corso universitario.

public String email

Questo attributo rappresenta l'indirizzo di posta elettronica di un professore di un corso.

4.8.3.2 Metodi della classe

public void setIDProfessore(int IDProfessore)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDProfessore*) all'attributo *IDProfessore*.

public void setCognome(String cognome)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String cognome*) all'attributo *cognome* dell'oggetto rappresentante il professore.

public void setNome(String nome)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String nome*) all'attributo *nome* del professore.

public void setTelefono(String telefono)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String telefono*) all'attributo *telefono* associato ad un professore.

public void setEmail(String email)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String email*) all'attributo *email* associato ad un professore.

4.8.3.3 Costruttore della classe

```
public Professore(int IDProfessore, String cognome, String nome, String telefono, String email)
```

Questo costruttore ha cinque parametri che servono ad inizializzare rispettivamente le variabili *int IDProfessore*, *String cognome*, *String nome*, *String telefono*, *String email*.

4.8.4 GestioneDati.ClienteLibreria

Si tratta di una classe entità che rappresenta il cliente che entra nella libreria per comprare un libro.

4.8.4.1 Attributi della classe

```
public int IDCliente
```

Questo attributo rappresenta l' ID Cliente, cioè un parametro che rappresenta in modo univoco un cliente della libreria all'interno del database.

```
public String nome
```

Questo attributo rappresenta il nome di un cliente.

```
public String cognome
```

Questo attributo rappresenta il cognome di un cliente.

public String indirizzo

Questo attributo rappresenta l'indirizzo del cliente della libreria.

public String citta

Questo attributo rappresenta il nome della città in cui vive il cliente.

public String cap

Questo attributo rappresenta il codice di avviamento postale della città in cui vive il cliente.

public String telefono

Questo attributo rappresenta il numero di telefono del cliente.

public String email

Questo attributo rappresenta l'indirizzo di posta elettronica del cliente.

4.8.4.2 Metodi della classe

public void setIDCliente(int IDCliente)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int ID Cliente*) all'attributo *IDCliente*.

public void setNome(String nome)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String nome*) all'attributo *nome*.

public void setCognome(String cognome)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String cognome*) all'attributo *cognome*.

public void setCap(String cap)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String cap*) all'attributo *cap*.

public void setIndirizzo(String indirizzo)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String indirizzo*) all'attributo *indirizzo*.

public void setCitta(String citta)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String citta*) all'attributo *citta*.

public void setTelefono(String telefono)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String telefono*) all'attributo *telefono* associato al cliente della libreria.

public void setEmail(String email)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String email*) all'attributo *email* associato al cliente della libreria.

4.8.4.3 Costruttore della classe

public ClienteLibreria(int IDCliente, String nome, String cognome, String indirizzo, String citta, String cap, String email, String telefono)

Questo costruttore ha otto parametri che servono ad inizializzare rispettivamente le variabili *int IDCliente*, *String nome*, *String cognome*, *String indirizzo*, *String citta*, *String cap*, *String email*, *String telefono*.

4.8.5 GestioneDati.Corsi

Si tratta di una classe entità che rappresenta un corso universitario.

4.8.5.1 Attributi della classe

public int IDCorsi

Questo attributo rappresenta l' ID Corso, cioè un parametro che rappresenta in modo univoco un corso universitario all'interno del database.

public String nomeCorso

Questo attributo rappresenta il nome di un corso.

public int semestre

Questo attributo rappresenta il semestre in cui si svolge il corso universitario.

public String anno

Questo attributo rappresenta l'anno in cui si svolge il corso.

public int IDProfessore

Questo attributo rappresenta l'ID associato al professore che tiene il corso.

public int IDFacolta

Questo attributo rappresenta l'ID associato alla facoltà in cui si tiene il corso.

4.8.5.2 Metodi della classe

public void setIDCorsi(int IDCorsi)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int ID Corso*) all'attributo *IDCorso*.

public void setNomeCorso(String nomeCorso)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String nomeCorso*) all'attributo *nomeCorso*.

public void setSemestre(int semestre)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int semestre*) all'attributo *semestre*.

public void setAnno(String anno)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String anno*) all'attributo *anno*.

public void setIDProfessore(int IDProfessore)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDProfessore*) all'attributo *IDProfessore*.

public void setIDFacolta(int IDFacolta)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDFacolta*) all'attributo *IDFacolta*.

4.8.5.3 Costruttore della classe

```
public Corsi(int IDCorsi, String nomeCorso, int semestre, String anno, int  
IDProfessore, int IDFacolta)
```

Questo costruttore ha sei parametri che servono ad inizializzare rispettivamente le variabili *int IDCorsi*, *String nomeCorso*, *int semestre*, *String anno*, *int IDProfessore*, *int IDFacolta*.

4.8.6 GestioneDati.Impiegati

Si tratta di una classe entità che rappresenta un impiegato della libreria (Amministratore, Addetto agli Ordini, Addetto alle Vendite oppure Addetto al Magazzino).

4.8.6.1 Attributi della classe

```
public int IDImpiegato
```

Questo attributo rappresenta l' ID Impiegato, cioè un parametro che rappresenta in modo univoco i dati relativi ad un impiegato all'interno del database.

public String nome

Questo attributo rappresenta il nome di un impiegato.

public String cognome

Questo attributo rappresenta il cognome di un impiegato.

public String ruolo

Questo attributo rappresenta il ruolo svolto dall'impiegato all'interno della libreria.

public String reparto

Questo attributo rappresenta il reparto della libreria in cui lavora l'addetto.

public String internoTelefono

Questo attributo rappresenta il numero telefonico interno dell'impiegato.

public String password

Questo attributo rappresenta la password assegnata all'impiegato. Questa password dovrà essere inserita dall'impiegato all'avvio dell'applicativo.

4.8.6.2 Metodi della classe

public void setIDImpiegato(int IDImpiegato)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int ID Impiegato*) all'attributo *IDImpiegato*.

public void setNome(String nome)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String nome*) all'attributo *nome*.

public void setCognome(String cognome)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String cognome*) all'attributo *cognome*.

public void setRuolo(String ruolo)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String ruolo*) all'attributo *ruolo*.

public void setReparto(String reparto)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String reparto*) all'attributo *reparto*.

public void setInternoTelefono(String internoTelefono)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String internoTelefono*) all'attributo *internoTelefono*

public void setPassword(String password)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String password*) all'attributo *password*.

4.8.6.3 Costruttore della classe

public Impiegati(int IDImpiegato, String nome, String cognome, String ruolo, String reparto, String internoTelefono, String password)

Questo costruttore ha sette parametri che servono ad inizializzare rispettivamente le variabili *int IDImpiegato*, *String nome*, *String cognome*, *String ruolo*, *String reparto*, *String internoTelefono*, *String password*.

4.8.7 GestioneDati.Ordine

Si tratta di una classe entità che rappresenta un ordine spiccato dalla libreria verso un fornitore (Casa Editrice).

4.8.7.1 Attributi della classe

public int IDOrdine

Questo attributo rappresenta l' ID Ordine, cioè un parametro che rappresenta in modo univoco i dati relativi ad un ordine all'interno del database.

public String numeroOrdine

Questo attributo rappresenta il numero d'ordine associato ad un ordine.

public int IDCasaEditrice

Questo attributo rappresenta l'ID associato alla casa editrice a cui inviare l'ordine.

public String dataOrdine

Questo attributo rappresenta la data in cui viene fatto l'ordine.

public String dataSpedizione

Questo attributo rappresenta la data in cui viene spedito l'ordine.

public String statoOrdine

Questo attributo rappresenta lo stato dell'ordine, che può essere “Aperto”, “Pendente”, oppure “Chiuso”.

4.8.7.2 Metodi della classe

public void setIDOrdine (int IDOrdine)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int ID Ordine*) all'attributo *IDOrdine*.

public void setNumeroOrdine(String numeroOrdine)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String numeroOrdine*) all'attributo *numeroOrdine*.

public void setIDCasaEditrice (int IDCasaEditrice)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDCasaEditrice*) all'attributo *IDCasaEditrice*.

public void setDataOrdine(String dataOrdine)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String dataOrdine*) all'attributo *dataOrdine*.

public void setDataSpedizione(String dataSpedizione)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String dataSpedizione*) all'attributo *dataSpedizione*.

public void setStatoOrdine(String statoOrdine)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String statoOrdine*) all'attributo *statoOrdine*.

4.8.7.3 Costruttore della classe

public Ordine(int IDOrdine, String numeroOrdine, int IDCasaEditrice, String dataOrdine, String dataSpedizione, String statoOrdine)

Questo costruttore ha sei parametri che servono ad inizializzare rispettivamente le variabili *int IDOrdine*, *String numeroOrdine*, *int IDCasaEditrice*, *String dataOrdine*, *String dataSpedizione*, *String statoOrdine*.

4.8.8 GestioneDati.PreordineCliente

Si tratta di una classe entità che rappresenta il preordine di un libro fatto dall'addetto alla libreria quando un cliente vuol comprare un libro non presente in libreria.

4.8.8.1 Attributi della classe

public int IDPreordine

Questo attributo rappresenta l' ID Preordine, cioè un parametro che rappresenta in modo univoco i dati relativi ad un preordine all'interno del database.

public int IDCliente

Questo attributo rappresenta l'ID associato al cliente che richiesto il libro non presente in libreria.

public int IDLibro

Questo attributo rappresenta l'ID associato al libro richiesto dall'utente.

public int IDImpiegato

Questo attributo rappresenta l'ID associato all'impiegato che ha fatto il preordine.

public int quantitaRichiesta

Questo attributo rappresenta la quantità di libri che il cliente vuole prenotare.

public float prezzoUnitario

Questo attributo rappresenta il prezzo unitario del libro richiesto dal cliente della libreria.

public float subtotale

Questo attributo rappresenta la somma dovuta dal cliente relativamente ad un tipo di libro prenotato.

public float acconto

Questo attributo rappresenta l'acconto (in euro) lasciato dal cliente al momento della prenotazione del libro.

public float rimanenza

Questo attributo rappresenta la rimanenza che il cliente deve dare alla libreria al momento del ritiro del libro ordinato.

public String dataPrenotazione

Questo attributo rappresenta la data in cui il libro è stato prenotato dal cliente.

public String statoPreordine

Questo attributo rappresenta lo stato del preordine che può essere “Evaso” oppure “Da evadere”.

4.8.8.2 Metodi della classe

public void setIDPreordine(int IDPreordine)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDPreordine*) all'attributo *IDPreordine*.

public void setIDLibro(int IDLibro)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDLibro*) all'attributo *IDLibro*.

public void setIDImpiegato(int IDImpiegato)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDImpiegato*) all'attributo *IDImpiegato*.

public void setIDCliente(int IDCliente)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDCliente*) all'attributo *IDCliente*.

public void setQuantitaRichiesta(int quantitaRichiesta)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int quantitaRichiesta*) all'attributo *quantitaRichiesta*.

public void setPrezzoUnitario(float prezzoUnitario)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *float prezzoUnitario*) all'attributo *prezzoUnitario* .

public void setSubtotale(float subtotale)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *float subtotale*) all'attributo *subtotale* .

public void setAcconto(float acconto)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *float acconto*) all'attributo *acconto*.

public void setRimanenza(float rimanenza)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *float rimanenza*) all'attributo *rimanenza* .

public void setDataPrenotazione(String dataPrenotazione)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String dataPrenotazione*) all'attributo *dataPrenotazione*.

public void setStatoPreordine(String statoPreordine)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String statoPreordine*) all'attributo *statoPreordine*.

4.8.8.3 Costruttore della classe

public PreordineCliente(int IDPreordine,int IDCliente, int IDLibro, int IDImpiegato, int quantitaRichiesta, float prezzoUnitario, float subtotale, float acconto, float rimanenza, String dataPrenotazione, String statoPreordine)

Questo costruttore ha undici parametri che servono ad inizializzare rispettivamente le variabili *int IDPreordine, int IDCliente, int IDLibro, int IDImpiegato, int quantitaRichiesta, float prezzoUnitario, float subtotale, float acconto, float rimanenza, String dataPrenotazione, String statoPreordine*.

4.8.9 GestioneDati.Transazione

Si tratta di una classe entità che rappresenta lo stato della transazione relativo ad un ordine.

4.8.9.1 Attributi della classe

public int IDTransazione

Questo attributo rappresenta l' ID Transazione, cioè un parametro che rappresenta in modo univoco i dati relativi ad una transazione all'interno del database.

public String dataTransazione

Questo attributo rappresenta la data in cui è avvenuta una determinata transazione.

public String tipoTransazione

Questo attributo rappresenta il tipo di transazione, che può essere del tipo “Invio Ordine” nel momento in cui viene spiccato un ordine da parte della libreria, oppure del tipo “Ricevo Libri” quando la libreria riceve dalla casa editrice dei libri precedentemente ordinati.

public int IDLibro

Questo attributo rappresenta l'ID associato al libro ordinato.

public int IDOrdine

Questo attributo rappresenta l'ID dell'ordine a cui si riferisce la transazione.

public int quantitaRichiesta

Questo attributo rappresenta la quantità di libri che è stata ordinata dalla libreria alla casa editrice.

public int quantitaRicevuta

Questo attributo rappresenta la quantità di libri che è stata spedita dalla casa editrice alla libreria.

public int quantitaMancante

Questo attributo rappresenta la quantità di libri che è stata ordinata dalla libreria alla casa editrice, ma che ancora la libreria non ha ricevuto.

4.8.9.2 Metodi della classe

public void setIDTransazione(int IDTransazione)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDTransazione*) all'attributo *IDTransazione*.

public void setIDLibro(int IDLibro)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDLibro*) all'attributo *IDLibro*.

public void setIDOrdine(int IDOrdine)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDOrdine*) all'attributo *IDOrdine*.

public void setDataTransazione(String dataTransazione)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String dataTransazione*) all'attributo *dataTransazione*.

public void setTipoTransazione(String tipoTransazione)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String tipoTransazione*) all'attributo *tipoTransazione*.

public void setQuantitaRichiesta(int quantitaRichiesta)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int quantitaRichiesta*) all'attributo *quantitaRichiesta*.

public void setQuantitaRicevuta(int quantitaRicevuta)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int quantitaRicevuta*) all'attributo *quantitaRicevuta*.

public void setQuantitaMancante(int quantitaMancante)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int quantitaMancante*) all'attributo *quantitaMancante*.

4.8.9.3 Costruttore della classe

public Transazione(int IDTransazione,String dataTransazione, String tipoTransazione, int IDLibro, int IDOrdine, int quantitaRichiesta, int quantitaRicevuta, int quantitaMancante)

Questo costruttore ha otto parametri che servono ad inizializzare rispettivamente le variabili *int IDTransazione*, *String dataTransazione*, *String tipoTransazione*, *int IDLibro*, *int IDOrdine*, *int quantitaRichiesta*, *int quantitaRicevuta*, *int quantitaMancante*.

4.8.10 GestioneDati.Transazione

Si tratta di una classe entità che rappresenta il libro.

4.8.10.1 Attributi della classe

public int IDLibro

Questo attributo rappresenta l' ID Libro, cioè un parametro che rappresenta in modo univoco i dati relativi ad un libro all'interno del database.

public String titolo

Questo attributo rappresenta il titolo di un libro.

public int annoCopyright

Questo attributo rappresenta l'anno di Copyright di un libro.

public String codiceISBN

Questo attributo rappresenta il codice ISBN associato ad un libro.

public String tipoEdizione

Questo attributo rappresenta il tipo di edizione del libro, come ad esempio Universitaria, Guida, ecc..

public int numeroEdizione

Questo attributo rappresenta il numero dell'edizione del libro (1, 2, 3, ecc.).

public int numeroPagine

Questo attributo rappresenta il numero di pagine di un libro.

public float prezzo

Questo attributo rappresenta il prezzo associato ad un libro.

public String sconto

Questo attributo rappresenta lo sconto che la libreria applica su quel determinato libro.

public int IDCategoria

Questo attributo rappresenta la categoria a cui appartiene un libro, come ad esempio Informatica, Matematica, Elettronica, ecc..

public String collocazione

Questo attributo rappresenta la collocazione del libro nel magazzino della libreria.

public int totaleCopie

Questo attributo rappresenta il numero di copie di un libro presenti nel magazzino della libreria.

public int numeroCopieDisponibili

Questo attributo rappresenta il numero di copie di un libro disponibili nel magazzino della libreria e che non sono state prenotate ancora da alcun cliente.

public int numeroCopiePrenotate

Questo attributo rappresenta il numero di copie di un libro disponibili nel magazzino della libreria, ma che sono state prenotate ancora da clienti.

public int totaleCopieVendute

Questo attributo rappresenta il numero totale di copie di un libro che la libreria ha venduto.

public int totaleCopieAcquistate

Questo attributo rappresenta il numero totale di copie di un libro che la libreria ha acquistato.

4.8.10.2 Metodi della classe

public void setIDLibro(int IDLibro)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDLibro*) all'attributo *IDLibro*.

public void setTitolo(String titolo)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String titolo*) all'attributo *titolo*.

public void setAnnoCopyright(int annoCopyright)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int annoCopyright*) all'attributo *annoCopyright*.

public void setIDCategoria(int IDCategoria)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int IDCategoria*) all'attributo *IDCategoria*.

public void setCodiceISBN(String codiceISBN)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String codiceISBN*) all'attributo *codiceISBN*.

public void setTipoEdizione(String tipoEdizione)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String tipoEdizione*) all'attributo *tipoEdizione*.

public void setNumeroEdizione(int numeroEdizione)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int numeroEdizione*) all'attributo *numeroEdizione*.

public void setNumeroPagine(int numeroPagine)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int numeroPagine*) all'attributo *numeroPagine*.

public void setPrezzo(float prezzo)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *float prezzo*) all'attributo *prezzo*.

public void setSconto(String sconto)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String sconto*) all'attributo *sconto*.

public void setCollocazione(String collocazione)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *String collocazione*) all'attributo *collocazione*.

public void setTotaleCopie(int totaleCopie)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int totaleCopie*) all'attributo *totaleCopie*.

public void setNumeroCopieDisponibili(int numeroCopieDisponibili)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int numeroCopieDisponibili*) all'attributo *numeroCopieDisponibili*.

public void setNumeroCopiePrenotate(int numeroCopiePrenotate)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int numeroCopiePrenotate*) all'attributo *numeroCopiePrenotate*.

public void setTotaleCopieVendute(int totaleCopieVendute)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int totaleCopieVendute*) all'attributo *totaleCopieVendute*.

public void setTotaleCopieAcquistate(int totaleCopieAcquistate)

Questo metodo viene utilizzato per assegnare un valore (il valore del parametro *int totaleCopieAcquistate*) all'attributo *totaleCopieAcquistate*.

4.8.10.3 Costruttore della classe

```
public Libro(int IDlibro,String titolo,int annoCopyright, String  
codiceISBN, String tipoEdizione, int numeroEdizione, int numeroPagine,  
float prezzo, int IDCategoria, String sconto, String collocazione, int  
totaleCopie, int numeroCopieDisponibili, int numeroCopiePrenotate, int  
totaleCopieVendute, int totaleCopieAcquistate)
```

Questo costruttore ha sedici parametri che servono ad inizializzare rispettivamente le variabili *int IDlibro,String titolo,int annoCopyright, String codiceISBN, String tipoEdizione, int numeroEdizione, int numeroPagine, float prezzo, int IDCategoria, String sconto, String collocazione, int totaleCopie, int numeroCopieDisponibili, int numeroCopiePrenotate, int totaleCopieVendute, int totaleCopieAcquistate*.

APPENDICE A

Materiale fornito

Il seguente materiale viene fornito insieme a questa documentazione :

[NuovaLibreriaUniversitariaG&R.mdl](#) : file Rational Rose relativo al progetto del sistema.

[Libreria.mdb](#) : file database contente i dati utilizzati dal programma

[Documentazione.doc](#) :file contenente tutta la documentazione qui presentata .

[ProgettoLibreriaG&R.mpp](#): file di pianificazione del progetto.

[File *.java](#): relativi al codice sorgente dell'applicazione sviluppata.

[File*.class](#): file java compilati costituenti l'applicazione.

[Altro](#) :tutto quello che si ritiene essere indispensabile per il corretto funzionamento dell'applicazione.

Tali file sono stati registrati sul supporto CD allegato a questa documentazione.

Come avviare l'applicazione :

- Copiare i file dal CD su c:\
- Avviare
- Eseguire da Dos Libreria.bat
- Ritornare nella cartella Applicazioni e :
- Eseguire :
 - Cliente.bat per l'applicazione che dovrà essere eseguita sul terminale cliente.

- Amministratore.bat per l'applicazione che dovrà essere eseguita sul terminale dell'amministratore.
- Magazzino.bat per l'applicazione che dovrà essere eseguita sul terminale del responsabile del magazzino.
- Ordini.bat per l'applicazione che dovrà essere eseguita sul terminale del responsabile degli ordini.
- Vendita per l'applicazione che dovrà essere eseguita sul terminale del responsabile alle vendite.
- Server per l'applicazione che dovrà essere eseguita sulla postazione server.