UNIVERSITA' DEGLI STUDI DI PADOVA

FACOLTA' DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Relazione di Tirocinio Lungo

UN SISTEMA DI GESTIONE INTEGRATA D'IMPRESA

(An integrated enterprise management system)

Laureando: Daniele Barilaro

Relatore: Ing. Franco Bombi

Corso di Laurea Triennale in Ingegneria Informatica

Anno accademico 2009/2010

RINGRAZIAMENTI

Un doveroso ringraziamento all'azienda AddSource che è stata sede del mio tirocinio, in particolare al mio tutore aziendale Dott. Fabio Buttari che mi ha seguito attentamente e con dedizione. Inoltre ringrazio l' ing. Adriano Luchetta , tutore universitario, e l'ing. Franco Bombi, relatore della presente tesi, per la cortese disponibilità.

SOMMARIO

PREMESSA

E' stato svolto un lavoro di tirocinio lungo(500 ore in 6 mesi) presso l'azienda ADD SOURCE Gestione Integrata d'Impresa di Dosson di Casier (Treviso). AddSource si occupa di fornire consulenza, formazione e soluzioni alle piccole e medie imprese per la gestione dei fornitori, vendite, clienti e tutti gli aspetti legati all'attività economica aziendale.
Interesse dell'azienda era quello di realizzare un framework che consenta di sviluppare velocemente software gestionali fruibili tramite portale web e mediante esso sviluppare un applicativo per la gestione dei centri di cura del corpo, chiamato AddCenter, da adattare successivamente ad un centro benessere SPA (Salus per aquam) che ne ha fatto richiesta.
Si è voluto realizzarlo come applicativo web, sfruttando le tecnologie ed i vantaggi messi a disposizione dal Web 2.0 che forniscono un elevato livello di iterazione sito-utente.
E' nata un'applicazione internet ricca, che fornisce cioè le stesse caratteristiche di un'applicazione desktop, e di livello enterprise, che aggiunge tutte quelle caratteristiche richieste dalle imprese quali l'efficienza nella sua amministrazione e manutenzione, l'alta fruibilità e velocità.
L'applicativo è residente interamente su server (aziendale o web) ed è sviluppato su piattaforma Java EE. Essa è la versione progettata per le aziende della piattaforma Java che aggiunge a quest'ultima funzionalità per creare software distribuito, multi-livello, efficiente e tollerante ai guasti, basato su moduli eseguiti su un server chiamato application server. Quello utilizzato è GlassFish che è l'implementazione di riferimento di tale piattaforma.
L'applicativo utilizza il framework Java Server Faces (JSF), che è una tecnologia Java Enterprise basata sul design-pattern architetturale modello-vista-controllore per lo sviluppo di interfacce web, e il framework ICEfaces che viene utilizzato per creare applicazioni internet ricche sopra a JSF. Come sistema per la gestione della base di dati relazionale dell'applicativo viene utilizzato MySql, uno dei più noti database open source, famoso per la sua velocità, affidabilità e facilità d'uso.

Nella progettazione e sviluppo di tale applicazione si è svolta la mia attività di tirocinio, che mi ha visto impegnato nella scelta delle tecnologie da utilizzare, nell'intera ideazione e realizzazione della base di dati per i centri di cura del corpo e nella progettazione e scrittura dei moduli che costituiscono il framework per lo sviluppo rapido di software gestionali.
Nel primo capitolo vengono descritte le scelte tecnologiche effettuate per realizzare l'applicativo. Ho contribuito all'adozione del framework ICEfaces per la realizzazione dell'interfaccia grafica ricca, voluto in quanto è un framework che offre caratteristiche molto superiori rispetto agli altri della stessa categoria per lo sviluppo di applicazioni JavaEE AJAX. Inoltre ho proposto la realizzazione di un gestionale valido generalmente per la gestione dei centri di cura del corpo, non solo centri benessere come inizialmente era stato pianificato. Questo perché nella fase di raccolta ed analisi dei requisiti della progettazione della base di dati per centri benessere ho notato molte similitudini di questi coi centri di cura del corpo di altro tipo, quali centri estetici, barbieri, parrucchieri e saune. Ho avuto quindi la responsabilità di progettare completamente la base di dati rendendola flessibile per tutte queste realtà. L'analisi, progettazione e implementazione (mediante mappatura oggetto/relazionale dello strato di persistenza Java) è descritta nel secondo capitolo di tale relazione.
Ho curato infine lo sviluppo del framework per la creazione rapida di software gestionali, che consiste principalmente di normali classi Java, classi Enterprise Java Beans e modelli di pagine JSF-Facelets che consentono la creazione automatica di viste per le anagrafiche delle entità dello strato di persistenza e forniscono un'interfaccia applicativa per innestare all' interno di ciascuna vista un editor per il tipo di entità gestita.
Nel dettaglio ho scritto un piccolo framework (QuerySystem) per la creazione guidata di interrogazioni per il motore di persistenza Java, che permette di specificarle mediante linguaggio di programmazione (metodi di classi), piuttosto che tramite il linguaggio dichiarativo JPQL (il corrispettivo Java di SQL). JPQL, come SQL, è un linguaggio statico che ha la necessità di essere codificato in fase di programmazione nel codice sorgente del programma e quindi non è adatto per la creazione in fase di esecuzione (run-time) di interrogazioni. Invece la possibilità di definire in linguaggio di programmazione le interrogazioni comporta il grande vantaggio di poterle creare dinamicamente, controllando tutti gli aspetti della loro definizione. Questo lo rende ideale per generarle a tempo di esecuzione. Ciò è richiesto dalle viste dell'applicativo, che essendo create dinamicamente, devono avere la possibilità di specificare in modo arbitrario le entità che devono recuperare e visualizzare, ed eventualmente specificare su di esse filtri di ricerca (impostati lato utente o applicativo).
Correlato a QuerySystem ho realizzato il modulo, presente nelle viste, per la ricerca delle entità di un determinato tipo, che permette di specificare con un altro livello di dettaglio, ma in modo semplice, i filtri di ricerca.
Ho implementato l'API per l'innesto nelle viste degli editor (EntityEditor API), con la relativa gestione degli errori ed eccezioni che questi possono generare, e scritto alcuni editor di base, quali quelli per i trattamenti e prodotti.
La struttura di questi moduli è descritta, fornendo alcuni esempi di utilizzo, nei capitoli quarto e quinto.
Infine ho curato l'interazione di tutti i moduli che compongono la vista con essa affinché questa sia operativa.

CAPITOLO 1

1 INTRODUZIONE E SPECIFICHE

1.1 Introduzione

Presso AddSource è stato richiesto lo sviluppo di un applicativo per la gestione di un centro benessere.
Dopo una pianificazione preliminare sono emerse varie necessità e specifiche che si volevano raggiungere.

1.2 Specifiche di progetto

Si descrivono nel seguito le specifiche più importanti emerse in fase di progettazione dell'applicativo.

1.2.1 Prima specifica: Creare un framework che consenta di sviluppare velocemente software gestionali RIA basati sulla tecnologia Java EE. L'applicativo per il centro benessere verrà successivamente costruito sopra tale framework.

Si vuole creare un framework che consenta di rendere più rapida la costruzione di applicativi gestionali che utilizzano la piattaforma Java EE, semplificando tutte quelle procedure e logiche di business che si ripetono spesso in tali applicativi, nonché fornendo uno strumento per costruire velocemente le sezioni desiderate. Comunemente un gestionale è diviso in sezioni per visualizzare, creare e modificare le varie entità della realtà che rappresenta. Ad esempio un software gestionale utilizzato per un negozio di abbigliamento potrà avere una sezione dedicata ai fornitori, una dedicata al magazzino, un'altra ancora dedicata agli articoli in vendita nel negozio e così via. Tutte queste sezioni hanno in comune un elenco di tutte le entità dello specifico tipo associato. Ad esempio la sezione dedicata ai fornitori visualizzare in qualche modo tutti i fornitori inseriti nel sistema informatico.
Per ciascun tipo di entità è necessario un editor specifico per visualizzare, creare, modificare ed eliminare le istanze di entità di quel tipo.
L'obiettivo del framework è perciò quello di fornire strumenti e modelli per costruire tali sezioni e per inserire in esse editor creati appositamente per le entità di tali sezioni.
L'applicativo utilizzerà il framework per la creazione delle sezioni di gestione dei clienti, appuntamenti, ricevute e tutte le altre della realtà da gestire, garantendo all'utente un'interfaccia grafica comune per tutte le sezioni. Ciò garantirà un più facile apprendimento da parte dell'utente finale del funzionamento del gestionale, dovendo apprendere principalmente il funzionamento di un'unica interfaccia comune.

1.2.2 Seconda specifica: progettare un applicativo valido generalmente per la gestione di centri di cura del corpo di vario tipo, non solo centri benessere.

Dopo una breve analisi del settore di riferimento è emerso che vi sono una moltitudine di attività commerciali che per la gestione hanno requisiti simili a quelli di un centro benessere. Tali attività sono tutte quelle che riguardano la cura del corpo in generale. Rientrano perciò nella categoria, oltre al centro benessere, centri estetici, solarium, barbieri, parrucchieri, terme e simili. Tutti effettuano dei trattamenti sui loro clienti ed eventualmente possono anche vendere prodotti per la cura del corpo, ad esempio un barbiere può vedere degli shampoo e un centro estetico una crema antirughe. Inoltre devono gestire gli appuntamenti con i clienti, combinando gli ambienti disponibili per effettuare il trattamento richiesto con l'operatore disponibile per eseguirlo. Tutti questi centri richiedono la gestione delle ricevute e probabilmente anche una gestione di carte fedeltà e sconti su prodotti e trattamenti
Ecco quindi che è possibile sviluppare, con uno sforzo aggiuntivo minimo, un applicativo valido per centri di cura del corpo di ogni genere che si adatterà facilmente alla realtà nella quale si andrà ad integrare. L'importante è che esso rappresenti e gestisca tutti i concetti comuni ai vari tipi di centro, e possieda sezioni aggiuntive che si possono utilizzare per gestire aspetti particolari a ciascun centro. Ad esempio l'applicativo dovrà disporre di una sezione per la gestione delle carte fedeltà, anche se poi non tutti i centri avranno bisogno di esse.
Ecco quindi che da questo requisito nasce il nome dell'applicativo *AddCenter*, che specifica la sua natura universale per gestire i centri di vario tipo. Si adatterà poi, molto facilmente, tale applicativo alla gestione del centro benessere.

1.2.3 Terza specifica: i clienti del sistema devono poter accedere alle loro informazioni (ricevute, appuntamenti ecc.), in sola lettura, tramite portale web.

Tale requisito è semplice da soddisfare data la natura web del software da progettare. Bisognerà semplicemente introdurre una gestione dettagliata delle utenze e diversificare le sezioni di portale alle quali ciascuna di esse può accedere. Se un segretario effettua l'accesso, potrà gestire tutto l'applicativo, mentre se un cliente entra nel sistema, potrà visualizzare in sola lettura le informazioni che lo riguardano, ad esempio potrà stampare una copia delle ricevute che lo riguardano. Grazie a Java EE, che possiede un potente sistema di autenticazione ed autorizzazione chiamato JAAS [1] (Java Authentication and Authorization System), è possibile implementare in maniera standard, semplice e potente la gestione delle utenze e della sicurezza.

1.3 Scelte tecnologiche

In questa sezione vengono descritte le scelte tecnologiche effettuate per la realizzazione dell'applicativo. In particolare è effettuata un'analisi delle possibili tecnologie, presenti nel panorama software, con le quali poteva essere sviluppato l'applicativo e vengono descritte, con relativa motivazione, quelle scelte per realizzarlo.

1.3.1 Prima scelta: Rich Internet Application

Si è scelto di sviluppare l'applicativo secondo una metodologia diversa da quella comunemente usata per gestionali di questo tipo che prevede lo sviluppo di un'applicazione desktop stand alone da installare in un computer in locale. Con un occhio alle nuove tecnologie e metodologie di sviluppo software si è deciso di progettare l'applicazione sotto forma di applicazione per internet ricca, in inglese Rich Internet Application (RIA) [2]. Ciò significa lo sviluppo di un software con le stesse caratteristiche di un'applicazione desktop ma accessibile tramite rete e visualizzata in un browser sotto forma di contenuto web. Nel mondo del web vi sono molti portali che offrono applicazioni RIA di varia natura, da quelli per l'online gaming (lo standard Adobe Flash predomina in questo campo), a quelli di posta elettronica quali Gmail di Google, Hotmail di Microsoft, Yahoo mail, fino ai portali di social network quali Facebook [3](1), Twitter [4](2), MyVip[5](3).
Le RIA si caratterizzano per la multimedialità, velocità di interazione con l'utente e di esecuzione. Solo l'interfaccia grafica è trasferita al client, mentre la maggior parte della logica di business e i dati rimangono sul server remoto. Le RIA si basano perciò su un' architettura di tipo distribuito.

Vi sono svariati framework (4) per lo sviluppo di applicazioni RIA Open Source (5) e commerciali, i quali utilizzano diverse tecnologie, linguaggi di programmazione e scripting. Alcuni dei principali linguaggi sono Java, Php, Asp .NET. Conosciuti framework di sviluppo sono Adobe Flex, .NET Framework, Google Web Toolkit, ICEfaces, RichFaces.
Portali nei quali si trovano le suddette tecnologie sono svariati, ad esempio Google utilizza in ampio modo Java e il suo toolkit Google Web Toolkit per creare interfacce grafiche. Microsoft nel suo portale e in Hotmail utilizza la tecnologia di cui è proprietaria .NET. Facebook invece utilizza principalmente PHP e framework sviluppati ad hoc.

1.3.2 Requisito prima scelta: Utilizzare un framework che consenta di scrivere un'applicazione RIA leggera, supportata da qualsiasi browser senza la necessità di installare plug-in di terze parti.

Tale requisito è soddisfatto utilizzando come tecnica di sviluppo AJAX, acronimo di Asynchronous JavaScript and XML [6]. Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è asincrono nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript.

[1] Facebook è un sito web di social network, di proprietà della Facebook, Inc., ad accesso gratuito. È nel 2010 il secondo sito più visitato del mondo dopo Google
[2] Twitter è un servizio gratuito di social network e microblogging che fornisce agli utenti una pagina personale aggiornabile tramite messaggi di testo
[3] MyVip è un social network ungherese gratuito
[4] Nella produzione del software, il framework è una struttura di supporto su cui un software può essere organizzato e progettato. Alla base di un framework c'è sempre una serie di librerie di codice utilizzabili con uno o più linguaggi di programmazione, spesso corredate da una serie di strumenti di supporto allo sviluppo del software, come ad esempio un IDE, un debugger, o altri strumenti ideati per aumentare la velocità di sviluppo del prodotto finito.
[5] In informatica, open source (termine inglese che significa sorgente aperto) indica un software i cui autori (più precisamente i detentori dei diritti) ne permettono, anzi ne favoriscono il libero studio e l'apporto di modifiche da parte di altri programmatori indipendenti. Questo è realizzato mediante l'applicazione di apposite licenze d'uso.

Tuttavia, e a dispetto del nome, l'uso di JavaScript[6] e di XML [7] [7] non è obbligatorio, come non è necessario che le richieste di caricamento debbano essere necessariamente asincrone. AJAX è una tecnica multi-piattaforma utilizzabile su molti sistemi operativi, architetture informatiche e browser web, ed esistono numerose implementazioni open source di librerie e framework.
Sono stati scartati perciò, come possibili framework di sviluppo, tutti i framework che richiedono l'installazione di plug-in nel broswer, molto spesso onerosi in termini di risorse e non universalmente compatibili.
Framework di questo tipo sono Adobe Flex che richiede il plug-in Flash Player, .NET framework con l'ambiente di runtime Silverlight, e JavaFX le quali applicazioni sono eseguite sotto forma di Applet.

La scelta è quindi ricaduta su uno dei framework AJAX attualmente esistenti. Alcuni di questi sono:

- ICEfaces [8]: un toolkit per Java
- AA: un toolkit Ajax per PHP
- Sajax: un semplice toolkit Ajax per PHP
- Xajax: un toolkit in PHP
- Google Web toolkit: un toolkit Ajax per Java sviluppato da Google
- Ajax ASP .NET: Ajax per il framework di Microsoft .NET.

1.3.3 Seconda scelta: Applicazione di livello Enteprise

Tale applicazione oltre al requisito di presentare un'interfaccia grafica ricca deve essere di classe enterprise, cioè progettata per i requisiti delle imprese.
Questa classe di applicazioni deve soddisfare importanti e stringenti requisiti in modo tale che l'applicazione risultante sia efficiente dal punto di vista della gestione, aggiornamento, amministrazione, manutenzione, leggerezza di esecuzione e fruibilità. Infatti un'applicazione enterprise è presente in un server remoto accessibile tramite rete, il che significa che qualsiasi pc client dotato di web broswer possa accederci. Se tale server è localizzato in un Intranet (rete aziendale locale), allora tutti i pc di tale Intranet potranno accedere alle risorse del server, mentre se è connesso ad Internet è possibile accedere a tale server da qualsiasi PC nel mondo connesso ad Internet. Ciò rende possibile de-localizzare il lavoro accedendo al gestionale in qualunque momento e in qualunque luogo, non solo all'interno del centro benessere. Avere l'applicativo residente in un server significa anche liberare i pc locali dalla necessità di installare in essi un software che necessiterà poi di manutenzione e aggiornamenti continui. Infatti l'applicativo, essendo residente in un solo computer, il server, avrà bisogno di esser gestito e aggiornato solo in esso. Cosa molto auspicabile, in quanto è una procedura di solito effettuata da tecnici, programmatori ed amministratori. Ciò libera l'utente finale da tutte le problematiche che possono sorgere dall'installazione in locale di software (corruzione di files dell'applicativo, aggiornamenti ecc).
Inoltre la maggior parte del carico di lavoro richiesto dal software è assorbito dal server, mentre ai pc client è richiesta solo la visualizzazione dell'interfaccia grafica dell'applicazione web mediante broswer. Di solito si usa chiamare tali client "Thin client", cioè client leggeri, in quanto la richiesta di risorse per l'applicazione è minima. Questo significa che qualsiasi PC, che non dispone di grandi risorse hardware, può visualizzare l'applicazione. Unico requisito è che sia in grado di eseguire fluidamente un browser di recente generazione. Il fatto che l'applicazione sia residente in remoto su uno o più application server [9]([8])(vedere l'introduzione agli application server) apporta molti altri vantaggi. E' infatti possibile il clustering [10][(9)]dell'applicazione su più servers in modo tale che se uno di essi va in guasto, l'applicazione sia accessibile dagli altri computer nel cluster, ciò significa avere un'applicazione ad alta disponibilità. Inoltre permette il bilanciamento del carico tra i calcolatori nel cluster, in modo che all'aumentare delle richieste da parte dei client, non vi sia un degrado delle prestazioni, bilanciando la loro elaborazione tra i diversi server.
Come aspetto negativo da tenere in considerazione vi è il fatto che se l'utente ha fuori uso l'accesso ad internet non potrà usare l'applicativo. Ma tale aspetto è considerato di minor importanza rispetto ai vantaggi che una web application fornisce, in quanto è evento raro ed eccezionale il disservizio di una connessione ad internet quale l'ADSL.

Ampia è anche la scelta di piattaforme/server Enterprise, che differiscono per le caratteristiche fornite e il linguaggio di programmazione adottato.
Alcune di esse sono:

- Java EE [11] di Sun Microsystem : Piattaforma per lo sviluppo di applicazioni Enterprise
- .NET di Microsoft [12]: Piattaforma enterprise, supporta di base i linguaggi della piattaforma .NET come C# e J#.

---

[6] JavaScript è un linguaggio di scripting orientato agli oggetti comunemente usato nei siti web.
[7] XML (acronimo di eXtensible Markup Language) è un metalinguaggio di markup, ovvero un linguaggio marcatore che definisce un meccanismo sintattico che consente di estendere o controllare il significato di altri linguaggi marcatori.
[8] Un application server è un software che fornisce un'infrastruttura per lo sviluppo di applicazioni aziendali.
[9] Un **computer cluster**, o più semplicemente un **cluster** (dall'inglese *grappolo*), è un insieme di computer connessi tramite una rete telematica. Lo scopo di un cluster è quello di distribuire una elaborazione molto complessa tra i vari computer componenti il cluster.

- Adobe Flex[13]: piattaforma per lo sviluppo di applicazioni enterprise basate su Adobe Flash.Supporta Java EE e JSP per lo sviluppo.
- Zend Server[14]: utilizza il linguaggio PHP fornendo le espansioni per scrivere applicazioni di classe enterprise.
- Zope [15]: un application server per il linguaggio Python

1.3.4 Requisito seconda scelta: utilizzare una piattaforma Enterprise Open Source per ridurre i costi che sia ben definita, supportata, aggiornata e che utilizzi un linguaggio di programmazione già conosciuto per lo sviluppo.

Per quanto riguarda il requisito Open Source la scelta è ricaduta tra la piattaforma Java EE ed i vari server che forniscono estensioni enterprise al linguaggio PHP. Nella realtà gli Application Server più diffusi ed utilizzati sono basati sulla piattaforma Java EE. Questo perché Java EE è una specifica formale per la costruzione di applicazioni enterprise ed application server, appositamente studiata a tale scopo. I framework PHP invece non sono dotati di alcuna specifica formale ed inoltre, in genere, offrono caratteristiche molto inferiori alle loro controparti Java. Comunque sia, in generale, tutte le offerte di application server non-Java non hanno alcuna specifica di interoperabilità formale alla parti con JSR [16], i documenti di standardizzazione delle specifiche Java. Come risultato, l'interoperabilità con i prodotti non Java è molto povera e difficile comparata con i prodotti basati su Java EE.
Inoltre esistono numerosi server enterprise sviluppati da vari produttori e community Open Source, il che consente un'ampia scelta e garantisce una buona documentazione a supporto della tecnologia. Unico svantaggio rispetto ad application server PHP è la più lenta velocità di esecuzione dovuta al fatto che Java è un linguaggio interpretato, mentre PHP è un linguaggio di scripting pre-processato che garantisce velocità molto più elevata.
Nella Figura 1-1: volumi di ricerca sull'argomento in Google [], tratta da Google Trends, è visualizzata una comparazione dell'interesse globale che vi è nei linguaggi Java, PHP, .NET, Pyhton e Ruby. Si nota che il linguaggio Java è il più diffuso.
Si è scelto di sviluppare l'applicativo in Java EE per tali motivi e per il fatto che è il linguaggio di programmazione maggiormente conosciuto in azienda.
Dato l'utilizzo di Java si è scelto, dopo attenta analisi, come framework AJAX ICEfaces. Le motivazioni dettagliate che hanno portato al suo utilizzo sono illustrate nel capitolo 2.

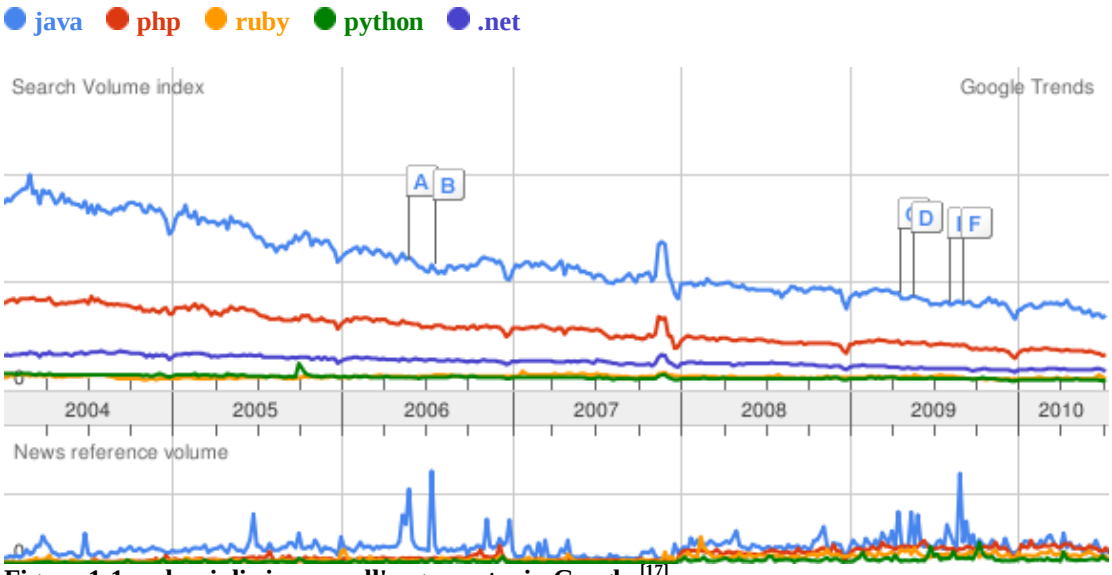


Figura 1-1: volumi di ricerca sull'argomento in Google [17].

1.4 Soddisfare le specifiche

Le specifiche dell'applicativo e devono essere soddisfatte mediante una progettazione e sviluppo attento dello stesso. In particolare è previsto che richiederà un elevato impegno progettuale il requisito 1.2.1, in quando si deve creare dal nulla un framework specifico per lo scopo. Il requisito 1.2.2 è rispettato mediante un'attenta progettazione della base di dati in modo tale che essa sia valida generalmente per i centri di cura del corpo e mediante lo sviluppo di regole di business e di interfaccia grafica che sia adattabile a tutte le realtà in considerazione. L' ultimo requisito 1.2.3 è soddisfatto mediante la creazione di una sezione dell'applicativo accessibile dai clienti dal web.

1.5 Organizzazione del testo

Nella trattazione si procedono ad introdurre, nel Capitolo 2, tutte le tecnologie utilizzate per lo sviluppo dell'applicativo. Ciò tornerà utile durante la descrizione dell'applicativo vero e proprio per capire le metodologie di sviluppo e quali toolkit e framework sono stati utilizzati, e in che modo, nei vari moduli del software.
Nel terzo capitolo è eseguita un' analisi ed implementazione dettagliata della base di dati che rappresenta la realtà del centro estetico(e tutti gli altri tipi di centro). Si procede ad introdurre per prima cosa la base di dati perché tramite la sua analisi si apprende meglio la realtà analizzata. Essa comprende le entità in gioco, le relazioni e vincoli tra esse ed in generale tutti gli aspetti che riguardano un centro di cura del corpo. Per la progettazione della base di dati è stata utilizzata la metodologia classica che divide le fasi in analisi dei requisiti, progettazione concettuale, progettazione logica e progettazione fisica.
Nel capitolo 4 si procede alla descrizione ad altro livello dell'applicativo, illustrandone la struttura. Si analizzano le specifiche ed è illustrato con quali idee e tecniche, successivamente implementate, esse sono state soddisfatte.
Nel capitolo 5 si descrivono i moduli specifici dell'applicativo, approfondendo ciò che è stato introdotto nel capitolo 4.

CAPITOLO 2

2 Tecnologie utilizzate

2.1 Java EE

Java EE è l'acronimo di 'Java Enterprise Edition' cioè la versione enterprise della piattaforma Java. E' costituita da un insieme di specifiche (JSR[18]) che definiscono le caratteristiche, interfacce e strati di un insieme di tecnologie ideate per creare applicazioni di tipo enterprise. Tali specifiche sono aperte e chiunque può fornire la propria implementazione, anche se l'implementazione di riferimento è fornita da Sun.
E' una piattaforma ampiamente utilizzata per la programmazione lato server. Essa stessa si basa sulla piattaforma Java SE ma differisce in quanto aggiunge librerie e funzionalità per creare software distribuito, multi-livello, efficiente e tollerante ai guasti basato largamente su componenti modulari eseguiti su un server chiamato application server.

Java EE dispone di molte API quali: JDBC, e-mail, JMS, RMI, web services, XML e definisce come queste interagiscono tra loro. Fornisce anche specifiche per i suoi componenti che includono EJB, Connector, servlets, portlets, JSP e altre tecnologie web. Queste consentono la creazione di applicazioni enterprise portabili e scalabili.
L'application server che implementa le specifiche Java EE alleggerisce il lavoro dei programmatori occupandosi di molti aspetti trasversali e comuni alle applicazioni enterprise. Infatti gestisce transazioni, sicurezza, scalabilità, concorrenza e la gestione dei componenti che sono pubblicati in esso in modo che i programmatori si possano concentrare in maggior modo sulla logica di business dei componenti piuttosto che sull'infrastruttura e integrazione dei compiti.

La specifica Java EE include molte tecnologie che estendono le funzionalità di base della piattaforma Java. La specifica descrive i seguenti componenti:

- Gli Enterprise JavaBeans definiscono un sistema a componenti distribuito che rappresenta il cuore della specifica Java EE. Tale sistema, infatti, fornisce le tipiche caratteristiche richieste dalle applicazioni *enterprise*, come scalabilità, sicurezza, persistenza dei dati e altro. Essi gestiscono principalmente le sessioni (con e senza stato) dei clienti fornendo i loro servizi sotto forma di metodi Java controllati di oggetti controllati dal container [19].
- Il JNDI [20] definisce un sistema per identificare e elencare risorse generiche, come componenti software o sorgenti di dati.
- Il JDBC è un'interfaccia per l'accesso a qualsiasi tipo di basi di dati. (Compresa anche nella standard edition).
- Il JTA [21] è un sistema per il supporto delle transazioni distribuite.
- Il JAXP è un API per la gestione di file in formato XML.
- Il Java Message Service (JMS) [22] descrive un sistema per l'invio e la gestione di messaggi.
- JSP: è un framework per lo sviluppo di applicazioni web.

L'ultima versione Java EE, la sesta, introduce inoltre i seguenti principalmente i seguenti componenti [23] [24]:

- Managed Beans: bean gestiti in generale dal container senza dover essere necessariamente EJB o Managed Bean della tecnologia JSF.
- Contexts and Dependency Injection: un meccanismo universale per la gestione del contesto di un bean e per l'inserimento di risorse tra i vari beans
- Bean Validation: un meccanismo universale per la validazione dei campi di un bean che attraversa tutti gli strati di un'applicazione enterprise.
- Nuove caratteristiche per JSF: nuovo modello di vista basato su Facelets.

Per documentarsi in maniera dettagliata sulla versione enterprise di Java si consiglia di visitare il sito web di Sun http://java.sun.com/javaee/, ed in particolar modo la lettura del tutorial ufficiale per Java EE 5 e 6 scaricabile alla pagina http://java.sun.com/javaee/reference/tutorials/, di fatto due veri e propri libri gratuiti sull'argomento.
Inoltre è suggerita la lettura del libro *'EJB 3 in action', Debu Panda, Reza Rahman, Derek Lane, (2007). Edizioni Manning*. per documentarsi in modo esaustivo sulla tecnologia EJB.

2.2 Application Server Java EE

Un **application server** è un software progettato per fornire tutta l'infrastruttura di supporto per lo sviluppo ed esecuzione di applicazioni e componenti server [25]. Esso fornisce servizi per la realizzazione di applicazioni enterprise costruite su più livelli. Tali servizi includono la gestione delle connessioni con le basi di dati, gestione delle transazioni tra più risorse (transazioni estese), servizi di e-mail, di sicurezza quali l'autenticazione e autorizzazione degli utenti e tanti altri. In generale tali applicazioni sono orientate al mondo del web.
In generale le applicazioni sviluppate mediante l'ausilio dell' application server sono costruite seguendo l'architettura a tre livelli nella quale un computer client (chiamato thin client) che non incorpora nessuna logica, ma solo elementi di interfaccia grafica, si connette all'application server che implementa la logica applicativa (business logic) la quale a sua volta transitivamente comunica con lo strato di persistenza (database).

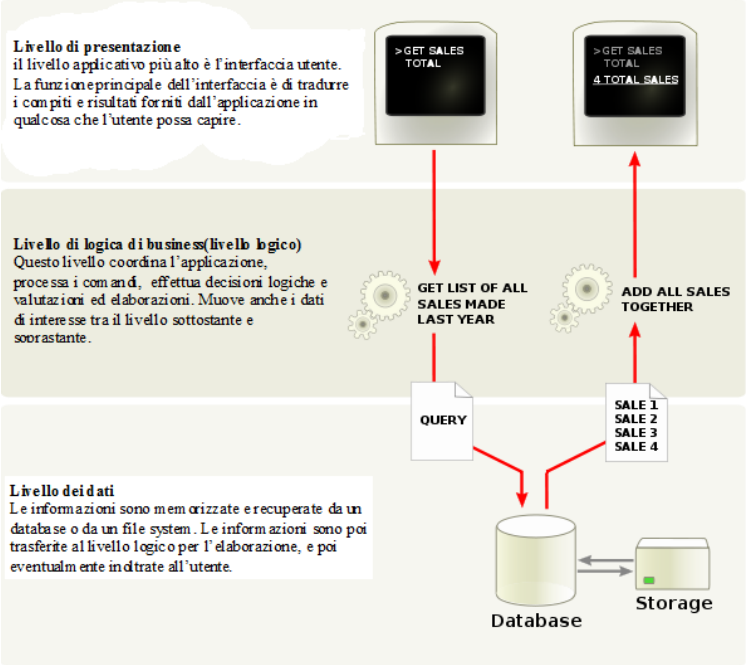


Figura 2-1: Architettura a tre livelli di un' applicazione enterprise

L'application server dispone anche di moduli per distribuire le applicazioni mediante clustering, moduli per il bilanciamento del carico e per massimizzare le prestazioni regolando i parametri del server.

Java EE dispone di svariati application server conformi alle specifiche, a seconda della versione di riferimento.
Alcune delle implementazioni correntemente certificate per la versione 5 sono:

- JBoss [26] , una implementazione rilasciata sotto licenza GPL, sviluppata da RedHat
- GlassFish [27], una implementazione *open source* di Sun
- Geronimo, una implementazione open source realizzata dalla fondazione Apache
- NetWeaver, realizzato da SAP
- JEUS 6, un *application server* specifico per Linux realizzato da TmaxSoft

Mentre le versioni correntemente certificate per la versione 6(le cui specifiche sono state rilasciate ufficialmente nel dicembre 2009) sono:

- GlassFish Server Open Source Edition 3.0.x
- Oracle GlassFish Server 3, based sull'application server GlassFish Open Source
- JEUS 7, un application server sviluppato TmaxSoft. Secondo il loro sito web "è stato pianificato che JEUS sarà rilasciato alla fine del 2010."

Tutte le implementazioni sono conformi alle specifiche, quindi un qualsiasi applicativo sviluppato per essere eseguito sotto l'ambiente Java EE deve poter essere fatto girare su un qualsiasi server.
Ogni application server offre caratteristiche diverse per quanto riguarda la gestione delle applicazioni, la sua amministrazione, le prestazioni offerte e moduli aggiuntivi per personalizzare le applicazioni.
Ogni sviluppatore infatti è libero di offrire caratteristiche aggiuntive per il suo application server che potenziano i già ricchi moduli di Java EE 6. Per esempio JBoss [28] tramite il suo Hibernate [29] offre una gestione più sofisticata e dettagliata dello strato di persistenza di un'applicazione con l'aggiunta di annotazioni proprietarie.
Il problema con le caratteristiche aggiuntive proprietarie è che legano strettamente l'applicazione che si sta sviluppando ad uno specifico application server. Qualora si volesse successivamente cambiare application server sarebbe necessario un grande sforzo e perdita di tempo per riscrivere le parti di codice legate alla tecnologia proprietaria, sempre sia possibile effettuare la traduzione.
Il consiglio è di non usare mai moduli e librerie specifiche di un application server per non incorrere nei problemi visti.
Utilizzare sempre e solo le librerie e moduli che fanno parte delle specifiche ufficiali Java EE.
Con tali librerie è possibile sviluppare senza alcun problema tutta la logica applicativa necessaria, magari dovendosi concentrare di più sullo sviluppo di determinate parti che un modulo aggiuntivo di un application server consentirebbe di scrivere più velocemente, ma col vantaggio che tali porzioni di codice risulteranno universali.

Si discuterà brevemente l'application server GlassFish, in quanto è stato scelto come application server di riferimento per lo sviluppo dell'applicazione AddCenter.

2.3 GlassFish

GlassFish, come sopra affermato, è un application server Open Source sviluppato da Sun Microsystems e dalla comunità GlassFish per la piattaforma Java EE. Esso è l'implementazione di riferimento per tale piattaforma, cioè per ogni versione Java EE la pubblicazione delle sue specifiche è seguita dalla pubblicazione dell'implementazione che le soddisfa [30]. La versione proprietaria di GlassFish è chiamata Oracle GlassFish Enterprise Server.
GlassFish è un software libero, rilasciato sotto doppia licenza di software libero: la licenza di sviluppo e distribuzione comune (CDDL) e la licenza pubblica GNU (GPL) con l'eccezione dei link(classpath exception).Tale eccezione permette al software sotto GPL di essere incluso o di includere (in linguaggio tecnico, *"collegato a"*) altro software avente licenza non compatibile con la GPL.
GlassFish utilizza come servlet container per il contenuto web un derivato di Apache Tomcat [31], con l'aggiunta di un componente chiamato Grizzly, che utilizza le nuove librerie Java NIO per la gestione dei Threads, che garantiscono scalabilità e velocità.
Utilizza come libreria di persistenza una modifica del sistema di persistenza TopLink, rilasciato da Sun e Oracle. (la nuova versione 3.x di GlassFish utilizza EclipseLink[32] , basato su TopLink).
Correntemente vi sono due versioni di GlassFish la 2 , conforme allo standard Java EE 5, e la 3, conforme allo standard Java EE 6.
GlassFish possiede molte caratteristiche che lo rendono idoneo all'uso aziendale in piccole e grandi aziende. Le più importanti sono le seguenti:

- Scalabilità ed alta disponibilità: se l'application server deve gestire applicazioni critiche per business, deve assicurare che tale applicazione sia altamente disponibile. Esso deve essere scalabile per fare fronte all'aumento del carico di lavoro. GlassFish assicura alta disponibilità e scalabile tramite il clustering(scalabilità) e la teconologia di database ad alta disponibilità (chiamata HADB: High-Availability Database [33])
- Interoperabilità col framework .NET [1] di Microsoft: L'interoperabilità è un requisito molto importante in un'impresa. Questo perché in essa le risorse sono distribuite in una vasta gamma di ambienti operativi. Per esempio può esserci un software per gestire il magazzino, un altro per l'amministrazione, uno per il settore commerciale e un altro per la gestione di alto livello dell'azienda (Business Intelligence). Un altro esempio è che la parte client dell'applicazione sia in un ambiente, per esempio Java EE, e i servizi web necessari siano in un altro ambiente, per esempio il framework .NET di Microsoft. GlassFish abilita le applicazioni basate su servizi web ad interoperare tra Java EE e .NET.
- Gestione molto efficiente dei servizi di messaggistica asincrona: un servizio di messaggistica efficiente è molto importante per connettere il software di business per formare un'impresa efficiente. GlassFish fornisce Open MQ (Open Message Queue) [34], una completa implementazione servizio di messaggistica Java (JMS) per l'integrazione di sistemi basati sulla messaggistica. JMS è un sistema di messaggistica che permette le applicazioni Java EE di creare, inviare, ricevere e leggere messaggi in modo asincrono.

[1] La suite di prodotti **.NET** è un progetto all'interno del quale Microsoft ha creato una piattaforma di sviluppo software, .NET, la quale è una tecnologia di programmazione ad oggetti.

- Amministrazione centralizzata: permette di gestire un intero cluster di istanze dell'application server mediante una console di amministrazione unica.

La scelta di utilizzare GlassFish è stata determinata da vari fattori. Il primo è dato dalla sua relativa facilità di apprendimento, grazie alla sua semplicità concettuale ed ampia documentazione presente (sono presenti guide per l'installazione, amministrazione, pubblicazione delle applicazioni e per lo sviluppatore e molte altre). Il secondo fattore è dato dalla sua facilità di gestione mediante una comoda interfaccia a linea di comando(chiamata asadmin) o grafica (console di amministrazione web). Alcuni application server infatti permettono la configurazione tramite file XML e solo la loro versione a pagamento consente altri strumenti di configurazione. Terzo fattore è dovuto al fatto che GlassFish costituisce l'implementazione di riferimento per la piattaforma Java EE, continuamente mantenuta ed aggiornata da Sun Microsystem e dalla community di sviluppatori. Ultimo fattore è dovuto all'alto livello di prestazioni e caratteristiche di cui dispone.

Per lo sviluppo dell'applicativo in particolare è stata scelta la versione 2 di GlassFish, principalmente perché al momento della progettazione di esso la versione 3 era ancora in fase di beta test. E' stato comunque tenuta in considerazione la compatibilità con la versione successiva, scrivendo codice compatibile completamente con entrambi i server.

Per documentarsi su GlassFish si rimanda al sito web ufficiale https://glassfish.dev.java.net/ dove si possono trovare le guide introduttive, tutti i download dell'application server nelle varie versioni e la documentazione specifica di ogni genere, dalle guide, ai manuali di riferimento fino ai tutorial.

2.4 Enterprise Java Beans

Gli **Enterprise JavaBean** (**EJB**) sono i componenti che implementano, lato server, la logica di business all'interno dell'architettura Java EE [35]. Le specifiche per gli EJB definiscono diverse proprietà che questi devono rispettare, tra cui la persistenza(Entity Beans), il supporto alle transazioni, la gestione della concorrenza e della sicurezza(Session Beans) e l'integrazione con altre tecnologie, come JMS[(2)], JNDI[(3)], e CORBA[(4)]. Tali specifiche sono progettate per fornire una metodologia comune e ben progettata per lo sviluppo di applicazioni enterprise. Gli enterprise beans forniscono una soluzione efficiente alle varie problematiche che si possono incontrare nello sviluppo di applicazioni di questo tipo.

Gli EJB sono ampiamente utilizzati nell'applicazione AddCenter per lo strato di persistenza e di logica di business.
Per documentarsi riguardo tale tecnologia si rimanda al tutorial ufficiale Java EE, nonché alla pagina ufficiale di Java EE come specificato nella sezione riguardante Java EE.

2.5 JSF

JavaServer Faces (JSF) è una tecnologia Java Enterprise basata sul design pattern architetturale Model-View-Controller (MVC) e descritta da un documento di specifiche (vedere a riguardo il documento di specifiche Java JSR 127) alla cui stesura hanno partecipato aziende quali IBM, Oracle, Siemens e Sun Microsystems. Il suo scopo è di semplificare lo sviluppo dell' interfaccia utente (UI) di una applicazione Web; può quindi essere considerata un framework per componenti lato server di interfaccia utente [36].
Come detto è un framework web MVC guidato dalle richieste utente basato sul modello di interfaccia grafica guidato dai componenti. Con ciò si intende che si sviluppano le interfacce mediante componenti di varia natura (pulsanti, campi di testo, link , menù e altro) legati tra loro in vario modo formando una struttura ad albero e il comportamento di questi è gestito mediante il pattern MVC. La struttura ad albero dei componenti forma la *vista* di tale pattern.
La vista è specificata mediante file XML (in termini pratici di solito i file sono basati sullo standard XHTML) chiamati modelli di vista o viste Facelets (a seconda della tecnologia utilizzata all'interno del file per specificare la vista).
Le richieste sono elaborate dal FacesServlet, che carica l'appropriato modello di vista, costruisce l'albero dei componenti, processa gli eventi e interpreta la risposta per il client, tipicamente in HTML. Lo stato dei componenti è salvato alla fine di ogni richiesta e ripristinato alla successiva creazione della vista.
JSF 1.x puro (senza l'ausilio di frame work quali ICEfaces) usa la tecnologia JSP (Java Server Pages) come tecnologia di default per la visualizzazione. JSP è una tecnologia molto usata per la costruzione di applicazioni web dinamiche, ma è da ritenersi, a parere dell'autore, molto obsoleta per la costruzione di siti web dinamici. Infatti lo sforzo per sviluppare applicazioni web ricche e altamente dinamiche in JSP è molto elevato ed inefficiente in quanto legato al vecchio

[2] JMS: **Java Message Service** (o **JMS**) è l'insieme di API, appartenente a Java EE, che consente ad applicazioni Java presenti in una rete di scambiarsi messaggi tra loro.
[3] The **Java Naming and Directory Interface** (**JNDI**) è l'API Java per un servizio di directory(come LDAP) che permette i client Java di scoprire e risolvere dati e oggetti tramite nomi.
[4] **CORBA** (**Common Object Request Broker Architecture**) è uno standard sviluppato da OMG per permettere la comunicazione fra componenti indipendentemente dalla loro distribuzione sui diversi nodi della rete o dal linguaggio di programmazione con cui siano stati sviluppati.

modello di sviluppo di applicazioni web. Inoltre tale tecnologia è molto lenta in quanto richiede la traduzione ed elaborazione delle pagine web create con essa in Servlet e l'output di ciascun componente mediante istruzioni di stampa del Servlet. Per ottenere maggiori informazioni riguardo a JSP e al suo funzionamento si rimanda a documentazione specifica.
JSF 2.0 [37] è l'ultima versione attualmente disponibile, uscita molto recentemente rispetto alla data di tale trattazione. Esso utilizza come tecnologia di visualizzazione Facelets (5), presente in JSF 1.x solo come framework di templating aggiuntivo.
Esso fornisce un più efficiente, semplice e potente linguaggio di descrizione delle viste.

2.5.1 Vantaggi di JSF

- Controlli GUI personalizzati: fornisce un insieme di API e associa tags personalizzati per creare form HTML che hanno interfacce complesse.
- Gestione degli eventi: rende facile progettare codice Java che viene invocato quando i forms sono inviati. Il codice può essere invocato da pulsanti, cambiamenti in particolari valori (ad esempio l'utente modifica il valore di un campo)
- Managed Beans: JSF estende le caratteristiche dei managed beans di JSP, semplificando di molto l'elaborazione dei parametri.
- Expression Language: JSF fornisce un potente e conciso linguaggio per accedere alle proprietà dei beans e agli elementi delle collezioni di oggetti.
- Validazione e conversione dei form: JSF ha la capacità di controllare che i valori dei form siano nel corretto formato e convertire le stringhe in altri tipi di dato. Se un qualsiasi valore manca o è nel formato errato, il form può essere automaticamente riviusalizzato con messaggi di errore e con i valori precedentemente inseriti mantenuti.
- Configurazione basata su file centralizzato: molti valori JSF sono rappresentati in XML o file di proprietà, piuttosto che codificarli nei programmi. Questo accoppiamento leggero significa che molte delle modifiche possono essere fatte senza modificare o ricompilare il codice Java e che tutte le modifiche di alto livello possono essere fatte modificando un singolo documento. Questo approccio consente agli sviluppatori di concentrarsi sui loro compiti specifici senza la necessità di conoscere tutto il layout del sistema.
- Approccio coerente: JSF incoraggia l'utilizzo del MVC per l'applicazione.
- Supporto per altre tecnologie di visualizzazione: JSF non è limitato ad HTML e HTTP, permette infatti diverse traduzioni delle viste a seconda del client utilizzato.

Per documentarsi su JSF si consigliano in particolar modo i libri della serie "Core servlet" (Marty Hall and Larry Brown) e il loro sito *http://www.coreservlets.com/*, e i siti ufficiali *http://java.sun.com/javaee/javaserverfaces/* e *https://javaserverfaces.dev.java.net/*.

2.6 Introduzione a ICEfaces

ICEFaces è un framework Open Source Java basato su AJAX che viene utilizzato per creare Rich Internet Application (RIA), cioè applicazioni web che possiedono le caratteristiche e le funzionalità delle applicazioni desktop, senza però necessitare dell'installazione sul disco fisso [38]. Infatti esse vengono visualizzate tramite l'ausilio di un broswer. Le applicazioni sono scritte in puro linguaggio Java.

ICEFaces sfrutta tutti i tools ed ambienti di esecuzione basati su standard JavaEE (Quali JSF e gli application server). Permette di sviluppare applicazioni RIA con numerose caratteristiche sviluppate in Java senza bisogno di applet o plugin proprietari da integrare nel Browser. Le applicazioni ICEFaces sono applicazioni JSF così che non ci sia bisogno dell'utilizzo di Javascript scritto lato utente , inoltre il meccanismo che sta alla base (Ajax) è completamente trasparente allo sviluppatore.

A livello di architettura ICEFaces utilizza ed espande la tecnologia JSF(Java Server Faces) [39] introducendo migliorie e nuovi componenti con caratteristiche avanzate.
JSF è basata sul design pattern MVC (Model-View-Controller) [40] che è molto efficiente e pratico per lo sviluppo di interfacce grafiche basate sugli eventi(programmazione orientata agli eventi).

[5] Facelets è un framework web open source sotto la licenza Apache. Facelets è nato per sostituire JSP nella creazione delle pagine e fornisce un potente meccanismo di templating e definizione di nuovi componenti grafici.

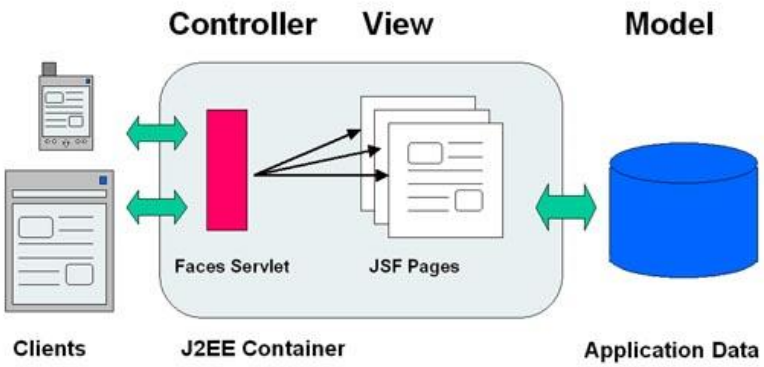


Figura 2-2: Implementazione JavaServer Faces del MVC [41]

Essa però soffre di tutte le limitazioni legate allo sviluppo dei siti web mediante semplice HTML e metodi HTTP per comunicare col server web. Quando infatti viene richiesta una risorsa un normale server web si limita ad inviarla per intero (ad esempio è richiesta una pagina web, ed essa viene spedita al broswer dell'utente) e quando si vogliono inviare dati al server è necessario farlo in un unico passo mediante un metodo (POST o PUT) del protocollo HTTP. Si immagini la compilazione di un modulo per la sottoscrizione ad un portale, è necessario compilare tutti i campi necessari e al termine inviare tutti i dati in esso al server. Non è possibile in alcun modo interagire col server web durante la compilazione degli stessi.
JSF, basandosi su questa tecnologia, permette di comunicare con l'application server solo al momento del subit di un form e inoltre soffre della necessità di dovere caricare sempre per intero le pagine visualizzate all'utente. Soffre perciò di una comunicazione inefficiente tra server e client(con relativo spreco di banda) e riduce il dinamismo delle pagine non potendo caricare parte di esse.
Nel tempo si è sviluppata la tecnologia nota col nome di AJAX [42], acronimo di *Asynchronous JavaScript and XML*. Essa è costituita da un insieme di tecnologie web unite tra loro (JavaScript, XML, HTML, XHTML, CSS, DOM) usate lato client per creare applicazioni web interattive. Con essa le applicazioni possono ricevere in modo asincrono dati dal server senza disturbare la visualizzazione e il comportamento delle pagine esistenti presenti nel client(eliminando di conseguenza il problema dell'interazione col server anche in momenti diversi dai submit e il problema del caricamento di frazioni di pagine). I dati sono recuperati dal server utilizzando l'oggetto XMLHttpRequest del linguaggio di scripting JavaScript, che fornisce un metodo per scambiare informazioni tra broswer e server senza la necessità di caricamenti di pagina completi.
Se si completa la tecnologia JSF con tutti i vantaggi e migliorie fornite da AJAX si possono ottenere applicazioni web dinamiche molto sofisticate. Questo è ciò che è effettuato da ICEFaces che fonde JSF e AJAX. Esso inoltre permette di scrivere applicazioni in puro linguaggio Java, accollandosi la responsabilità di convertire il codice scritto in AJAX.

ICEfaces rimpiazza i renderers JSF basati su HTML, che traducono i componenti dei documenti in tags HTML, con i renderers Direct-to-DOM (D2D), che traducono i componenti direttamente in oggetti DOM. Esso introduce un bridge Ajax leggero che riporta i cambiamenti alla presentazione al browser client e comunica gli eventi utente indietro all'applicazione JSF residente sul server. In aggiunta ICEfaces fornisce un vasto insieme di componenti che facilitano il rapido sviluppo di applicazioni RIA. Nel seguito si introduce l'architettura base di ICEfaces.
Ricapitolando con ICEfaces possiede principalmente le seguenti caratteristiche:

- Aggiornamenti di pagina incrementali che non richiedono il refresh completo della pagina per ottenere i cambiamenti nell'applicazione. Solo gli elementi della presentazione che sono cambiati sono aggiornati durante la fase di rendering.
- Durante l'aggiornamento delle pagine il contesto utente (l'insieme di tutti i dati dell'utente) è preservato. L'aggiornamento della presentazione non interferisce con l'interazione dell'utente con l'applicazione.
- Permette l'interazione con l'utente anche fuori dal normale ciclo di vita JSF. E' possibile aggiornare la presentazione in modo asincrono, senza la necessità che l'utente effettui un'iterazione. Inoltre è possibile controllare a un livello molto fine componente per componente utilizzando un meccanismo chiamato 'partial submit', che permette il submit al server solo del campo correntemente editato dall'utente.

ICEFaces è formato principalmente da tre elementi:

1. **Il Framework ICEFaces**
   E' un'estensione del framework standard JSF con la fondamentale differenza con cui viene trattata la fase di rendering. Diversamente da JSF il rendering avviene nel DOM lato server e solo cambiamenti parziali sono lasciati al browser ed in seguito assemblati con un bridge Ajax molto leggero, cioè solo i cambiamenti necessari nel DOM sono inviati dal server e riportati lato client. Il risultato è un render fluido,perché eseguito dal server, effettuato solo su certi elementi della pagina, quindi in modo incrementale. Ajax utilizza le Api inizializzate dal server(il carico di lavoro demandato al client è molto basso) ed integra il meccanismo similmente al ciclo di JSF.
2. **Il Bridge Ajax**
   Presenta elementi lato server e lato client che coordinano la comunicazione (basata su Ajax) fra il browser del client e l'applicazione lato server. Il Bridge si occupa di apportare i cambiamenti alla presentazione(pagina web del broswer) dalla fase di rendering(eseguita nel server) al browser del client e del riassemblamento di questi cambiamenti nel DOM del browser per applicare i cambiamenti. Inoltre ha il compito di rilevare le interazioni dell'utente con l'interfaccia grafica e di portare le azioni dell'utente all'applicazione presente lato server per essere processate dal ciclo di vita JSF. Un meccanismo chiamato *partial submit* è integrato nei componenti di ICEFaces e facilita la generazione di eventi attraverso il bridge. La prima volta che la pagina viene caricata viene creato il bridge Ajax e coordina gli aggiornamenti della presentazione e la trasmissione degli eventi dell'utente per tutto il ciclo di vita dell'applicazione.
3. **La Suite di componenti di ICEFaces**
   La suite di componenti fornisce tutti i componenti per la costruzione dell'interfaccia grafica dell'applicazione. Include sia i componenti standard JSF che una vasta gamma di componenti che consente allo sviluppatore di costruire applicazioni sofisticate e dall'interfaccia intuitiva. Oltre al meccanismo dell'interazione diretta con il DOM i componenti possono utilizzare un set di effetti come il drag and drop, tutto ciò con la semplice modifica di attributi così che lo sviluppatore si debba mai programmare a basso livello Javscript per ottenere caratteristiche dinamiche da un componente.

2.6.1 Architettura di ICEFaces

Per sviluppare applicazioni ICEFaces utilizzando tutti i benefici che esso apporta non è necessario conoscere la sua intera architettura, è sufficiente conoscere solo la struttura principale.
E' molto utile sapere in che modo rende possibile l'integrazione tra JSF e AJAX per permettere update incrementali delle pagine e la continua interazione tra server e client. La seguente trattazione è tratta e riassume la guida dello sviluppatore di ICEfaces [43].

La Figura 2-3: Architettura di ICEfaces mostra l'architettura base:

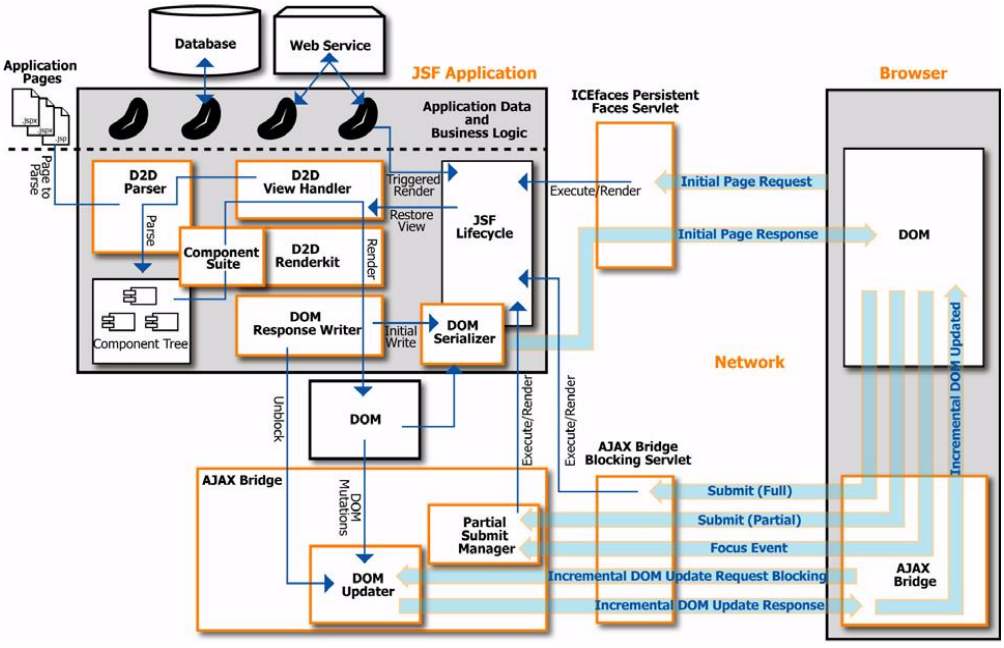


Figura 2-3: Architettura di ICEfaces

Gli elementi fondamentali che si notano sono:

- **Persistent Faces Servlet:** Gli URL con l'estensione ".iface" sono mappati col Persistent Faces Servlet. (Ma anche altre eventuali altre estensioni possono esser configurate in fase di mappatura col servlet). Quando una richiesta iniziale di pagina e' effettuata, tale servlet e' responsabile dell'esecuzione del ciclo di vita JSF associato con la richiesta.
- **Blocking Servlet:** responsabile della gestione di tutte richieste bloccanti e non bloccanti successive al rendering iniziale della pagina. E' responsabile dell'inoltro delle richieste al bridge AJAX quando necessario. Esso gestisce le seguenti richieste:
    - Full submit: viene eseguito il normale ciclo di vita JSF. E' l'unico caso in cui l'inoltro al bridge ajax non è necessario, in quanto viene effettuato il submit di una pagina completa.
    - Partial submit: Il submit viene inoltrato al bridge, necessario per un aggiornamento parziale della pagina dovuto al particolare componente associato col partial submit(quale ad esempio un campo geenrico o un pulsante)
    - Focus Event: eventi di focus generati dall'utente ,ad esempio selezione di un campo.
    - Incremental DOM update request blocking: richieste bloccanti di update incrementale della pagina. Il broswer dell'utente mediante DOM effettua periodicamente il polling dell'applicazione lato server per richiedere eventuali update della pagina mediante tali chiamate.
    - Incremental DOM update response: risposta lato server alla richiesta di aggiornamento.
    -
- **D2D ViewHandler:** responsabile della creazione dell'ambiente di rendering D2D, inclusa l'inizializzazione del D2D response writer.Invoca anche il parser per il parsing iniziale della pagina nel JSF components tree.
- **D2D Parser:** Crea il component tree dal documento JSP. Esso legge la pagina JSP richiesta ed effettua la sua traduzione per creare l'albero dei componenti associato con i componenti specifici di ICEFaces e JSF in generale.
- **D2D RenderKit:** Responsabile del rendering del components tree nel DOM tramite il DOM response writer.
- **DOM Response Writer:** Responsabile della scrittura dentro il DOM. Inizializza anche la serializzazione DOM per il primo rendering e sblocca il DOM Updater per gli update DOM incrementali.
- **DOM Serializer:** responsabile della serializzazione del DOM per la pagina di risposta iniziale (e' l'unica volta che viene iniviata l'intera pagina al broswer)
- **DOM Updater:** responsabile dell'assemblaggio dei cambiamenti DOM in un singolo aggiornamento DOM incrementale. Esso e' bloccato dal response writer fino a quando la fase di rendering non e' stata completata. Le richieste bloccanti di update DOM incrementale rimangono bloccate fino a quando non e' eseguita tale fase.
- **Partial submit manager:** gestisce la validazione parziale dei form, validando solo i campi su cui è attiva tale proprietà. Permette quindi un più alto livello di interazione col client.
- **Component Suite:** Fornisce un ampio insieme di componenti JSF arricchiti che forniscono le caratteristiche per l'AJAX bridge e i blocchi base di costruzione per le applicazioni ICEFaces.
- **Client-side Ajax Bridge:** Responsabile della generazione delle richieste di DOM update e dell'elaborazione delle risposte. E' anche responsabile della gestione del focus e del processo di submit.

2.6.2 Concetti chiave di ICEfaces

- Direct to DOM (D2D) rendering: ICEFaces esegue il rendering di un JSF component tree direttamente in una struttura dati DOM standard, a differenza di JSF che effettua il rendering basato su HTML. In Figura 2-4:Direct-to-DOM Rendering si può osservare come vengano tradotti i componenti nel DOM corrispondente. Vi è un componente PanelGrid che contiene un InputText e un PanelGrid che a sua volta contiene un CommandButton e un InputText.

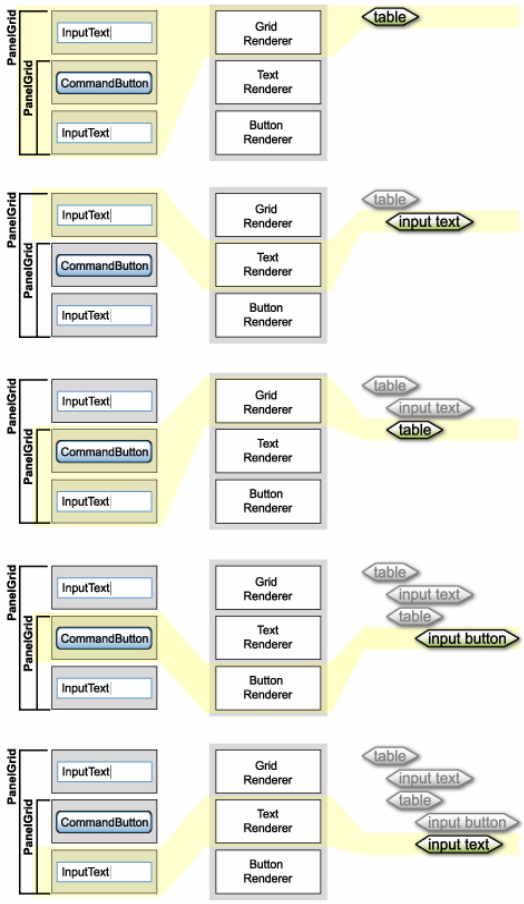


Figura 2-4:Direct-to-DOM Rendering

- Aggiornamenti di pagina locali e incrementali: è possibile effettuare l'update completo di una pagina o localizzato a un sottoinsieme dei suoi elementi.

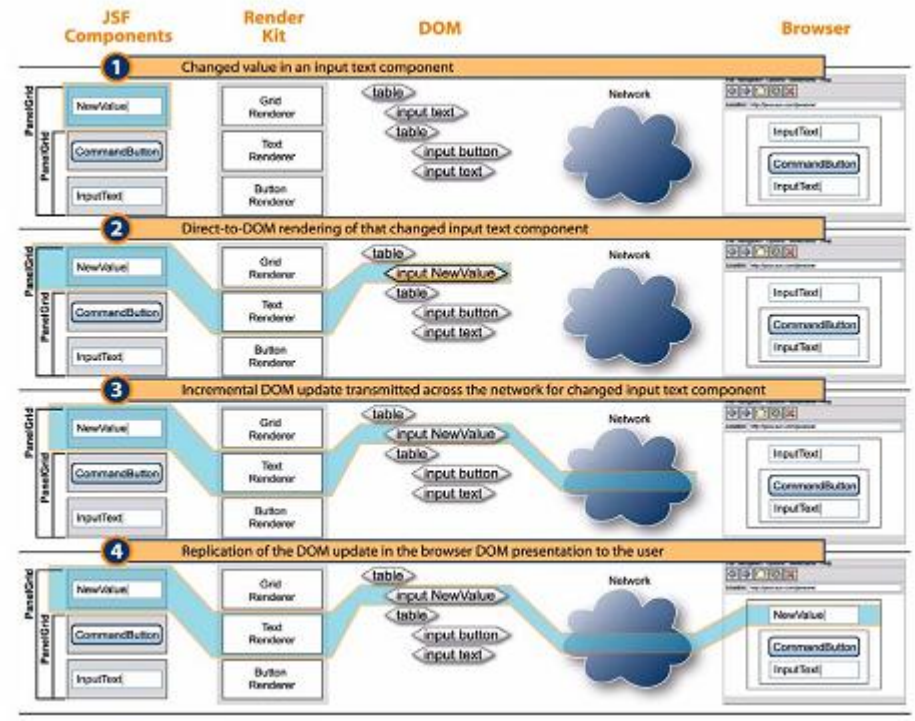


Figura 2-5: Incremental Update with Direct-to-DOM Rendering

- Aggiornamenti sincroni e asincroni:
  - Aggiornamento sincrono: Normalmente JSF aggiorna la presentazione come parte del ciclo standard di richiesta/risposta. L'aggiornamento e' quindi sempre dovuto a qualche evento generato dall'utente. Tale aggiornamento e' chiamato "aggiornamento sincrono". In Figura 2-6 è rappresentato il meccanismo che permette l'aggiornamento sincrono. Molto semplicemente ad ogni interazione dell'utente (submit) viene eseguito il ciclo di vita JSF che aggiorna il component tree e questo viene tradotto nel DOM. Infine i cambiamenti in questo sono inviati al broswer dell'utente che li riporta nella pagina associata.

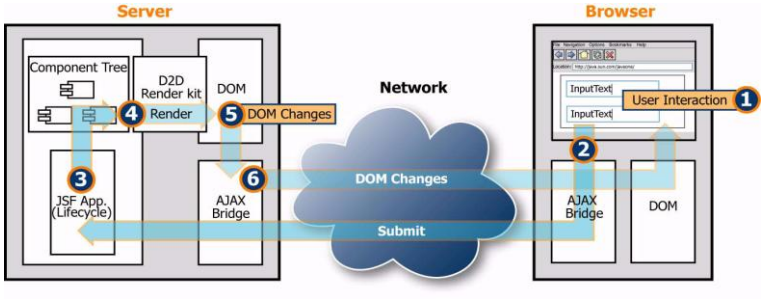


Figura 2-6: Aggiornamenti sincroni

  - Aggiornamento asincrono: un grave problema con l'aggiornamento sincrono e' che l'applicazione necessita di una richiesta generata dall'utente. Se un cambio dello stato dell'applicazione avviene durante il periodo di inattività dell'utente non v'è alcun modo di mostrare tali aggiornamenti all'utente. Ad esempio un'applicazione di e-commerce vuole informare l'utente sulla pagina che sta attualmente visualizzando di un oggetto da lui osservato in breve scadenza per l'acquisto. ICEFaces toglie questo limite introducendo gli aggiornamenti asincroni mediante l'AJAX bridge. In Figura 2-7 è rappresentato il meccanismo che consente gli aggiornamenti asincroni. Il client mediante polling invia una richiesta di aggiornamento asincrona(Asynchronous XMLHttelReuqest) al server il quale controlla se si sono verificati cambiamenti nello stato dell'applicazione. In caso affermativo apporta le modifiche necessarie al DOM. Tali modifiche,se presenti, sono inviate indietro al client mentre se non vi sono stati cambiamenti è inviata una modifica vuota. Esse sono inseguito processate per essere riportate nel DOM visualizzato dall'utente.

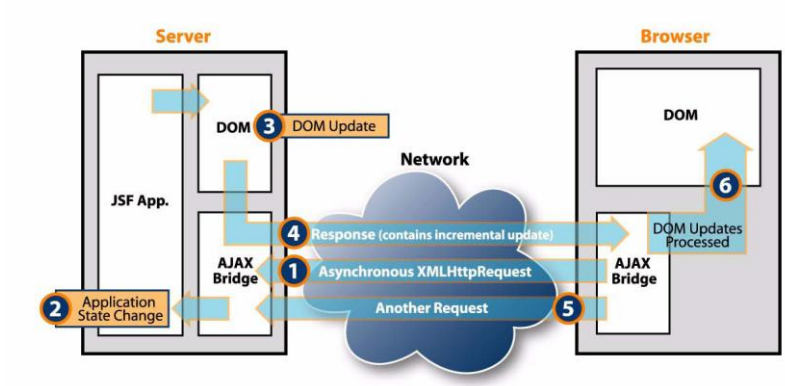


Figura 2-7: Aggiornamenti asincroni

- Gestione della connessione: la connessione tra client e server e' fondamentale perche' ICEFaces funzioni. Per questo provvede strumenti di controllo della connessione e monitoraggio nel lato client. Questi vanno dalla modifica dei parametri per ottimizzare la comunicazione col server quali l'intervallo di polling a component grafici che informano l'utente sullo stato della comunicazione.
- Rendering inizializzato lato server: aggiornamento della presentazione lato client inizializzato dal server in seguito a un evento dell'applicazione.
- Partial submit: validazione parziale dei form. Quando tale proprietà e' attiva in un campo, quando l'utente genera un evento su tale campo può esser invocato il submit solo su di esso.
- CSS: ai componenti ICEFaces si possono assegnare degli stili mediante CSS. Se a un componente non è assegnato alcun stile, ICEFaces assegna uno stile predefinito.
- Drag and drop: e' incluso il supporto per il drag and drop dei componenti del tutto simile al drang n' drop presente nelle applicazioni desktop.
- Effetti: ai componenti e' possibile dare effetti di animazione quale lampeggi e faders.

- Viste DOM concorrenti: per default ogni utente può avere una sola pagina aggiornata dinamicamente per ogni applicazione web. In questa configurazione un singolo DOM e' mantenuto per ogni sessione utente. L'apertura di una nuova pagina del broswer nella stessa applicazione può portare alla corruzione della pagina. Per consentire finestre multiple all'interno di una singola applicazione deve esser attivata la proprietà — "concurrent DOM views".

2.6.3 Perché la scelta di ICEfaces

ICEfaces è stato scelto per tutte le caratteristiche sopra illustrate, che lo rendono una tecnologia unica nel suo campo.
Solo tramite ICEfaces è possibile disporre di un framework AJAX basato su JSF che offre update di pagina incrementali, aggiornamenti asincroni, validazione parziale dei form, effetti grafici avanzati e componenti evoluti che supportano submit parziali, il tutto eseguito da un meccanismo molto leggero e veloce.
Inoltre,qualche mese dopo la pianificazione delle tecnologie da usare, ICEfaces è stata sponsorizzata da Sun Microsystem per diventare la tecnologia di rimpiazzo del progetto Woodstock, il progetto di Sun per sviluppare interfacce grafiche ricche utilizzando JSF ed AJAX.
Articolo sulla migrazione da Woodstock ad ICEfaces:
http://www.computerworld.com/s/article/9124022/Sun_s_Woodstock_Web_dev_effort_shifts_to_Icesoft
ICEfaces quindi è diventata la tecnologia ufficiale di Sun per lo sviluppo di applicazioni AJAX RIA in JSF e questo fatto è un motivo aggiuntivo per il suo utilizzo.

2.7 Introduzione a MySQL

MySQL [44] è un Relational database management system (*RDBMS*[6]). E' il più famoso database open source, e' dotato di elevata velocità, affidabilità e facilità d'uso. Esso è ampiamente utilizzato nel mondo del Web come RDBMS per i siti web, dalle compagnie di telecomunicazioni e all'interno delle aziende come database aziendale. Famose aziende che lo adottano sono Wikipedia, YouTube, Nokia e Google [45].
Esso è composto principalmente da un'interfaccia client a caratteri e da un server. MySql è disponibile per una svariata serie di sistemi operativi. Si può installare su sistemi basati su Unix e Windows. Esso è anche famoso per essere parte chiave dell'usatissima piattaforma LAMP [7](Linux, Apache, MySql, PHP), il più usato e in continua crescita stack enterprise Open Source.
MySql è stato scelto come DBMS del software da sviluppare per le sue potenti caratteristiche, facilità di gestione, ampio supporto e documentazione nonché per la sua universale diffusione. Inoltre MySql è gestito dalla stessa società (Sum Microsystem) che sviluppa e mantiene Java, la quale offre perciò una totale compatibilità con tale linguaggio.
Come riportato nel sito di MySql, il 57% degli sviluppatori usa Java per sviluppare applicazioni MySql, perciò l'integrazione tra Java e MySql viene considerata importante e continuamente evoluta e perfezionata.
MySql fornisce drivers per gli standard di connessione a database quali JDBC, ODBC, .NET, C++, C e altri [46]. In particolare per il linguaggio Java mette a disposizione Connector/J, il driver per JDBC(Java Database Connectivity), lo standard Java di connessione alle basi di dati.

MySql è dotato di un vasto numero di tools di supporto che ne facilitano la gestione e offrono supporto per lo sviluppo di database con esso. Alcuni dei più noti tools grafici di gestione sono 'MySql Administrator' sviluppato da MySql e 'phpMyAdmin' [47], un frontend web in PHP. Per la creazione e gestione dei database è molto diffuso il software open source 'MySQL Workbench' [8](che nella versione attuale include i tools grafici di gestione MySql). Esso permette lo sviluppo e modellazione di database, lo sviluppo di codice SQL e l'amministrazione dei database. Si ricorrerà al suo utilizzo per la progettazione della base di dati.

[6] Il termine Relational database management system (RDBMS) (sistema per la gestione di basi di dati relazionali) indica un database management system basato sul modello relazionale. Un Database Management System (abbreviato in DBMS) è un sistema software progettato per consentire la creazione e manipolazione efficiente di database (ovvero di collezioni di dati strutturati) solitamente da parte di più utenti.
[7] E' un'installazione di MySql e Apache HTTP Server in un sistema operativo Linux come ambiente di esecuzione di applicazioni web.
[8] Tutta la documentazione relativa a MySQL Workbench è disponbile nel sito di MySql

2.8 Introduzione a NetBeans

NetBeans [48] (9) è un ambiente di sviluppo multi-linguaggio Open Source scritto interamente in Java nato nel giugno 2000. È l'ambiente scelto dalla Sun Microsystems come IDE ufficiale per lo sviluppo in Java,esso ha come "concorrente" il più diffuso Eclipse. Possiede molti plug-in per lo sviluppo in Java ed altri linguaggi che lo rendono molto valido per lo sviluppo di applicazioni. Il uso principale problema è che richiede molte risorse hardware per essere eseguito perché utilizza le librerie grafiche standard Java (Swing), che sono molto pesanti essendo eseguite in una Virtual Machine(la JVM[10]). Esso è un applicativo completamente Open Source e scaricabile gratuitamente dal sito http://www.netbeans.org/ . NetBeans offre il massimo supporto per tutte le tecnologie Java direttamente sotto il controllo di Sun Microsystem, in quanto adottato da essa, come detto, come IDE ufficiale. Esso quindi supporta pienamente e nel modo più naturale possibile l'integrazione con GlassFish , MySql e ICEfaces. Quest' ultima è la tecnologia scelta da Sun per le sue applicazioni JSF AJAX, ed è stata quindi integrata completamente col suo IDE di riferimento.
Utilizzando interamente tecnologie Sun, o supportate da essa, la scelta è si focalizzata per l'utilizzo di NetBeans.
Inoltre, come sempre, è stata valutata la natura Open Source dell'IDE, il che abbatte i costi di sviluppo dell'applicativo, disponendo di un IDE gratuito.
NetBeans inoltre offre di base un supporto più ampio alla programmazione Java rispetto a quello fornito di base dalle controparti Open Source, quali Eclipse [49]. NetBeans dispone di molti moduli che per esempio in Eclipse sono aggiuntivi o disponibili solo commercialmente.
Infine è stato scelto NetBeans anche per una questione prettamente di interfaccia grafica, ritenuta più intuitiva e comoda, nonostante sia più lenta, rispetto a quella di applicativi simili come Eclipse.

[9] Fare riferimento al portale di NetBeans per scaricare l'applicativo, la documentazione associata e i moduli aggiuntivi.
[10] JVM: La macchina virtuale Java, detta anche Java Virtual Machine o JVM, è la macchina virtuale che esegue i programmi scritti in bytecode Java.

CAPITOLO 3

3 PROGETTAZIONE DELLA BASE DI DATI

3.1 ANALISI PRELIMINARE

Definizione strategica:

Realizzare una base di dati per la gestione informatizzata tramite software gestionale di un centro di cura del corpo.

Pianificazione generale:

E' necessario automatizzare e gestire tutte le principali operazioni che si effettuano in un centro benessere ed archiviare elettronicamente tutte le informazioni riguardanti l'esercizio dell'attività.
Bisogna gestire i clienti, gli operatori (massaggiatori ecc), gli appuntamenti , le ricevute , gli ambienti di lavoro e le carte fedeltà.
Particolare attenzione dovrà essere posta alle ricevute per garantire una gestione accurata degli aspetti economici dell'esercizio. Per legge inoltre è richiesto che ogni giorno sia riepilogato l'incasso della giornata, separando accuratamente l'importo di ogni ricevuta dall'imponibile (valore su cui è applicata la tassa IVA).

3.2 RACCOLTA ED ANALISI DEI REQUISITI

3.2.1 Acquisizione informale dei requisiti

3.2.1.1 Requisiti generali

Si vogliono organizzare i dati del centro benessere per automatizzare la gestione dei clienti e di tutta l'attività che si svolge all'interno del centro. In stretta relazione con la gestione dei clienti c'è la gestione dell'agenda degli appuntamenti e degli operatori del centro che ad ogni appuntamento seguono il cliente. Inoltre è necessario gestire gli ambienti del centro benessere, ad esempio la/le sala/e per i massaggi,il numero di lettini disponibili, le cabine solarium ed altro ancora.
E' anche necessario automatizzare il processo di stampa e gestione delle ricevute effettuate agli utenti.
Si devono memorizzare le informazioni relative alle carte fedeltà associate ad ogni cliente. Ogni carta fedeltà dà il diritto a sconti su particolari prodotti e pacchetti decisi in fase di acquisto della card.
Il database dovrà essere il livello di persistenza di un applicativo nel quale è richiesta la gestione delle utenze. In tale software gli utenti potranno accedere a diverse sezioni a seconda del loro livello di autenticazione. In particolare un segretario potrà accedere a tutta la base di dati per gestire l'attività aziendale nel suo complesso, mentre un cliente potrà autenticarsi per vedere i suoi appuntamenti , le ricevute ad esso emesse ed eventualmente il listino dei prodotti e pacchetti offerti dal centro benessere.

3.2.1.2 Raccolta della documentazione e modulistica esistente

Si è raccolta la documentazione esistente utile al fine della progettazione della base di dati.
In allegato vi è il listino servizi e il modulo registrazione cliente.

3.2.1.3 Requisiti specifici

E' necessario gestire l'anagrafica clienti per l' inserimento, modifica e recupero dei dati anagrafici, lo storico dei servizi e prodotti acquistati e dei servizi non ancora erogati o pagati.
Per ogni cliente bisogna memorizzare il nome, cognome, il nome da visualizzare nell'applicativo (screen name), eventuale e-mail, eventuale password se il cliente vuole accedere al sistema, data di nascita, luogo di nascita,sesso,codice fiscale , recapiti telefonici(fisso o mobile), lo stato civile e professione. Si è anche interessati a sapere eventuali relazioni di parentela tra i clienti.

Per ogni utente si vuole anche almeno un suo indirizzo. Ogni indirizzo è costituito da città , codice di avviamento postale, via , numero via e località. E' auspicabile mantenere un database con tutte le città italiane all'interno dell'applicativo per velocizzare le procedure.
Per l'accesso al sistema l'utente deve poter usare come nome utente lo screen-name o la mail.
Ad ogni cliente, all'atto dell'iscrizione, si pongono alcune domande a scopo statistico che è necessario raccogliere per una successiva analisi.

Vanno gestiti anche gli impiegati per i quali è necessaria un'anagrafica simile a quella dei clienti, per quanto riguarda le informazioni di base, e ad essa si aggiungono informazioni sull'orario di lavoro e sul controllo delle presenze.
Per ogni impiegato l'orario di lavoro deve contenere il periodo di validità e gli orari giornalieri per ogni giorno della settimana, suddivisi in orario mattina e orario pomeriggio. Un impiegato può avere più orari di lavoro in relazione a diversi periodi temporali.
Per il controllo delle presenze si deve registrare per ogni giorno l'orario di accesso e uscita dall'ambiente di lavoro dell'impiegato.
Vi è una classe specializzata di impiegati, chiamati operatori, per i quali è necessario sapere la loro qualifica e specializzazione. Ogni operatore è abilitato ad eseguire dei particolari trattamenti. Tale controllo è effettuato per segnalare in fase di creazione di un appuntamento se l'operatore selezionato è idoneo ad eseguire il trattamento prefissato.

Si gestisce l'anagrafica dei trattamenti che si possono effettuare nel centro. Ogni trattamento deve avere nome, durata, prezzo,imponibile in percentuale sul prezzo,posti in cui può essere eseguito ed un'eventuale categoria e descrizione. Ogni categoria trattamenti deve avere un nome e una descrizione. I trattamenti possono contenere i prodotti necessari per la loro esecuzione, in modo tale che sia possibile scalare dall'anagrafica prodotti le quantità utilizzate nel trattamento. In tale modo si facilita e si rende più efficiente la gestione dei prodotti.

Si gestisce l' anagrafica dei prodotti. Ogni prodotto ha un nome, una descrizione, una quantità acquistata,una quantità attuale, un'unità di misura , un costo di acquisto per unità di misura ed un eventuale costo di vendita, se si prevede di venderlo. L'unità di misura specifica come vengono gestite le quantità di prodotto. Se per esempio si prevede di vendere il prodotto in flaconi si avrà come u.m.(unità di misura) [p.z.] (pezzo), mentre se il prodotto per esempio è venduto a peso, si può specificare come u.m. il [Kg] (chilogrammo). Quantità acquistata specifica la quantità di prodotto che è stata acquistata fino al momento corrente, mentre quantità attuale specifica le rimanenze in magazzino. Il costo di acquisto per u.m. specifica il prezzo unitario del prodotto per [pz] o [kg] o altra u.m. specificata. I prodotti si prevede di poterli vendere e quindi sono dotati di un prezzo di vendita. I prodotti, come precedentemente detto, possono essere associati ai trattamenti.

I trattamenti possono essere raggruppati in *pacchetti* da vendere al cliente. Ogni pacchetto è quindi composto da una serie di trattamenti. Esso deve avere un nome, un prezzo, un imponibile in percentuale, una categoria ed un'insieme di trattamenti in esso compresi. Nel caso in cui il pacchetto fornisca un certo numero di trattamenti , riportare tale quantità. Il prezzo del pacchetto è separato dal prezzo dei trattamenti. Si prevede infatti di dare un prezzo unitario al pacchetto diverso dalla somma dei prezzi dei trattamenti in esso contenuti. Ad esempio se un trattamento "massaggio Shiatsu" costa €30,00 si può prevedere un pacchetto "Natale Shiatsu" contenente 5 trattamenti Shiatsu al prezzo di €120,00 invece di €30,00 * 5 = €150,00.

Si devono gestire le ricevute, indicando per ogni ricevuta il numero progressivo, il cliente associato a tale ricevuta e i dettagli ricevuta. I dettagli ricevuta possono essere inseriti a mano, in tal caso per ognuno di essi è necessario indicare il nome, una eventuale descrizione, il prezzo , l'eventuale sconto e l'iva.
Ad ogni ricevuta può essere anche associato un prodotto acquistato. In tale caso è necessario indicare l'imponibile, lo sconto(se effettuato), e la quantità di prodotto acquistata. I prodotti inseriti nella ricevuta devono esser tra quelli presenti nell'anagrafica prodotti. Se si vuole inserire un prodotto liberamente è necessario utilizzare un dettaglio ricevuta.
Tramite le ricevute i clienti pagano i pacchetti acquistati. Importante notare che l'utente non può acquistare direttamente un trattamento, ma solo pacchetti. Nel caso in cui si voglia acquistare un solo trattamento, è necessario creare un pacchetto con un solo trattamento associato.
Per ogni pacchetto acquistato si indica il prezzo di vendita, composto da imponibile e da iva e si indica anche un eventuale sconto applicato.
Nelle ricevute si ricarica il credito delle carte fedeltà specificando l'importo versato in un dettaglio movimento. La data dell'ultima ricarica fa fede per quanto riguarda la validità della carta fedeltà.

Bisogna rendere disponibile un' agenda appuntamenti per il planning giornaliero delle prenotazioni.
Ad ogni appuntamento corrisponde un cliente associato a un operatore che esegue un trattamento in un determinato posto. Per ogni appuntamento è necessario indicare la data, l'ora di inizio,l'ora di fine ed eventuali note associate. L'agenda deve essere flessibile e per ogni evento l'associazione con cliente/operatore/trattamento/posto è opzionale.

Un cliente quando acquista un pacchetto non è tenuto a pagarlo immediatamente, quindi per ogni pacchetto acquistato è necessario memorizzare il saldo attuale del pagamento, l'eventuale sconto effettuato, se il pacchetto è stato acquistato completamente (in modo indipendentemente dal raggiungimento del saldo) e se tale pacchetto è stato completamente erogato al cliente.

Inoltre per ogni pacchetto acquistato bisogna tenere traccia dei trattamenti ad esso associati e di essi si vuole sapere se sono stati erogati o meno e in quale appuntamento. Si deve aver la possibilità di segnare un trattamento come eseguito, indipendentemente dalla sua reale associazione con un appuntamento, per mantenere la flessibilità nella gestione.

Ogni cliente può possedere una o più carte fedeltà ciascuna delle quali dà diritto a sconti su particolari pacchetti e prodotti. In generale le carte fedeltà si organizzano in categorie.
Ciascuna categoria di carta definisce gli sconti base per ogni carta appartenente ad essa. Essa specifica una percentuale di sconto che si applica ad ogni pacchetto e prodotto specificato.
Per ogni carta fedeltà si possono aggiungere sconti specifici per il cliente che la possiede(indipendenti dal tipo di carta).
I pacchetti e prodotti si possono comprare tramite ricevuta o effettuando un addebito sul credito di una carta fedeltà. Le carte fedeltà possono essere ricaricate, tale operazione è effettuata emettendo una ricevuta per il cliente. Le ricevute tengono nota del movimento di denaro effettuato in cassa, mentre tutte le operazioni effettuate tramite card sono registrate separatamente in quanto rappresentano un movimento di credito "immaginario". Infatti solo il caricamento della carta tramite ricevuta è un movimento di denaro vero tra il cliente e il centro benessere.

3.2.2 Rappresentazione dei concetti più importanti della realtà di interesse

In figura 1 sono rappresentati i principali concetti da tradurre nella base di dati.
Sono rappresentati i clienti con la loro relazione con le ricevute, carte fedeltà e appuntamenti ,gli operatori con i loro appuntamenti e trattamenti a cui sono abilitati, le carte fedeltà con la loro relazione coi prodotti e pacchetti , i trattamenti associati a ciascun pacchetto e le ricevute.

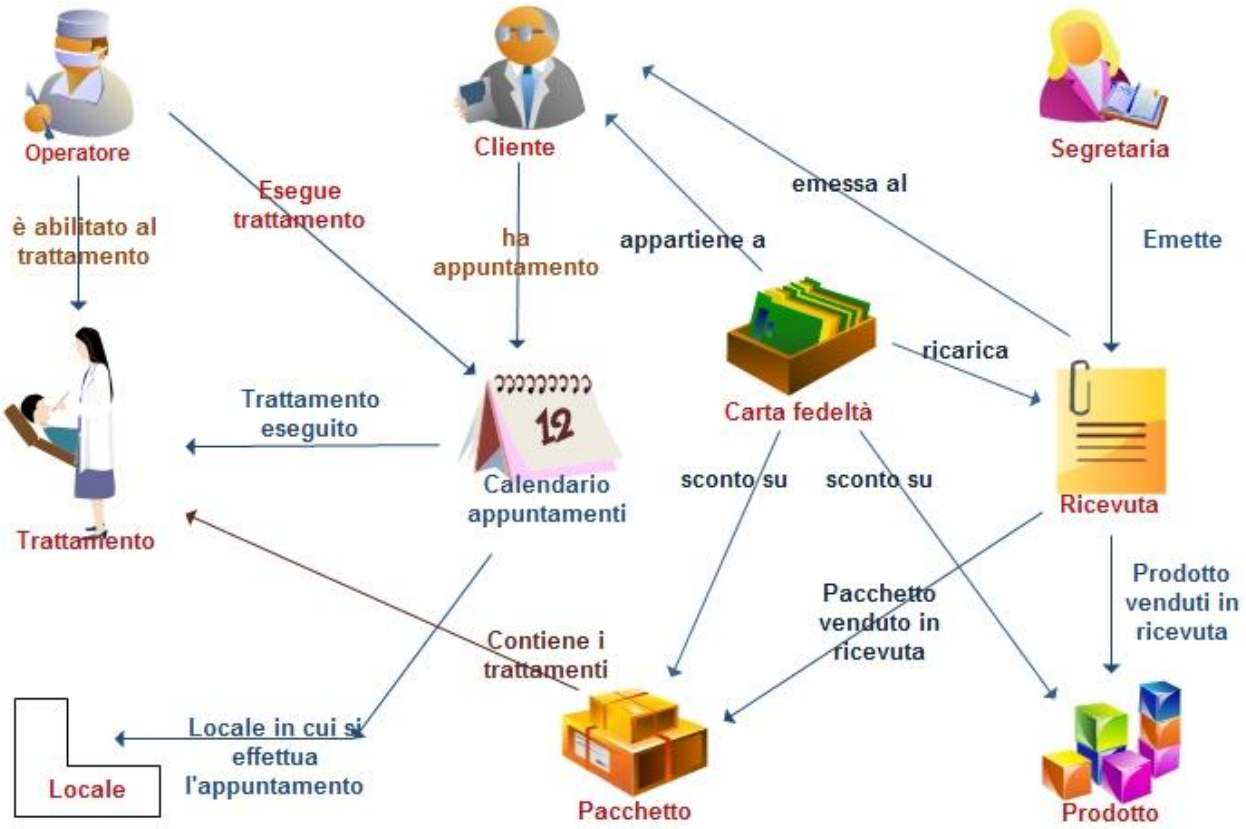


Figura 3-1: Principali concetti del centro benessere da rappresentare

3.2.3 Glossario dei termini, omonimi e sinonimi

Si riporta in seguito l'elenco contenente i termini individuati dalla descrizione informale dei requisiti precedentemente effettuata. Per ogni termine se ne fornisce una breve descrizione, eventuali sinonimi riscontrati nella realtà esaminata e i riferimenti con gli altri termini.

Tabella 1:Glossario dei termini

Termine	Descrizione	Sinonimi	Collegamenti
Cliente	Cliente del centro benessere.		-Cliente(relazione parentela tra clienti) -Appuntamento -Ricevuta(servizi e prodotti acquistati) -Carta fedeltà
Impiegato	Impiegato generico del centro benessere quale può essere un segretario o un operatore.		
Operatore	Impiegato specializzato nella erogazione dei trattamenti ai clienti.	Massaggiatore, terapeuta, dermatologo ecc…	Impiegato
Trattamento	Trattamenti eseguiti nel centro benessere, quali possono essere massaggi, fanghi per le dermatiti, cura in acqua termale ed altro.	Terapia,servizio	-Operatore(abilitato ad eseguire il trattamento) -Posto (dove può essere eseguito)
Prodotto	Prodotto utilizzato all'interno del centro, di cui c'è la necessità di mantenere un'anagrafica. Il prodotto può anche essere venduto ad un cliente.	Merce	-Ricevuta(nel quale è venduto) -Carta fedeltà(nel quale è scontato)
Posto	Località del centro benessere nel quale è erogato un determinato servizio.	Luogo,locale, lettino, sedile,vasca ecc…	-Trattamento(che può essere eseguito in tale posto) -Appuntamento(che si svolge in tale posto)
Appuntamento	Appuntamento nel centro benessere relativo a un cliente.		-Posto(nel quale è eseguito tale appuntamento) -Operatore (responsabile dell'appuntamento) -Trattamento(erogato ad un cliente) -Cliente(al quale si eroga il trattamento in tale appuntamento)
Pacchetto	Collezione di trattamenti da vendere come singola unità ad un cliente.		-Trattamento(presente all'interno del pacchetto) -Ricevuta(trattamento venduto ad un cliente) -Carta fedeltà(nel quale è scontato)
Ricevuta	Ricevute erogate ai clienti.		-Cliente(a cui corrisponde la ricevuta) -Pacchetto (eventualmente pagato in tale ricevuta) -Carta fedeltà(ricaricata in ricevuta)

Carta fedeltà	Carta associata a un cliente per sconti su trattamenti e prodotti.		-Cliente(proprietario della card) -Pacchetto(scontato se pagato con la carta) -Prodotto(scontato se pagato con la carta) -Ricevuta(Nella quale si ricarica la carta)

3.2.4 Requisiti aggiuntivi

3.2.4.1 Requisiti sui pacchetti

I pacchetti sono un multi - insieme di trattamenti che si possono vendere ad un cliente.
Per esempio un pacchetto può comprendere due massaggi, un trattamento viso, una ricostruzione unghie ed un trattamento fanghi.
E' importante distinguere tra i pacchetti che si possono vendere ed i pacchetti effettivamente venduti ad un cliente. I pacchetti definiscono la struttura dell'insieme di trattamenti vendibili come unica unità, con informazioni aggiuntive quale il prezzo complessivo e una descrizione dettagliata. Un pacchetto venduto invece è un insieme di servizi acquistati dal cliente, dei quali è necessario sapere l'importo attualmente versato, in quali ricevute e tutti i trattamenti associati al pacchetto, dei quali è necessario sapere per ognuno se è stato erogato o meno, ed eventualmente in quale appuntamento.
Si distinguerà perciò l'entità pacchetto dall'entità pacchetto acquistato.

3.2.4.2 Requisiti sui trattamenti

Per i requisiti al punto 3.2.4.1 è fondamentale distinguere anche i trattamenti dai trattamenti acquistati. I primi forniscono una descrizione del trattamento con tutte le proprietà specifiche riguardanti(ad esempio nome , descrizione, imponibile, iva), i secondi sono associati ad un 'pacchetto acquistato' e vengono utilizzati distintamente con gli appuntamenti.

3.2.4.3 Requisiti sulle ricevute

Nelle ricevute bisogna riportare l' imponibile, cioè il credito sul quale sarà calcolata la tassa IVA, assieme alla percentuale dell'importo su cui è applicata. Per motivi commerciali è inoltre necessario riportare lo sconto eventualmente applicato sul prezzo finale.
Si sceglie di memorizzare nella base di dati tutti gli importi senza iva, ed indicare separatamente la percentuale su cui si applica la tassa(normalmente I.V.A. al 20% [50]).
E' importante memorizzare gli sconti applicati in quanto danno al commerciante un'idea più accurata delle tecniche commerciali utilizzate per fidelizzare il cliente, nonché una misura quantitativa del bilancio d'esercizio indicando le riduzioni di prezzo utilizzare per incentivare la vendita.

Ogni ricevuta deve comporsi delle seguenti voci:

- Imponibile (senza iva)
- Sconto (senza iva)
- Percentuale IVA

Da queste voci base poi nell'applicativo si ricaveranno:

- Prezzo (con IVA) = Imponibile * (1 + IVA/100)
- Sconto (con IVA) = Sconto(Senza IVA) * (1 + IVA/100)
-

La ricevuta è composta da voci singole(dettagli ricevuta) e quindi ,per ottenere in giusta misura gli importi, è necessario avere importo, sconto e percentuale IVA per ogni singola voce.

3.2.4.4 Requisiti sulle carte fedeltà

Per ogni tipo di carta fedeltà si memorizza il nome, una descrizione, lo sconto percentuale, se le carte associate sono ricaricabili ed i pacchetti e prodotti scontati.
Per ogni carta fedeltà bisogna registrare il cliente associato,il credito caricato(mediante le operazioni di ricarica effettuate tramite ricevute),la data di creazione e di scadenza, il credito speso e la cauzione. Si memorizzano i pacchetti e prodotti scontati e due valori booleani che specificano se sono scontati tutti i prodotti e pacchetti specificati nel suo tipo.

3.3 PROGETTAZIONE CONCETTUALE

Si fornisce una descrizione ad alto livello della base di dati del centro benessere utilizzando il ben noto modello Entity-Relationship [(1)]. La notazione qui utilizzata è quella descritta nel libro 'Sistemi di basi di dati'.Fondamenti di Elmasri e Navathe [51].
Si produce lo schema utilizzando una strategia di progettazione mista, individuando prima i concetti principali e aggiungendo poi quelli correlati.

3.3.1 Schema ER scheletro

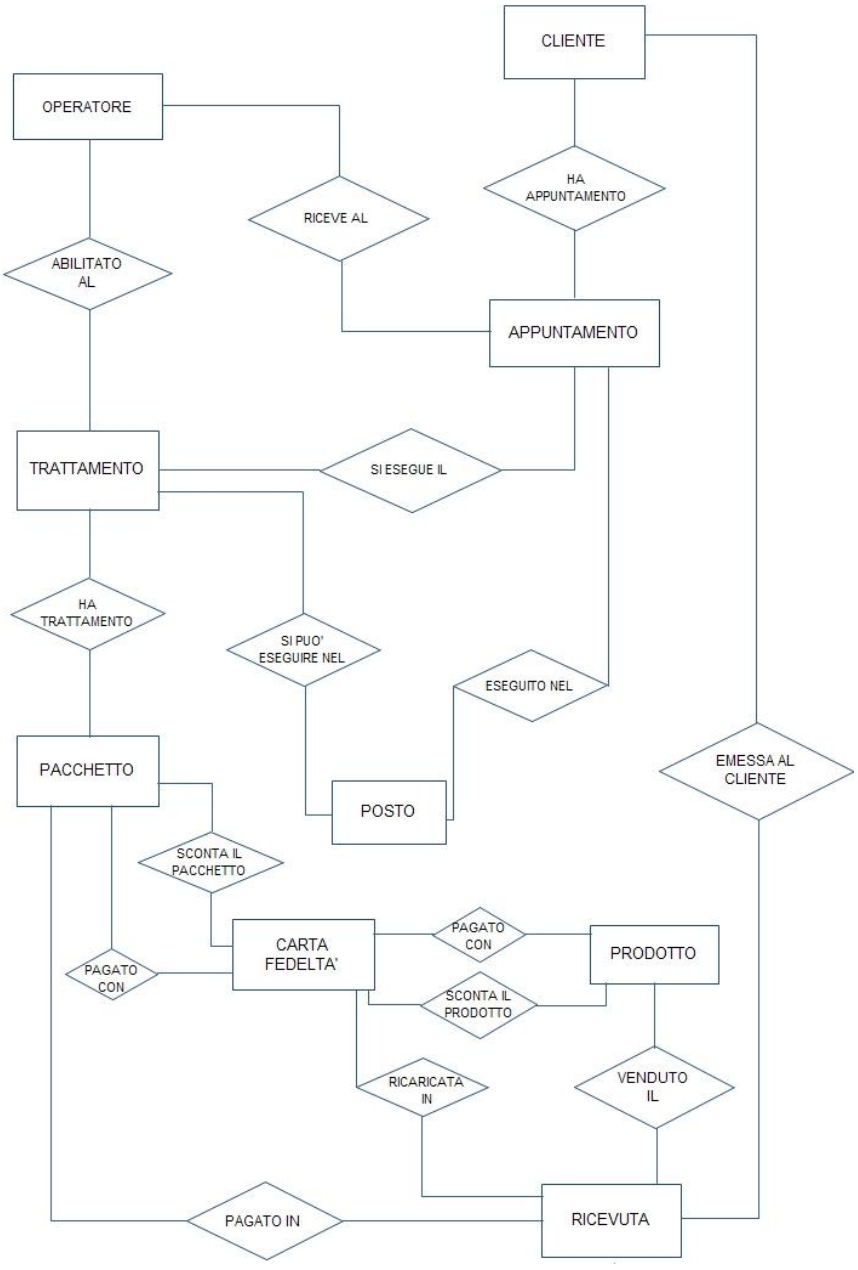


Figura 3-2: Schema scheletro di partenza

In figura 2 si ha lo schema scheletro di partenza ricavato dalla fase di raccolta ed analisi dei requisiti. In esso sono riportati i concetti base analizzati nel dizionario dei termini. Sono state esplicitate le associazioni di partenza individuate tra i vari concetti.
Nel seguito si raffinano i vari concetti visti in precedenza.

[1] Nel contesto della progettazione dei database, il modello entity-relationship (anche detto modello entità-relazione, modello entità-associazione o modello E-R) è un modello per la rappresentazione concettuale dei dati ad un alto livello di astrazione.

3.3.2 Raffinamento ed espansione delle entità ed associazioni individuate nello schema scheletro

Nel seguito si procede al raffinamento delle varie entità , nelle frasi si userà la seguente notazione:

- Attributi: sottolineato
- Entità: MAIUSCOLO
- Associazione: *corsivo*

3.3.2.1 Raffinamento ed espansione dell'entità cliente

E' necessario gestire l'anagrafica CLIENTI per l' inserimento, modifica e recupero dei dati anagrafici, lo storico dei servizi acquistati in precedenza e dei servizi non ancora saldati.
Per ogni cliente bisogna memorizzare il nome, cognome, il nome da visualizzare nell'applicativo (screen name), eventuale email, eventuale password se il cliente vuole accedere al sistema, data di nascita, luogo di nascita,sesso,codice fiscale , recapiti telefonici(fisso o mobile), lo stato civile e professione. Si è anche interessati a sapere *eventuali relazioni di parentela* tra i CLIENTI.

Per ogni UTENTE si vuole anche almeno un suo indirizzo. Ogni indirizzo è costituito da città, codice di avviamento postale, via , numero via e località. E' auspicabile mantenere un database con tutte le CITTA' italiane all'interno dell'applicativo per velocizzare le procedure.[..] Ad ogni cliente, all'atto dell'iscrizione, si pongono alcune DOMANDE a scopo statistico che è necessario raccogliere per una successiva analisi.

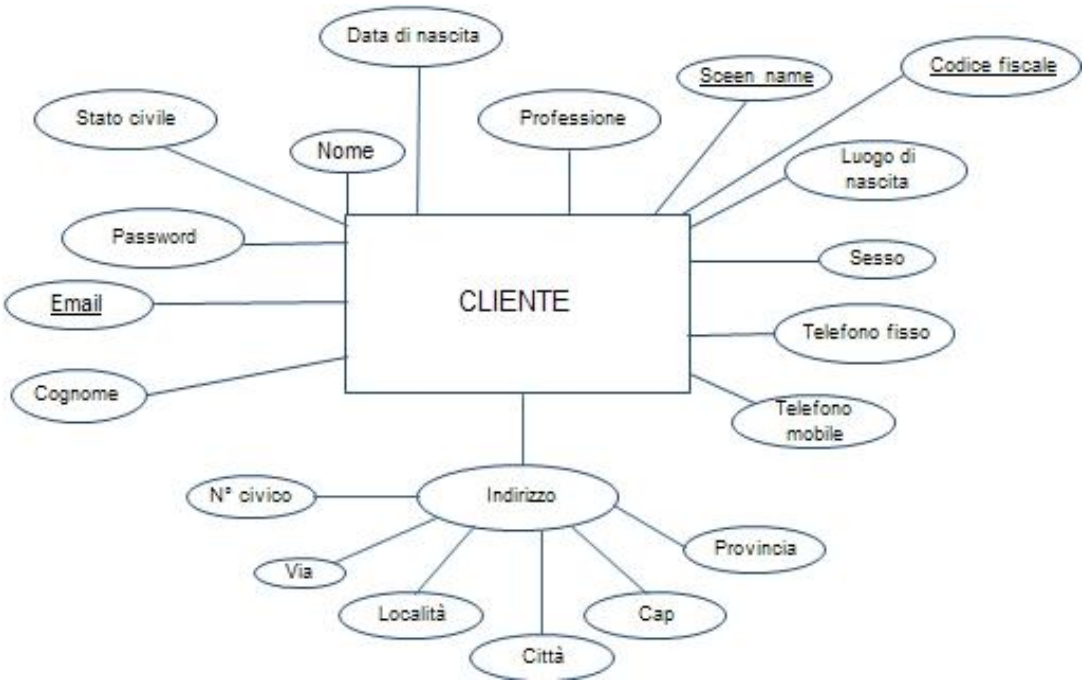


Figura 3-3: Raffinamento di Cliente

Dalla descrizione emerge la necessità della memorizzazione dati relativi alle città, è quindi necessario creare entità separate per tale esigenza.
Si raffina ed espande quindi l'attributo indirizzo come in Figura 4. In essa si nota l'introduzione delle nuove entità CITTA',PROVINCIA,REGIONE e NAZIONE e le associazioni esistenti tra loro.

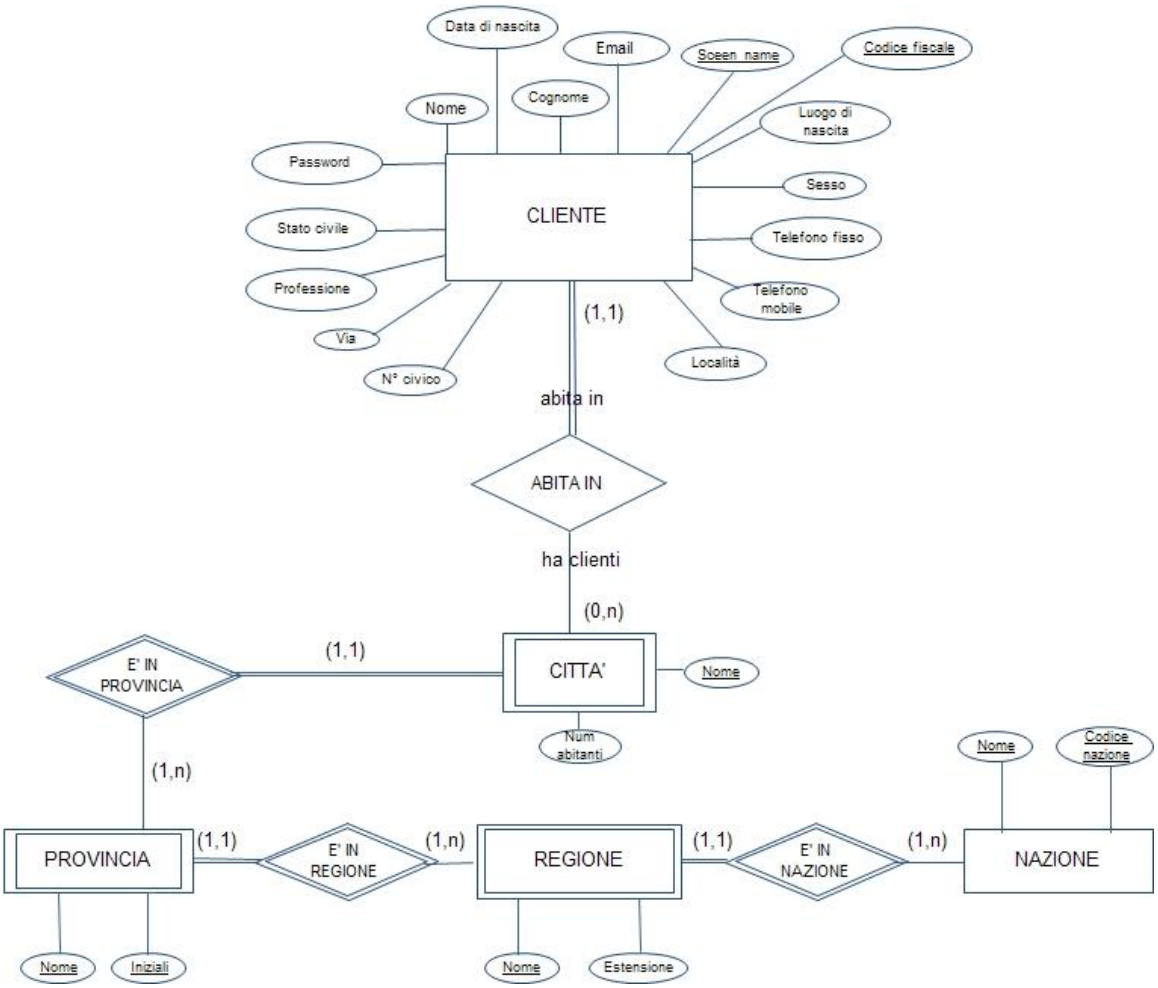


Figura 3-4:raffinamento con entità per il recapito

3.3.2.2 Gerarchia specializzazione tra operatori, impiegati , clienti ed in generale degli utenti della base di dati

Si osserva che molte delle proprietà degli operatori, impiegati e in generale utenti della base di dati sono in comune con le proprietà dei clienti. Ci si riferisce in particolar modo al recapito e informazioni anagrafiche. E' quindi possibile introdurre una gerarchia di generalizzazione/specializzazione per tali entità.
La classe generalizzata è data dal generico UTENTE della base di dati. Sottoclassi di questo saranno IMPIEGATO, OPERATORE E CLIENTE.
Si introduce inoltre a questo punto la relazione di parentela tra i clienti.

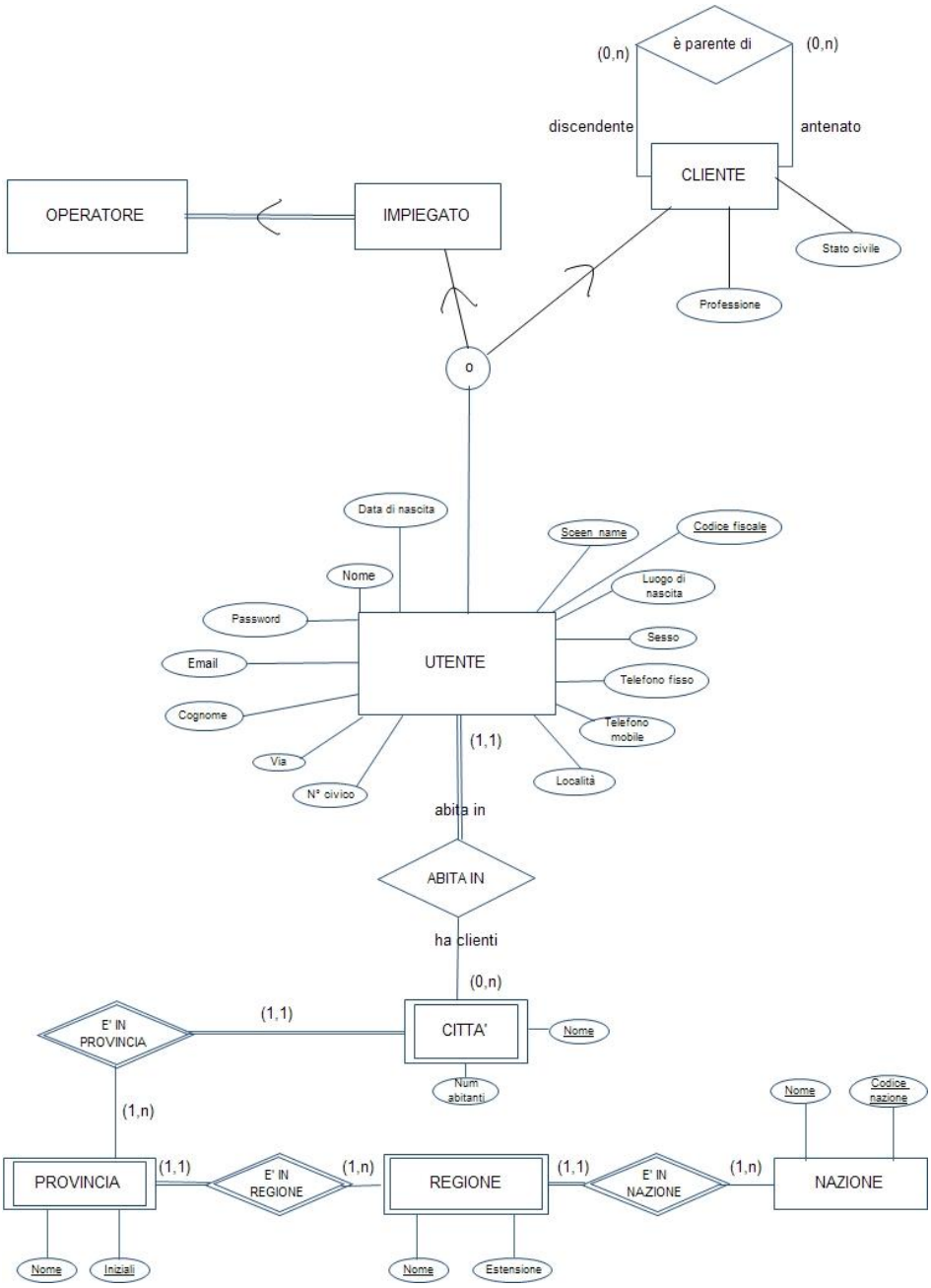


Figura 3-5:gerarchia specializzazione degli utenti

3.3.2.3 raffinamento ed espansione dell'entità Impiegato

Frase relativa agli impiegati:

Vanno gestiti gli IMPIEGATI del centro per i quali è necessaria un'anagrafica simile a quella dei clienti per quanto riguarda le informazioni di base e ad essa si aggiungono informazioni sull'orario di lavoro e sul controllo delle presenze.
Vi è una classe specializzata di IMPIEGATO, chiamati OPERATOR, per la quale è necessario sapere la loro qualifica e specializzazione. Ogni operatore è abilitato ad eseguire dei particolari trattamenti.

Si introdurranno due nuove entità:

- Entità ORARIO LAVORO: descrive l'orario lavoro di ogni impiegato, anche in relazione a diversi periodi temporali.
- Entità PRESENZA: descrive le presenze giornaliere di un impiegato

Entrambe le entità sono entità deboli, avente come entità identificante IMPIEGATO.

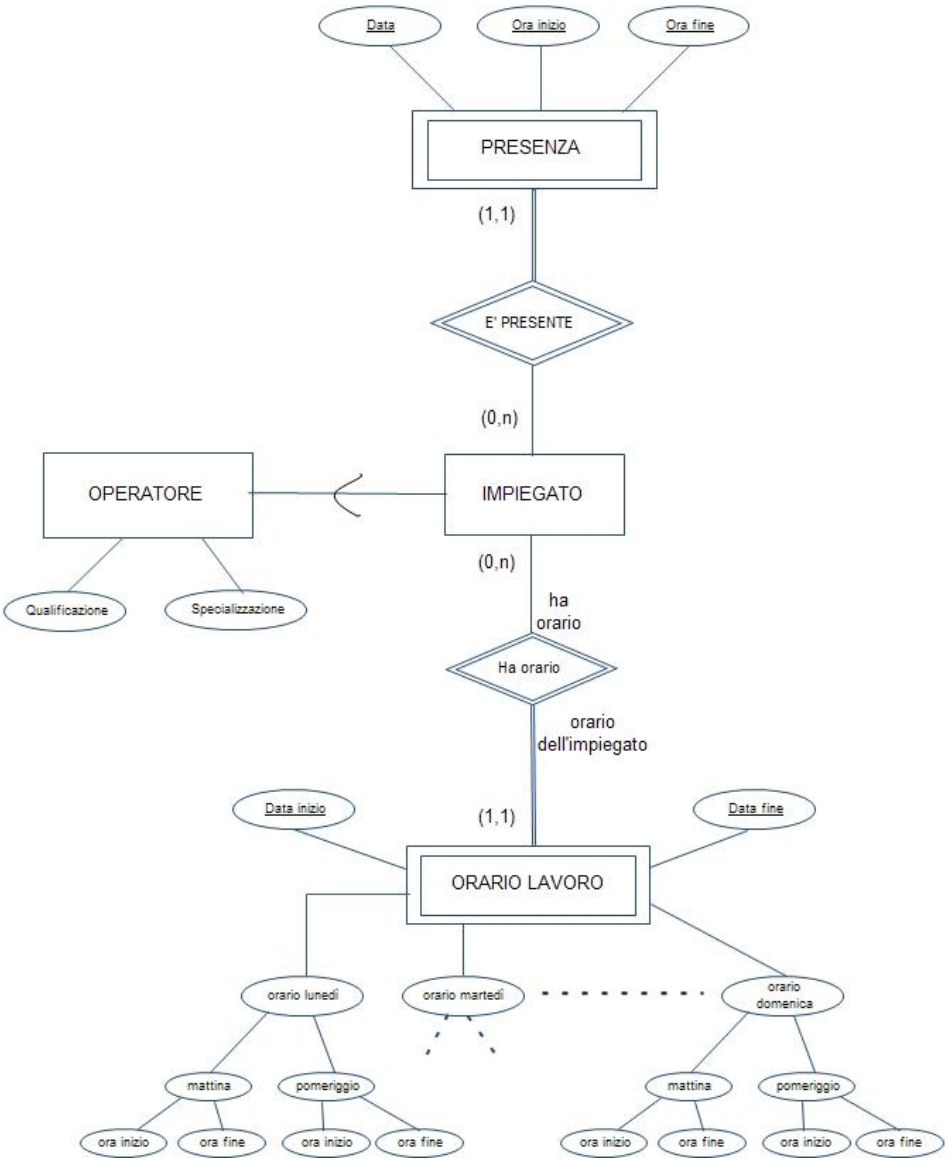


Figura 3-6:raffinamento di IMPIEGATO

3.3.2.4 raffinamento dell'entità Posto

Ad ogni appuntamento corrisponde un CLIENTE associato a un OPERATORE che esegue un determinato TRATTAMENTO in un determinato POSTO. Si crea l'entità POSTO che descrive un ambiente utilizzato nel centro benessere.

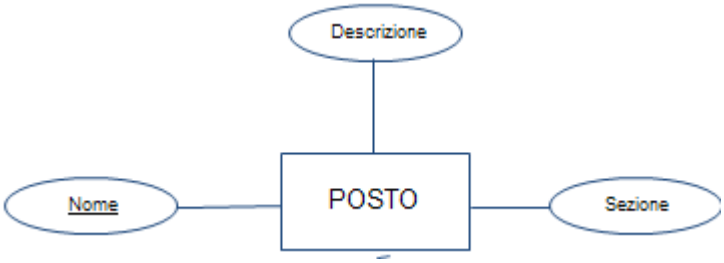


Figura 3-7: Raffinamento di POSTO

L'attributo sezione descrive in maniera più esaustiva dove si trova il locale, la sezione la parte del centro dove è collocato.

3.3.2.5 Raffinamento dell'entità Prodotto

Ogni PRODOTTO ha un nome, una descrizione, una quantità iniziale,quantità attuale, un'unità di misura , un costo di acquisto per unità di misura ed un eventuale costo di vendita, se si prevede di venderlo. In modo naturale si utilizza come attributo chiave 'Identificativo prodotto'.

Figura 3-8:Raffinamento di Prodotto

3.3.2.6 Raffinamento dell'entità Trattamento

Ogni TRATTAMENTO deve avere un nome, durata, prezzo, imponibile in percentuale sul prezzo,[…] ed un'eventuale CATEGORIA e descrizione.
Ogni CATEGORIA TRATTAMENTO deve avere un nome ed una descrizione.
In figura 9 sono rappresentate tali entità e l'associazione tra esse. I vincoli di cardinalità specificano che un TRATTAMENTO può avere una CATEGORIA TRATTAMENTO, mentre una CATEGORIA TRATTAMENTO può avere zero o più TRATTAMENTI.

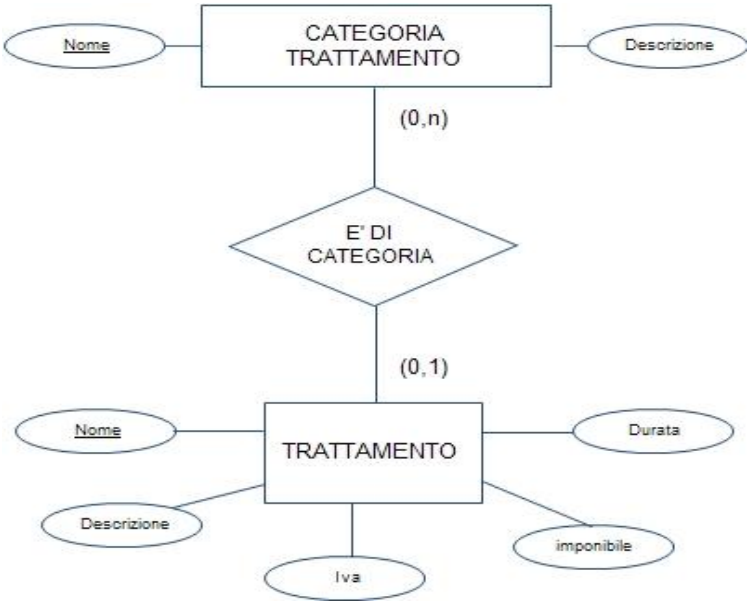


Figura 3-9:Raffinamento di TRATTAMENTO

3.3.2.7 Raffinamento dell'entità Pacchetto

Ogni PACCHETTO […] deve avere un nome, un prezzo, un imponibile in percentuale, una CATEGORIA …[…].

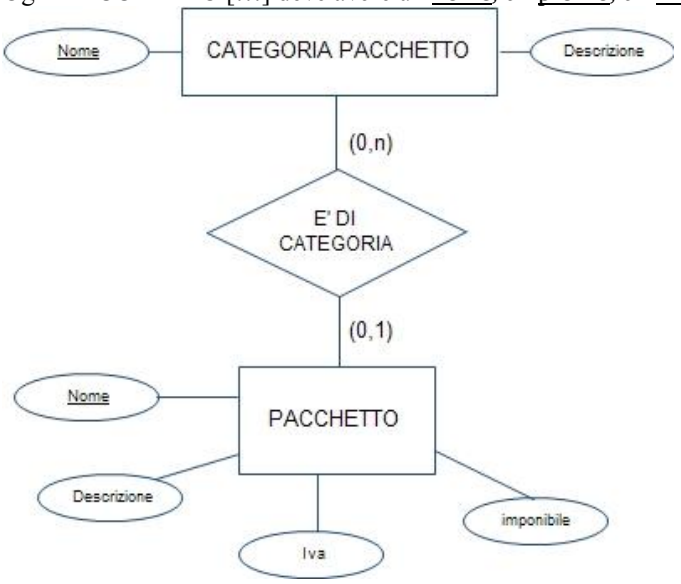


Figura 3-10:Raffinamento di pacchetto

3.3.2.8 raffinamento dell'entità Appuntamento

Per ogni APPUNTAMENTO è necessario indicare la data, l'ora di inizio, l'ora di fine ed eventuali note associate. In non tutti gli appuntamenti vi è l'obbligatorietà di indicare il posto dove si svolge, il trattamento acquistato svolto, il cliente che lo svolge e l'operatore. Ciò perché gli appuntamenti possono essere usati come una agenda nel senso più generale del termine. E' perciò necessario introdurre un attributo identificativo progressivo per ogni evento.
In figura 11 lo schema di APPUNTAMENTO.

Figura 3-11:Raffinamento di appuntamento

3.3.2.9 Raffinamento ed espansione dell'entità ricevuta

Si devono gestire le ricevute, indicando per ogni RICEVUTA il numero progressivo,[…] i dettagli ricevuta. I DETTAGLI RICEVUTA possono essere inseriti a mano, in tal caso per ognuno di essi è necessario indicare il nome, una eventuale descrizione, il prezzo e l'iva. In figura 12 la porzione di schema ER di RICEVUTA e DETTAGLI RICEVUTA.

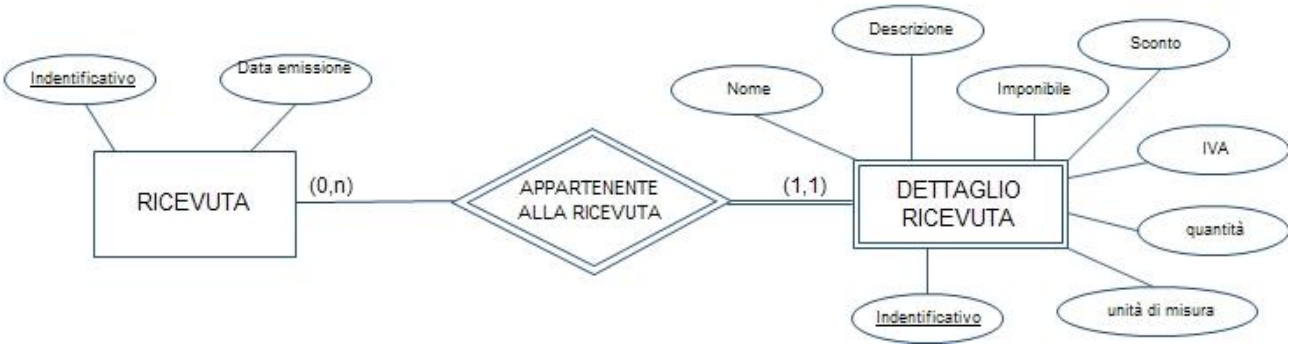


Figura 3-12:RICEVUTA e DETTAGLI RICEVUTA

3.3.2.10 Espansione delle entità TRATTAMENTO e PACCHETTO per soddisfare i requisiti specifici

[…]E' importante distinguere tra i PACCHETTI che si possono vendere ed i PACCHETTI VENDUTI ad un CLIENTE. I pacchetti definiscono "la struttura" dell'insieme di trattamenti vendibili come unica unità, con informazioni aggiuntive quale il prezzo complessivo ed una descrizione dettagliata. Un PACCHETTO "VENDUTO" invece è un insieme di servizi acquistati dal cliente, dei quali è necessario sapere l'importo attualmente versato, in quali *RICEVUTE* o *CARTE FEDELTA'* è stato fatto l'addebito e tutti i *TRATTAMENTI* associati al pacchetto, dei quali è necessario sapere per ognuno se è stato erogato o meno, e in quale *APPUNTAMENTO*. In figura 13 lo schema ER derivante.
Nell'entità pacchetto venduto vi sono gli attributi:

- Imponibile: valore commerciale del pacchetto senza iva
- Imponibile pagato : imponibile saldato fino ad ora
- Sconto: sconto senza iva applicato all'imponibile
- Iva: valore percentuale dell'iva applicata al pacchetto
- Pagamento completato: valore booleano che indica il saldo del pagamento
- Pacchetto completato: valore booleano per sapere se sono stati eseguiti tutti i trattamenti associati al pacchetto

Si distingue quindi anche l'entità TRATTAMENTO dall'entità TRATTAMENTO VENDUTO, che rappresenta un trattamento associato a un pacchetto venduto.

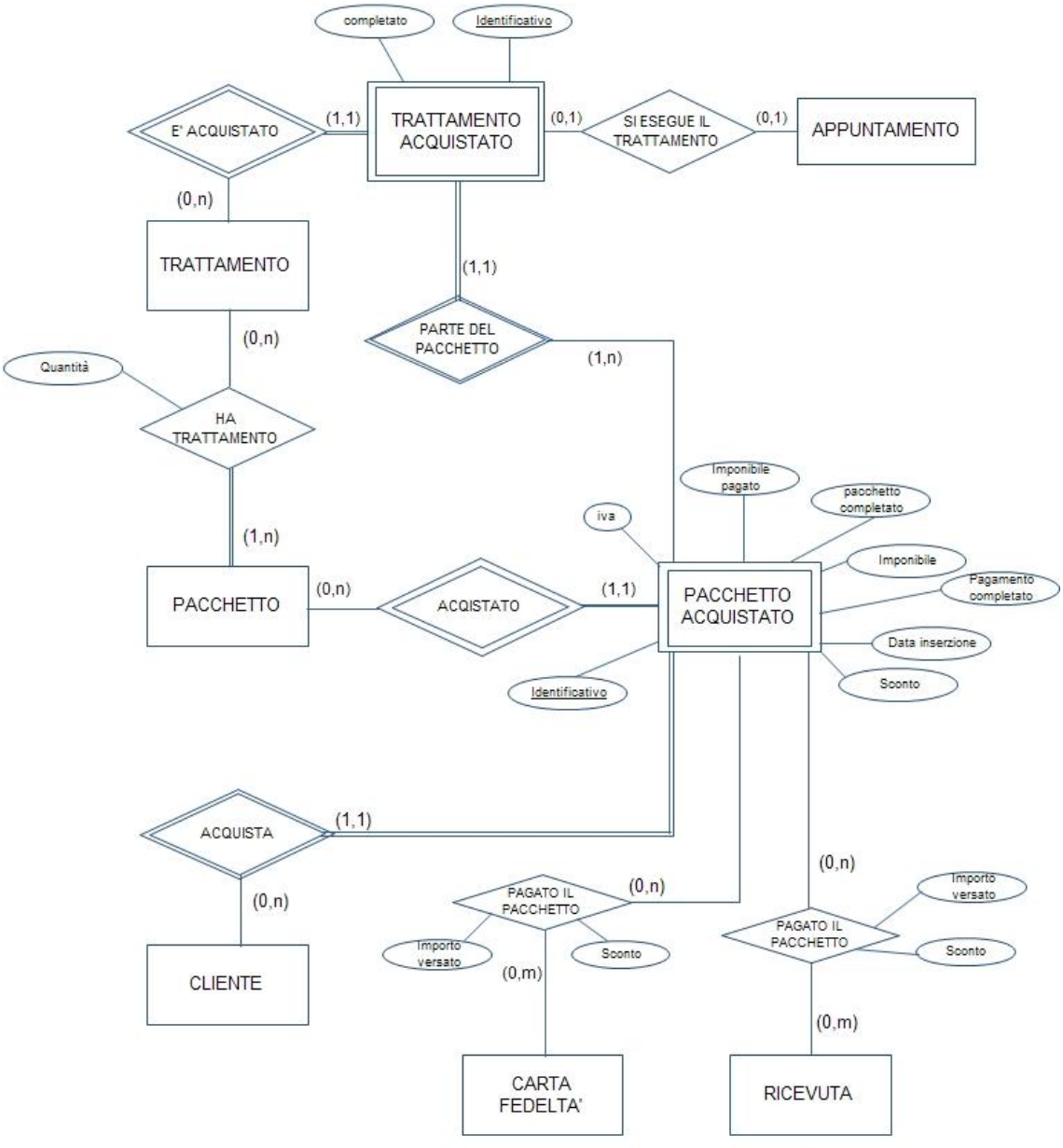


Figura 3-13:Schema dettagliato dei pacchetti e trattamenti

3.3.2.11 Espansione delle entità TIPO CARTA FEDELTA' e CARTA FEDELTA'

Ogni CLIENTE può possedere una o più CARTE FEDELTA' ciascuna delle quali dà diritto a sconti su particolari PACCHETTI e PRODOTTI. In generale le CARTE FEDELTA' si organizzano in CATEGORIE.
Ciascuna CATEGORIA DI CARTA definisce gli sconti base per ogni CARTA appartenente ad essa. Essa specifica una percentuale di sconto che si applica ad ogni PACCHETTO e PRODOTTO specificato.
Per ogni CARTE FEDELTA' si possono aggiungere *sconti specifici* per il cliente che la possiede(indipendenti dal tipo di carta). I PACCHETTI e PRODOTTI si possono comprare tramite RICEVUTA o effettuando un ADDEBITO sul credito di una carta fedeltà. Le CARTE FEDELTA' possono essere ricaricate, tale operazione è effettuata emettendo una RICEVUTA per il cliente. Le ricevute tengono nota del movimento di denaro effettuato in cassa, mentre tutte le operazioni effettuate tramite CARD sono registrate separatamente in quanto rappresentano un movimento di credito "immaginario". Infatti solo il caricamento della CARTA tramite RICEVUTA è un movimento di denaro vero tra il cliente e il centro benessere.

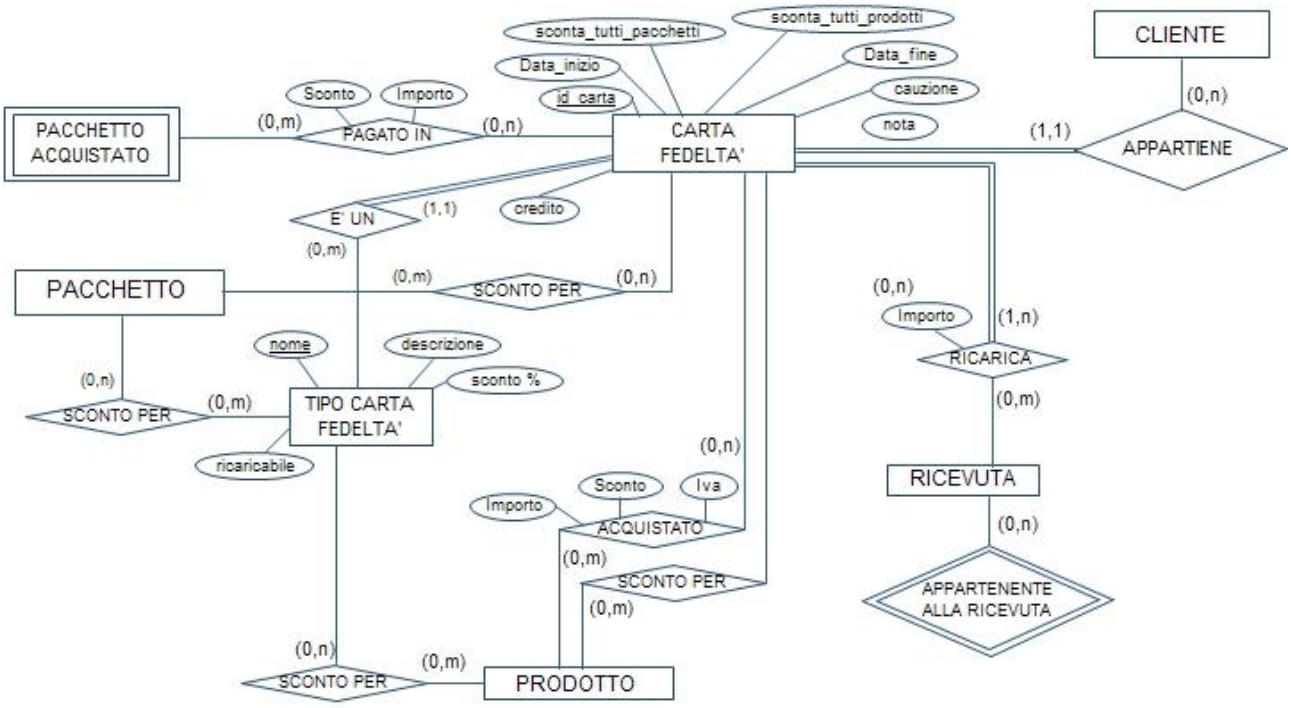


Figura 3-14:Schema dettagliato della CARTA FEDELTA'

3.3.3 Schema EER completo

Nel seguito si riporta lo schema Entity Relationship esteso [52] (Figura 14) derivato dall'analisi precedente per integrazione dei vari sottoschemi analizzati.
Sono stati tratteggiati, per semplicità di lettura, i concetti riguardanti l'anagrafica delle città, dei trattamenti/prodotti e delle ricevute. Dove necessario, sono stati riportati gli attributi delle associazioni.
NOTA: Per semplicità di lettura non si riportano nello schema completo gli attributi delle entità derivati al punto precedente(3.2), sono rappresentate solo le entità e le associazioni tra esse.

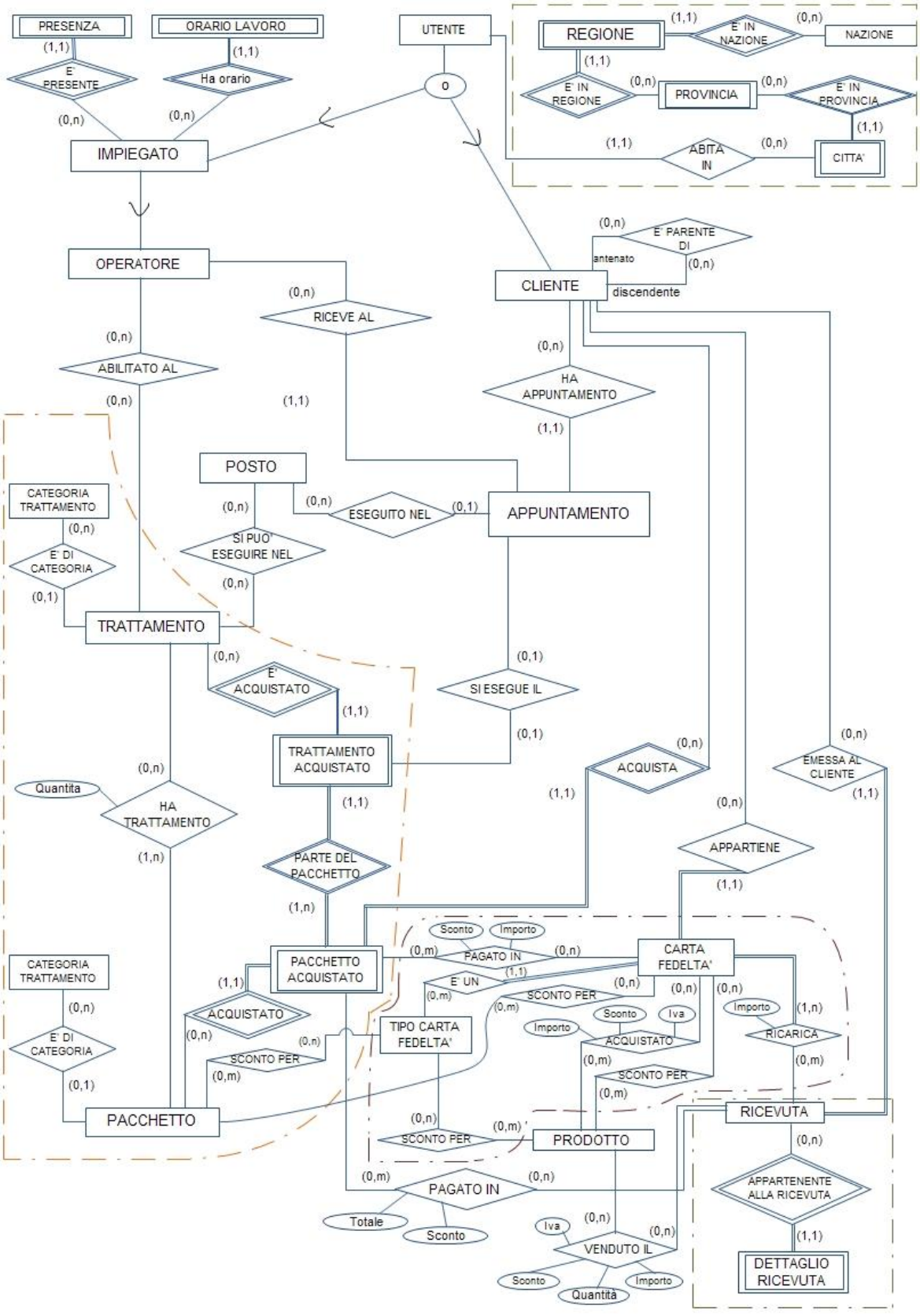


Figura 3-15:Schema ER completo

3.4 PROGETTAZIONE LOGICA

Si procede alla ristrutturazione e traduzione dello schema concettuale nel modello relazionale.

LEGENDA IMMAGINI:

- Simbolo chiave(giallo): attributo chiave primaria
- Rombo vuoto(ciano): normale attributo non obbligatorio
- Rombo pieno(ciano): normale attributo obbligatorio
- Rombo vuoto scuro(rosso): attributo chiave esterna non obbligatorio
- Rombo pieno(rosso): attributo chiave esterna obbligatorio

Nell'analisi successiva per gli attributi che hanno a che fare con la valuta si utilizza il tipo di dati INT, rappresentando gli importi in centesimi di euro. L'iva inoltre è INT essendo una percentuale compresa tra 0-100.

3.4.1 Ristrutturazione e traduzione della gerarchia degli utenti del sistema

Si sceglie di tradurre la gerarchia di specializzazione dell' utente: impiegato-operatore/cliente utilizzando relazioni multiple con superclassi e sottoclassi. Data infatti la natura dell'applicativo che utilizza il database, scritto in Java e utilizzante la mappatura oggetti/relazionale, si preferisce questo approccio in quanto rispecchia in modo naturale la gerarchia tra oggetti in Java. Inoltre è preferibile al caso di rappresentazione tramite singola relazione, per evitare un gran numero di associazioni in un'unica relazione che non hanno a che fare con tutti i concetti rappresentati.

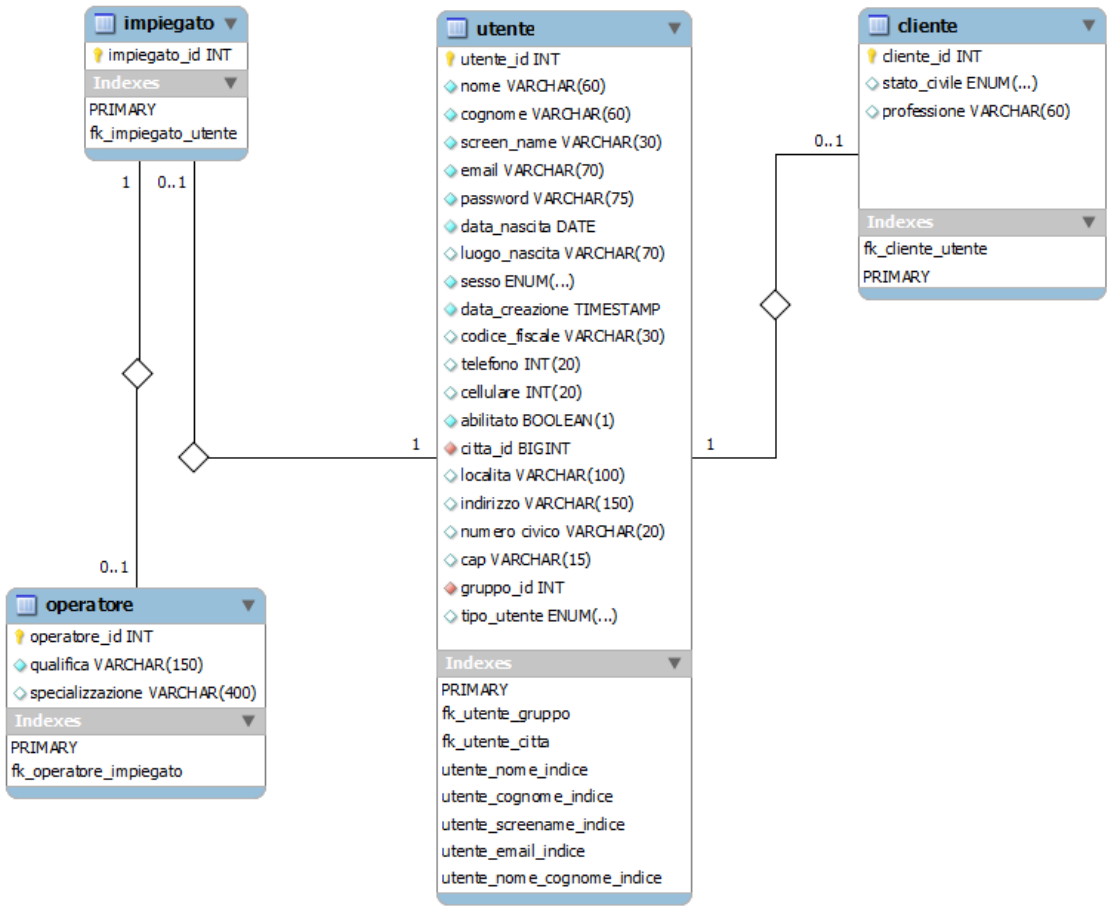


Figura 3-16:Gerarchia utenze

Si nota in figura 15 la traduzione della gerarchia di specializzazione. Vi è una singola relazione per ogni entità rappresentata, ognuna avente i suoi attributi specifici. Gli attributi generali sono raggruppati nella relazione utente.

Ogni relazione ha come chiave primaria la chiave esterna che fa riferimento alla relazione padre(eccetto per la relazione utente,radice della gerarchia).
In utente sono stati introdotti i seguenti campi:

- Id_utente: si è scelto di distinguere tramite id ogni utente, introducendo quindi una chiave primaria surrogata. Si è fatta tale scelta per integrare la base di dati con l'applicativo e ottimizzare le operazioni usando una chiave di tipo intero.
- Data_creazione: definisce la data di inserimento dell'utente nella base di dati
- Tipo_utente: necessario per identificare facilmente il tipo di tupla presente
- Gruppo: chiave esterna che identifica il gruppo nell'applicativo al quale appartiene l'utente

Nel software gestionale è richiesta una gestione delle utenze (autenticazione, autorizzazione) basata sui gruppi di utenti. E' necessario perciò introdurre una nuova relazione per la rappresentazione di tali gruppi.

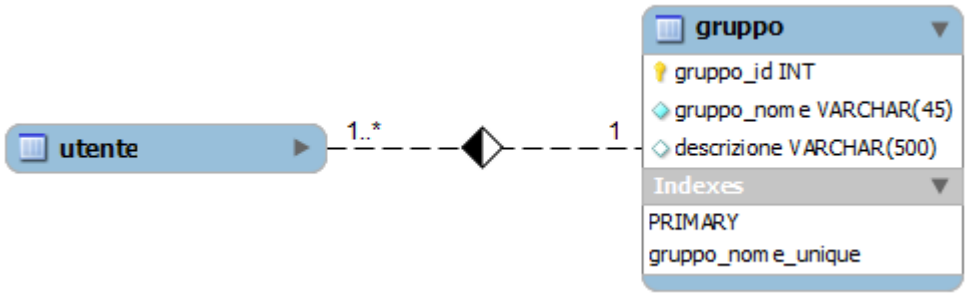


Figura 3-17:Gruppi di utenti per l'applicativo

Si rappresenta tramite una relazione associazione la relazione di parentela tra i clienti, indicando anche il tipo di parentela.

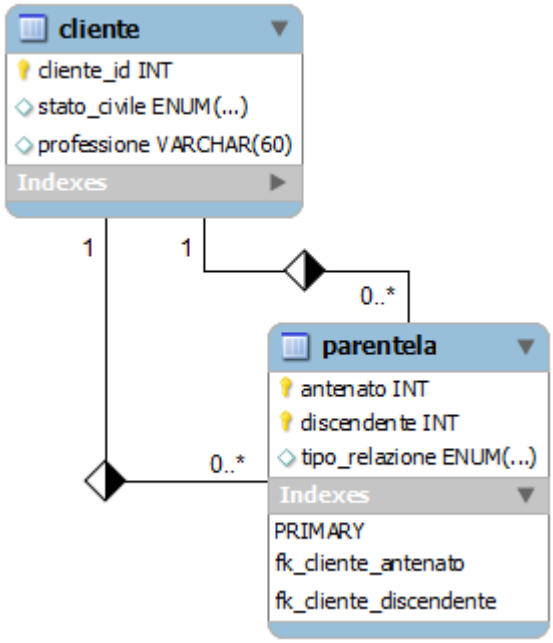


Figura 3-18:relazione 'parentela' tra clienti

-Relazioni referenzianti la relazione impiegato:

All'entità impiegato sono associate l'entità presenza e l'entità orario di lavoro . Esse hanno come entità identificante 'impiegato'.
Tuttavia dati gli attributi chiave presenti in presenza , cioè giorno, ora inizio e ora fine assieme all'associazione identificante, si nota che questi non forniscono una chiave primaria di facile utilizzo, né tantomeno un vincolo di univocità che permette agli orari di non sovrapporsi.
Infatti è possibile per un impiegato avere due orari sovrapposti nell'insieme di entità.

Si sceglie perciò di introdurre una chiave surrogata chiamata ‘presenza_id’ ed associare tramite chiave esterna la relazione alla relazione ‘impiegato’. Si demanda quindi alle regole di business la verifica della sovrapposizione degli orari e della consistenza della relazione.
Per un simile motivo si introduce una chiave surrogata ‘orario_lavoro_id’ per la relazione ‘orario_lavoro’.

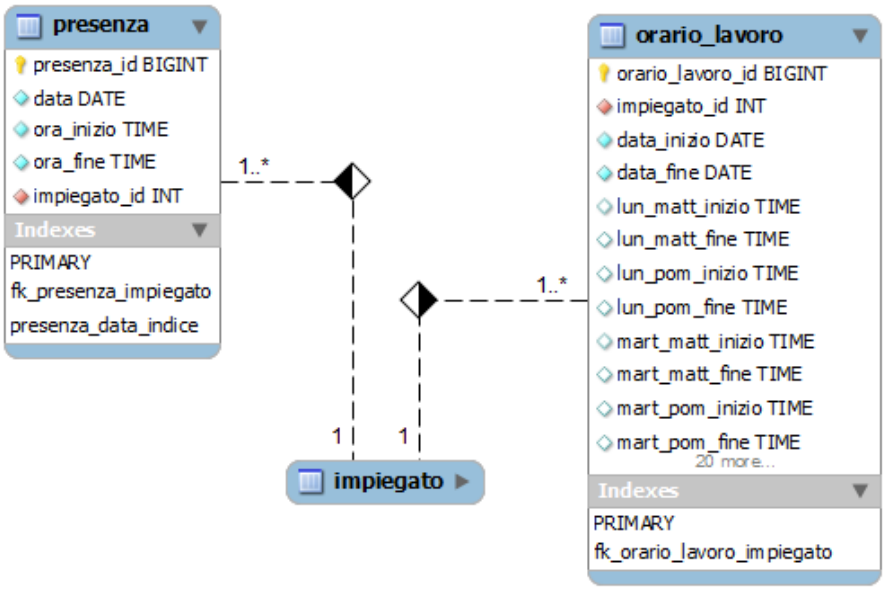


Figura 3-19:relazioni referenzianti impiegato

3.4.2 Ristrutturazione e traduzione delle entità per l’anagrafica città

Si traducono le entità NAZIONE,REGIONE,PROVINCIA e CITTA’ e le relative associazioni.
Le associazioni sono associazioni binarie di tipo uno-molti e si sceglie di tradurle nel modo più semplice possibile inserendo , come chiave esterna, in ogni relazione del lato molti la chiave primaria del lato uno.
Dato che l’entità nazione è identificante per l’entità regione, questa a sua volta per l’entità provincia ed infine provincia è identificante per città, si ha un gran numero di entità identificanti in cascata, il che comporterebbe l’introduzione di molti attributi chiave primaria/esterna in ogni entità.
Ad esempio città avrebbe come chiave primaria (NAZIONE.nome, REGIONE.nome, PROVINCIA. nome, CITTA.nome) .
Ciò rende molto inefficiente la gestione della base di dati, si pensi ad esempio l’introduzione di tale chiave per associare una città ad un utente. Inoltre essendo tutti gli attributi componenti la chiave di città delle stringhe, le operazioni sulla relazione sarebbero inefficienti.
Si è scelto perciò di introdurre per ognuna di queste entità una chiave surrogata identificatrice.

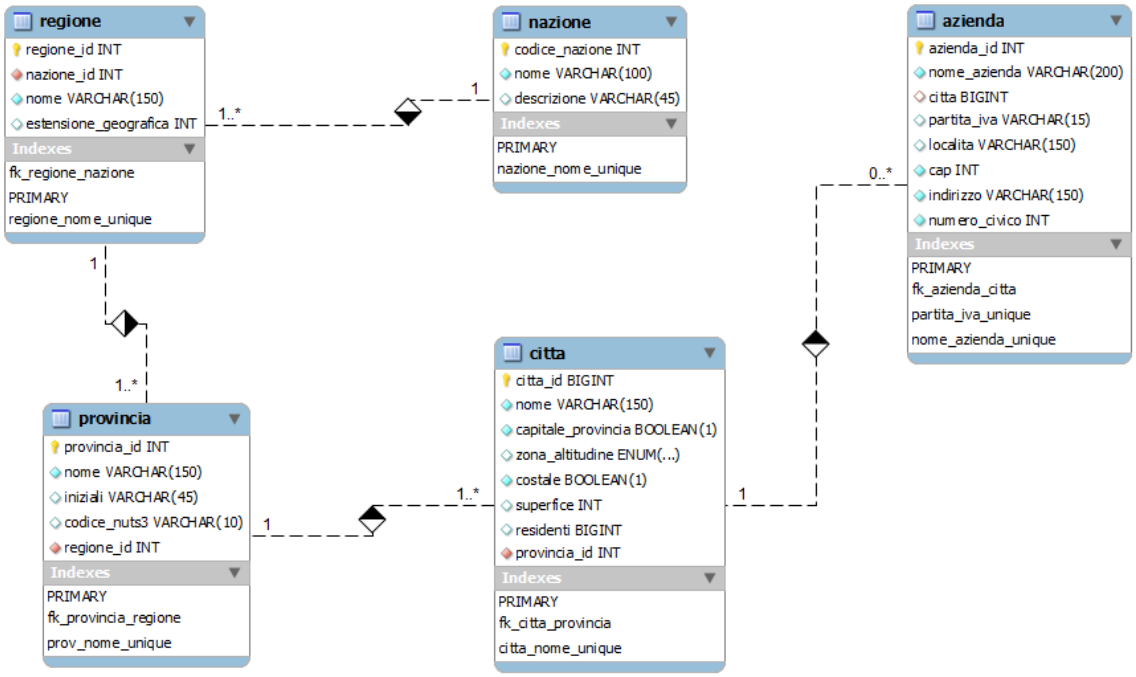


Figura 3-20:Relazioni anagrafica città

3.4.3 Ristrutturazione e traduzione delle entità trattamento e pacchetto

Si procede alla semplice traduzione di ciascuna entità nella relazione corrispondente.
L'associazione tra i trattamenti e i pacchetti viene tradotta mediante una relazione associazione avente come chiavi esterne le chiavi di trattamento e pacchetto e come chiave primaria l'unione di quest'ultime. Si aggiungono alla nuova relazione l'attributo dell'associazione , quantità e si aggiunge una nota.

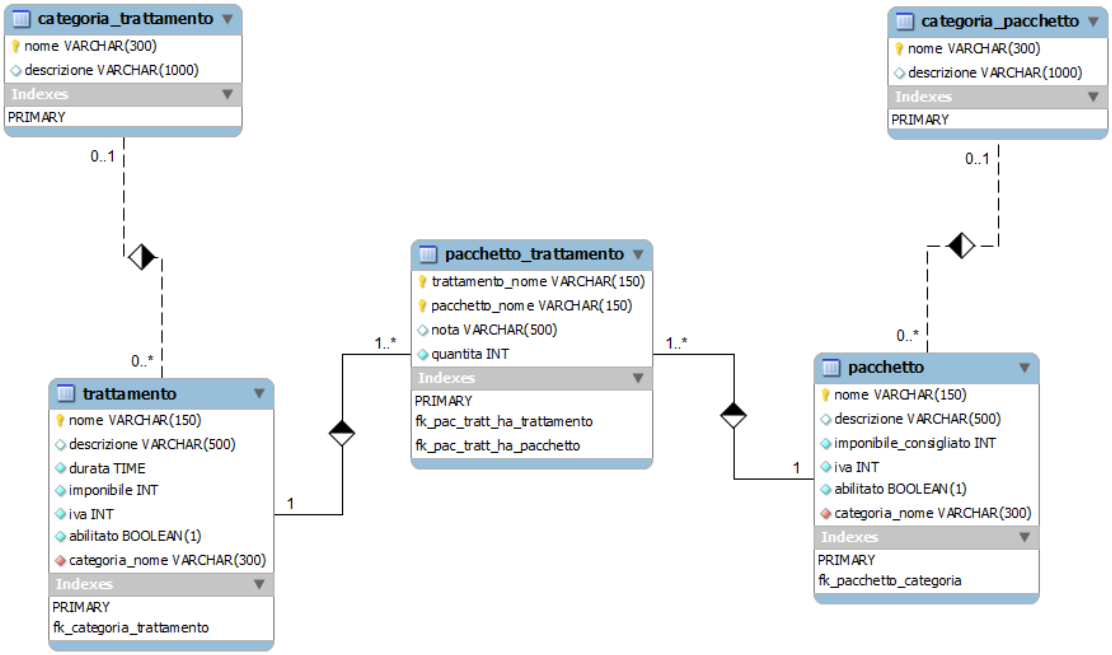


Figura 3-21:relazioni per pacchetti e trattamenti

3.4.4 Ristrutturazione e traduzione delle entità pacchetto acquistato e trattamento acquistato

Si nota che PACCHETTO ACQUISTATO è un'entità debole, avente come entità identificatrici PACCHETTO e CLIENTE. Dato però che un'utente può acquistare più volte uno stesso tipo di PACCHETTO è necessario introdurre anche un numero progressivo per identificare tra loro tali pacchetti. Quindi data comunque la necessità di identificare tramite ID i pacchetti comprati, si utilizza come chiave primaria l'attributo id inteso in senso generale, cioè univoco per ogni tupla nella relazione e non solo per ogni tupla corrispondente alla stessa combinazione pacchetto-cliente.

TRATTAMENTO ACQUISTATO è anch'essa un'entità debole, avente come entità identificatrici PACCHETTO COMPRATO e TRATTAMENTO. Dato però che un trattamento può comparire più volte in uno stesso PACCHETTO ACQUISTATO è necessario introdurre anche un numero progressivo per identificare tra loro i trattamenti acquistati. Quindi data comunque la necessità di identificare tramite id i trattamenti comprati, si utilizza come chiave primaria l'attributo id inteso in senso generale, cioè univoco per ogni tupla nella relazione e non solo per ogni tupla corrispondente alla stessa combinazione pacchetto comprato-trattamento.

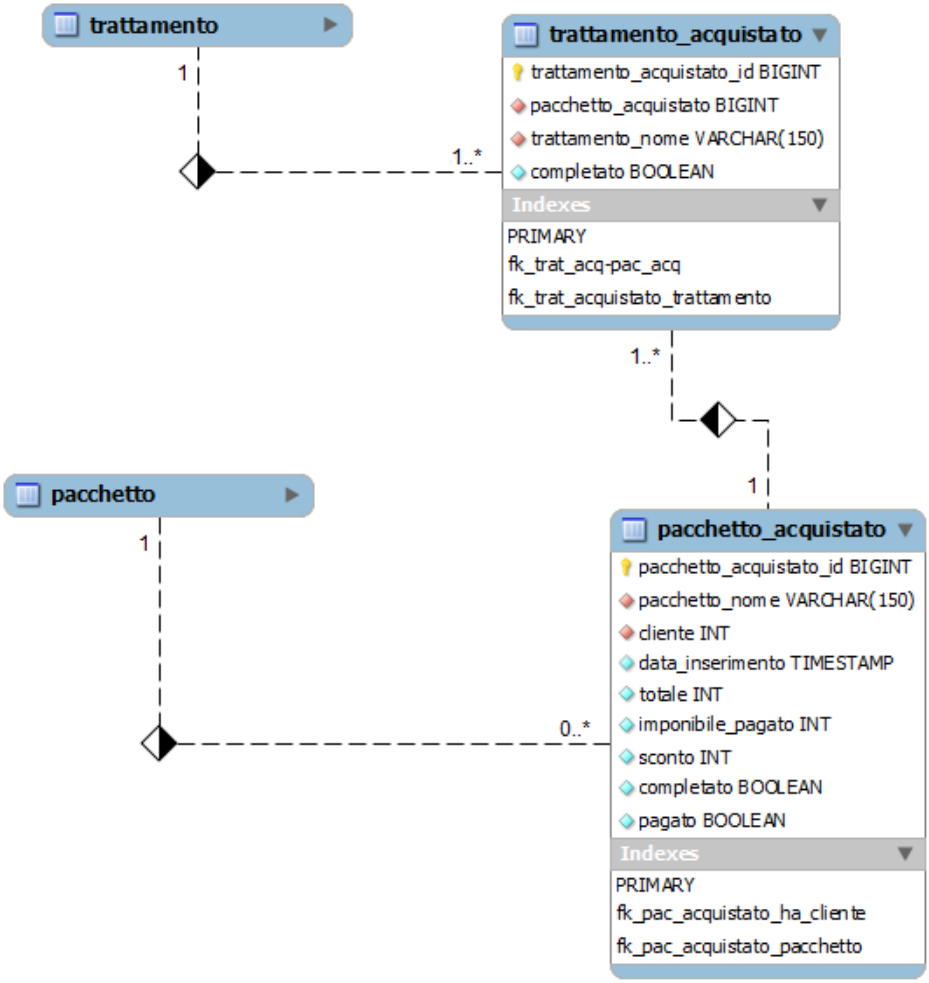


Figura 3-22:pacchetti e trattamenti acquistati

In figura 20 si notano le chiavi primarie ID delle due relazioni.

3.4.5 Ristrutturazione e traduzione delle entità posto e appuntamento

Banalmente si traduce l'entità POSTO nella corrispondente relazione e l'entità APPUNTAMENTO nella relazione associata. Essa ha come chiavi esterne le chiavi primarie di impiegato, cliente, trattamento acquistato e posto.

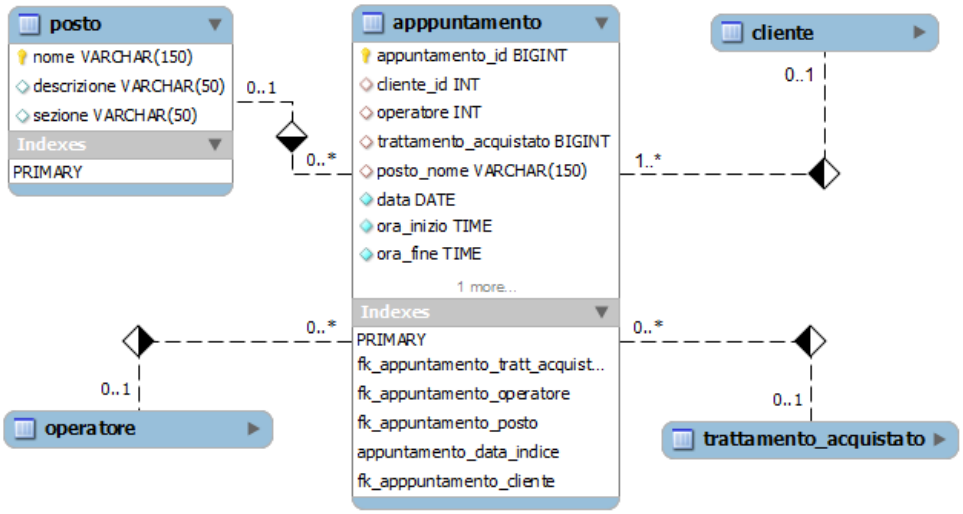


Figura 3-23:relazione appuntamento e posto

3.4.6 Traduzione delle entità ricevuta e prodotto e delle entità ad esse associate

Si traduce l'entità RICEVUTA con la relazione corrispondente, avente come chiave primaria il progressivo della ricevuta. L'entità PRODOTTO è mappata nella relazione relativa avente come chiave primaria l'ID prodotto.
Le associazioni PAGATO IN e VENDUTO IL contengono attributi e devono essere tradotte in due relazioni associazioni(rispettivamente 'ricevuta_pacchetto_acquistato' e 'ricevuta_prodotto') aventi come chiave primaria la combinazione delle chiavi delle relazioni che relazionano.
In ricevuta si aggiunge l'attributo 'metodo di pagamento'. In ricevuta_pacchetto_acquistato si notano i due attributi obbligatori imponibile e sconto. L'iva viene calcolata risalendo alla relazione 'pacchetto acquistato'.
Nella relazione 'ricevuta_prodotto' vi sono gli attributi imponibile, sconto e quantità, derivanti direttamente dall'associazione dello schema ER. Si aggiunge anche l'attributo iva per separare i dati della vendita effettuati da quelli di 'prodotto', in modo tale che se iva dovesse cambiare, l'iva al momento della vendita rimane invariata.

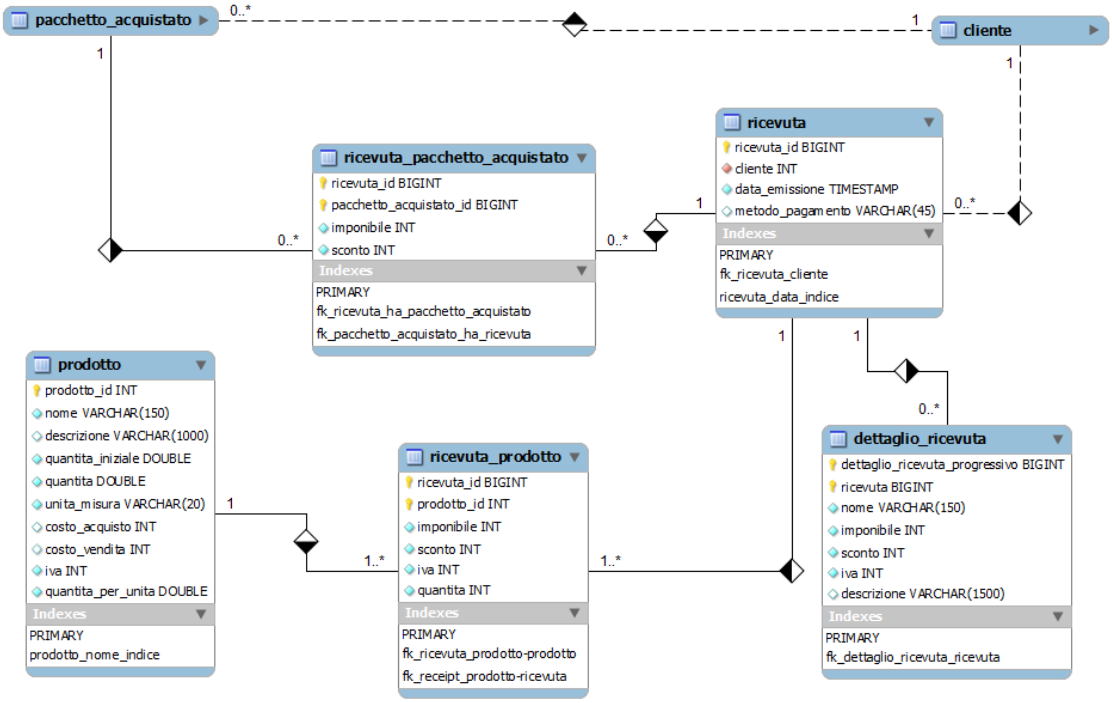


Figura 3-24:Relazione ricevuta,prodotto e relazioni referenzianti

3.4.7 Traduzione delle entità ricevuta e prodotto e delle entità ad esse associate

Si traducono le entità TIPO CARTA FEDELTA' e CARTA FEDELTA' con le relazioni corrispondenti,chiamate per semplicità tipo_carta e carta. Per TIPO CARTA FEDELTA' si introduce un identificativo seriale come chiave primaria, tipo_carta_id.

Le associazioni SCONTO PER relative alle combinazioni prodotto/tipo_carta, prodotto/carta, pacchetto/tipo_carta e pacchetto/carta sono tradotte nelle relative relazioni associazione chiamate rispettivamente prodotto_tipo_carta, prodotto_carta, pacchetto_tipo_carta e pacchetto_carta. L'associazione RICARICA è tradotta nella relazione associazione ricarica avente come chiave primaria una chiave surrogata chiamata ricarica_id e come chiavi esterne le chiavi della carta e ricevuta associata. Ha come attributo "importo" che definisce l'importo della ricarica in centesimi di euro. L'associazione PAGATO IN è tradotta nella relazione pagato_in, avente come chiave primaria una chiave surrogata. Essa permette di pagare in diversi tempi lo stesso pacchetto con la stessa carta fedeltà. Ha come chiavi esterne le chiavi primarie di pacchetto_acquistato e carta e come attributi i due attributi della relazione. Similmente l'associazione ACQUISTATO è tradotta nella relazione corrispondente.

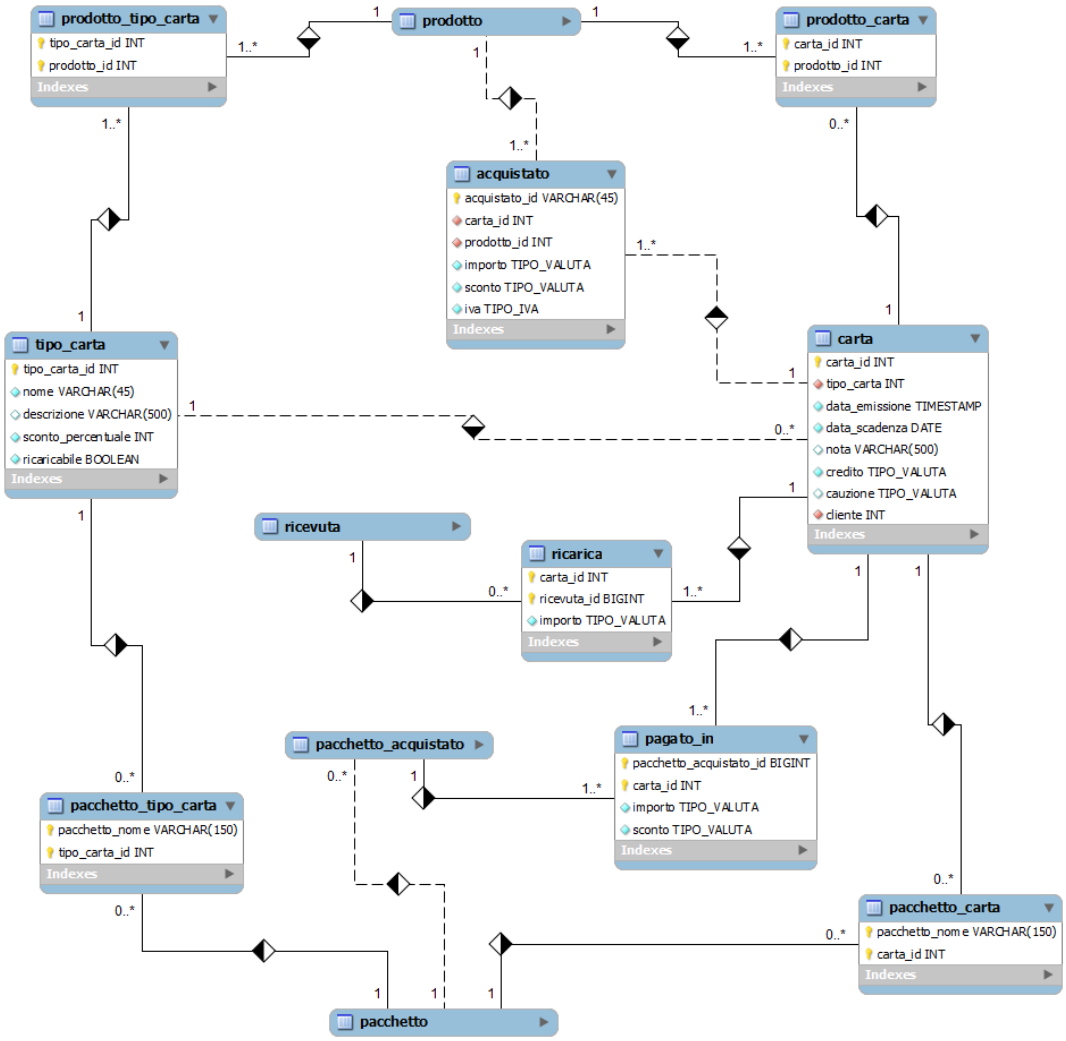


Figura 3-25:Relazioni riguardanti la gestione carte fedeltà

3.4.8 Schema relazionale completo della base di dati

In figura 24 si riporta lo schema relazionale completo della base di dati del centro benessere, da ora chiamata 'centroBenessere'.
Per la sua progettazione si è utilizzato il programma Open Source MySql Workbench [53]. Esso è un IDE visuale per la progettazione dei database che permette di progettare schemi relazionali, sviluppare in SQL e creare, gestire mantenere database MySQL. Esso è il successore DBDesigner 4 ed è supportato dalla community di sviluppo di MySQL.
Nello schema per i vicoli di integrità referenziale tra tabelle si utilizza la notazione referenziale.
Si raffina lo schema relazionale introducendo due nuovi domini:

- TIPO_VALUTA : dominio per gli attributi valuta(importo,sconto). Esso definisce un insieme di valori interi senza segno e concettualmente indica che le valute sono in centesimi.
- TIPO_IVA: indica il tipo di dati per gli attributi iva. Esso è l'insieme di valori interi compreso tra 0 e 100 inclusi.

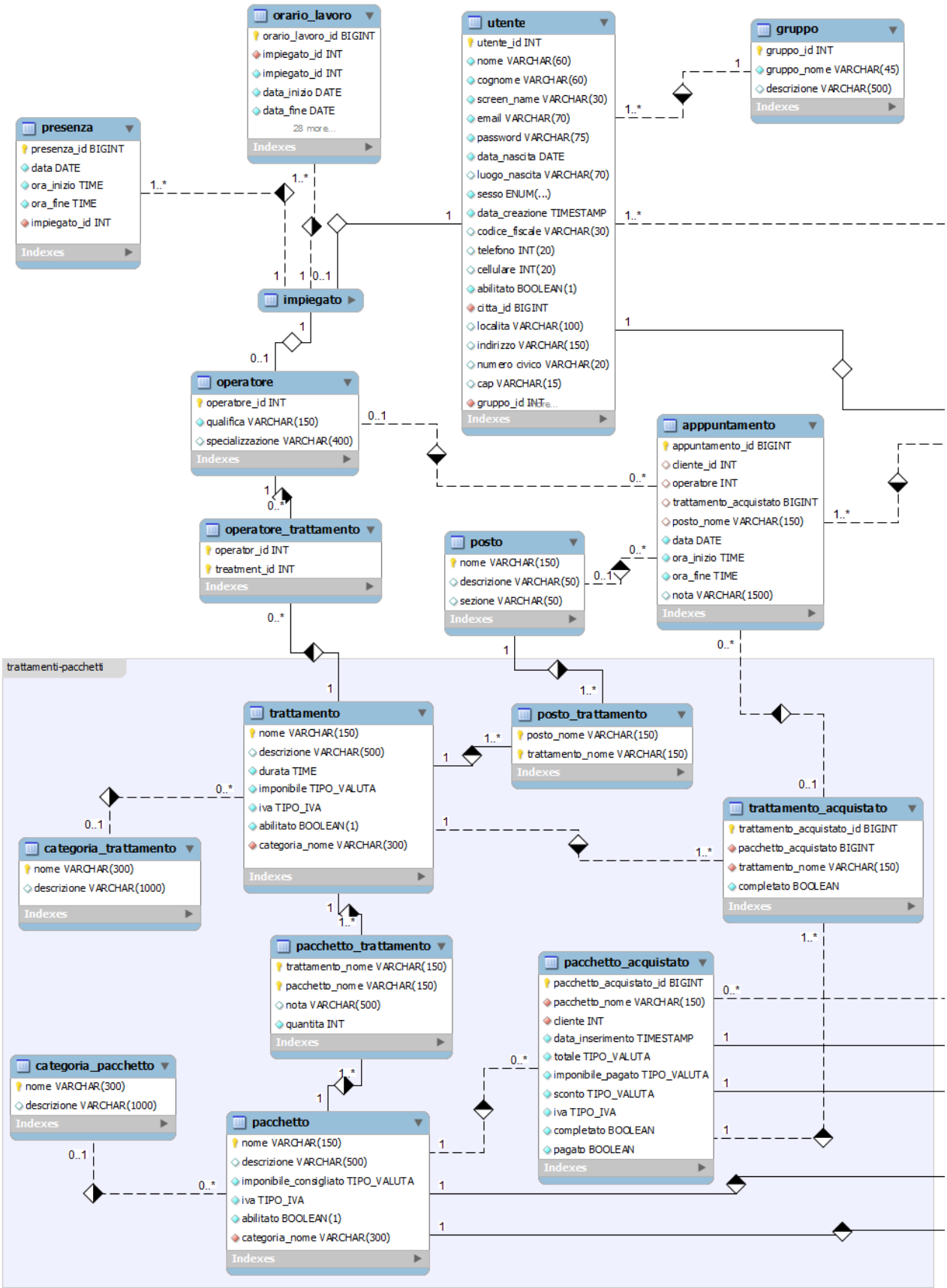


Figura 3-26-a: Schema SQL

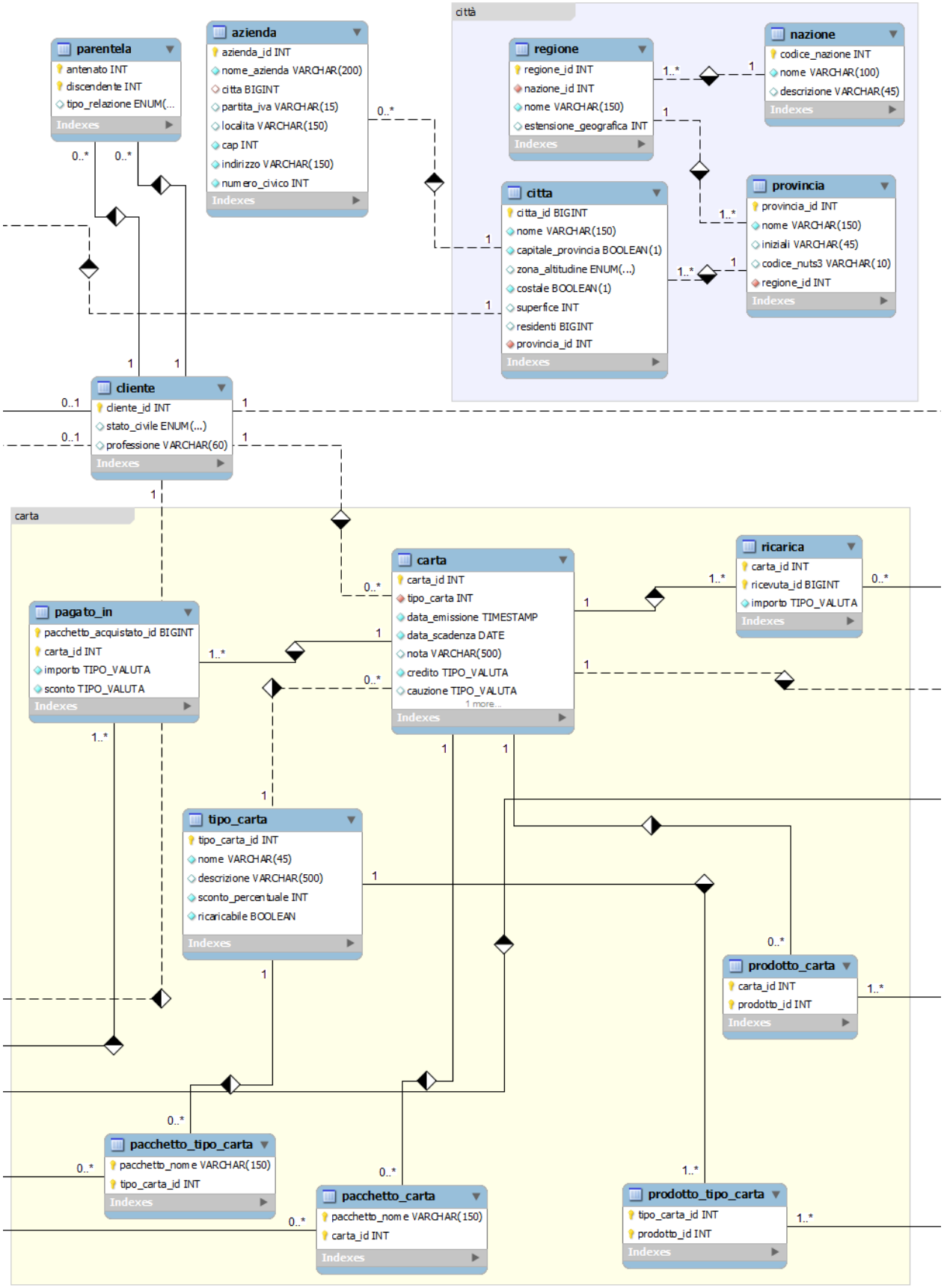


Figura 3-27-b:Schema SQL

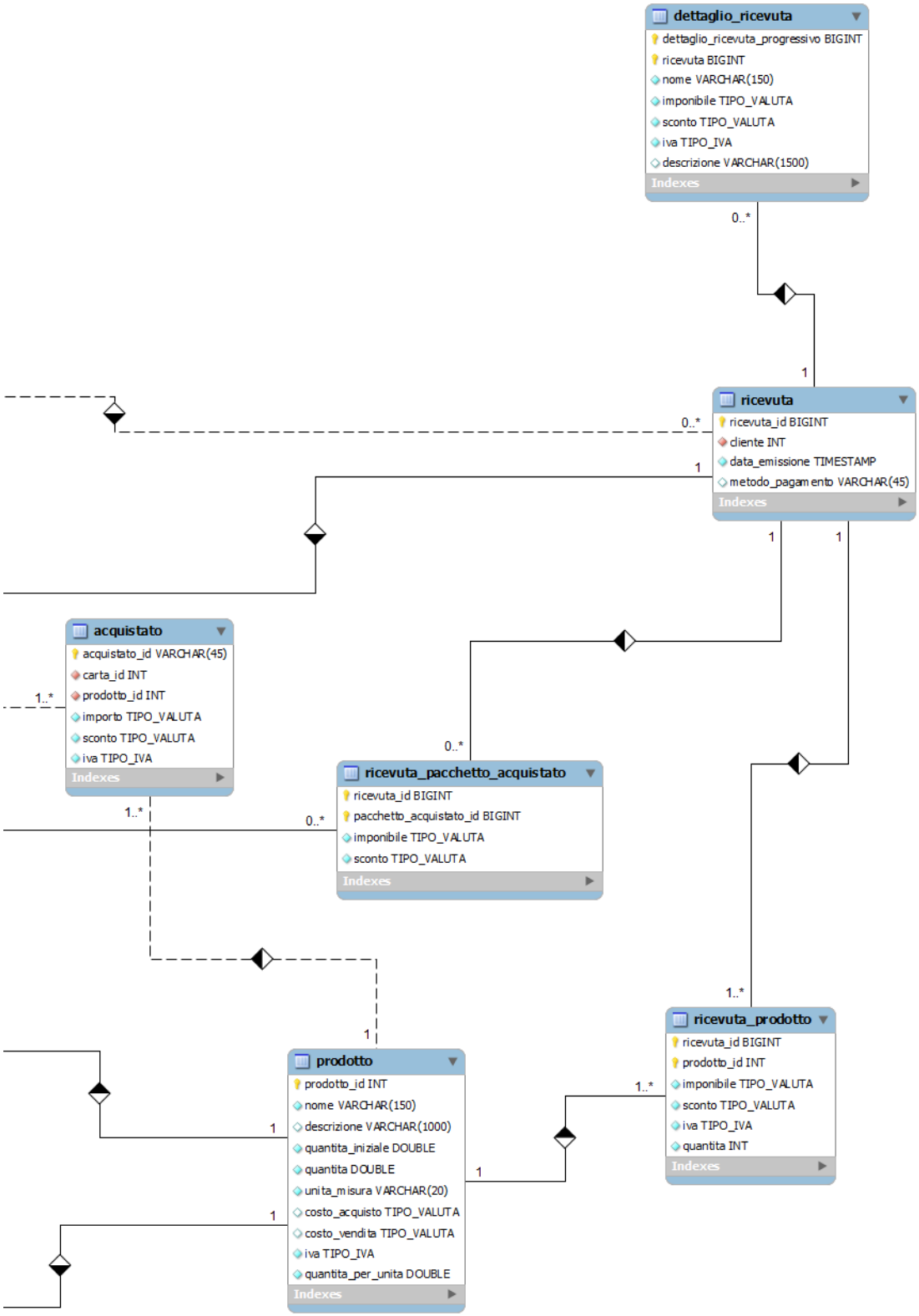


Figura 3-28-c:Schema SQL di AddCenter

3.5 Realizzazione

Per l'implementazione della base di dati si è scelto, come detto, il DBMS 'MySql Server 5.1' e come tool di sviluppo 'MySQL Workbench'. Per la progettazione fisica e definizione della base di dati non è stato tuttavia creato uno script di definizione dello schema SQL. Questo perché per lo sviluppo dell'applicativo è stato utilizzato Java EE che ha come standard una potente tecnologia di Object-Relational Mapping (ORM) [54].
La maggior parte dei sistemi enterprise salva i loro dati in un database relazionale. Questo è il motivo per cui la persistenza, cioè il processo svolto da un'applicazione di salvare e recuperare i dati da una memoria permanente, è diventato un concetto tra i più importanti nello sviluppo delle applicazioni.
ORM è una tecnica di programmazione che favorisce l'integrazione di sistemi software aderenti al paradigma della *programmazione orientati agli oggetti* con sistemi *RDBMS*. E' possibile gestire le istanze del database direttamente sotto forma di oggetti, piuttosto che sotto forma di record di tabelle. Ciò permette una gestione di alto livello e veloce della base di dati all'interno di applicativi sviluppati con un linguaggio di programmazione ad oggetti.
Per la definizione della base di dati è stata quindi utilizzata la mappatura oggetti/relazionale fornita da Java EE mediante il framework JPA, acronimo di Java Persistence API [55]. Mediante JPA è possibile tradurre direttamente gli schemi di relazione definito nel modello relazionale in classi di oggetti. Ciò è molto intuitivo da capire ed interpretare, se si pensa che gli schemi di relazione definiscano la struttura di entità del mondo reale, esattamente come le classi di oggetti. Gli oggetti a loro volta sono istanze di classi, e rappresentano le entità del mondo reale. Essi corrispondono alle tuple di una relazione, le quali definiscono le entità a loro volta. Schema relazionale e schema ad oggetti dopotutto sono atti semplicemente a rappresentare la realtà di interesse, anche se secondo diversi formalismi e vincoli.
In modo non troppo complicato è quindi possibile passare da una rappresentazione all'altra della realtà analizzata, facendo attenzione alla traduzione di quei formalismi non direttamente rappresentabili in un modello o nell'altro.
Mediante la mappatura oggetti/relazionale è possibile definire nelle classi di oggetti come mapparle in relazioni di uno schema relazionale. Tramite speciali annotazioni sugli attributi di una classe , si mappano tali attributi negli attributi corrispondenti di una relazione. Inoltre sempre tramite annotazioni si possono definire le associazioni e in che modo implementare la gerarchia di ereditarietà tra classi, formalismo che non è possibile specificare direttamente mediante il modello relazionale.

Si riportano nel seguito alcuni esempi di come è stato utilizzato JPA per lo sviluppo dell'applicativo AddCenter per capire come è stata definita la base di dati e in modo tale da introdurre anche ,molto ad alto livello, tale tecnologia. Ovviamente per motivi di copyright e spazio non è riportata l'intera definizione della base di dati trattata fino ad ora, ma solo una piccola parte illustrativa del lavoro svolto.

3.5.1 Mappatura della classe Utente

Si illustra di seguito come è stata definita la relazione 'utente' nel modello ad oggetti. E' stata per prima cosa creata la classe User (nell'applicativo è stata utilizzata la lingua inglese per lo sviluppo) e inseriti come attributi tutti gli attributi della relazione utente. In seguito si è proceduto a definire le associazioni con gli altri oggetti.
Dalla riga #1 a #12 si è specificato il package di appartenenza della classe e gli import delle classi e package utilizzati.
In riga #17 l'annotazione @Entity specifica che la classe sopra la quale è apposta deve essere trattata dal container manager come un'entità che può essere memorizzata tramite persisenza del RDBMS [56].
In riga #18 l'annotazione @Table(name = 'user_') specifica il nome della tabella del database relazionale nella quale la classe è mappata [57].Si nota l'utilizzo del carattere di underscore dopo 'user'. Questo è necessario in quanto la parola 'user' è riservata in SQL e non può essere specificata come nome di tabella .
Dalla riga #19 alla riga #20 vi sono le annotazioni che specificano come deve essere mappata nella base di dati relazionale la gerarchia d'ereditarietà (specializzazione) tra le classi che estenderanno User. In questo caso in riga #19 l'annotazione @Inheritance(strategy=InheritanceType.JOINED) specifica che la strategia usata è quella di creare una tabella per ogni classe e di convertire le associazioni di generalizzazione/specializzazione tramite associazioni uno-uno tra queste tabelle [58]. Riga #20 specifica un campo aggiuntivo che verrà utilizzato come attributo di appartenenza di un oggetto specifico della classe a una determinata sottoclasse. Questo è utilizzato solo dal meccanismo di persistenza per velocizzare le operazioni di individuazione della classe corretta di appartenenza. In questo caso il nome dell'attributo discriminatore è 'user_type', di tipo stringa e lunghezza 4 caratteri. L'annotazione @DiscriminatorValue [59] specifica il valore che deve avere l'attributo discriminatore per gli oggetti di classe User, in questo caso si è scelto semplicemente come valore 'user'. In riga #23 si nota che la classe User implementa l'interfaccia Serializable, questo perché tutte le entità devono poter esser serializzate. Da riga #25 a #29 si nota la specificazione dell'attributo userId, che è la chiave primaria identificatrice degli utenti. @Id specifica che è la chiave primaria della relazione, @ GeneratedValue(strategy = GenerationType.IDENTITY) [60] specifica che è un valore identità auto generato dal sistema in modo progressivo, le altre due annotazioni specificano la colonna in cui è mappato, la possibilità che tale campo sia null , in questo caso falsa, e l'opzionalità del campo, sempre falsa in quanto campo chiave. Da riga #31 a riga #82 sono specificati gli

attributi(campi) della classe che sono mappati nei campi della tabella ‘user_’ della base di dati. In ciascuno di essi si specifica mediante l’annotazione @Column [61] il nome della colonna in cui è mappato, se i valori nulli sono consentiti,il vincolo di univocità e nel caso di campi stringa, la lunghezza massima del campo nella base di dati. Si noti che non è necessario specificare il tipo di dati in cui è mappato l’attributo. Questo perché il motore di persistenza analizza il tipo di dati Java e lo traduce automaticamente nel campo più opportuno del database. Vi sono solo alcune eccezioni a riguardo.
La prima si può notare per l’attributo ‘birthDate’ che è di classe Calendar. Come è noto in SQL vi sono tre diversi tipi di campo per un attributo temporale che sono DATE , TIME e TIMESTAMP. Calendar specifica un momento temporale sotto forma di data e ora, ed è quindi necessario sapere se memorizzare nella base di dati sia la data che l’ora o alternativamente una o l’altra. L’annotazione @Temporal quindi determina il tipo di dati nel database. Per ‘birthDate’ si è specificato di memorizzare solo la data.
La seconda eccezione si ha per i tipi di dati enumerativi. Si può notare l’attributo Gender che è enumerativo. E’ necessario dire al motore di persistenza come memorizzare tali valori. L’annotazione @Enumerated(EnumType.STRING) specifica che il valore enumerativo è memorizzato come stringa nella base di dati.
Da riga #85 a riga #87 è definito un campo speciale utilizzato dal motore di persistenza per gestire la concorrenza mediante lock ottimistici. @Version definisce il campo utilizzato per mantenere il numero progressivo della versione dell’oggetto.
Da riga #89 a riga #93 sono specificate le associazioni di User con le altre entità. L’attributo ‘userGroupCollection’ definisce la collezione di oggetti UserGroup associate all’oggetto User. Si ricorda che UserGroup nella base di dati è la relazione associazione tra utenti e gruppi. L’attributo @OneToMany(cascade = CascadeType.ALL, mappedBy = "user”) specifica l’associazione uno-a-molti [62] tra utente e utente_gruppo. Cioè un utente può avere più utente_gruppo e viceversa ciascun utente_gruppo può avere un solo utente associato. Cascade specifica in quale caso intraprendere un azione referenziale innescata quando il gestore delle entità memorizza, modifica o cancella un’entità User. In questo è innescata ogni tipo di azione. Ad esempio quando si rimuove un’ entità User , vengono eliminate tutte le entità UserGroup collegate. L’attributo mappedBy specifica che l’associazione è definita tramite l’attributo user dell’oggetto UserGroup. Si notano da riga #95 a #149 i costruttori e alcuni metodi getter e setter per gli attributi di User.
Il meccanismo di persistenza di Java è molto potente non solo perché permette di mappare classi a tabelle SQL, ma anche perché permette di definire un modello di dominio che non è,come si dice il letteratura, “anemico”. Una semplice mappatura da attributi di una classe a campi di tabelle non consente altro se non il recupero e memorizzazione delle informazioni nella base di dati, e ciò è considerato un modello di dominio “anemico”, cioè impossibilitato a fare altro oltre la banale mappatura. Ma , come si sa, un oggetto Java può fare molto di più che semplicemente memorizzare attributi.
E’ possibile infatti mediante metodi definire il suo comportamento, essi descrivono i servizi e le funzioni che rende disponibili all’esterno. E’ possibile quindi definire per le entità mappate anche un comportamento, e ciò rende il modello di dominio un modello ricco, non anemico.
Esempio di caratterizzazione del comportamento dell’oggetto è data dal metodo setPassword di riga #151.
Nella base di dati infatti le password degli utenti non sono memorizzate mediante testo in chiaro, se così non fosse ciò costituirebbe un rischio per la sicurezza del sistema informatico. Esse sono memorizzate mediante tecnica Hash. Quando l’utente inserisce la sua password , ad essa viene applicata una funzione hash e il valore ottenuto è confrontato con quello memorizzato nel database. Se essi coincidono allora l’utente avrà inserito la sua password correttamente.
Il metodo setPassword allora riceve in ingresso una stringa contente la password in chiaro e ne produce il suo codice hash, infine memorizza esso nella base di dati. Per l’applicativo è stata utilizzata come funzione hash SHA a 384 bit [63] (per offrire un’elevata sicurezza lo si è preferito a SHA standard a 160 bit attaccabile con ‘l’attacco del compleanno’) e il valore dell’hash è memorizzato utilizzando la codifica in Base 64 [64] nella base di dati.
I rimanenti metodi della classe si spiegano da soli.

```
 1 package it.addsource.addcenter.persistence.users;
 2
 3 import it.addsource.addcenter.persistence.geography.Address;
 4 import java.io.Serializable;
 5 import java.util.Date;
 6 import java.util.Set;
 7 import javax.persistence.*;
 8 import java.security.*;
 9 import it.sauronsoftware.base64.*;
10 import java.util.Calendar;
11 import java.util.GregorianCalendar;
12 import java.util.StringTokenizer;
13 /**
14  *
15  * @author Daniele Barilaro < daniele.barilaro@addsource.it >
16  */
17 @Entity
18 @Table(name = "user_")
19 @Inheritance(strategy=InheritanceType.JOINED)
```

```
20 @DiscriminatorColumn(name="user_type",discriminatorType=DiscriminatorType.STRING,length=4)
21 @DiscriminatorValue(value="user")
23 public class User implements Serializable {
24     private static final long serialVersionUID = 1L;
25     @Id
26     @GeneratedValue(strategy = GenerationType.IDENTITY)
27     @Basic(optional = false)
28     @Column(name = "user_id", nullable = false)
29     private Integer userId;
30 
31     @Basic(optional = false)
32     @Column(name = "name", nullable = false,length=40)
33     private String name;
34 
35     @Basic(optional = false)
36     @Column(name = "surname", nullable = false,length=40)
37     private String surname;
38 
39     @Basic(optional = false)
40     @Column(name = "screen_name", nullable = false,length=60,unique=true)
41     private String screenName;
42 
43     @Basic(optional = false)
44     @Column(name = "email", nullable = false,length=60,unique=true)
45     private String email;
46 
47     @Basic(optional = false)
48     @Column(name = "password", nullable = false, length = 75)
49     private String password;
50 
51     @Basic(optional = false)
52     @Column(name = "birth_date", nullable = false)
53     @Temporal(TemporalType.DATE)
54     private Calendar birthDate;
55 
56     @Column(name = "birth_place", length = 100)
57     private String birthPlace;
58 
59 
60     @Basic(optional = false)
61     @Enumerated(EnumType.STRING)
62     @Column(name = "gender", nullable = false, length = 6)
63     private Gender gender;
64 
65 
66     @Basic(optional = false)
67     @Column(name = "creation_date", nullable = false)
68     @Temporal(TemporalType.TIMESTAMP)
69     private Calendar creationDate;
70 
71     @Column(name = "tax_code", length = 30)
72     private String taxCode;
73 
74     @Column(name = "phone",length=30)
75     private String phone;
76 
77     @Column(name = "mobile_phone",length=30)
78     private String mobilePhone;
79 
80     @Basic(optional = false)
81     @Column(name = "enabled", nullable = false)
82     private Boolean enabled;
83 
84 
85     @Version
86     @Column(name="version")
87     private Long version;
88 
89     @OneToMany(cascade = CascadeType.ALL, mappedBy = "user")
90     private Set<UserGroup> userGroupCollection;
91 
92     @OneToMany(cascade = CascadeType.ALL, mappedBy = "user")
93     private Set<Address> addressCollection;
94 
95     public User() {
96     }
97 
```

```
 98    public User(Integer userId, String name, String surname, String screenName, String email, String password, Calendar
birthDate, Gender gender, Calendar creationDate, boolean enabled) {
 99        this.userId = userId;
100        this.name = name;
101        this.surname = surname;
102        this.screenName = screenName;
103        this.email = email;
104        this.password = password;
105        this.birthDate = birthDate;
106        this.gender = gender;
107        this.creationDate = creationDate;
108        this.enabled = enabled;
109    }
110
111    public Integer getUserId() {
112        return userId;
113    }
114
115    public String getName() {
116        return name;
117    }
118
119    public void setName(String name) {
120        this.name = name;
121    }

[..... ]

146
147    public String getPassword() {
148        return password;
149    }
150
151    public void setPassword(String password) throws NoSuchAlgorithmException{
152
153        try {
154          MessageDigest md = MessageDigest.getInstance("SHA-384"); //-384
155          md.reset();
156          //calcolo il message digest
157          byte[] pwdDigest = md.digest(password.getBytes());
158          //lo codifico in base64
159          byte[] baseEncodedPwd = Base64.encode(pwdDigest);
160          this.password = new String(baseEncodedPwd);
161
162        } catch (NoSuchAlgorithmException e) {
163            throw new NoSuchAlgorithmException("impossible to set " +
164                    this + "pwd. " + e.getMessage());
165        }
166
167    }
168
[..... ]
207    public Gender getGender() {
208        return gender;
209    }
210
211    public void setGender(Gender gender) {
212        this.gender = gender;
213    }
214
215
216    public String getGenderChar(){
217        if(gender == null){
218            return null;
219        }else{
220            return gender.toString().substring(0,1);
221        }
222    }
223
224    public void setGenderChar(String chr){
225        if(chr.equals("M")){
226            gender = Gender.MALE;
227        }else if(chr.equals("F")){
228            gender = Gender.FEMALE;
229        }else{
230          throw new InvalidParameterException(chr + " is not a gender character.It must be M or F");
231        }
```

```
232
233     }
234
[..... ]
275
276
277     public Set<UserGroup> getUserGroupCollection() {
278         return userGroupCollection;
279     }
280
281     public void setUserGroupCollection(Set<UserGroup> userGroupCollection) {
282         this.userGroupCollection = userGroupCollection;
283     }
284
285     public Set<Address> getAddressCollection() {
286         return addressCollection;
287     }
288
289     public void setAddressCollection(Set<Address> addressCollection) {
290         this.addressCollection = addressCollection;
291     }
292
293
[..... ]
337
338 }
```

3.5.2 Mappatura della classe Card

Si illustra di seguito come è stata definita la relazione 'Carta' nel modello ad oggetti. Essa corrisponde alla classe Card ed in maniera del tutto simile alla classe User si è proceduto alla mappatura dei campi di Card nei corrispondenti campi della relazione. Si salterà la spiegazione dei concetti introdotti nella sezione precedente , in quanto leggendo il codice si auto illustrativi. Si spiegerà solo la nuova annotazione presente in tale classe @ManyToMany [65].
Tale annotazione serve a definire l'associazione molti-a-molti definita tra due entità.
Per Card viene utilizzata in riga #61 definendo l'associazione molti-a-molti tra Card e Packs e in riga #65 definendo l'associazione molti-a-molti tra Card e Products. In riga #61, mediante l'attributo mappedBy, è specificato che l'associazione è definita dall'attributo 'cardCollection' dell'entità Products.
In riga #65 invece è la classe Card che definisce l'associazione tra le due entità partecipanti. Mediante l'annotazione @JoinTable viene definita la tabella di JOIN (detta anche tabella associazione) nella quale sono memorizzate le coppie di chiavi che definiscono l'associazione. La tabella di join è nominata 'card_discount_product' e possiede due campi chiavi esterne, ciascuno dei quali referenzia le chiavi primarie delle due tabelle partecipanti all'associazione.Per l'attributo joinColumns l'annotazione @JoinColumn specifica che la chiave esterna che collega Card è chiama 'card_id' e fa riferimento al campo 'card_id' di Card. Per l'attributo inverseJoinColumns l'annotazione @JoinColumn specifica che la chiave esterna che collega Product è chiama 'product_id' e fa riferimento al campo 'product_id' di Product.

```
 1
 2 package it.addsource.addcenter.persistence.card;
 3 import it.addsource.addcenter.persistence.inventory.Product;
 4 import it.addsource.addcenter.persistence.*;
 5 import it.addsource.addcenter.persistence.users.Customer;
 6 import java.io.Serializable;
 7 import java.util.Date;
 8 import java.util.Set;
 9 import javax.persistence.*;
10
11 import it.addsource.addcenter.persistence.inventory.Pack;
12
13 /**
14  *
15  * @author Daniele Barilaro < daniele.barilaro@addsource.it >
16  */
17 @Entity
18 @Table(name = "card")
20 public class Card implements Serializable {
21     private static final long serialVersionUID = 1L;
22     @Id
23     @GeneratedValue(strategy = GenerationType.IDENTITY)
24     @Basic(optional = false)
```

```
25    @Column(name = "card_id", nullable = false)
26    private Integer cardId;
27
28    @Basic(optional = false)
29    @Column(name = "start_date", nullable = false)
30    @Temporal(TemporalType.DATE)
31    private Date startDate;
32
33    @Basic(optional = false)
34    @Column(name = "expiry_date", nullable = false)
35    @Temporal(TemporalType.DATE)
36    private Date expiryDate;
37
38    @Basic(optional = false)
39    @Column(name = "note", nullable = false, length = 500)
40    private String note;
41
42    @Basic(optional = false)
43    @Column(name = "credit", nullable = false)
44    private int credit;
45
46    @Basic(optional = false)
47    @Column(name = "used_credit", nullable = false)
48    private int usedCredit;
49
50    @Column(name = "bail")
51    private Integer bail;
52
53    @Basic(optional = false)
54    @Column(name = "discount_all_packages_in_type", nullable = false)
55    private boolean discountAllPackagesInType;
56
57    @Basic(optional = false)
58    @Column(name = "discount_all_products_in_type", nullable = false)
59    private boolean discountAllProductsInType;
60
61    @ManyToMany(mappedBy = "cardCollection")
62    private Set<Pack> packageCollection;
63
64    @JoinTable(name = "card_discount_product", joinColumns = {@JoinColumn(name = "card_id",
          referencedColumnName = "card_id", nullable = false)}, inverseJoinColumns =
        {@JoinColumn(name = "product_id", referencedColumnName = "product_id", nullable = false)})
65    @ManyToMany
66    private Set<Product> productCollection;
67
68
69    @OneToMany(cascade = CascadeType.ALL, mappedBy = "card")
70    private Set<BoughtPackCardReceipt> boughtPackageCardReceiptCollection;
71
72    @JoinColumn(name = "customer", referencedColumnName = "user_id", nullable = false)
73    @ManyToOne(optional = false)
74    private Customer customer;
75
76    @JoinColumn(name = "card_type", referencedColumnName = "card_type_id", nullable = false)
77    @ManyToOne(optional = false)
78    private CardType cardType;
79
80    @OneToMany(cascade = CascadeType.ALL, mappedBy = "card")
81    private Set<ProductCardReceipt> productCardReceiptCollection;
82
83    @OneToMany(cascade = CascadeType.ALL, mappedBy = "card")
84    private Set<CardRecepit> cardRecepitCollection;
85
86    public Card() {
87    }
88
89    public Card(Integer cardId) {
90        this.cardId = cardId;
91    }
92
93    public Card(Integer cardId, Date startDate, Date expiryDate, String note, int credit, int usedCredit, boolean
discountAllPackagesInType, boolean discountAllProductsInType) {
94        this.cardId = cardId;
95        this.startDate = startDate;
96        this.expiryDate = expiryDate;
97        this.note = note;
98        this.credit = credit;
```

```
99        this.usedCredit = usedCredit;
100       this.discountAllPackagesInType = discountAllPackagesInType;
101       this.discountAllProductsInType = discountAllProductsInType;
102   }
103
104   public Integer getCardId() {
105       return cardId;
106   }
107
108   public void setCardId(Integer cardId) {
109       this.cardId = cardId;
110   }
[..... ]
238
239   @Override
240   public boolean equals(Object object) {
241
242       if (!(object instanceof Card)) {
243           return false;
244       }
245       Card other = (Card) object;
246       if ((this.cardId == null && other.cardId != null) || (this.cardId != null && !this.cardId.equals(other.cardId))) {
247           return false;
248       }
249       return true;
250   }
251
252   @Override
253   public String toString() {
254       return "it.addsource.addcenter.persistence.Card[cardId=" + cardId + "]";
255   }
256
257 }
258
259
```

CAPITOLO 4

4 STRUTTURA DELL'APPLICATIVO

Nel capitolo precedente è stata trattata la progettazione della base di dati per l'applicativo, base di dati generale per i centri di cura del corpo. Tramite essa si è soddisfatta la settima specifica, di progettazione di un software generalmente valido per i centri di cura del corpo, per quanto riguarda lo strato di persistenza, mentre successivamente si discuterà il suo soddisfacimento nella logica applicativa.
Nel capitolo corrente si descriverà la struttura ad alto livello dell'applicativo, descrivendo i moduli in cui è composto e con quali tecnologie e strategie è stato sviluppato ciascuno di essi. Nella descrizione emergerà in modo naturale come determinate specifiche ,descritte nel primo capitolo, sono state soddisfatte.
Si inizierà descrivendo i concetti generali alla base dei software gestionali, per poi spiegare come essi sono stati riassunti, analizzati e implementati per AddCenter in modo tale da sviluppare un framework di aiuto allo sviluppo di software gestionali RIA.

4.1 CONCETTI BASE DEI SOFTWARE GESTIONALI

Ogni programma gestionale è suddiviso in più sezioni, ognuna delle quali permette di amministrare dati o processi dell'azienda a cui sono associate [66]. A prescindere da come è strutturata l'azienda e il software è possibile trovare degli aspetti comuni ad ogni software gestionale. E' possibile, in linea di principio, dividere i software gestionali in:

- Software di Contabilità (ERP[1])
- Software per il magazzino (MRP[2])
- Software per la produzione (ERP)
- Software per il budgeting (ERP e BI[3])
- Software di gestione a analisi finanziaria (BI)

Ognuno di questi deve gestire entità all'interno dell'azienda che rappresenta; entità che possono esser concrete (ad esempio i dipendenti o i prodotti in magazzino) o astratte (ad esempio i gruppi in cui sono suddivisi i dipendenti).
Esempi di gestione di entità:

- Gestione dipendenti
- Gestione clienti
- Gestione fornitori materie prime e semilavorati
- Gestione prodotti di magazzino
- Le entità descritte nel capitolo precedente sono un buon esempio di entità controllate da un gestionale.

Inoltre devono gestire le relazioni che intercorrono tra le entità nei modi e tempi adeguati.
Esempi di relazioni che intercorrono tra entità sono:

- Relazione di gerarchia tra dipendenti (capi progetto, dipendenti, dirigenti, ecc)
- Relazione tra prodotti e fornitori dei prodotti
- Relazione tra clienti e fatture emesse
- Relazione tra appuntamenti e clienti
- In un gestionale per officina meccanica, la relazione tra componenti meccanici e automobili
- Relazione tra posizioni in magazzino e prodotti
- Relazione tra ricevute emesse e clienti

Inoltre tali software devono gestire i processi e flussi informativi interni all'azienda.
Esempi di gestione di flussi informativi sono:

- Pianificazione della produzione in base alle risorse disponibili (disponibilità di magazzino, disponibilità di forza lavoro, dimensionamento catena di produzione).
- Emissione degli ordini d'acquisto ai fornitori e loro controllo(conferma,rifiuto o modifica dell'ordine da parte del fornitore) seguiti dalla gestione degli arrivi degli ordini in magazzino e pagamento del fornitore.

[1] ERP è l'acronimo di Enterprise Resource Planning, cioè pianificazione delle risorse aziendali
[2] MRP è l'acronimo di Materials Requirements Planning, cioè pianificazione del fabbisogno di materiali
[3] BI è l'acronimo di Buisness Intelligence, cioè intelligenza del commercio

- Gestione del ciclo di lavorazione di un determinato prodotto governando i flussi di informazione tra le varie fasi del ciclo.
- In un applicativo di e-commerce acquisto della merce da parte di un cliente, che viene inviata ,dopo il pagamento della stessa, mediante l'emissione di un ordine di spedizione.
- In una centrale del pronto soccorso, smistamento delle chiamate in arrivo agli operatori disponibili e gestione delle risorse disponibili (ambulanze, paramedici, medici) in modo che l'operatore possa avviare la gestione dell'emergenza nel modo più veloce possibile e con le risorse ottimali per farlo.

Infine spesso i gestionali devono analizzare i dati che possiedono, ai fini riassuntivi, statistici ed economici.
Esempio di analisi dei dati:

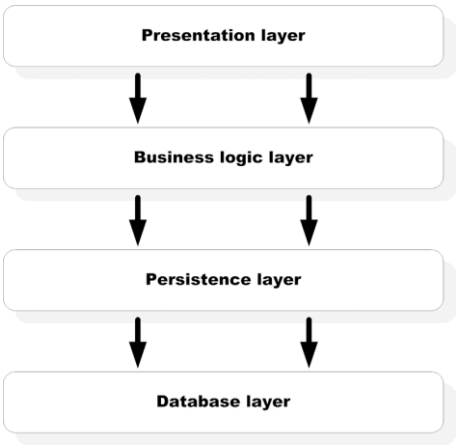
- Creazione di statistiche sulle fatture emesse ad incasso immediato e posticipato.
- Creazione di statistiche sul tempo di rotazione del magazzino
- Analisi della qualità dei prodotti realizzati
- Analisi del tempo di rottura medio di un prodotto
- Analisi dei problemi di maggior rilievo riscontrati nei prodotti ricevuti in assistenza
- Determinazione del budget dell'impresa
- Trend dei prodotti più venduti in un determinato periodo

In tutti i compiti dei software gestionali si notano,come detto, caratteristiche comuni. E' possibile ricondurre tutti questi software ai sistemi informatici[4].
E' perciò possibile analizzarli mediante la teoria dei sistemi informatici e dei DBMS(DataBase managment system).
In particolare è comune sviluppare le applicazioni di questo genere seguendo l'architettura a tre livelli descritta nel paragrafo 2.2.

Tutti i software presi in considerazione avranno quindi un database che memorizza le entità , le relazioni tra le entità e i vincoli che queste devono rispettare (mediante un DBMS di solito relazionale [(5)] o ad oggetti [(6)]), un linguaggio per gestire la persistenza dei dati (comunemente SQL o nel caso di JAVA il JPQL [(7)]) e dei metodi per garantire le comunI operazioni sui dati garantendo la transazionalità delle operazioni.

Sopra lo strato di persistenza si trova la logica per eseguire le operazioni sui dati, tale logica è comunemente detta logica di business , in inglese *Business Logic* [(8)].

L'ultimo livello è quello di presentazione dei dati, strettamente legato all'interfaccia grafica utente. Tale livello chiama le funzioni del livello di Business Logic per eseguire le operazioni sui dati.
Nella seguente immagine si possono osservare i livelli che costituiscono un'applicazione enterprise:



La maggior parte delle applicazioni enterprise hanno quattro livelli.
1) Il livello di presentazione. È l'interfaccia utente e può essere sia un broswer che un'applicazione desktop.
2) Il livello di business logic definisce le regole di business.
3) Il livello di persistenza ha a che fare con le interazioni con il database.
4) Il livello di database consiste di un database (comunemente relazionale) che memorizza le entità.

Figura 4-1:Architettura a tre livelli

[4] Per sistema informatico si intende un insieme di computer, composti da hardware e software che elaborano dati e informazioni per restituire altri dati ed informazioni utili
[5] DBMS che utilizza come modello dei dati il modello relazionale
[6] DBMS che utilizza come modello dei dati il modello ad oggetti
[7] JPQL: Java Persistence Query Language, il linguaggio java per l'interrogazione del database ad oggetti dello strato di persistenza
[8] Con il termine Business Logic ci si riferisce a tutta quella logica applicativa che rende operativa un'applicazione. È un termine largamente utilizzato nella progettazione del software per individuare un componente software, un *layer* (o *tier*, cioè livello) di una architettura software, ecc. La business logic è spesso associata ad architetture software di tipo three-tier.

4.2 Componenti comuni dei software gestionali

Dal punto di vista dell'utente , esso si troverà sempre ad eseguire una serie di operazioni comuni sui dati, in particolare:

-  Creazione di una nuova entità (ad esempio può essere creata una fattura, inserito nel database un nuovo dipendente, cliente o fornitore ecc.)
-  Modifica di un'entità (modifica del recapito di un dipendente, modifica della quantità in magazzino di un determinato prodotto ecc.)
-  Eliminazione di un'entità(eliminazione di un cliente dal database, rimozione di un ordine al fornitore errato o rifiutato ecc)
-  Ricerca di un'entità all'interno del database(ricerca di tutti i clienti nati prima del 1/1/1970 per mandare a questi un determinato tipo di offerte, ricerca di tutti gli operai che hanno fatto più di 50 giorni di malattia in un anno per controlli medici ecc.)

In generale quindi sarà necessario:

-  Visualizzare le istanze d'entità per tipo(ad esempio la visualizzazione dei prodotti, dei clienti, dei fornitori, dei trattamenti)
-  Nelle varie sezioni avere una ricerca per selezionare solo le entità i cui attributi soddisfano particolari condizioni(ad esempio cercare i prodotti con un livello di riordino [9]minore di 2 o cercare gli utenti con debiti non saldati)
-  Un editor per ciascun insieme d' entità, che permetta di creare nuove istanze(ad esempio un nuovo prodotto) , modificarle o eliminarle.

Inoltre nei software gestionali multiutente è consigliabile o necessario controllare l'accesso degli utenti e le autorizzazioni di questi ad eseguire operazioni e visualizzare certi dati. Ad esempio un dirigente o capo personale può esser autorizzato a visualizzare e modificare lo stipendio di un dipendente, ma quest'ultimo è autorizzato solo alla visualizzazione. Analogamente un normale dipendente può accedere al software gestionale ed è autorizzato a visualizzare solo i dati che lo riguardano, mentre un segretario può invece esser in grado di visualizzare tutti gli utenti ed effettuare alcune operazioni su questi, ad esempio la modifica di alcune anagrafiche.
Quindi in una stessa sezione dell'applicativo un utente può visualizzare dei dati che un altro utente non può.

Per ottenere ciò si possono utilizzare diverse tecniche tra le quali:

- Creare sezioni separate per ciascuna categoria di utenti, ad esempio un cliente potrà accedere alla sue ricevute tramite una pagina web progettata appositamente che le visualizza in sola lettura, mentre un segretario potrà accedere a tutte le ricevute emesse tramite una pagina web che permette di effettuare l'inserimento e la visualizzazione di tutte le ricevute.
- All'ingresso di ogni sezione si ha il controllo delle credenziali dell'utente e in base a queste si filtrano i dati visualizzati.
- Per ogni dato si verifica se l'utente è autorizzato a visualizzarlo, ad esempio il campo "note di merito" del profilo di un impiegato può esser nascosto ad esso, ma accessibile ai dirigenti.

[9] Livello di riordino: numero di articoli di prodotto in magazzino sotto il quale si deve eseguire una nuovo ordine.

4.3 Esempio di software gestionale

Si riportano alcune immagini di esempio per mostrare la struttura dei software gestionali, introdotti nel paragrafo precedente, tratte da Zoho© [67] (10), un CRM commerciale.

Nella Figura 4-2: Esempio di anagrafica prodotti, si osserva l'anagrafica prodotti di un software CRM. Si nota che i prodotti sono riportati in una tabella che contiene i campi più significativi, al fine di visualizzarli facilmente. Nell'esempio in tabella vi sono i campi nome, codice e prodotto attiva(che definisce se il prodotto è abilitato all'uso nel sistema).

Figura 4-2: Esempio di anagrafica prodotti

In Figura 4-3 un esempio di editor prodotti in stato di visualizzazione. Nell'editor, a differenza della tabella anagrafica, sono riportati tutti gli attributi del prodotto.

Figura 4-3: Esempio di editor per prodotti in visualizzazione

In Figura 4-4 l'editor prodotti in stato di modifica. In esso tutti i campi sono editabili mediante caselle di testo, liste, valori vero-falso ecc.

[10] Zoho CRM è un programma di gestione delle relazioni coi clienti sviluppato da Zoho Inc. Si può trovare al sito

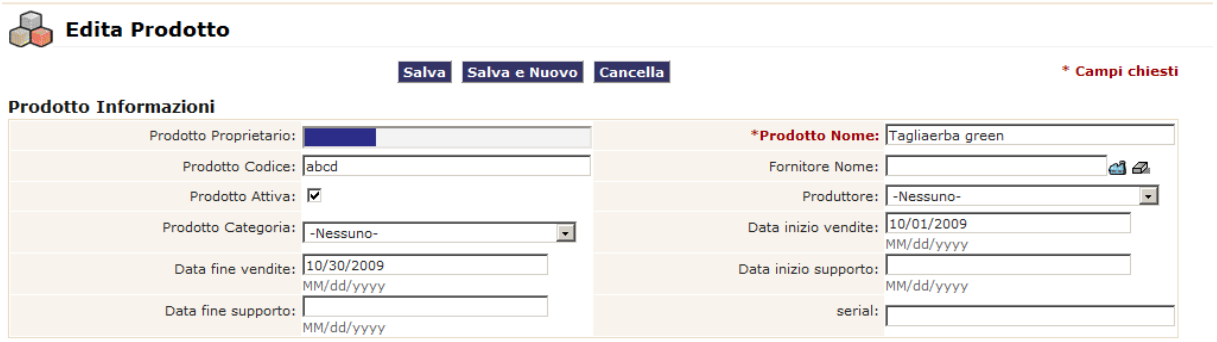


Figura 4-4:Esempio di editor per prodotti in modifica

In Figura 4-5 è riportato un esempio di ricerca del cliente che contiene “Mario” nel nome. Si nota vicino all’anagrafica un campo per effettuare la ricerca dei clienti.

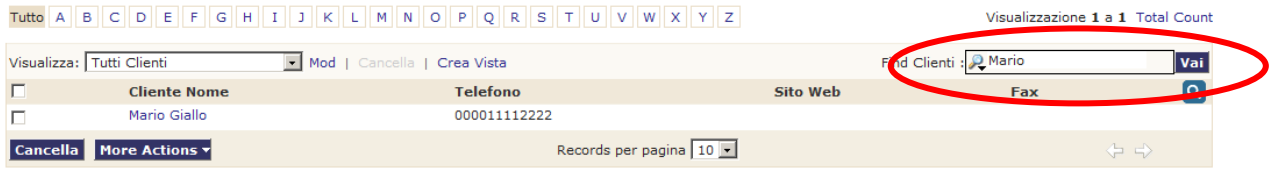


Figura 4-5: Esempio di ricerca. Si selezionano i clienti di nome Mario.

4.4 Concetti progettuali alla base di AddCenter

Dall'analisi è maturato lo sviluppo di AddCenter, che si è posto i seguenti obiettivi principali:

- Creazione di un sistema standard per la visualizzazione degli oggetti di un tipo d'entità in modo indipendente dal loro tipo. Esso può essere qualunque, ad esempio un prodotto, un utente o una fattura, ma l'interfaccia grafica ed il metodo di presentazione dei dati all'utente è comune per ognuna di queste categorie. Cambieranno solo gli attributi specifici da visualizzare a seconda dell' entità. Ciò corrisponde, in termini pratici, ad avere in ogni sezione una tabella per gli oggetti di quel tipo (spesso chiamata anagrafica) , che differisce dalle tabelle delle altre sezioni per il tipo di attributi presentati e, banalmente, dall'entità visualizzata. Tale tabella potrà anche contenere una selezione o proiezione dei dati a seconda delle autorizzazioni dell'utente che vi accede e dal filtro di ricerca impostata su di essa. Esempio di proiezione è l'esclusione da un'entità prodotto del costo di acquisto per visualizzare al cliente solo il listino col prezzo di vendita. Esempio di filtro può essere la selezione dei soli prodotti di categoria 'crema corpo'.

- Creazione di un sistema standard di ricerca. La ricerca dovrà esser effettuata sempre con le medesime modalità indipendentemente dal tipo di entità cercata. Essa deve essere facilmente comprensibile all'utente, ma fornire allo stesso tempo un livello di precisione tale da poter formulare interrogazioni dettagliate. Per questo si è deciso di costruire il sistema di ricerca sopra al linguaggio JPQL, il linguaggio di interrogazioni dello strato di persistenza del Java, fornendo un interfaccia grafica che permetta di specificare interrogazioni che verranno direttamente mappate in una interrogazione JPQL.

- Creazione di un sistema standard per innestare gli editors per le varie classi di oggetti. Gli editors dovranno esser sviluppabili in modo a se stante, indipendente dalla sezione in cui verranno utilizzati, concentrando la progettazione solo sulle operazioni che devono eseguire sull'entità gestita. Inoltre avranno a disposizione un API per dialogare con il gestionale. Il gestionale comunicherà a questi l'entità da editare e le operazioni che si possono effettuare dopo l'editing (salvataggio, annulla, eliminazione).

- Un sistema comune per gestire gli errori e le eccezioni che eviti il blocco dell'applicativo e che permetta di comunicare all'utente l'eventuale problema riscontrato. In particolare si sviluppano separatamente dal framework gli editors, i quali possono essere soggetti ad errori logici che devono essere catturati in maniera efficiente dal framework, senza causare il malfunzionamento del resto dell'applicativo.

- Un sistema di help comune alle varie sezioni, associabile a queste includendo normali file XHTML contenenti la pagina di supporto. L'help delle sezioni potrà essere scritto separatamente ed integrato in esse facilmente.

- Un metodo comodo di selezione delle entità , utile in particolare per la creazione di relazioni. Ad esempio nella creazione di una ricevuta si ha la necessità di selezionare i prodotti acquistati visualizzando la sezione prodotti, selezionandoli e ritornando nell'editor della ricevuta. Altro esempio è la creazione di un pacchetto trattamenti, nella quale è necessario selezionare i singoli trattamenti da includere in esso, visualizzando l'anagrafica trattamenti, e successivamente tornando all'editor del pacchetto per completare la creazione.

4.5 *L'interfaccia grafica di AddCenter*

La via più semplice per capire il funzionamento di AddCenter è quella di osservare il modo d'uso dell'interfaccia grafica dal punto di vista dell'utente.
L'interfaccia grafica è suddivisa in "**viste**", ognuna contenente un dato tipo di anagrafica. L'anagrafica non è altro che "l'elenco" degli oggetti di una data classe(istanze di una data entità). Le viste sono costruite mediante il sistema standard di visualizzazione delle entità di AddCenter, che integra al suo interno la ricerca,l'help, il sistema per innestare gli editors e quello per gestire le eccezioni.

In *Figura 4-6: Anagrafica clienti* è riportata una vista di AddCenter, specializzata per la visualizzazione e gestione dell'anagrafica Clienti.

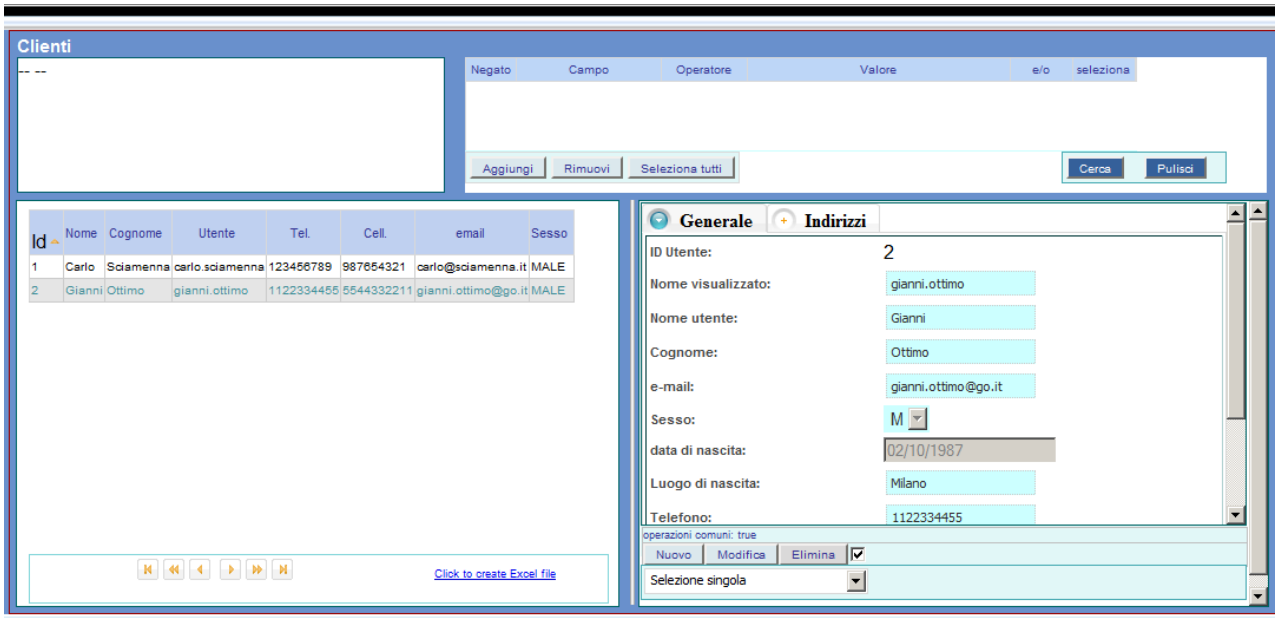


Figura 4-6: Anagrafica clienti

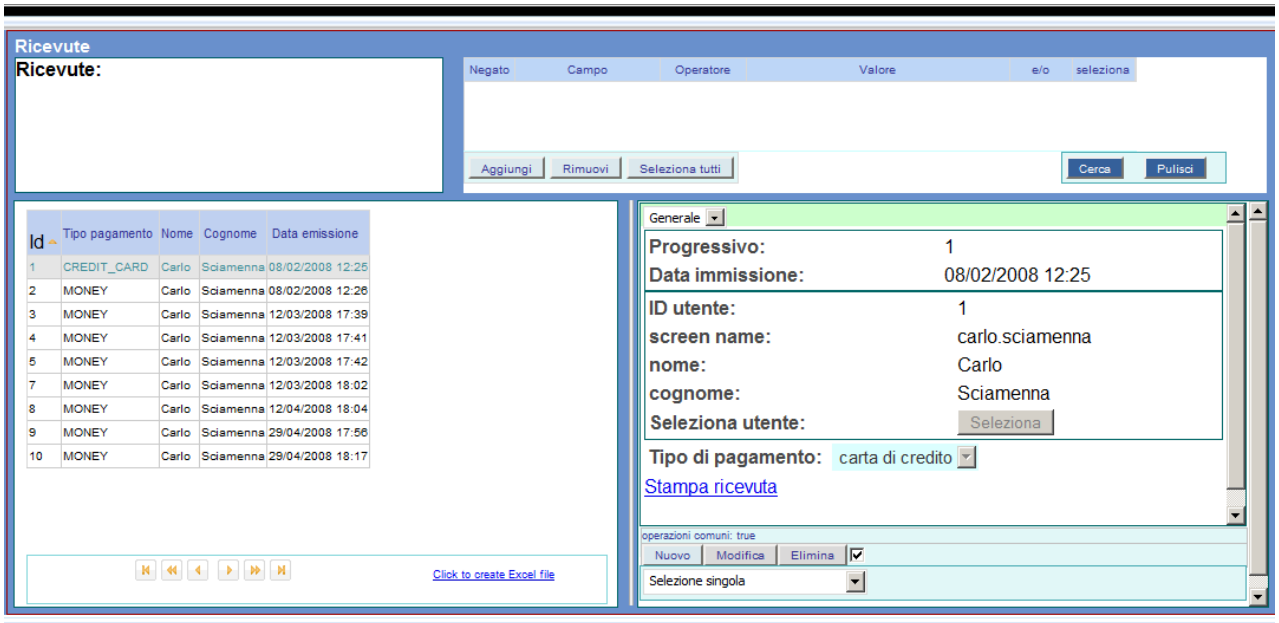


Figura 4-7: Vista ricevute

In *Figura 4-7: Vista ricevute* è riportata la vista per la gestione delle ricevute.
Si può osservare dalle due figure la struttura delle viste di AddCenter. Esse forniscono in alto una sezione orizzontale per il titolo della vista , ad esempio "Ricevute" o "Clienti", sotto ad esso vi sono due rettangoli, il primo sulla sinistra è una sezione che fornisce un'eventuale descrizione aggiuntiva della vista. Ciò può essere utile, ad esempio, se si vuole informare l'utente di eventuali restrizioni sulla visualizzazione delle entità, cioè si riportano in linguaggio facilmente comprensibile i filtri applicati alla visualizzazione lato applicativo. Un cliente è autorizzato a visualizzare solo le sue ricevute, quindi in tale sezione può essere riportato il filtro "Ricevute emesse al cliente X".
Tali filtri sono ovviamente specificati lato applicativo e non sono modificabili dall'utente.
Il secondo riquadro sulla destra contiene la sezione per la ricerca delle entità specificabile dall'utente del sistema. Si forniranno in seguito maggiori dettagli su tale sezione.
In basso a sinistra è presente il riquadro che contiene la tabella dell'anagrafica per gli oggetti della classe specificata per vista. Si nota in esso la tabella in alto ed in basso i pulsanti per la navigazione dell'anagrafica a pagine. A destra dei pulsanti vi è un link per scaricare l'anagrafica sotto formato di file Excel (.xls).
L'ultimo riquadro,in basso a sinistra, è il riquadro dell'editor delle entità. Per ogni vista si può innestare un editor appositamente sviluppato. Nelle due immagini si può osservare l'editor per i clienti e quello per le ricevute.

A sinistra dell'interfaccia è presente la barra dei menù, organizzata mediante una struttura ad albero. Il menù ha presente varie voci e tra esse ci sono quelle per selezionare le viste.
In Figura 4-8:Barra dei menù è riportato il menù.
Ad esempio clienti e ricevute/giorno sono viste entità, mentre ricevute/statistiche visualizza la sezione per le statistiche delle ricevute.

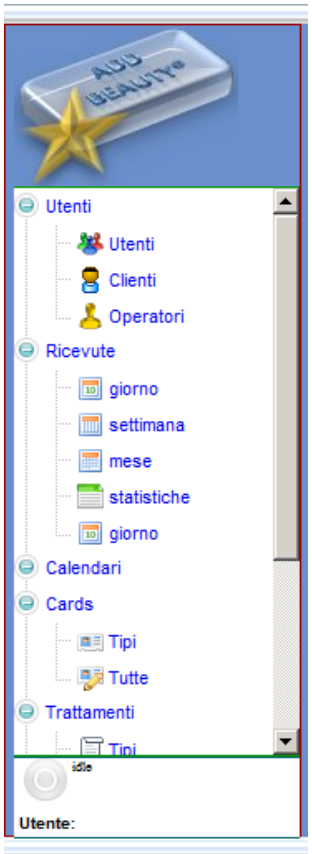


Figura 4-8:Barra dei menù

4.5.1 Editor

Un editor è un componente software che permette l'inserimento,modifica o eliminazione delle informazioni. AddCenter fornisce una API per la creazione ed innesto di editor al suo interno, in modo tale che essi siano sviluppati con una metodologia standard.
In Figura 4-9: Editor delle categorie trattamento , c'è un semplice esempio di editor per creare, modificare ed eliminare le categorie trattamento. E' composto da tre campi, il primo, non modificabile, è l'identificativo (id) della categoria, il secondo è il nome della categoria e il terzo campo è una descrizione della categoria.

Sotto l'editor si notano tre pulsanti, chiamati "Nuovo", "Modifica" ed "Elimina", essi fanno parte del gestore dell'editor e servono rispettivamente per creare una nuova entità e modificarne ed eliminarne una già esistente.

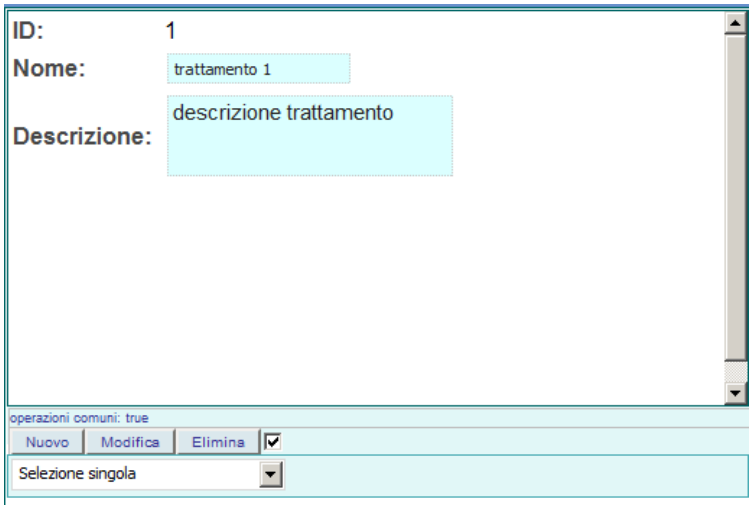


Figura 4-9: Editor delle categorie trattamento

In Figura 4-10 è mostrata la sezione per l'inserimento manuale delle voci di dettaglio nell'editor delle ricevute da emettere ai clienti. Si nota la tabella contenente le voci di dettaglio, in questo caso due, 'crema corpo' e 'crema viso', sotto ad essa i pulsanti per la modifica delle voci ed infine il riquadro riassuntivo degli importi.

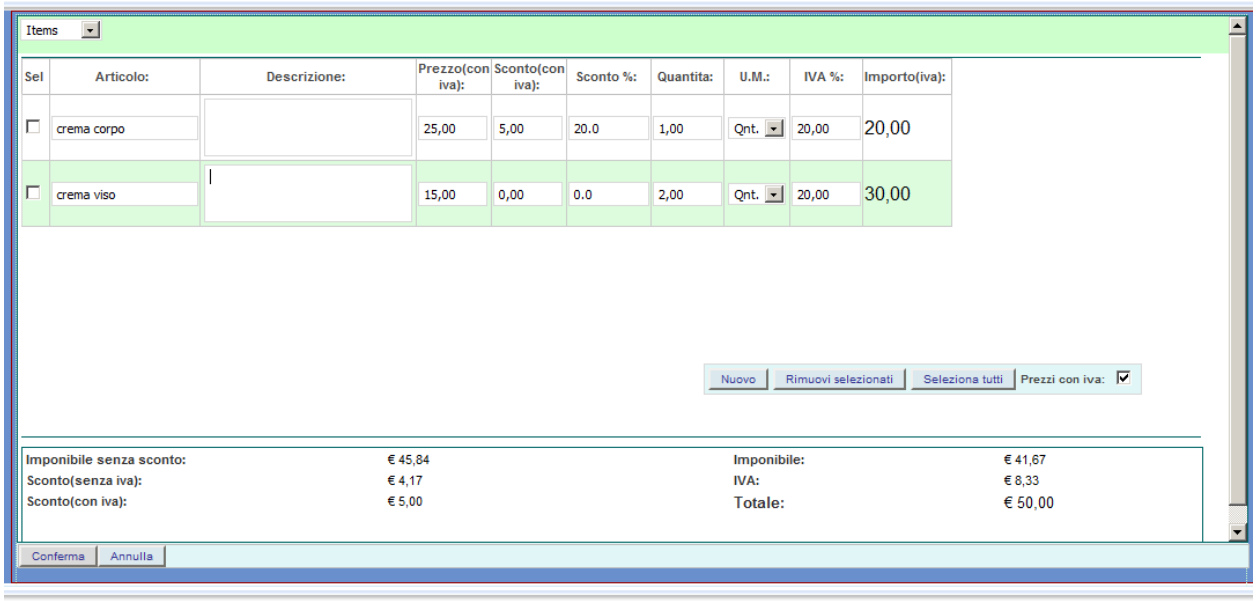


Figura 4-10:Voci di dettaglio manuali dell'editor ricevute.

4.5.2 Ricerca

AddCenter fornisce un sistema standard per la ricerca delle istanze d'entità nelle viste. In ogni vista, come prima accennato, è presente una parte dedicata alla ricerca. Tale ricerca permette di specificare con un alto livello di dettaglio i criteri di filtraggio delle entità. Essa infatti è praticamente un'interfaccia grafica per specificare interrogazioni in JPQL, il linguaggio Java per la creazione di query per il suo strato di persistenza.

In Figura 4-11 è riportata la sezione di ricerca per la vista ricevute. In essa si notano due criteri di ricerca. Il primo specifica che si vogliono selezionare le ricevute il cui cliente ha cognome uguale a *Belli*. Si nota la colonna per la selezione del campo,in questo caso 'customer.surname', quella per selezionare l'operatore di uguaglianza e quella per scrivere il valore, in questo caso *Belli*.
La colonna e/o serve per specificare il connettivo logico con il criterio successivo. In questo caso è presente 'E' per specificare che le entità devono soddisfare entrambi i criteri per essere selezionate.
Il secondo criterio specifica che la data di emissione della ricevuta deve essere precedente(minore) al giorno 01/01/2009 ore 00:00.

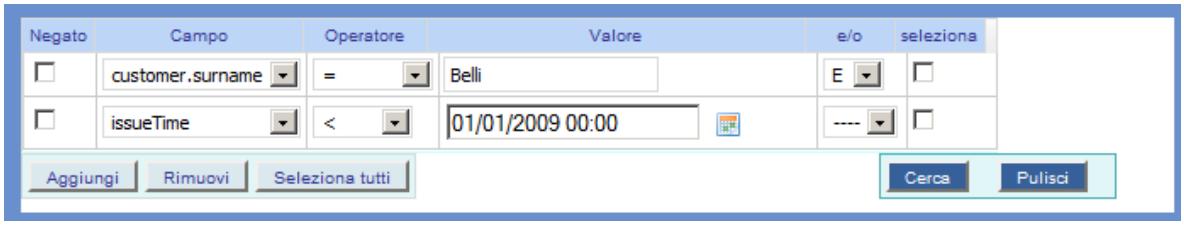


Figura 4-11: Sezione di ricerca per la vista *ricevute*

4.5.3 Gestione degli errori

In AddCenter è integrato un sistema di gestione e segnalazione degli errori che controlla gli editor. Esso permette all'applicativo di continuare la sua esecuzione anche nel caso si verifichino eccezioni all'interno degli editor. Di seguito si presenta un esempio di tale meccanismo.

Due segretari, Mario e Pietro vogliono modificare contemporaneamente i dati di uno stesso cliente "Gianni Ottimo". Mario apre l'editor dei clienti per modificare i dati di "Gianni Ottimo" e Pietro fa lo stesso. A questo punto Pietro per primo modifica il luogo di nascita e salva. Mario rimane in modifica e vuole aggiungere il codice fiscale , non sapendo che nel frattempo la residenza dell'utente è stata aggiornata.
A questo punto se Mario salvasse i dati, il codice fiscale risulterebbe calcolato errato sulla città di nascita sbagliata. Per questo quando Mario salva, l'operazione gli viene negata segnalandogli che i dati dell'utente sono stati modificati nell'arco di tempo in cui stava effettuando l'editing.
Nelle seguenti figure(da figura 4-12 a figura 4-15) è illustrato quanto descritto:

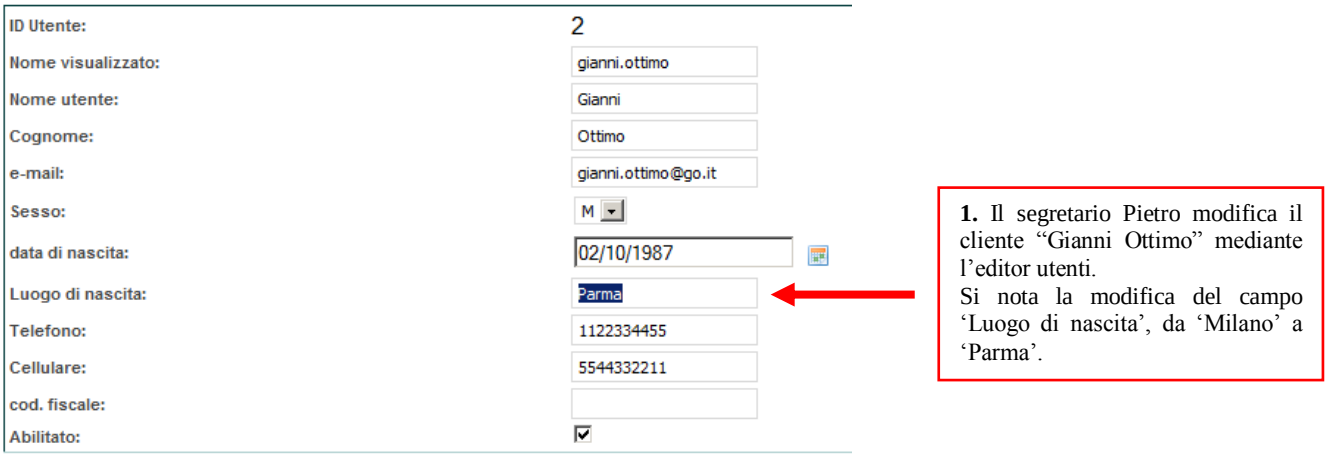
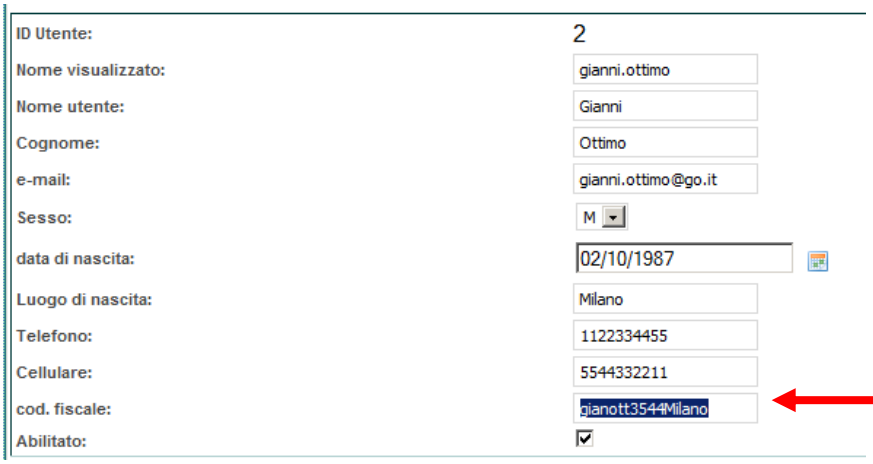


Figura 4-12: Pietro modifica il luogo di nascita



2. Anche il segretario Pietro modifica il cliente "Gianni Ottimo" mediante l'editor utenti.
Si nota la modifica del campo 'cod.fiscale', calcolato sulla base della città 'Milano'.
Nota: il codice fiscale è inventato a solo scopo illustrativo.

Figura 4-13: Mario modifica il codice fiscale

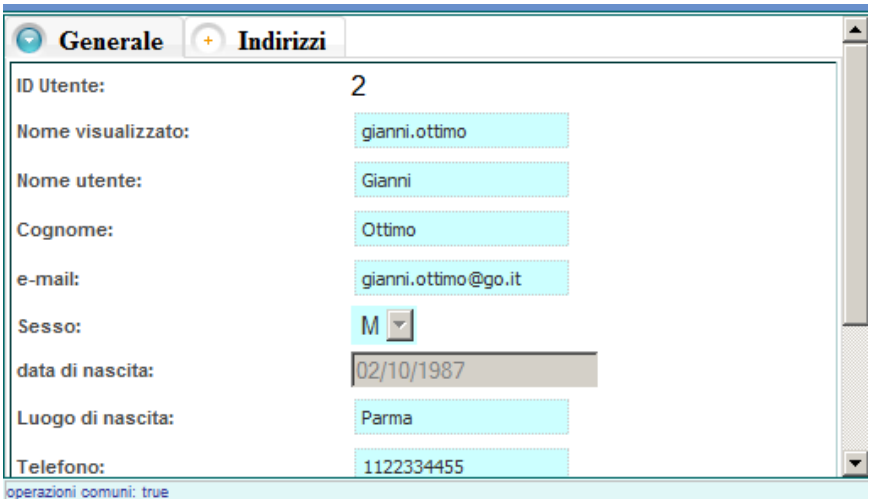


Figura 4-14: Pietro salva con successo

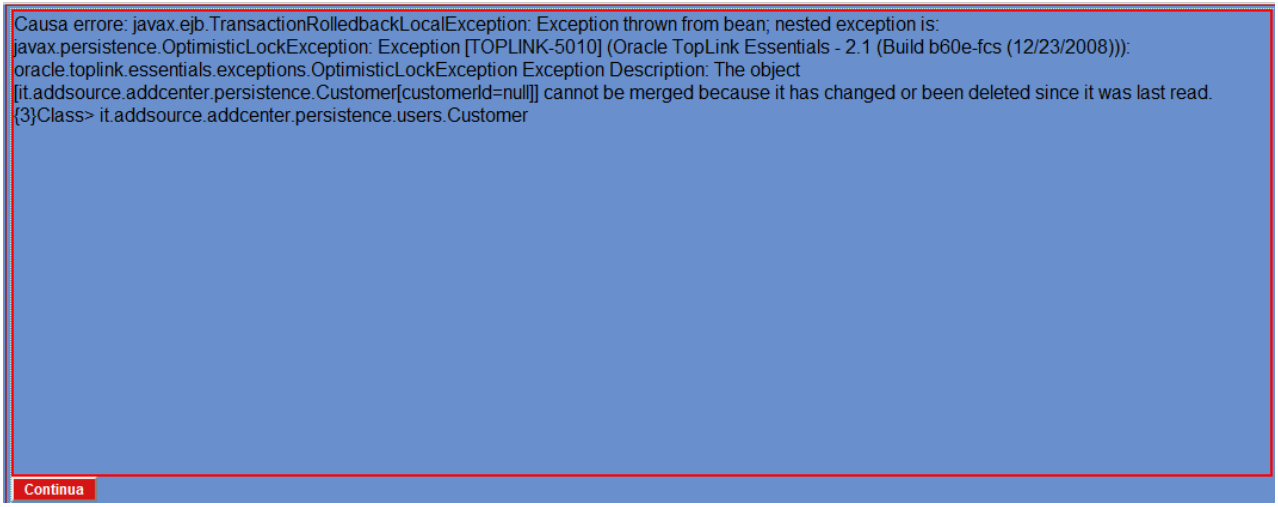


Figura 4-15: Segnalazione a Mario di un errore nella modifica

A N viene segnalata l'impossibilità di continuare con l'operazione in quanto l'oggetto che si vuole modificare è stato modificato dall'ultima sua lettura.

CAPITOLO 5

5 MODULI DI AddCenter

Si descriverà ora in dettaglio il funzionamento di alcuni moduli chiave dell'applicativo. In particolare il modulo, con le relative API, per l'innesto degli editor, il gestore delle ricerche e il sistema di creazione automatica delle interrogazioni per lo strato di persistenza di Java.

5.1 QuerySystem, un framework di supporto a JPQL

JPQL, come accennato nel capitolo precedente, è il linguaggio di interrogazione dello strato di persistenza di Java. Tale strato è gestito dall'API di persistenza Java (Java Persistence API), chiamata con l'acronimo di JPA.
Essa è un framework Java per gestire i dati relazionali nelle applicazioni che usano la piattaforma Java, sia standard che enterprise. E' formata principalmente da tre componenti:

- L' API, definita nel pacchetto javax.persistence
- Il linguaggio di interrogazione dello strato di persistenza Java, chiamato Java Persistence Query Language (JPQL)
- I metadati per la mappatura oggetti/relazionale

Si rimanda a letteratura specifica sull'argomento per approfondire JPA. In questa trattazione si discuterà solo di alcune sue caratteristiche,in particolare di JPQL, e le necessità che hanno portato alla creazione di un framework di supporto a tale linguaggio.

5.1.1 Caratteristiche principali di JPQL

JPQL(Java persistence query language) permette di effettuare:

- interrogazioni, tramite il motore di persistenza, sulle entità dell'unità di persistenza [68](1) specificata. Esse sono create "manualmente" mediante stringhe.
- operazioni di:
  - selezione (clausola WHERE)
  - proiezione (clausola SELECT)
  - ordinamento (ORDER BY)
  - navigazione delle relazioni tra entità (mediante navigazione delle proprietà degli oggetti)
  - Aggregazione (funzioni di aggregazione)
- Le query restituiscono insiemi di oggetti anziché multi-insiemi di records di basi di dati.

JPQL non permette:

- Creazione delle interrogazioni in modo ***dinamico***, infatti è sempre necessario codificarle in una stringa di testo
- Controllo rigoroso della sintassi di un' interrogazione
- Controllo dell'esistenza dei campi specificati nell'interrogazione
- Controllo,nella clausola WHERE delle interrogazioni, del tipo di dato degli oggetti forniti nelle operazioni di confronto con i campi dell'entità.

5.1.2 Perché QuerySystem

I limiti di JPQL descritti nel paragrafo precedente lo rendono un linguaggio prevalentemente statico, cioè che ha bisogno di essere codificato in fase di scrittura del codice sorgente dal programmatore. Questo significa che le interrogazioni ad una base di dati,di qualunque tipo, non potranno essere specificate dall'utente finale dell'applicativo, ma dovranno sempre essere scritte dallo sviluppatore.

[1] Unità di persistenza: in inglese "Persistence unit", fornisce un modo convenzionale per specificare un insieme di files di metadati, classi e pacchetti jars che contengono le classi che devono essere ibernate in gruppo. Ad essa viene fornito un nome che viene utilizzato per identificarla.

Sarebbe possibile utilizzare JPQL a livello utente dell' applicativo, ma esso dovrebbe scrivere manualmente le stringhe di interrogazione. Ciò implica che l'utente debba conosca la base di dati sottostante e il linguaggio JPQL, e ciò rende JPA poco flessibile e dinamico. Infatti un utente arbitrario non sarà in grado di specificare interrogazioni alla base di dati sottostante. Inoltre, come per SQL, non è possibile effettuare un controllo approfondito della correttezza delle interrogazioni scritte, avendo esse tutte le limitazioni delle stringhe di testo. Solo nella fase della loro esecuzione è possibile verificare se esse sono state scritte correttamente, nel quale caso viene restituito il risultato richiesto, o in modo errato, nel quale caso viene fornito un messaggio di errore in formato testo.
QuerySystem è un piccolo framework che è stato ideato e sviluppato per risolvere tali problematiche e per permettere ad un programmatore di creare interrogazioni in JPQL in modo controllato utilizzando interamente il linguaggio di programmazione Java. QuerySystem si assume completamente il compito di tradurre tale interrogazioni nel linguaggio JPQL. Quindi esso aumenta la potenza di JPQL eliminando i problemi visti, infatti grazie ad esso è possibile:

- Creare delle interrogazioni dinamicamente
- Controllare la sintassi delle interrogazioni
- Controllare i campi specificati per il filtraggio delle interrogazioni
- Controllare il tipo di dati degli oggetti verificando che l'oggetto passato sia della stessa classe del campo specificato per il confronto.

Fornisce inoltre:

- Un meccanismo per creare interrogazioni solo su un sottoinsieme dei campi delle entità, ciò risulta utile per la protezione della base di dati e per il controllo delle autorizzazioni a visualizzare ,filtrando, solo determinati dati.
- Un meccanismo per costruire le interrogazioni specificando filtri modulari. La clausola di selezione dell'interrogazione è suddivisa in due o più "parti". Le condizioni di filtraggio sono divise in blocchi legati da AND in modo tale da poter specificare regole di interrogazione su sottoselezioni successive dei dati. E' quindi possibile per ogni interrogazione mantenere una parte fissa e una parte variabile. Ciò è utile se si vuole specificare in fase di programmazione delle determinate condizioni di filtraggio e lasciare all'utente finale la possibilità di filtrare ulteriormente l'insieme di entità restituite. In tal modo è possibile garantire la protezione della base di dati, presentando solo le entità volute all'utente finale.

5.1.3 Struttura di QuerySystem

Classi ed interfacce principali:

EntityFieldsDescriptor:
Classe astratta chiave per il funzionamento di QuerySystem. Come dice il nome è l'astrazione di "un descrittore di campi d' entità", cioè una classe che la estende descrive una determinata entità. Specifica l'entità che descrive ed i campi,insieme al loro tipo, accessibili tramite essa.

QueryItemsList:
Classe per la creazione di interrogazioni. Il costruttore riceve come parametro un oggetto di classe EntityFieldsDescriptor che definisce il tipo (detto anche classe o tipo di entità) delle entità da cercare e i campi di queste sui quali è possibile effettuare l'ordinamento ed utilizzare gli operatori di confronto. E' dotato di metodi per specificare filtri di selezione basati sui valori di determinati campi e di metodi per ottenere l'insieme risultato dell'interrogazione ordinato basandosi sui valori di determinati campi delle entità. L'interrogazione si costruisce a "blocchi" specificando individualmente gli elementi che la compongono creando una lista di criteri di filtro ed ordine. Da qui il nome QueryItemsList, lista di elementi di interrogazione.

QueryExecutorLocal:
Interfaccia locale implementata dal bean di sessione (Statless Session EJB) che esegue le interrogazioni specificate mediante gli oggetti di tipo QueryItemsList.

QueryExecutor:
Classe dei Beans di sessione senza stato (Statless Session Bean) che implementano l'interfaccia QueryExecutorLocal ed eseguono le interrogazioni definite negli oggetti di tipo QueryItemsList.

Eccezioni:

- IllegalFieldException: Lanciata da QueryItemsList quando nelle creazione dell'interrogazione si specificano campi non permessi o non esistenti nel tipo di oggetto da cercare. IllegalFieldTypeException: Eccezione ritornata dal metodo addSearchItem di QueryItemsList quando si specifica un criterio di ricerca su un campo che ha tipo di dato differente da quello del valore da confrontare fornito al metodo.
- IllegalSearchType: Eccezione lanciata dal metodo addSearchItem di QueryItemsList quando l'operatore di confronto specificato nel criterio di ricerca è errato per il tipo di dato del campo specificato (ad esempio l'operatore LIKE può essere utilizzato solo coi tipi di dato String)
- InvalidBracketException: Lanciata quando si posiziona erroneamente una parentesi in una interrogazione.
- InvalidSearchExpression: Eccezione lanciata dal metodo addSearchItem di QueryItemsList quando l'item di ricerca è posizionato erroneamente all'interno della query.

Per effettuare un'interrogazione su una tabella di una base di dati , mappata dal motore di persistenza di Java in una classe annotata come Entity, è per prima cosa necessario creare una classe che implementa la classe astratta EntityFieldsDescriptor in modo tale da avere un descrittore di tale entità che QuerySystem possa utilizzare per formulare interrogazioni. Si supponga di avere una base di dati che gestisca un negozio di videonoleggio.
Si avrà per esso una tabella SQL che memorizza l'anagrafica dei DVD. In Java tale tabella può essere mappata con una classe entità di nome Dvd.
Si può quindi creare una classe DvdEntityFieldsDescriptor che implementa EntityFieldsDescriptor e descrive totalmente o parzialmente l'entità Dvd. Si può ad esempio specificare che i campi 'nome_film' , 'anno','durata','regista','genere','trama' siano accessibili, ma i campi 'prezzo_acquisto' , 'data_Acquisto' e 'numero noleggi' non siano descritti.
Successivamente si istanzia un oggetto di classe di tipo QueryItemsList passando al suo costruttore un oggetto di tipo DvdEntityFieldsDescriptor. Con tale oggetto QueryItemsList si può creare l'interrogazione voluta.
Ad esempio si può specificare un'interrogazione per selezionare i DVD acquistati prima del 15/05/2010.
Tale interrogazione viene successivamente eseguita ottenendo un statless session bean di tipo QueryExecutor e passandogli in ingresso l'interrogazione da eseguire.

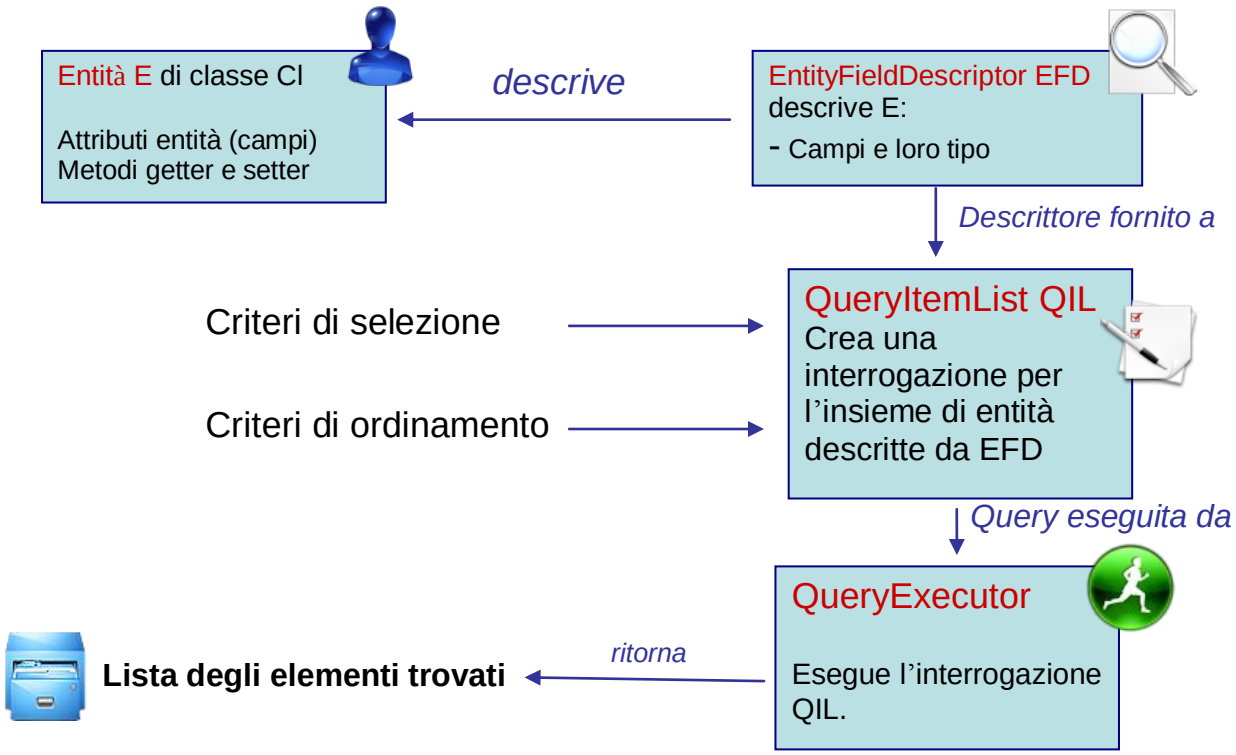


Figura 5-1: Funzionamento delle classi di QuerySystem

Si tratterà nel seguito il funzionamento di tali componenti ed il loro uso per creare un'interrogazione ed eseguirla.

EntityFieldsDescriptor

Di seguito sono descritti i metodi di tale classe:

public String getIdFieldName(): Restituisce il nome del campo impostato come identificatore primario.

protected boolean setIdFieldName(String field): Imposta un campo come identificatore primario. Metodo da chiamare dopo l'aggiunta del campo che si vuole impostare come id tramite il metodo addField. Il nome del campo deve contenere all'interno la sottostringa 'Id' per essere impostato come campo Id. Il parametro 'field' è il nome del campo Id, il metodo restituisce il valore booleano true se il campo è stato impostato con successo, false altrimenti.

protected boolean addField(Class type,String completeName):

Aggiunge un campo al descrittore di entità. Riceve in ingresso il parametro type, che è la classe del campo da aggiungere e completeName, che è il nome del campo da aggiungere. E' possibile aggiungere solo campi realmente esistenti dell'entità che si vuole descrivere e specificata tramite il metodo getDescribedEntity. Il metodo controlla che il campo fornito sia corretto e che il tipo di dato corrisponda al tipo di campo (con alcune eccezioni descritte in seguito).
Data la possibilità di specificare interrogazioni tramite il linguaggio JPQL che coinvolgono non solo campi base, ma anche metodi in generale, addField permette di specificare come campi metodi generali dell'entità descritta e permette inoltre l'annidamento delle proprietà, chiamando metodi degli oggetti restituiti da altri metodi. In pratica dato per esempio un oggetto di classe 'Automobile' che possiede come metodo 'getProprietario' che restituisce un oggetto 'Proprietario', quest'ultimo può avere come metodi 'getNome' e 'getCognome'.
E' possibile aggiungere come campo dell'entità 'Automobile' il nome e cognome del proprietario come se fossero campi nativi suoi, specificando come attributo "completeName" di addField le stringhe 'proprietario.nome' e 'proprietario.cognome'.

Particolare attenzione bisogna porre all'aggiunta dei campi rispettando il loro tipo, ma come alcune eccezzioni.
In JPQL le date vengono trattate mediante il tipo di dati Calendar, ma in SQL le date vengono memorizzate sotto i tipi Date,Time e Timestamp. La differenza principale è che un oggetto Calendar memorizza sia la data che il tempo, mentre in SQL questi possono essere divisi rispettivamente in Date e Time. E' quindi necessario specificare il tipo di dato del campo che è mappato in SQL. In presenza quindi di un campo di tipo Calendar è necessario fornire in ingresso, a seconda di quanto specificato nell'annotazione dell'entità, le classi:

- java.sql.Date: se il campo è annotato come data
- java.sql.Time: se il campo è annotato come tipo ora
- java.sql.Timestamp: se il campo è annotato per memorizzare sia data che ora

Altra eccezione riguarda il tipo di dato valuta. Nel sistema i tipi di dato valuta sono memorizzati tramite numeri interi (normali o lunghi).
Ad esempio se la valuta è l'euro, si memorizzano gli importi in centesimi di euro. Dato ad esempio un importo di € 5.67, questo è memorizzato con un intero di valore 567. Il sistema deve distinguere tra in normali valori interi e i valori valuta e ciò deve essere specificato separatamente dalle annotazioni di persistenza.
E' perciò necessario passare in ingresso come tipo java.util.Currency, nel qual caso si voglia far assumere ad un campo intero il significato di valuta.
Il metodo restituisce true se il campo è stato aggiunto, false altrimenti.

public Map<String, Set> getFieldsTypeMap(): Restituisce la mappa dei campi, utilizzando il loro tipo come chiave.

public Set<String> getTypeSet(Class cl): Restituisce l'insieme dei nomi dei campi aventi come tipo la classe cl.

public Set<String> getTypeSet(String cl): Restituisce l'insieme dei nomi dei campi aventi come tipo la classe di nome cl.

public Class getFieldClass(String field): Restituisce la classe assegnata al campo passato in ingresso. In particolare se il campo è stato segnato come valuta, restituirà il tipo java.util.Currency, mentre se è un tipo temporale potrà essere java.sql.Date, java.sql.Time o java.sql.Timestamp. Negli altri casi restituisce il tipo effettivo del campo.

public abstract Class getDescribedEntity(): Metodo da implementare che restituisce la classe dell'entità descritta.

QueryItemsList

L'interrogazione si costruisce a "blocchi" specificando individualmente gli elementi che la compongono creando una lista di criteri di filtro ed ordine. Da qui il nome QueryItemsList, lista di elementi di interrogazione.
La modalità di creazione delle interrogazioni segue in modo molto simile la sintassi utilizzata da JPQL e quindi SQL. E' possibile specificare condizioni di selezione delle entità creandole in modo pilotato mediante i metodi addSearchItem, addLogicItem e addBracketItem.
Per capire come usarli è consigliabile conoscere il funzionamento della clausula WHERE di SQL.Il passaggio dalla sintassi SQL all'utilizzo di tali metodi avviene in modo semplice. Si descrive come utilizzarli per creare condizioni di selezione.
Una condizione è formata da atomi (voci o items), che possono sono del tipo "nome_a op VAL" dove op è uno degLi operatori di confronto nell'insieme (=,<,<=,>,>=,!=,LIKE,IS NULL) , nome_a è il nome di un attributo dell'entità considerata e VAL è il valore di confronto.
Ogni atomo ha valore VERO o FALSO per una data entità. Ciò è chiamato valore di verità dell'atomo.
La condizione, come detto è formata da uno o più atomi, legati tra loro tramite gli operatori logici AND,OR,NOT e si definisce ricorsivamente:

1. ogni atomo è una formula
2. se M ed N sono formule, lo sono anche (M AND N) , (M OR N), NOT(M), NOT(F).

- Per selezionare le entità sulla base del valore di un campo, ad esempio si vogliono selezionare tutti gli Utenti che hanno l'età minore di 30 anni,si deve chiamare il metodo addSearchItem specificando il nome del campo, 'età' in questo caso, l'operatore di confronto, '<' in questo caso, e il valore di confronto, che è 30 nell'esempio.
La chiamata quindi è:

```
addSearchItem("età",new Integer(30),SearchType.LESS).
```

- Si possono specificare condizioni di selezione più complesse. Ad esempio si potrebbero voler selezionare tutti gli Utenti di età minore di 30 anni, ma di sesso maschile.

Allora l'espressione logica sarà: età < 30 AND sesso = 'Maschio'
E si specifica con le seguenti righe di codice:

```
qil.addSearchItem("età",new Integer(30),SearchType.LESS);
qil.addLogicItem(LogicOperator.AND);
qil.addSearchItem("sesso","Maschio",SearchType.EQUAL);
```

- Infine è possibile specificare raggruppamenti delle condizioni di selezione tramite parentesi per modificare l'ordine di valutazione logica. Il metodo addBracketItem è utilizzato per inserire parentesi nel predicato di selezione.
Se ad esempio si vogliono selezionare gli utenti di età inferiore ai 30 anni e che sono di sesso femminile e nubili o di sesso maschile e celibi, la seguente condizione:

```
età < 30 AND sesso = 'F' AND stato='nubile' OR sesso = 'M' AND stato='celibe'
```

è errata in quanto vengono selezionati gli utenti femmina,nubili e di età inferiore ai 30 anni e gli utenti maschi celibi.
Il predicato di selezione corretto è:

```
età < 30 AND (sesso = 'F' AND stato='nubile' OR sesso = 'M' AND stato='celibe')
```

che si traduce in:

qil.addSearchItem("età",new Integer(30),SearchType.LESS);	età < 30
qil.addLogicItem(LogicOperator.AND);	AND
qil.addBracketItem(Bracket.OPEN);	(
qil.addSearchItem("sesso","F",SearchType.EQUAL);	sesso = 'F'
qil.addLogicItem(LogicOperator.AND);	AND
qil.addSearchItem("stato","nubile",SearchType.EQUAL);	stato='nubile'
qil.addLogicItem(LogicOperator.OR);	OR
qil.addSearchItem("sesso","M",SearchType.EQUAL);	sesso = 'M'
qil.addLogicItem(LogicOperator.AND);	AND
qil.addSearchItem("stato","celibe",SearchType.EQUAL);	stato='celibe'
qil.addBracketItem(Bracket.CLOSE);	)

In generale per formulare condizioni di selezione,come visto, è necessario e sufficiente conoscere l'algebra booleana

(per la risoluzione delle condizioni di verità delle espressioni), e gli operatori aritmetici, (per specificare le voci di confronto).

Di seguito sono descritti i metodi di tale classe:

public QueryItemsList(EntityFieldsDescriptor efd) : In ingresso il descrittore di entità per la quale si vuole effettuare l'interrogazione.

public EntityFieldsDescriptor getEfd() : Restituisce il descrittore di entità associato

public boolean isSearchEmpty() : Dice se l'interrogazione è vuota, cioè non ha items.

public ArrayList<QueryInterface> getSearchItems() : restituisce la copia non profonda(shallow) degli item di ricerca (non sono modificabili quindi la copia shallow va bene)

public synchronized void clearSearchItems() :Pulisce la lista degli atomi di ricerca. Metodo sincronizzato per gestire l'accesso concorrente alla lista di selezione.

public synchronized void clearOrderItems():Pulisce la lista dei campi di ordinamento

public boolean addItemsFromQueryItemsList(QueryItemsList qil): Aggiunge i filtri di ricerca da un altro QueryItemsList.E' possibile farlo solo se i due QueryItemsList hanno lo stesso descrittore di entità.
Il parametro 'qil' è l'interrogazione che contiene gli elementi da aggiungere. Il metodo restituisce vero se è possibile effettuare l'aggiunta, false altrimenti.

public synchronized void addSearchItem(String field,Serializable value,SearchType type)
throws IllegalFieldException,IllegalFieldTypeException,IllegalSearchType,InvalidSearchExpression:

Aggiunge una voce di ricerca. Essa è composta da un attributo(campo) da confrontare con un valore.
Nella voce si specifica il campo da confrontare (field) , il valore di confronto (value) e il tipo di confronto da effettuare specificando un operatore di confronto (<,<=,>,>=,=,!=,LIKE,IS NULL).
L'eccezzione IllegalFieldException viene lanciata se il campo specificato non è specificato nel descrittore di entità (perchè non esiste o non lo si vuole rendere accessibile).

L'eccezzione IllegalFieldTypeException viene lanciata se il tipo di dati del valore da confrontare è diverso da quello del campo.
L'eccezzione InvalidSearchExpression viene lanciata se l'espressione di ricerca viene formulata in modo errato. Questo può accadere se non viene rispettata l'ordine delle voci di ricerca con i connettivi logici. Ad esempio chiamate X,Y,Z tre voci di ricerca, un'espressione logica del tipo X AND Y OR Z è corretta, mentre X AND Y Z non lo è.
Nell'ultimo caso viene lanciata l'eccezzione InvalidSearchExpression.
Essa viene lanciata anche nel caso in cui non si rispetti l'ordine delle parentesi nelle espressioni. Per esempio un'espressione del tipo (X AND Y)) porta all'insorgere dell'eccezione.

public synchronized void addBracketItem(Bracket brk) throws InvalidBracketException : Aggiunge una parentesi alla lista di selezione dell'interrogazione. Lancia InvalidBracketException se la parentesi è posizionata erroneamente.

public synchronized void addEnclosingBrackets() throws InvalidBracketException : Aggiunge parentesi che rinchiudono l'interrogazione creata fino ad ora.

public synchronized void addOrderItem(String field,boolean ascending) throws IllegalFieldException : Aggiunge un criterio di ordinamento basato su un campo dell'entità. L'attributo ascending specifica se ordinare le entità in modo ascendente o discendente sulla base dei valori di tale campo.Lancia IllegalFieldException se il campo specificato non esiste.

public synchronized void addLogicItem(LogicOperator lo) throws Exception : Aggiunge un operatore logico nella clausula di selezione. E' necessario seguire le seguenti regole per il suo uso:

1. Se non vi sono voci nella lista di ricerca, l'operatore può essere solo NOT.
2. Se l'elemento precedente è un operatore logico AND od OR,il successivo operatore può essere solo NOT.
3. Se l'elemento che precede è una parentesi chiusa l'operatore può essere AND od OR.
4. Se l'elemento che precede è una parentesi aperta l'operatore può essere solo NOT.

In caso di violazione di queste regole verrà lanciata un'eccezione.

public static String convertSearchType(SearchType st): Converte un tipo di ricerca nella relativa stringa.

QueryExecutorLocal

Si descrivono i due metodi di tale interfaccia:

public List executeQuery(QueryItemsList qil,int firstResult,int maxResult) throws IllegalStateException :

Esegue l'interrogazione passata in ingresso come parametro nell'oggetto di tipo QueryItemsList e restituisce la lista degli oggetti ottenuti.
Ha i parametri:

- *QueryItemsList* qil: la lista degli elementi che compongono l'interrogazione da eseguire.
- Int firstResult: l'indice del primo elemento da restituire dalla lista dei risultati.
- Int maxResult: il numero massimo di elementi da ritornare nel risultato.

Lancia l'eccezione IllegalStateException nel caso l'interrogazione sia formulata in modo errato. E' lanciata per segnalare la presenza di eventuali problemi di formulazione dell'interrogazione a livello di linguaggio JPQL.

public Long queryResultsCount(QueryItemsList qil):

Fornisce il numero di risultati che l'esecuzione della interrogazione restituirebbe.

5.1.4 QuerySystem: Esempio

Data un'entità che rappresenta un'utente (classe User) con i seguenti campi:
- Integer: userId
- String: name, surname, phone, mobilePhone, taxCode, birthPlace
- Calendar(Date): birthDate
- Enum: gender

si vuole creare un descrittore di entità che permetta di effettuare interrogazioni analizzando i soli campi:
userId, name, surname ,birthDate, Gender

In seguito si vuole creare una interrogazione che restituisca tutti gli utenti di cognome "Verdi" che siano di sesso maschile e nati prima del 1/1/1991, ordinati per data di nascita crescente.

Per prima cosa si crea una classe che implementa EntityFieldsDescriptor per descrivere l' entità di classe utente:

```
public class UserFieldsDescriptor extends EntityFieldsDescriptor
{
        → Classe descritta
        public Class getDescribedEntity() {
           return User.class;
        }

        → Definisco nel costruttore i campi dell'entità:
        public UserFieldsDescriptor() {
           →Campi di tipo intero:
           addField(Integer.class,"userId");
           →Specifico quale campo è l'id dell'entità:
           setIdFieldName("userId");
           →Campi di tipo stringa:
           addField(String.class,"name");
           addField(String.class,"surname");
           →Campi di tipo data:
           addField(java.sql.Date.class,"birthDate");
           →Campi di tipo enumerativo:
           addField(Enum.class,"gender");
        }

}
```

Successivamente si progetta l'interrogazione voluta. La si descrive in linguaggio JPQL, per poi passare al suo corrispondente usando la classe `QueryItemList` di QuerySystem.

Si vuole creare la seguente interrogazione:

SELECT u FROM Utenti u
WHERE u.cognome ="Verdi" AND u.gender = "MALE"
AND u.birthDate < "1/1/1991"
ORDER BY u.birthDate ASC

Si crea un oggetto di tipo QueryItemList passandogli come descrittore di entità UserFieldsDescriptor e si specifica l'interrogazione aggiungendo le voci di ricerca ed ordinamento:

```
#1 QueryItemList qil = new QueryItemList(new UserFieldsDescriptor());

#2  try{
#3   qil.addSearchItem("cognome","Verdi", SearchType.EQUAL);
#4   qil.addLogicItem("LogicOperator.AND");
#5   qil.addSearchItem("gender",Gender.MALE, SearchType. EQUAL);
#6   qil.addLogicItem("LogicOperator.AND");
#7   Calendar cal = new GregorianCalendar(1991,1,1);
#8   qil.addSearchItem("birthDate",cal, SearchType.LESS);
#9   //aggiungo clausula di ordinamento su birthDate,ordine ascendente = true
#10  qil. addOrderItem ("birthDate",true);
#11 }catch(...){...}
```

Si osservino le chiamate ai metodi di `QueryItemList` nel blocco try. In particolare in riga #3 viene aggiunta una voce di ricerca per selezionare gli utenti di cognome Verdi (condizione cognome = 'Verdi'). In riga #4 viene impostato il connettore logico AND in modo che siano ritornate solo le entità che soddisfano la condizione in #3 e la successiva che verrà impostata. In riga #4 si aggiunge il criterio di selezione degli utenti di sesso maschile (condizione gender = "Gender.MALE"), in riga #6 si imposta un altro connettore AND per la voce di riga #8. Infine in riga #10 si aggiunge l'ordinamento del risultato sul campo data di nascita (birthDate) in ordine ascendente.

Per potere eseguire le interrogazioni è necessario chiamare l' EJB QueryExecutor utilizzando l'interfaccia locale QueryExecutorLocal.
Da un bean EsBean gestito da EJB container o dal Web Container, utilizzando la dependance injection(DI), richiamo lo stateless bean QueryExecutor, questo è così accessibile nei metodi del bean EsBean per eseguire interrogazioni.

```
@ManagedBean          //bean gestito dal web container
@RequestScoped        //bean con scope di richiesta

public class EsBean{
       @EJB     //DI, inietto un bean qe
       QueryExecutor qe;

       public void someMethod(){
              QueryItemList qil;

              ... Creazione query , descritta precedentemente ...

              //chiamo executeQuery per eseguire la query creata, il secondo
              //parametro indica l'indice del primo risultato da restituire,
              //il secondo il massimo numero di oggetti da restituire nella query.
              List<User> users = qe.executeQuery(qil,-1,-1);

              // esamino i risultati ottenuti
              for(User u: users){
                ... Fai qualcosa su u ...
              }
              ...
       }
       ... ...
}
```

5.2 SearchHandler, il gestore delle ricerche

SearchHandler è il gestore delle ricerche di AddCenter ed è associato alla pagina template Search.xhtml.
Esso è una classe Java e il suo funzionamento è strettamente legato a QuerySystem, in quanto, dato in ingresso un tipo di entità descritto da un oggetto di classe EntityFieldsDescriptor , crea l'interfaccia grafica che permette all'utente di specificare i criteri di ricerca sui campi specificati dal descrittore.
Se ne descriverà ora il funzionamento esterno e si forniranno alcuni frammenti di codice interno per mostrare come è stato sviluppato.

5.2.1 Interfaccia grafica di SearchHandler

In Figura 5-2 è mostrata la sezione di ricerca vuota. Infatti si può osservare che nella tabella di ricerca non è specificato alcun criterio. Essa è composta da una barra di titolo utilizzabile in modo arbitrario dal programmatore, una sezione sulla sinistra che fornisce una descrizione aggiuntiva della vista, che serve per informare l'utente di eventuali filtri impostati lato applicativo sulla visualizzazione delle entità, e una sezione sulla destra che permette la creazione e modifica di criteri di ricerca.

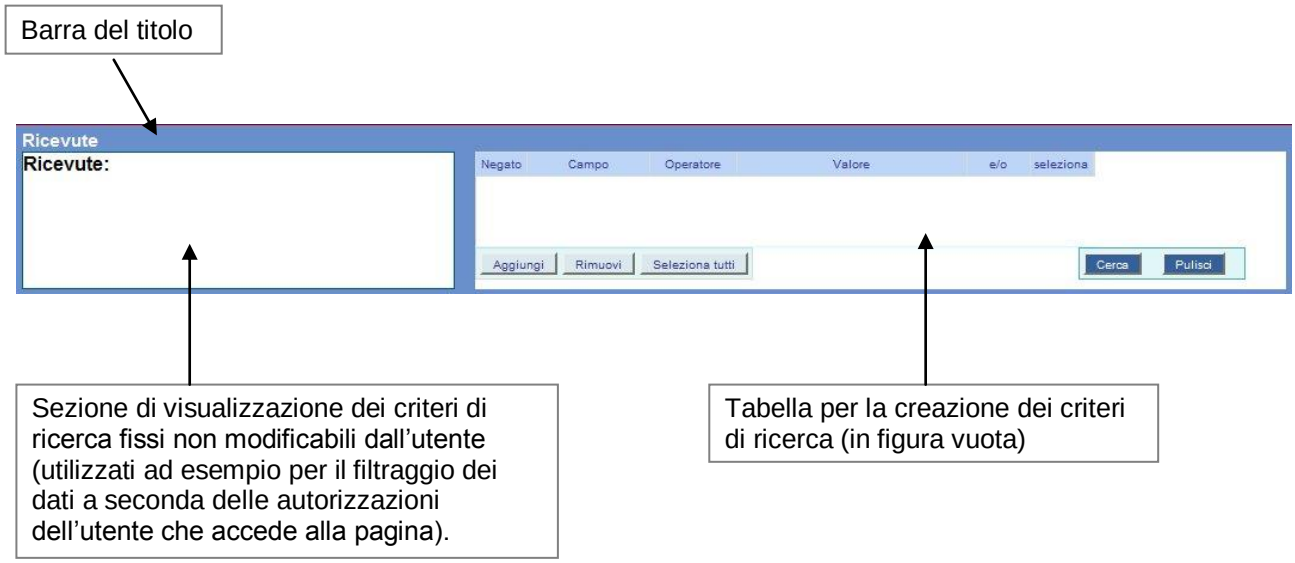


Figura 5-2:Interfaccia grafica di SearchHandler

In Figura 5-3 si osserva la tabella di ricerca in cui sono specificati i criteri per trovare tutte le ricevute emesse per il cliente di cognome "Belli" prima del 1 gennaio 2009.

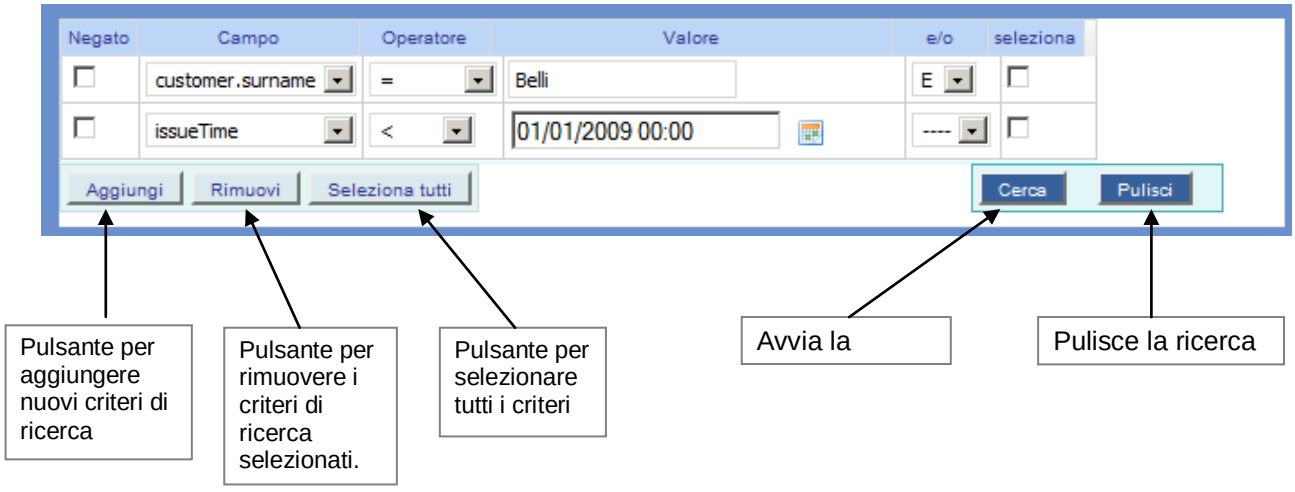


Figura 5-3: Ricerca delle ricevute emesse al cliente "Belli" prima del 1/1/2009

Si mostra ora un esempio completo di ricerca. In Figura 5-4 è riportata un'ipotetica anagrafica delle ricevute. Si possono notare in essa il campo Id,che specifica il progressivo della ricevuta,il campo tipo pagamento, che specifica come è stato pagato da parte del cliente l'importo della ricevuta, i campi Nome e Cognome del cliente a cui è intestata e il campo data emissione.

Id	Tipo pagamento	Nome	Cognome	Data emissione
1	CREDIT_CARD	io	io	08/02/2008 12:25
2	MONEY	io	io	08/02/2008 12:26
3	MONEY	io	io	12/03/2008 17:39
4	MONEY	io	io	12/03/2008 17:41
5	MONEY	io	io	12/03/2008 17:42
7	MONEY	io	io	12/03/2008 18:02
8	MONEY	io	io	12/04/2008 18:04
9	MONEY	io	io	29/04/2008 17:56
10	MONEY	io	io	29/04/2008 18:17

Figura 5-4: Anagrafica delle ricevute

Ora si vogliono visualizzare solo le ricevute pagate con carte di credito o emesse dopo il giorno 12/3/2008 ore 18:00. Si specificano quindi nella ricerca tali criteri, impostando "Tipo pagamento = CREDIT_CARD" o "Data emissione > 12/3/2008 18:00". In Figura 5-5 è mostrata la condizione di selezione per l'interrogazione.

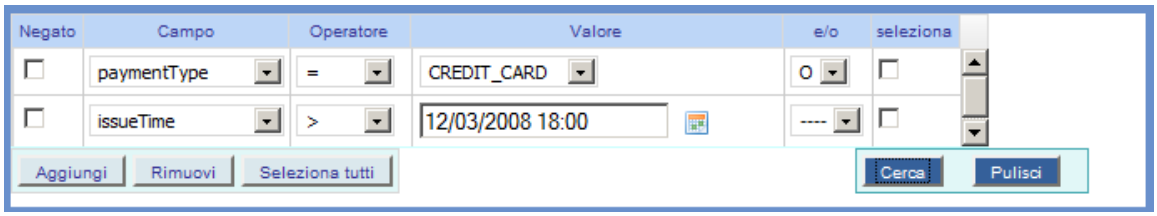


Figura 5-5: Selezione delle ricevute pagate con carta di credito o emesse dopo il giorno 12/03/2008 ore 18:00

Dopo la selezione del pulsante "cerca" vengono restituite le ricevute che soddisfano tali criteri. In Figura 5-6 è riportata la tabella dei risultati di tale ricerca.

Id	Tipo pagamento	Nome	Cognome	Data emissione
1	CREDIT_CARD	io	io	08/02/2008 12:25
7	MONEY	io	io	12/03/2008 18:02
8	MONEY	io	io	12/04/2008 18:04
9	MONEY	io	io	29/04/2008 17:56
10	MONEY	io	io	29/04/2008 18:17

Figura 5-6: Risultato dell'interrogazione sulle ricevute

Si è visto il funzionamento della ricerca, l'interfaccia grafica è definita dalla pagina Search.jspx, sviluppata interamente in JSF e ICEfaces. Essa è una pagina modello nella tecnologia Facelet, utilizzata come tecnologia di default per la visualizzazione delle pagine JSF.

Tale pagina è controllata dal gestore SearchHandler, che gli viene associato mediante un backing bean. SearchHandler è una normale classe Java e quindi per poter essere utilizzata all'interno di JSF deve sempre "vivere" all'interno di un bean gestito dal container.

In termini di progettazione MVC(Model View Controller) Search.jspx costituisce la vista e SearchHandler il controller. In questo caso il modello non è presente in quanto il SearchHandler crea soltanto le interrogazioni, senza prelevare dallo strato di persistenza dati.

In Figura 5-7 è schematizzato il funzionamento di SearchHandler. Gestisce la pagina Search.xhtml, ad esso associata tramite un bean JSF che lo contiene e lo associa mediante l'istruzione Facelets 'param', utilizzata per il passaggio parametri tra pagine modello. SearchHandler contiene un descrittore di entità (EntityFieldsDescriptor) che viene utilizzato per creare la ricerca per tale tipo di entità. In particolare vengono analizzati tutti i campi descritti, col loro relativo tipo, e da questi vengono create le voci di ricerca.
Quando vengono specificati dei criteri di ricerca, essi sono mantenuti in SearchHandler che li analizza non appena viene premuto dall'utente il pulsante "Cerca" [1] . Finita la loro elaborazione viene generato un oggetto QueryItemsList contenente l'interrogazione di ricerca. Tale oggetto è passato a tutti gli ascoltatori SearchListener del gestore di ricerca [2] tramite eventi di tipo SearchEvent[3] . Gli ascoltatori sono oggetti interessati a sapere quando viene modificata la condizione di ricerca del SearchHandler, ad esempio il gestore di una vista deve essere associato come ascoltatore di SearchHandler per sapere quando aggiornare la tabella delle entità.

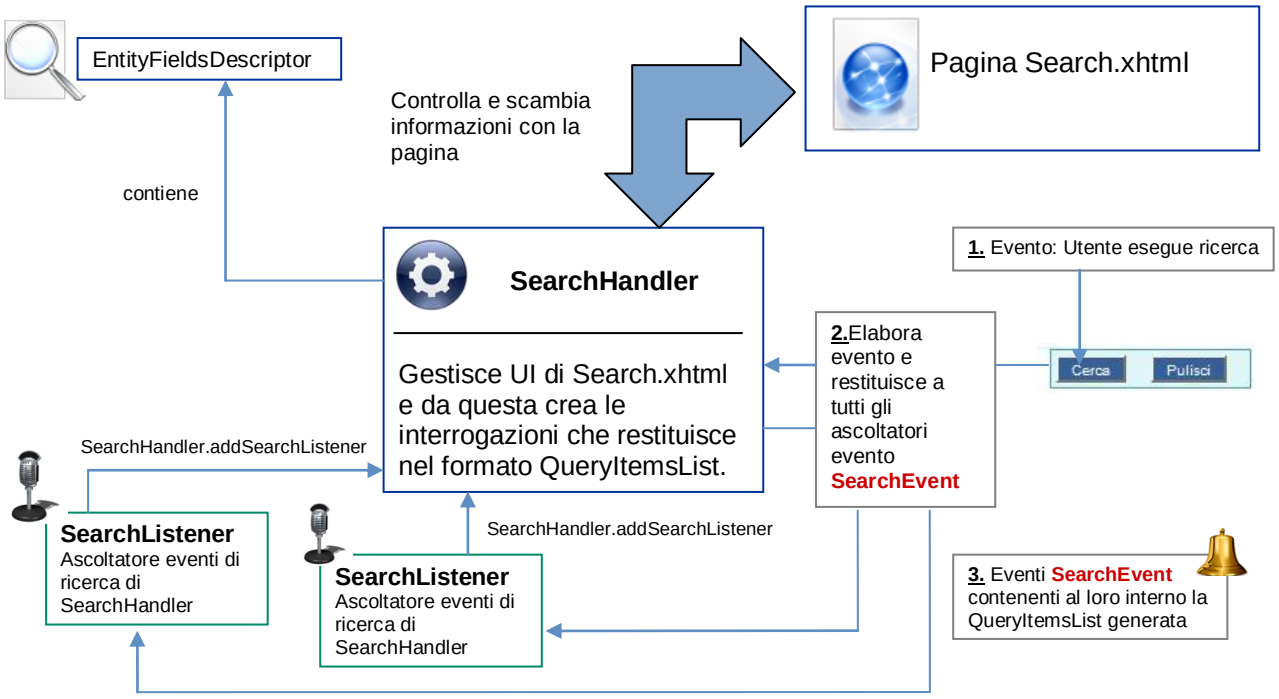


Figura 5-7: Schema funzionamento SearchHandler

Si descrivono brevemente le classi associate al gestore di ricerca.

SearchHandler: Come illustrato in questa sezione, è la classe che controlla la formulazione di interrogazioni per la ricerca di entità di un dato tipo.

SearchListener: Interfaccia che un ascoltatore di SearchHandler deve implementare per catturare le ricerche effettuate. Deve implementare il metodo XYZ, che è chiamato da SearchHandler ogni qual'volta viene svolta una ricerca. Tale metodo riceve in ingresso un oggetto di tipo SearchEvent, che contiene al suo interno l'interrogazione generata, memorizzata in un oggetto di tipo QueryItemsList.

SearchEvent: un evento semantico che indica che è stata effettuata una ricerca. E' generato da SearchHandler ogni volta che l'utente ricerca le entità secondo i criteri impostati.

In Figura 5-8 si mostrano alcune parti di codice della pagina Search.jspx. Essa è stata scritta utilizzando interamente il framework ICEfaces, si notino a proposito i tag di prefisso 'ice', e la tecnologia di visualizzazione Facelets, si noti il suo uso per l'inclusione della pagina FixedSearchView.xhtml (tramite il tag 'ui:include').
Il gestore SearchHandler è mappato nella pagina mediante il JSF Expression Language, è possibile notarlo nei frammenti di codice dichiarati con #{}, che è l'annotazione per l'Expression Language.
Nel caso specifico si ha #{searchHandler.proprietà}, dove searchHandler è la varabile utilizzata per il gestore della pagina e proprietà è una generica proprietà del gestore definita mediante la convenzione dei bean, cioè tale proprietà è dotata di un metodo getter e di un metodo setter.

```
<ice:form xmlns:h="http://java.sun.com/jsf/html" xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ice="http://www.icesoft.com/icefaces/component"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:t="http://myfaces.apache.org/tomahawk"  id="Search_CostructionForm">

     <ice:panelGroup id="searchPanelGroup" styleClass="searchPanel">

       <!-- Pannello del titolo contenente il valore fornito dall'attributo htmlTitle di searchHandler -->
       <ice:panelGroup id="searchTitlePanel" styleClass="searchTitlePanel">
            <ice:outputText id="searchTitle" value="#{searchHandler.htmlTitle}" escape="false"/>
       </ice:panelGroup>

       <!-- Pannello esterno -->
       <ice:panelGroup id="searchExternalPanel" styleClass="searchExternalPanel">

         <!-- Pannello contenente la pagina di visualizzazione dei parametri fissi -->
         <ice:panelGroup id="fixedSearchParametersPanel" styleClass="fixedSearchParametersPanel">
             <!-- Si include la pagina per la visualizzazione dei parametri di ricerca fissi associando all'alias per il formatter
                  il formatter contenuto in searchHandler. variabile alias per il FixedSearchFormatter: fixedSearchFormatter
             -->
           <ui:include src="/mngs/includes/FixedSearchView.xhtml">
             <ui:param id="fixedSearchFormatterAlias" name="fixedSearchFormatter"
                       value="#{searchHandler.fixedSearchFormatter}" />
           </ui:include>

        </ice:panelGroup> <!-- fine fixedSearchParametersPanel -->

        <!-- Pannello contenente la sezione di ricerca -->
       <ice:panelGroup id="statementsSearchPanel" styleClass="statementsSearchPanel">

    ... Codice non riportato ...

         </ice:panelGroup>
           <!-- GRIGLIA PULSANTI AGGIUNTA O RIMOZIONE STATEMENT E CONFERMA RICERCA -->
           <ice:panelGrid columns="3" id="manageButtomsGrid" styleClass="searchManageButtonsGrid">
                  <ice:commandButton actionListener="#{searchHandler.addStatement}"
                 id="addStatementCommand" type="submit" value="Aggiungi"
                 styleClass="searchManageButton"/>
                  <!-- si noti l'attributo immediate posto a true per permettere la rimozione anche in caso di dati non validi -->
              <ice:commandButton actionListener="#{searchHandler.removeSelectedStatements}"
                id="removeSelectedCommand" immediate="true" type="submit" value="Rimuovi"
                styleClass="searchManageButton"/>
              <ice:commandButton actionListener="#{searchHandler.selectAllStatements}"
                id="selectAllCommand" immediate="true" type="submit" value="Seleziona tutti"
                styleClass="searchManageButton"/>
           </ice:panelGrid>

           <ice:panelGrid columns="2" id="searchButtomsGrid" styleClass="searchButtonsGrid">
               <!-- pulsante per avviare la ricerca delle entità secondo i criteri specificati --->
              <ice:commandButton actionListener="#{searchHandler.searchEntities}"
              id="searchCommand" type="submit" value="Cerca" styleClass="searchButton"/>
              <ice:commandButton actionListener="#{searchHandler.clearSearch}" id="clearCommand"
                immediate="true" type="submit" value="Pulisci" styleClass="searchButton"/>
           </ice:panelGrid>

         </ice:panelGroup>
       </ice:panelGroup> <!-- fine statementSearchPanel -->

      </ice:panelGroup> <!-- fine pannello esterno -->
    </ice:panelGroup>  <!-- fine searchPanelGroup -->

</ice:form>
```

Figura 5-8: Pagina Search.xhtml

Si illustrano in Figura 5-9 le parti di codice di Search.xhtml che generano le corrispondenti parti di interfaccia grafica.

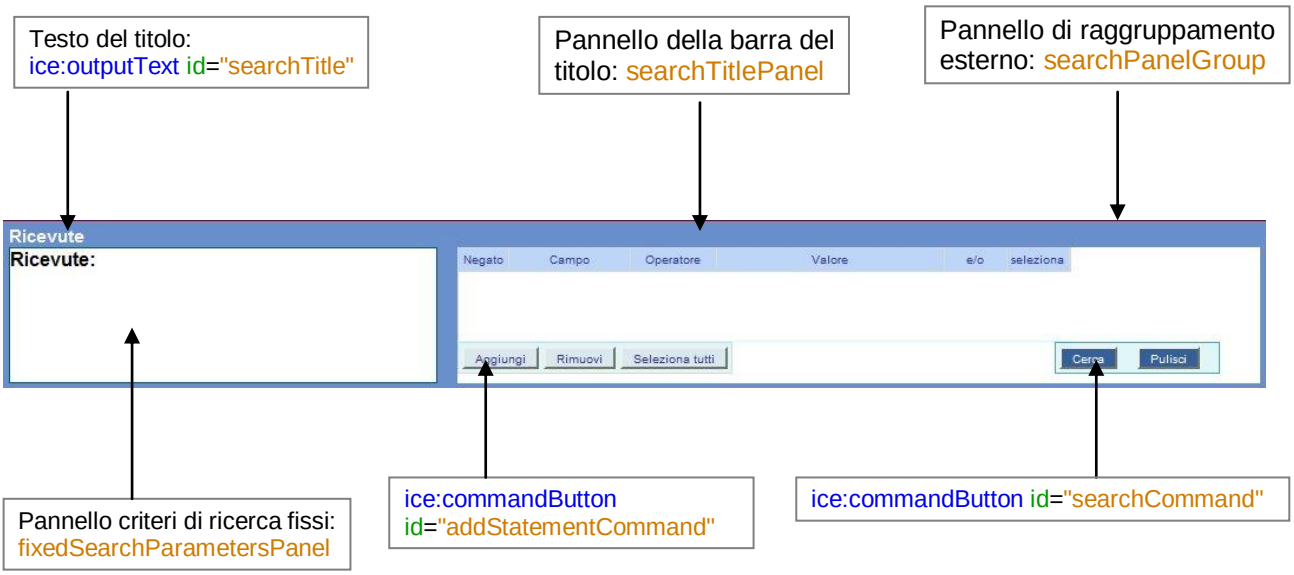


Figura 5-9: Corrispondenza tra interfaccia e codice di Search.xhtml

Infine si riportano alcuni frammenti di codice della classe SearchHandler per illustrare parzialmente il suo funzionamento e come questa dialoga con la vista definita in Search.xhtml.
Il costruttore riceve in ingresso un oggetto di tipo EntityFieldsDescriptor che, come illustrato precedentemente, serve a definire su quale entità e suoi attributi (campi primitivi o proprietà annidate ottenute mediante successive chiamate di metodo) è definita la ricerca. Nel costruttore è riportata l'assegnazione del descrittore all'attributo interno efd e la assegnazione, come titolo predefinito della sezione di ricerca, del valore "Add Source®" all'attriuto htmlTitle.
Sono riportatati i metodi ascoltatori dei pulsanti della ricerca (Aggiungi, Rimuovi, Seleziona tutti, Cerca e Pulisci).

```
public class SearchHandler  implements SearchHandlerInterface{
    ... variabili di classe non riportate ...
    private EntityFieldsDescriptor efd;
    private String htmlTitle;

    public SearchHandler(EntityFieldsDescriptor efDesc) {
        //imposto EntityFieldDescriptor interno
        efd = efDesc;
        ... ... ... ...
        htmlTitle = "Add Source&reg;";
    }

    ... ... ... METODI NON RIPORTATI ... ... ...

    public EntityFieldsDescriptor getFieldDescriptor() {
        return efd;
    }

    /* ***    LISTENERS PULSANTI ***    */
    /** * Avvia la ricerca */
    public void searchEntities(ActionEvent event) {
        QueryItemsList qil = createQueryItemsList(); //creo QueryItemsList
        //dispatch agli ascoltatori della ricerca
        for(SearchListener sl:  searchListeners){
            sl.searchPerformed(new SearchEvent(this, qil));
        }
    }
```

Il metodo searchEntities è l'ascoltatore del pulsante Cerca. In Figura 5-8 si può notare la sua associazione col pulsante di id "searchCommand" mediante l'attributo actionListener del tag 'ice:commandButton' (actionListener = #{searchHandler. searchEntities }).

```
    /** Ascoltatore per il pulsante 'Pulisci'.Pulisce i criteri di ricerca
      * e notifica l'evento agli ascoltatori */
    public void clearSearch(ActionEvent event) {
        statements.clear();
        QueryItemsList qil = createQueryItemsList();
        //dispatch agli ascoltatori solo la query fissa
        for(SearchListener sl:  searchListeners){
            sl.searchPerformed(new SearchEvent(this, qil));
        }
    }
    /** Ascoltatore del pulsante 'Rimuovi'. Rimuove i criteri selezionati
        e disabilita il connettore logico per l'ultima voce*/
    public void removeSelectedStatements(ActionEvent event){
       SearchStatement ss;
       int i = 0;
       while(i < statements.size()){
          ss = statements.get(i);
          if(ss.getSelected().booleanValue()){
              statements.remove(i);
          }else{
              i++;
          }
       }
       if(statements.size() > 0){
           ss = statements.get(statements.size() - 1);
           ss.setNextLogicOpEnabled(false);
       }
    }
    /** Seleziona tutti gli statements di ricerca */
    public void selectAllStatements(ActionEvent event){
        for(SearchStatement ss: statements){
            //se non selezionato, seleziono
            if(!ss.getSelected().booleanValue()){
                ss.setSelected(new Boolean(true));
            }
        }
    }
    /** Aggiunge un nuovo criterio di ricerca */
    public void addStatement(ActionEvent event){
        SearchStatement ss;
        statements.add(new SearchStatement(efd,Locale.ITALY));
        //se maggiore di uno il penultimo ha connettore logico
        if(statements.size() > 1){
            ss = statements.get(statements.size() - 2);
            ss.setNextLogicOpEnabled(true);
        }
    }
    /** Imposto titolo html  */
    public String getHtmlTitle() {
        return htmlTitle;
    }
    public void setHtmlTitle(String title) {
        htmlTitle = title;
    }
} // FINE classe SearchHandler
```

Quando l'utente preme il pulsante Cerca, viene generato un evento di tipo ActionEvent che viene passato in gestione al metodo searchEntities. Quest'ultimo chiama un metodo interno di nome `createQueryItemsList` che genera, dai criteri impostati dall'utente nell'interfaccia grafica, l'interrogazione voluta e la memorizza nella variabile 'qil'.
Successivamente inoltra a tutti gli ascoltatori di SearchHandler tale interrogazione generata. Si nota infatti il ciclo for che scandisce tutti gli ascoltatori e per essi inoltra un evento SearchEvent chiamando il loro metodo searchPerformed.

I successivi metodi clearSearch, removeSelectedStatements, selectAllStatements, addStatement sono collegati all'interfaccia grafica allo stesso modo di quello descritto per il metodo searchEntities.

Il metodo clearSearch è associato al pulsante "Pulisci". Esso serve per pulire la ricerca dalle voci di ricerca impostate.
Esso chiama statatements.clear() per pulire la lista delle istruzioni di ricerca e in seguito il metodo `createQueryItemsList` per rigenerare un interrogazione di base pulita, priva di voci di ricerca impostate lato utente. La nuova interrogazione così generata è passata agli ascoltatori.
Il metodo removeSelectedStatements è collegato al pulsante Rimuovi e rimuove le voci selezionate mediante la checkbox 'seleziona'. Analizza le voci una ad una mediante un ciclo for per cercare ed eliminare quelle che hanno la proprietà 'selected' vera. Successivamente disattiva la scelta dell'operatore logico (AND/OR) per l'ultima voce.
Il metodo selectAllStatements è associato al pulsante 'Seleziona tutti' e seleziona tutte le voci di ricerca impostando a true la loro proprietà 'selected'.
Il metodo addStatement è associato al pulsante 'Aggiungi' e aggiunge in fondo alla lista dei criteri di ricerca un nuovo criterio. Si nota che esso aggiunge un nuovo SearchStatement tramite il metodo 'add' di statements. Una volta aggiunto esso abilita l'operatore logico per il penultimo filtro impostato (se la lista non era vuota).

5.3 Editor

Un editor, come descritto nel paragrafo 4.5.1, è un componente software che permette l'inserzione,modifica o eliminazione delle informazioni.
Nei software gestionali si effettuano operazioni sulle istanze di entità, ed è necessario un editor per ogni entità da gestire.
AddCenter permette di creare editors da innestare nelle viste in modo indipendente da esse. L'editor può quindi esser sviluppato concentrandosi sul compito che dovrà svolgere e in seguito essere incluso nella sezione desidera.
AddCenter fornisce una comoda API chiamata EntityEditor API per sviluppare gli editors.
Un editor di entità è solitamente implementato, quando si utilizza il framework JSF, mediante un Managed Bean. Infatti questo comunica direttamente con l'interfaccia grafica, implementata mediante pagina JSF.
L'API EntityEditor fornisce l'interfaccia EntityEditorInterface che tutti gli editors devono implementare per poter interagire con AddCenter.
Nel seguito si illustrerà il funzionamento di tale interfaccia e se ne darà un esempio pratico di implementazione.
In Figura 5-10: Schema del gestore di entità è mostrata la struttura dell'API con le sue classi principali.
La classe EntityEditorHandler è la classe che gestisce l'editor e la sua iterazione col gestore della vista nel quale è incluso. Implementa l'interfaccia EntityEditorHandlerInterface che definisce le specifiche che deve soddisfare per poter interagire con tale gestore.
EntityEditorInterface è l'interfaccia che tutti gli editors che vogliono integrarsi con AddCenter devono implementare. In figura la classe EntityEditor, che implementa EntityEditorInterface, è un arbitrario editor per un entità ed è gestita da EntityEditorHandler.

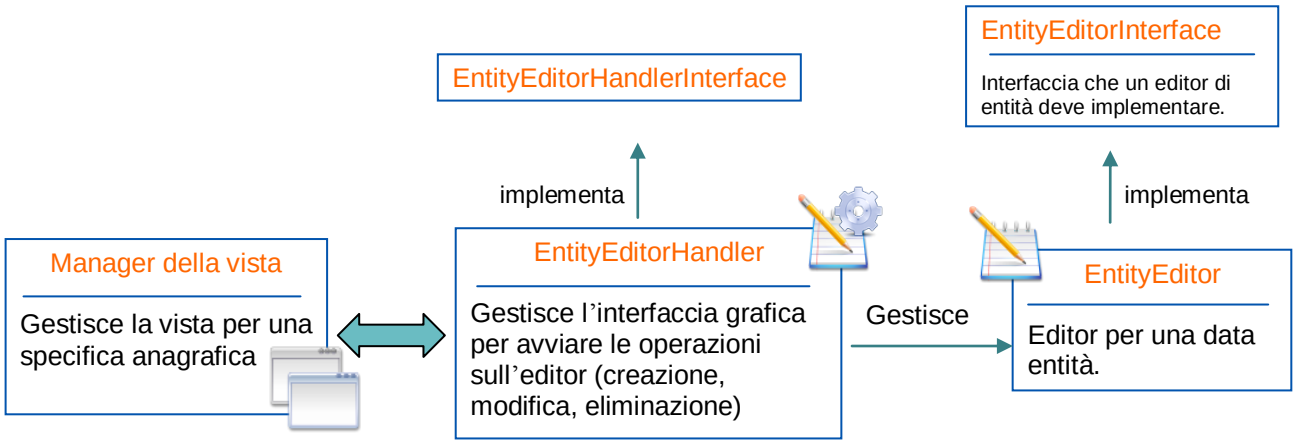


Figura 5-10: Schema del gestore di entità

Quindi per la creazione di un editor è necessario e sufficiente sviluppare un Managed Bean che implementi l'interfaccia EntityEditorInterface ed una pagina JSF, sviluppata col framework ICEfaces, da associare come vista a tale bean, e che costituisce l'interfaccia grafica dell'editor.

Si mostra ora l'interfaccia grafica di EntityEditorHandler, in quanto utile successivamente per capire come sviluppare un editor. In Figura 5-11 si visualizza un editor (in questo caso dei clienti) circondato dall'UI di EntityEditorHandler.

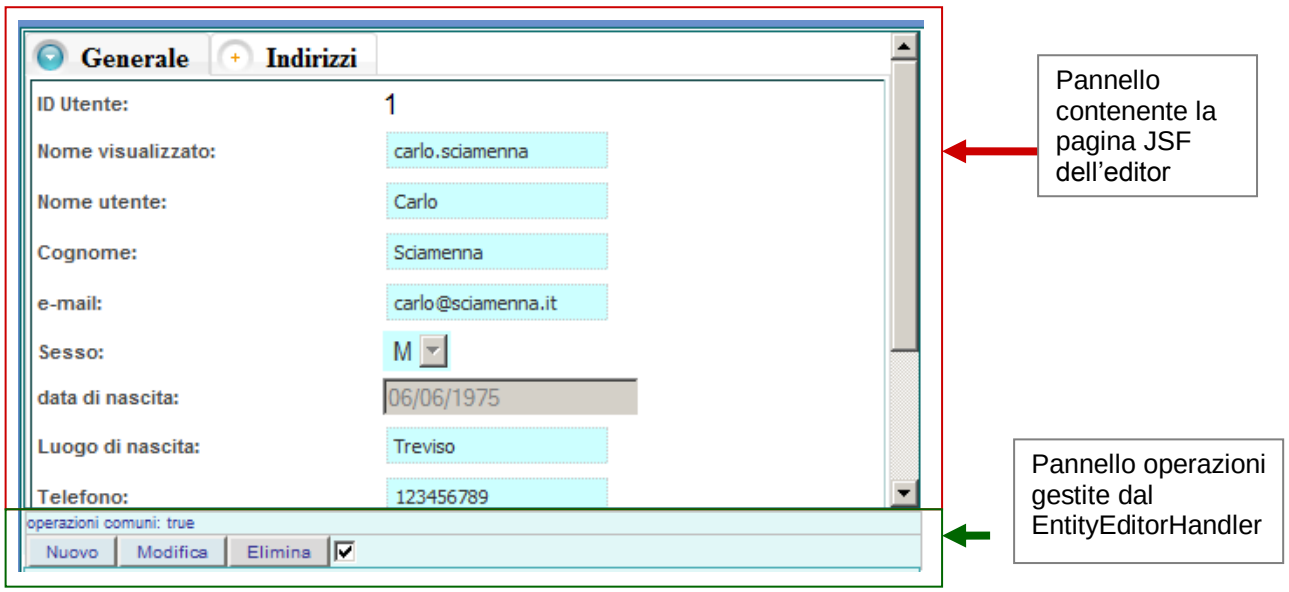


Figura 5-11: UI di EntityEditorHandler

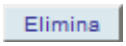
In rosso è evidenziato il pannello contenente la pagina dell'editor, mentre in verde il pannello contenente i pulsanti atti alla gestione delle operazioni nell'editor e controllati da EntityEditorHandler. Nella Figura 5-12 è riportata, ingrandita, la sezione con tali pulsanti.

Figura 5-12: Pulsanti del gestore di entità

Si osservano:

- 

Nuovo : avvia la creazione di una nuova istanza d'entità
- 

Modifica: modifica l'istanza correntemente selezionata.
- 

Elimina: elimina l'entità correntemente selezionata.
- La barra operazioni comuni, per ora vuota (in fase di sviluppo), conterrà le operazioni comuni effettuabili su un'entità selezionata.

Quando l'editor è in stato di editing la sezione dei pulsanti cambia e visualizza i pulsanti per annullare o confermare la modifica, inoltre l'editor è ingrandito ed occupa tutta la vista. In Figura 5-13 sono riportati tali pulsanti.

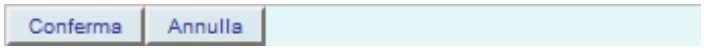


Figura 5-13:Pulsanti di conferma e annulla presenti in modifica

5.3.1 EntityEditorInterface : l'interfaccia per l'editor

Gli editors devono implementare l'interfaccia EntityEditorInterface per poter esser integrati nelle viste di AddCenter. Tali editors devono essere dei Managed Bean di JSF che interagiscono con la loro vista, da implementarsi tramite pagina JSF/ICEfaces.

L'interfaccia definisce i seguenti metodi, descritti nel seguito:

- public Class **getEntityClass**();
- public void **setEntityId**(Number id) throws EntityNotFoundException, SecurityException,ServerError;
- public Number **getEntityId**();
- public boolean **createNewEntity**();
- public void **deleteCurrentEntity**() throws EntityNotFoundException, SecurityException, ServerError;
- public void **modifyCurrentEntity**() throws EntityNotFoundException, SecurityException, ServerError;
- public boolean **confirmPerformed**() throws EntityNotFoundException, SecurityException, ServerError;
- public void **cancelPerformed**() throws EntityNotFoundException, SecurityException, ServerError;
- public boolean **isConfirmButtonEnabled**();
- public String **getEntityDescription**();
- public boolean **creationEnabled**();
- public boolean **modifyEnabled**();
- public boolean **deleteEnabled**();

public Class **getEntityClass**(): restituisce la classe dell'entità gestita

public void **setEntityId**(Number id) throws EntityNotFoundException, SecurityException,ServerError:

il gestore chiama tale metodo per impostare l'identificativo dell'oggetto da visualizzare/modificare, per non visualizzare alcuna entità si imposta id = null.

Le eccezioni che per contratto deve lanciare sono:

- javax.persistence.EntityNotFoundException: se I'id impostato non corrisponde ad alcuna istanza d'entità esistente.
- java.lang.SecurityException: se l'utente che vuole accedere all'entità non è autorizzato alla sua visualizzazione.
- it.addsource.addcenter.web.commonLogic.ServerError: quando vi è un errore da parte del software, ad esempio un errore di comunicazione con un session bean oppure un errore di concorrenza nell'accesso all'entità.

public Number **getEntityId**(): deve restituire l'id dell'entità impostata nell'editor. Deve ritornare null se non è impostato alcun oggetto, o se è in creazione.

public boolean **createNewEntity**(): il gestore chiama il metodo per avviare la creazione di una nuova entità. Deve essere restituito il valore booleano true se la creazione è effettuabile.

public void **deleteCurrentEntity**() throws EntityNotFoundException,SecurityException,ServerError:
Il metodo è chiamato per eliminare l'entità correntemente impostata. L'eliminazione è permanente se ha successo. Le eccezioni sono le stesse del metodo setEntityId e sono da generarsi nei seguenti casi:

- EntityNotFoundException: se l'entità impostata è già stata eliminata dal database
- SecurityException: se l'utente non è autorizzato ad eliminare l'entità
- ServerError: impossibile eliminare l'entità per altre cause dovute ad errori interni al server

public void **modifyCurrentEntity**() throws EntityNotFoundException,SecurityException,ServerError: chiamato dal gestore per avviare la modifica dell'entità correntemente impostata. Le eccezioni devono essere lanciate nei seguenti casi:

- EntityNotFoundException: se l'entità impostata che si vuole modificare è stata eliminata dal database
- SecurityException: se l'utente non è autorizzato ad modificare l'entità
- ServerError: impossibile modificare l'entità per altre cause dovute ad errori interni al server

public boolean **confirmPerformed**() throws EntityNotFoundException,SecurityException,ServerError: chiamato dal gestore per confermare la modifica. Deve restituire true se la modifica è stata effettuata con successo. Se il valore restituito è false il gestore mantiene l'editor nello stato di modifica.

public void **cancelPerformed**() throws EntityNotFoundException, SecurityException, ServerError: chiamato dal gestore per annullare l'operazione corrente di modifica. Le eccezioni devono essere lanciate nei seguenti casi:

- EntityNotFoundException: se l'entità in modifica è stata eliminata dal database
- SecurityException: se l'utente non può più accedere all'entità
- ServerError: impossibile ricaricare l'entità per altre cause, dovute ad errori interni al server

public boolean **isConfirmButtonEnabled**(): chiamato dal gestore per sapere se è possibile abilitare il pulsante di conferma dell'editing. Tale metodo funziona solo se i campi, che devono esser compilati e validi per poter procedere alla conferma, sono gestiti tramite il meccanismo di partialSubmit di ICEfaces. In caso contrario per eseguire il commit è necessario affidarsi totalmente al metodo confirmPerformed restituendo false se il commit non può esser eseguito. In implementazioni future di tale interfaccia si fornirà un meccanismo ad eventi per avvisare il gestore di eventi specifici.

public String **getEntityDescription**(): restituisce una stringa che fornisce una descrizione dell'istanza di entità correntemente gestita.

public boolean **creationEnabled**(): deve restituire true se l'editor permette la creazione di nuove entità.

public boolean **modifyEnabled**(): deve restituire true se l'editor permette la modifica dell'entità selezionata.

public boolean **deleteEnabled**(): deve restituire true se è possibile eliminare l'entità selezionata.

5.3.2 EntityEditor: Esempio

In tale sezione si illustrerà l'implementazione di un semplice editor per l'inserimento, modifica ed eliminazione delle categorie dei trattamenti nel gestionale AddCenter.
Il ManagedBean che implementa l'interfaccia EntityEditorInterface è TreatmentCategoryEditorBean ed è associato alla pagina TreatmentCategoryEditor.xhtml.
Nel seguito si riporta la classe Java di tale bean.

```
1  /*
2   * TreatmentCategoryEditorBean.java
3   */
4  package it.addsource.addcenter.web.editors.treatment;
5
6  import it.addsource.addcenter.logic.eao.TreatmentCategoryEAOLocal;
7  import it.addsource.addcenter.persistence.inventory.TreatmentCategory;
8  import it.addsource.addcenter.persistence.users.User;
9  import javax.ejb.EJB;
10 import javax.faces.FacesException;
11 import it.addsource.addcenter.web.commonLogic.editor.EntityEditorInterface;
12 import it.addsource.addcenter.web.commonLogic.ServerError;
13 import java.io.Serializable;
14 import javax.persistence.EntityNotFoundException;
15
16 /*
17 @ManagedBean(name="editors$treatment$TreatmentCategoryEditorBean")
18 @CustomScoped(value = "#{window}")
19  */
20 public class TreatmentCategoryEditorBean  implements EntityEditorInterface,Serializable {
21
22    //Inietto il SessionBean TreatmentCategoryEAO che è l'oggetto di accesso
23    //alle entità TreatmentCategory.
24    @EJB
25    private TreatmentCategoryEAOLocal treatmentCategoryEAO;
26
27    private TreatmentCategory treatmentCategory;
28
29    private boolean enabled;
30
31    /**Costruttore. Di default la proprietà enabled è falsa, cioè l'editor non è
32     * abilitato all'editing. */
33    public TreatmentCategoryEditorBean() {
34        enabled = false;
35    }
36    /** Restituisce l'oggetto TreatmentCategory associato ai campi visualizzati
```

```
37     */
38    public TreatmentCategory getTreatmentCategory() {
39        return treatmentCategory;
40    }
41    /** Imposta l'oggetto TreatmentCategory associato ai campi visualizzati. */
42    public void setTreatmentCategory(TreatmentCategory treatmentCategory) {
43        this.treatmentCategory = treatmentCategory;
44    }
45    /** Valore boolean che la vista utilizza per abilitare i campi di
46     *  inserimento/visualizzazione. Se enable = false, I campi non sono
47     *  abilitati all'inserimento.
48     */
49    public boolean isEnabled() {
50        return enabled;
51    }
52    public void setEnabled(boolean enabled) {
53        this.enabled = enabled;
54    }
55
```

In riga #17 è riportata l'annotazione @ManagedBean, presente nelle nuove specifiche JSF 2.0, che definisce la classe dichiarata come un bean JSF gestito dal container di nome "editors$treatment$TreatmentCategoryEditorBean" ed in riga #18 la visibilità del bean è dichiarata come personalizzata, di valore '#{window}'. Tali annotazioni sono commentate e hanno il solo scopo di ricordare che la classe è un bean gestito, l'effettiva dichiarazione del bean è effettuata tramite file di configurazione faces-config.xml. Si nota che la classe è dotata di tre attributi, il primo (riga #25), di nome `treatmentCategoryEAO`, è di tipo `TreatmentCategoryEAOLocal` ed è iniettato, mediante dependency injection, dal web container, il secondo (riga #27), di nome `treatmentCategory`, memorizza l'oggetto di tipo `TreatmentCategory` che è associato all'editor e i cui campi sono visualizzati nella vista. Il terzo attributo (riga #29), `enabled`, è booleano ed è utilizzato per abilitare/disabilitare la vista. Quando l'editor è in stato di visualizzazione esso ha valore '`false`', mentre quando è in stato di modifica ha valore '`true`'. I metodi `getTreatmentCategory` e `setTreatmentCategory` sono utilizzati dalla vista dell'editor per ottenere l'oggetto `treatmentCategory` e mappare gli attributi di questo con i campi di testo visualizzati.
I metodi `getEnabled` e `setEnabled` sono, banalmente, i metodi getter e setter della proprietà enabled e sono utilizzati dalla vista per controllare la sua abilitazione.
Nel seguito, da riga #56 a riga #200, è riportata l'implementazione dei metodi di EntityEditorInterface con la loro descrizione nei commenti.

```
56    /* **********************
57     *
58     * Implementazione dei metodi di EntityEditorInterface
59     *
60     */
61
62    /** Crea una nuova entità. Ciò si traduce nell' impostare un nuovo oggetto
63     *  TreatmentCategory e ad impostare la proprietà enable a true per
64     *  attivare i campi della vista.
65     */
66    @Override
67    public boolean createNewEntity() {
68        treatmentCategory = new TreatmentCategory();
69        setEnabled(true);
70        return true;
71    }
72    /** Restituisce l'id dell'entità correntemente impostata */
73    @Override
74    public Number getEntityId() {
75        if(treatmentCategory == null){
76          return null;
77        }else{
78          return treatmentCategory.getTreatmentCategoryId();
79        }
80    }
```

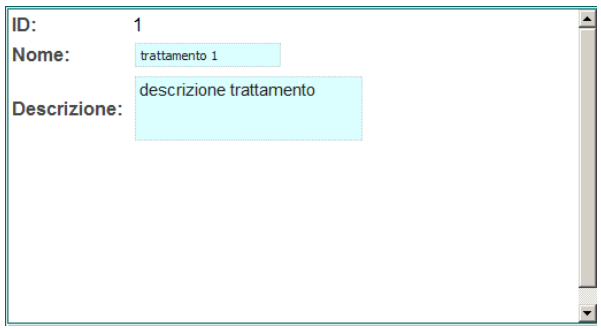
Continua a pagina seguente

```
81   /**
82    *  Chiamato dal gestore per annullare l'operazione corrente di editing.
83    *  Reimposta la visualizzazione dell'istanza d'entità in modifica. Se si
84    *  sta creando un nuovo oggetto, l'id sarà uguale a null e quindi l'editor
85    *  non visualizzerà alcun oggetto.
86    */
87   @Override
88   public void cancelPerformed() throws EntityNotFoundException, SecurityException, ServerError{
89      //se si stava creando una nuova entità reimposto editor,altrimenti si carica l'entità con l'id precedente
90      setEntityId(treatmentCategory.getTreatmentCategoryId());
91   }
92
93   /** Conferma le modifiche effettuate durante l'editing. Se l'oggetto
94    *  treatmentCategory ha id = null significa che si sta creando una nuova
95    *  entità e quindi viene chiamato il metodo "create" del SessionBean che
96    *  gestisce le categorie di trattamenti, altrimenti è invocato il suo
97    *  metodo "edit". Successivamente l'editor viene disabilitato.
98    */
99   @Override
100  public boolean confirmPerformed() throws EntityNotFoundException, SecurityException, ServerError{
101     //se si è in fase di creazione di una nuova entità invoco create, altrimenti edit
102     if(treatmentCategory.getTreatmentCategoryId() == null){
103       treatmentCategoryEAO.create(treatmentCategory);
104     }else{
105       treatmentCategoryEAO.edit(treatmentCategory);
106     }
107     setEnabled(false);
108     return true;
109  }
110
111  /**
112   * Invocato per la rimozione dell'entità correntemente selezionata. Esso è
113   * eseguito con successo solo se la categoria non ha trattamenti associati.
114   * Per la rimozione è chiamato il metodo remove del Session Bean associato.
115   */
116  @Override
117  public void deleteCurrentEntity() throws EntityNotFoundException, SecurityException, ServerError{
118     if(!deleteEnabled()){
119        throw new  SecurityException("Impossibile eliminare una categoria con trattamenti associati");
120     }else{
121        treatmentCategoryEAO.remove(treatmentCategory);
122        setEnabled(false);
123     }
124  }
125  /**Imposta l'id dell'oggetto gestito. Disabilito l'editor. Se l'id è null
126   * imposto tratmentCategory a null, altrimenti cerco l'oggetto con tale id.
127   * Se tale oggetto non esiste si ritorna l'eccezione EntityNotFoundException.
128   */
129  @Override
130  public void setEntityId(Number id) throws EntityNotFoundException, SecurityException, ServerError{
131     setEnabled(false);
132     if(id == null){
133        treatmentCategory = null;
134     }else{
135        treatmentCategory = treatmentCategoryEAO.find(id);
136        if(treatmentCategory == null){
137           throw new EntityNotFoundException("entità non trovata");
138        }
139     }
140  }
141  /** Chiamato per modificare l'entità correntemente impostata.
142   *   Se non è impostata alcuna entità si restituisce un errore. Altrimenti
143   *   è ricaricata l'entità ed abilitato l'editor.
144   */
145  @Override
146  public void modifyCurrentEntity() throws EntityNotFoundException, SecurityException, ServerError{
147     if(treatmentCategory == null){
148        throw new ServerError("entità non impostata");
149     }
150     setEntityId(treatmentCategory.getTreatmentCategoryId());
151     setEnabled(true);
152  }
```

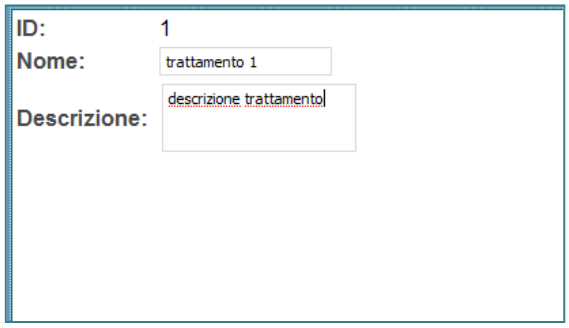
Continua a pagina seguente

```
153    /** La conferma delle modifiche è abilitata solo se il campo nome è diverso
154     *  da null e dalla stringa vuota. */
155    @Override
156    public boolean isConfirmButtonEnabled() {
157       if(treatmentCategory == null || treatmentCategory.getName() == null || treatmentCategory.getName().equals("") ) {
158          return false;
159       }else{
160          return true;
161       }
162    }
163    /** Restituisce la descrizione dell'entità corrente selezionata.
164     *  Qui banalmente è restituita come descrizione 'Categoria trattamento'.
165     */
166    @Override
167    public String getEntityDescription() {
168       return "Categoria trattamento";
169    }
170    /**
171     * La creazione di nuove entità è sempre possibile
172     */
173    @Override
174    public boolean creationEnabled() {
175       return true;
176    }
177    /** Solo se non vi sono trattamenti associati è possibile rimuovere
178     *   la categoria. */
179    @Override
180    public boolean deleteEnabled() {
181       if(treatmentCategory == null || treatmentCategory.getTreatmentCategoryId() == null ||
182             !treatmentCategory.getTreatmentCollection().isEmpty()){
183          return false;
184       }else{
185          return true;
186       }
187    }
188    /**
189     * La modifica è sempre effettuabile su qualunque entità
190     *
191     */
192    @Override
193    public boolean modifyEnabled() {
194       return true;
195    }
196    /**  Restituisce la classe dell'entità gestita.  */
197    @Override
198    public Class getEntityClass() {
199       return TreatmentCategory.class;
200    }
201
202 }
```

Si è visto il codice del backing bean dell'editor. Nel seguito si descrive la vista e come questa è mappata al bean.
In Figura 5-14 è riportata la pagina TreatmentCategoryEditor.xhtml visualizzata nel broswer Firefox.



Editor trattamento in stato di visualizzazione
(campi con disabled = true)



Editor trattamento in stato di editing
(campi con disabled = false)

Figura 5-14: Editor delle categorie dei trattamenti

In Figura 5-15 il codice xhtml della pagina TreatmentCategoryEditor.xhtml. Si nota l'utilizzo dei tag di ICEfaces di prefisso 'ice' per la sua definizione.

```
7 <ice:form    xmlns:h="http://java.sun.com/jsf/html"
 8             xmlns:f="http://java.sun.com/jsf/core"
 9             xmlns:ice="http://www.icesoft.com/icefaces/component"
10             id="treatmentCategoryEditorForm">
11
12  <ice:outputStyle href="/resources/css/center/editorStyle/treatmentEditorStyle.css" id="treatmentEditorStyle"/>
13
14    <ice:panelGroup id="treatmentCategoryEditorMainPanel" styleClass="treatmentCategoryEditorMainPanel">
15
16      <ice:panelGrid id="treatCatEditorGrid" columns="3" styleClass="treatCatEditorGrid">
17        <!--Riga-->
18        <ice:outputLabel id="treatCatIdLabel" value="ID:" for="treatCatId" />
19        <ice:outputText id="treatCatId"
                          value="#{editors$treatment$TreatmentCategoryEditorBean.treatmentCategory.treatmentCategoryId}"/>
20        <ice:outputText id="treatCatIdVoidText"/>
21
22        <!--Riga-->
23        <ice:outputLabel id="treatCatNameLabel" value="Nome:" for="treatCatName" />
24        <ice:inputText id="treatCatName" value="#{editors$treatment$TreatmentCategoryEditorBean.treatmentCategory.name}"
25                       maxlength="299"  disabled="#{!editors$treatment$TreatmentCategoryEditorBean.enabled}" required="true"
26                       partialSubmit="true" styleClass="treatCatField"/>
27        <ice:message id="treatCatNameMsg" for="treatCatName" />
28
29        <!--Riga-->
30        <ice:outputLabel id="treatCatDescriptionLabel" value="Descrizione:" for="treatCatDescription" />
31        <ice:inputTextarea id="treatCatDescription"
                             value="#{editors$treatment$TreatmentCategoryEditorBean.treatmentCategory.description}"
32                           disabled="#{!editors$treatment$TreatmentCategoryEditorBean.enabled}" partialSubmit="true"
                             styleClass="treatCatField"/>
33        <ice:message id="treatCatDescriptionMsg" for="treatDescriptionName" />
34
35      </ice:panelGrid>
36    </ice:panelGroup>
37
38 </ice:form>
```

Figura 5-15:TreatmentCategoryEditor.xhtml

In riga #7 è definito il form che gestisce la vista e nel suo campo di visibilità sono definiti gli spazi dei nomi dei tags html *(xmlns:h="http://java.sun.com/jsf/html")*, JSF *(xmlns:f=http://java.sun.com/jsf/core)* ed ICEfaces *(xmlns:ice=http://www.icesoft.com/icefaces/component)*. Il tag *ice:outputStyle* [69] in riga #12 connette il foglio di stile CSS "/resources/css/center/editorStyle/treatmentEditorStyle.css" alla pagina. Il tag *ice:panelGroup* in riga #14 definisce un pannello esterno che racchiude i componenti dell'editor. Al suo interno è definito il tag *ice:panelGrid* (riga #16) che definisce una griglia di tre colonne (tradotta in una tabella XHTML) nella quale saranno disposti i componenti dell'editor. I componenti definiti all'interno del tag sono disposti , a partire dal primo, in righe a gruppi di tre consecutivi e tale organizzazione è mostrata con i commenti *<!-- Riga -->* di linea #17,22 e 29.
La prima riga della griglia, definita nelle righe di codice #18,19,20, contiene i componenti per l'output dell'identificativo della categoria trattamento. In riga #18 il tag *ice:outputLabel* definisce l'etichetta 'ID:' che si può visualizzare in Figura 5-14 . In riga #19 il tag *ice:outputText* visualizza l'identificativo della categoria, associando il valore in uscita alla proprietà #{editors$treatment$TreatmentCategoryEditorBean.treatmentCategory. treatmentCategoryId}. Viene cioè chiamato tramite expression language il bean TreatmentCategoryEditorBean, ottenuto l'oggetto treatmentCategory ed infine prelevato da questo l'identificativo.
In riga #20 il tag visualizza un'etichetta vuota utilizzata per manetenere l'allineamento alla griglia.
La seconda riga della griglia, definita nelle righe di codice #23-27, ha i componenti per la gestione del nome della categoria trattamento. Il primo componente visualizza l'etichetta 'Nome:' , mostrata in Figura 5-14, mentre il secondo, in riga #24, è il tag *ice:inputText* e serve a visualizzare un campo (su singola linea) di inserimento testo per il nome della categoria.
Il suo valore (definito mediante l'attributo '*value'*) è mappato con l'attributo '*name'* dell'oggetto treatmentCategory corrente tramite l'espressione *#{editors$treatment$TreatmentCategoryEditorBean.treatmentCategory.name}*.
In riga #27 il tag *ice:message* defnisce un messaggio d'errore per il tag in riga #24 ed è usato da ICEfaces per segnalare eventuali errori di inserimento del valore.Ad esempio viene inserito un tipo di dati diverso da quello richiesto oppure al momento della convalida il campo, che è obbligatorio, è vuoto.
La terza riga della griglia ,definita nelle righe di codice #30-33, contiene i componenti per la gestione della descrizione della categoria. Questi sono molto simili a quelli precedentemente descritti per la seconda riga della griglia, con l'unica eccezione che il componente di input utilizzato è *ice:inputTextarea* invece di *ice:inputText*. La differenza tra i due è che

il secondo visualizza un campo di testo su un'unica riga, mentre il primo visualizza un’area di inserimento. In questo caso è stato utilizzato *ice:inputTextarea* perché la descrizione,se presente, è in genere più prolissa del nome della categoria.

In Figura 5-16 si può osservare la corrispondenza tra i marcatori ICEfaces descritti ed i componenti della vista dell’editor.

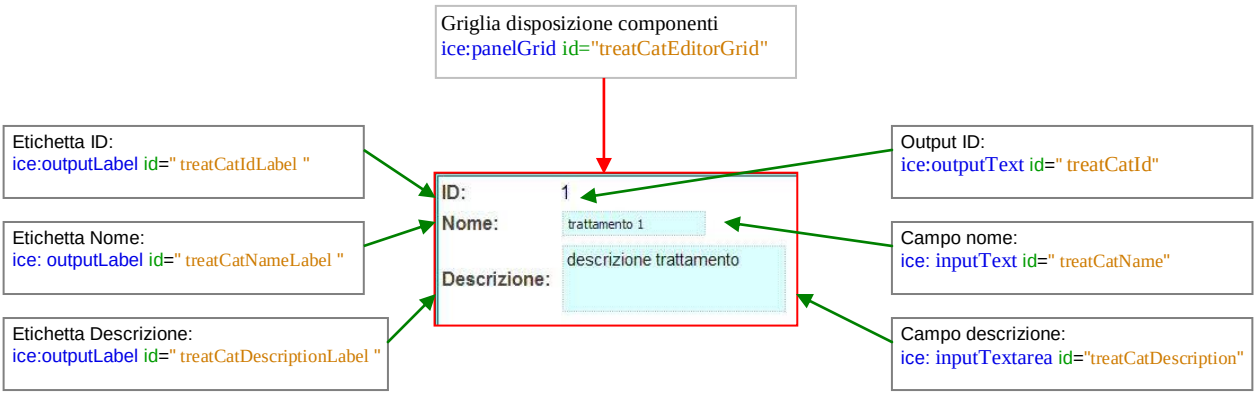


Figura 5-16: Corrispondenza tra componenti vista e tags

CONCLUSIONI

Nello svolgimento del tirocinio all'interno di AddSource ho appreso tecnologie di sviluppo di applicazioni enterprise basate sul web. In particolare ho imparato ad utilizzare e scrivere applicazioni per la piattaforma Java EE. In dettaglio ho studiato molti dei suoi moduli e librerie principali quali EJB (Entity e Session Beans), JMS, JTA, JNDI, Servlet, JSP e JSF. Ho appreso come amministrare l'application server Java EE GlassFish ed effettuare in esso il deploy delle applicazioni.
Ho poi avuto modo di progettare e realizzare una base di dati utilizzata in un ambiente reale, migliorando le conoscenze di basi di dati acquisite con gli studi ed applicandole in campo reale. Ho potuto in proposito relazionarmi con più persone ed acquisire reale documentazione descrittiva della realtà di interesse per l'analisi e progettazione.
Ho imparato a scrivere applicazioni internet ricche mediante il framework ICEfaces, apprendendo inoltre in generale le metodologie Java per lo sviluppo di interfacce web che separano la vista, dichiarata mediante file XHTML, dal controllore, implementato mediante classi Java bean gestite dal container.
L'esperienza mi è stata molto utile anche dal punto di vista della progettazione software. Ho appreso la programmazione modulare che consiste nello suddividere efficientemente in moduli il software da realizzare definendo in modo rigoroso le specifiche per ognuno di essi. A tale proposito ho imparato a relazionarmi con altre persone per scrivere in modo coordinato un applicativo, dividendolo in moduli da sviluppare per poi unirli per formare l'applicativo vero e proprio.
Il mio lavoro si è suddiviso in una prima fase di studio, dove ho appreso l'utilizzo di tutte le tecnologie necessarie. Si è caratterizzato dallo studio dei libri *'EJB in action' di Debu Panda, Reza Rahman, Derek Lane, 2007. Edizioni Manning.* e 'The Java EE 5 Tutorial,Volume I' di Sun Microsystem per quanto riguarda l'apprendimento degli EJB 3 comprendente la logica di business e l'API di persistenza e lo studio di tutorial e documentazione online per apprendere le tecnologie JSP, JSF ed ICEfaces. In particolare ho fatto riferimento ai siti web http://www.coreservlets.com/, http://www.icefaces.org/, http://dev.java.net e ai libri 'ICEfaces Developer's Guide v1.8' , ICEsoft Technologies, Inc. Inoltre ho imparato ad utilizzare l'application server GlassFish 2.1 studiando la guida di amministrazione 'SunGlassFish Enterprise Server 2.1 AdministrationGuide', Sun Microsystem. Infine ho studiato il mondo degli applicativi gestionali per capirne i concetti chiave, analizzando alcuni gestionali esistenti quali Zoho CRM.
La seconda fase del lavoro è stata entrare nello sviluppo dell'applicativo, iniziando dall'analisi della realtà da studiare dei centri di cura del corpo assieme al team di lavoro, per poi progettare ed implementare la base di dati.
Terminata questa fase di lavoro ho iniziato la progettazione ed implementazione dei moduli software assegnatomi, alcuni dei quali ho descritto nei capitoli precedenti. In particolare ho avuto modo di sviluppare logica di business mediante la scrittura di alcuni bean per l'accesso alle entità (per il framework QuerySystem ed alcuni editor) e di scrivere codice per realizzare interfacce grafiche web ricche.
Ho quindi potuto apprendere l'intera progettazione di un'applicazione enterprise sviluppando componenti per tutti i livelli in cui è suddivisa (livello di presentazione, livello logico e livello dei dati).
Tutti gli obiettivi che mi sono stati assegnati all'inizio del tirocinio sono stati raggiunti e hanno contribuito al funzionamento dell'applicativo.
In particolare il framework per la creazione rapida di software gestionali è completamente realizzato ed operativo e con esso si è sviluppata l'applicazione AddCenter.
L'obiettivo di renderla operativa e funzionante per la gestione di tutti gli aspetti principali dei centri di cura del corpo, quali la gestione dei clienti, ricevute, prodotti, trattamenti e pacchetti è stato pienamente completato.
Si prevede in futuro la sua espansione per controllare aspetti secondari di tali centri e l'evoluzione dell'interfaccia grafica e delle caratteristiche fornite.

LISTA DEGLI ACRONIMI

ACRONIMO	Definizione
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BI	Business intelligence
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
D2D	Direct-to-DOM
DOM	Document Object Model
EJB	Enterprise JavaBean
ERP	Enterprise Resource Planning
GUI	Gaphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JAAS	Java Authentication and Authorization Service
Java EE	Java Enterprise Edition
Java NIO	Java New Input/Output
JAXP	Java API for XML Processing
JDBC	Java Database Connectivity
JMS	Java Message Service
JNDI	Java Naming and Directory Interface (
JPA	Java Persistence API
JPQL	Java Persistence Query Language
JSF	Java Server Faces
JSP	Java Server Pages
JSR	Java Transaction aPLI
JTA	Java Transaction API
JVM	Java Virtual Machine
LAMP	Linux, Apache, MySql, PHP
MQ	Open Message Queue
MRP	Materials Requirements Planning
MVC	Model-View-Controller
ORM	Object Relational Mapping
PHP	Hypertext Preprocessor
RDBMS	Relational database management system
RIA	Rich Internet Application
RMI	Remote Method invocation
URL	Uniform Resource Locator
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language

INDICE DELLE FIGURE

BIBLIOGRAFIA E SITOGRAFIA

[1] http://java.sun.com/javase/technologies/security/

[2] http://it.wikipedia.org/wiki/Rich_Internet_application

[3] http://www.facebook.com/

[4] http://www.twitter.com/

[5] http://www.myvip.com/

[6] http://it.wikipedia.org/wiki/Computer_cluster/AJAX

[7] http://www.w3.org/XML/ . "Introduzione a XML" di Anders Møller,Micheal I. Schwartzbach. Pearson (2007)

[8] http://www.icefaces.org

[9] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 19.

[10] http://it.wikipedia.org/wiki/Computer_cluster

[11] http://java.sun.com/javaee/

[12] http://www.microsoft.com/net/

[13] http://www.adobe.com/it/products/flex/

[14] http://www.zend.com/products/server/

[15] http://www.zope.org/

[16] http://jcp.org/en/jsr/overview

[17] http://www.google.com/trends

[18] http://jcp.org/en/jsr/overview

[19] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 4.

[20] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 587-595.

[21] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 176-216.

[22] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 110-139.

[23] 'Introducing the Java EE 6 Platform'. By Ed Ort, 2009.
http://www.oracle.com/technetwork/articles/javaee/javaee6overview-141808.html

[24] Java EE 6 Tutorial,Volume . Sun Microsystem (2009).

[25] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 19-23

[26] http://jboss.org/

[27] https://glassfish.dev.java.net/

[28] http://www.jboss.com

[29] http://www.hibernate.org/

[30] 'Sun GlassFish Enterprise Server 2.1 Administration Guide', Sun Microsystem, 2008

[31] http://en.wikipedia.org/wiki/GlassFish

[32] http://www.eclipse.org/eclipselink/

[33] 'Sun GlassFish Enterprise Server 2.1 Administration Guide', Sun Microsystem, 2008. pp. 28

[34] https://mq.dev.java.net/

[35] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007.

[36] http://www.coreservlets.com/JSF-Tutorial/

[37] http://www.jcp.org/en/jsr/detail?id=314

[38] http://it.wikipedia.org/wiki/Icefaces

[39] 'ICEfaces Developer's Guide version 1.8', ICEfaces Technologies, Inc. (2009). pp. IV

[40] http://www.coreservlets.com/JSF-Tutorial/

[41] http://www.oracle.com/technology/tech/java/newsletter/articles/introjsf/index.html

[42] http://it.wikipedia.org/wiki/AJAX, http://www.w3schools.com/Ajax/Default.Asp

[43] 'ICEfaces Developer's Guide version 1.8', ICEfaces Technologies, Inc. (2009).

[44] http://www.mysql.it/

[45] http://www.mysql.it/why-mysql/

[46] http://www.mysql.it/downloads/connector/

[47] http://www.phpmyadmin.net/

[48] http://www.netbeans.org

[49] http://it.wikipedia.org/wiki/Eclipse_(informatica)

[50] http://it.wikipedia.org/wiki/Imposta_sul_valore_aggiunto

[51] 'Sistemi di basi di dati'.Fondamenti di Elmasri e Navathe . 5ª Edizione. Edizioni Pearson-Addison Wesley (2007)

[52] 'Sistemi di basi di dati'.Fondamenti di Elmasri e Navathe . 5ª Edizione. Edizioni Pearson-Addison Wesley (2007),pp. 105-136

[53] http://wb.mysql.com/

[54] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 217-292

[55] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 215,227

[56] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 228,229

[57] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 258-262

[58] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 284-292

[59] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 284-292

[60] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 284-292

[61] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 258-262

[62] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 278-281

[63] http://it.wikipedia.org/wiki/Secure_Hash_Algorithm

[64] http://it.wikipedia.org/wiki/Base64

[65] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 281-283

[66] http://it.wikipedia.org/wiki/Software_gestionale

[67] http://crm.zoho.com/crm/login.sas

[68] 'EJB in action'. Debu Panda, Reza Rahman, Derek Lane. Edizioni Manning, , 2007. pp 305,306

[69] http://www.icefaces.org/component-showcase