Università degli studi di Napoli Federico II
Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

TESI DI LAUREA

Una Web Application per la consultazione di una Data WareHouse di dati chimico-fisici sugli Alimenti

Relatore
Prof. Lucio Sansone

Correlatore
Prof. Antonio d'Acierno

Candidato
Fabio Neiviller
Matricola 041/2804

Anno Accademico 2005/2006

…ai miei genitori e a mia sorella
…a tutte le persone a cui voglio bene,
e che mi hanno sostenuto nel
lungo corso dei miei
studi universitari.

Ringraziamenti

Al termine di un lavoro appassionante, complesso e stimolante...
Eccomi giunto alla chiusura di un importante capitolo della mia vita: gli anni universitari. Ringraziare tutte le persone che in un modo o nell'altro mi hanno aiutato a scriverlo più che un dovere è una necessità.

Ringrazio il mio relatore Prof. *Lucio Sansone* per la sua grande disponibilità e serietà. Un ringraziamento speciale va al mio correlatore Prof. *Antonio d'Acierno*, una persona scrupolosa, intelligente e divertente, per avermi proposto una tesi molto interessante e per la libertà, la fiducia e i mirati suggerimenti che ha saputo darmi. Inoltre, ringrazio l'ing. *Fabrizio di Cesare* per l'ottimo lavoro svolto sulla progettazione della Data WareHouse.

Un sentito ringraziamento va alla mia famiglia, *Mamma*, *Papà*, mia sorella *Angela e Willy*, per aver sempre creduto in me e per avermi sostenuto in questo percorso universitario, che è stato lungo ma è arrivato finalmente alla fine. Con questo lavoro spero di poter ripagare almeno in parte tutti i sacrifici che hanno fatto per permettermi di arrivare a questo traguardo da me tanto desiderato, a loro dedico la mia tesi.

Un grazie anche ai miei parenti più stretti: a mio *Zio Salvatore* e *Zia Anna*, per avermi dato la possibilità di trascorrere una stupenda vacanza a Cuba; a *Marco e Marilena*, che mi hanno sopportato ogni domenica per le partite del Napoli; a *Zia Giovanna e Zio Mario*, per essere stati sempre a disposizione in qualsiasi circostanza; ringrazio, inoltre, *Zia Anna*, *Zio Franco* e tutti gli altri che hanno in qualche modo dato sempre il loro apporto.
Un ringraziamento particolare va a mio *Zio Mario* che ha sempre mostrato per quello che faccio una fiducia cieca e priva di incertezze, spronandomi sempre ad andare avanti per la mia strada.

Rispetto ai primi anni di università non sono soltanto cambiato io, sono cambiate anche le persone intorno a me.

Penso allora sia giusto esprimere un pensiero anche per quanti hanno condiviso con me questi ultimi anni, diventando pian piano dei punti di riferimento sui quali spero di poter contare ancora in futuro.

Un grazie di cuore alla mia amica *Francesca*, con cui ho condiviso gioie e dolori di questi anni universitari, che mi ha aiutato con discrezione e pazienza a superare le mie delusioni più grandi, e che ancora oggi mi incoraggia e mostra di credere in me e in quello che faccio. Grazie per essermi stata sempre vicina senza di te non sarei riuscito ad arrivare fino in fondo!

Sicuramente un ringraziamento particolare va a *Paolo*, un amico leale e sincero, abbiamo cominciato insieme questa lunga avventura partendo dalle superiori e affrontando insieme tutti i problemi che ci sono capitati di fronte.

Ringrazio *Paola e Nella*, per essermi state vicine in tutti i momenti di difficoltà e delle quali apprezzo sempre i consigli sinceri.

Un affettuoso pensiero va anche ai miei amici di sempre: *Alessandro, Gino, Mauro e Stefano*, con i quali sono praticamente cresciuto, e a coloro, tra quelli che ho conosciuto in questi anni, che da semplici conoscenti sono diventati degli amici che mi hanno dato affetto e amicizia sincera: senza di loro neanche una parola scritta in questa tesi avrebbe senso.

Grazie di cuore a Tutti.

Indice

Capitolo I – Introduzione ***1***

1. Il Problema Affrontato *1*
2. Stato dell'Arte *2*
3. Definizioni Fondamentali *3*
4. Linee Guida del Progetto *4*
5. Passi Fondamentali *5*
6. Struttura del Sistema *6*
7. Tecnologie Utilizzate *7*
8. L'architettura delle Applicazioni Web-based *10*
9. Request/Response *14*
10. Le basi del JavaServer Faces *17*
11. Java Servlet API *19*
12. JSP - Java Server Pages *21*
13. I Patterns *21*
14. Il pattern MVC (Model-View-Controller) *22*
14.1 Model 1 ..... *23*
14.2 Model 2 ..... *23*
14.3 MODEL *25*
14.4 VIEW *26*
14.5 CONTROLLER *26*

Capitolo II – Il framework JavaServer Faces ***27***

1 . Introduzione *27*
2. Framework Models *28*
2.1 Execution Model *29*
2.1.1 FacesServlet *29*
2.1.2 Lifecycle - PhaseListener *30*
2.1.3 Application *30*
2.1.4 FacesContext - ExternalContext *31*
2.2 User Interface Component Model *33*
2.3 Component Rendering Model *36*
2.4 Conversion Model *38*
2.5 Event and Listener Model *39*
2.5.1 FacesEvent e FacesListener *40*
2.6 Validation Model *41*
2.7 Navigation Model *43*
2.8 Backing Bean Management *43*
3. Ciclo di vita di una pagina JavaServer Faces *46*
3.1 Scenari di elaborazione di una richiesta *47*
3.2 Ciclo di vita Standard *48*
3.2.1 Restore View *49*

3.2.2 Apply Request Values *49*
3.2.3 Process Validation *50*
3.2.4 Update Model Values *50*
3.2.5 Invoke Application *51*
3.2.6 Render Response *51*
4. JSF Expression Language *52*
5. Espandibilità del framework *53*
5.1 Custom Converter *53*
5.2 Registrare un Custom Converter *54*
5.3 Event Listener *54*
5.3.1 Implementazione di un ActionListener/ValueChangeListener *55*
5.3.2 Backing Bean Method Listener *56*
5.4 Custom Validator *56*
5.4.1 Implementazione di un Validator *57*
5.4.1.1 Class Validator *58*
5.4.1.2 Tag Handler *58*
5.4.1.3 Tag Library Descriptor *59*
5.4.2 Backing Bean Method Validator *59*
5.4.3 Registrare un Custom Validator *60*
5.5 Custom UI Components *60*
5.5.1 Registrare un Custom Component *62*
5.5.2 Class Component *63*
5.5.3 Tag Handler *65*
5.5.4 Tag Library Descriptor *66*
5.5.5 Classe Renderer *66*
5.5.6 Registrare un Custom Renderer in un Renderer Kit *67*
6. Sicurezza *68*
7. Configurazione dell'applicazione *69*
7.1 Configurazione dei Backing Beans *69*
7.2 Localizzazione, Internazionalizzazione e Messaggi *71*
7.3 Configurare le Navigation Rules *71*

Capitolo III – Data WareHouse ***73***

1. Ricerca delle Fonti *73*
2. Fonti Selezionate *74*
3. Fonte INRAN *75*
3.1 Il Database *75*
3.2 Il Software di Consultazione *75*
3.3 Stato Iniziale del Database *75*
3.4 Tabelle Escluse dal Processo di Reverse Engineering *75*
3.5 Considerazioni sull'Uso del Database Come Fonte per il Datawarehousing *76*
3.6 Fonti di Dati INRAN Non Presenti nel Database *76*
3.7 Ristrutturazione dello Schema Concettuale *77*
3.7.1 Schema "INRAN Final Conceptual" *78*
3.8 Ulteriori Vincoli *78*

3.9 Analisi delle Ridondanze *79*
4. Fonte USDA *79*
4.1 Il Database *79*
4.2 Il Software di Consultazione *80*
4.3 Stato Iniziale del Database *80*
4.4 Tabelle Escluse dal Processo di Reverse Engineering *81*
4.5 Considerazioni sull'Uso del Database Come Fonte per il Datawarehousing *81*
4.6 Fonti di Dati USDA-NDL Non Presenti nel Database *81*
4.7 Ristrutturazione dello Schema Concettuale *82*
4.7.1 Schema "USDA Final Conceptual" *84*
4.8 Problemi di Ristrutturazione Risolti Parzialmente *84*
4.10 Analisi delle Ridondanze *85*
5 Fonte IEO *85*
5.1 Il Database *85*
5.3 Considerazioni sull'Uso del Database Come Fonte per il Datawarehousing *87*
5.4 Fonti di Dati IEO Non Presenti nel Database *87*
5.5 Creazione dello Schema Concettuale *87*
5.5.1 Schema "IEO Final Conceptual" *88*
5.6 Informazioni Non Presenti nel File ASCII *89*
5.8 Analisi delle Ridondanze *90*
6. Fonte ISA-CNR *90*
6.1 Il Database *90*
6.2 Stato Iniziale del Database *91*
6.3 Considerazioni sull'Uso del Database Come Fonte per il Datawarehousing *91*
6.4 Creazione dello Schema Concettuale *92*
6.4.1 Schema "ISA-CNR Final Conceptual" *93*
6.5 Analisi delle Ridondanze *93*
7. Il DataBase di Integrazione *94*
7.1 Le Entità Fondamentali *94*
7.2 Le Altre Entità *95*
7.3 Le Entità "Standard" *97*
7.4 Il Supporto Multilingua *98*
7.5 Dizionario dei Dati *101*
7.5.1 Alimento *101*
7.5.2 Categoria originaria *101*
7.5.3 Categoria standard *102*
7.5.4 Commento statistico *102*
7.5.5 Componente *102*
7.5.6 Componente standard *103*
7.5.7 Determinazione valore *103*
7.5.8 Fonte primaria *103*
7.5.9 Nota valore *104*
7.5.10 Fonte secondaria *104*
7.5.11 Tipo rifiuti *105*
7.5.12 Nota alimento *105*

7.5.13 Peso *106*
7.5.14 Valore *107*
7.5.15 Tipo di valore *108*
7.5.16 Descrizione generica *108*
7.5.17 Descrizione alimento *108*
7.5.18 Altre Descrizioni *109*
7.6 Informazioni sulle Singole Associazioni *109*
7.6.1 Origine alimento *109*
7.6.2 Origine categoria *109*
7.6.3 Origine componente *110*
7.6.4 Origine secondaria *110*
7.6.5 Assegnazione del Tipo agli Attributi *111*
7.7 Ulteriori Vincoli *111*
7.8 Diagrammi Completi *112*
7.8.1 Schema "Final Conceptual" *113*
7.8.2 Schema "Final Physical" *115*
8. I Moduli di Feeding *117*
8.1 Nota Iniziale *117*
8.2 I Requisiti Software *117*
8.2.1 Inserisci fonte *118*
8.2.2 Elimina fonte *119*
8.2.3 Aggiorna da fonte *119*
8.3 La Struttura delle Classi *119*
8.3.1 Un Ulteriore Sguardo alle Tecnologie *119*
8.3.2 La Gerarchia Principale *120*
8.3.3 La Gerarchia dei Buffer *121*
8.4 Informazioni sulle Singole Classi *123*
8.4.1 DWManager *123*
8.4.2 DWManagerFonteODBC *125*
8.4.3 DWManagerFonteASCII *126*
8.4.4 DWManagerINRAN, DWManagerUSDA, DWManagerIEO ..*127*
8.4.5 Tupla *127*
8.4.6 Le Classi Buffer *128*
8.5 L'Interfaccia Utente *128*
8.6 Aggiunta di Altre Fonti *128*
8.7 Architettura del Sistema *129*

Capitolo IV – Case Study : Analisi e Progettazione ..130

1. Introduzione *130*
2. Requisiti *131*
3. Progettazione *132*
3.1 Use Case Diagrams *133*
3.1.1 Generale *133*
3.1.3 Registrazione *134*
3.1.4 Caso d'uso comune:i Ricerca *135*
3.1.5 Gestione Utenti *136*

3.1.6 Gestione Alimenti *136*
3.1.7 Casi d'uso comuni : Login, Logout, Gestione Dati Personali ... *138*
3.2 Class Diagram *139*
3.3 Activity Diagrams – Sequence Diagrams *142*
3.3.1 Login *144*
3.3.2 Logout *148*
3.3.3 Registrazione *151*
3.3.4 Richiesta Username / Password *152*
3.3.5 Visualizzazione e Ricerca Alimenti *154*
3.3.6 Inserimento, Modifica ed Eliminazione Alimento *156*
3.3.7 Modifica Dati Personali *164*
3.3.8 Gestione Utenti *168*
3.4 Statechart Diagrams *170*

Capitolo V – Case Study : Implementazione 173

1. Introduzione *173*
2. Struttura dell'applicazione *174*
3. Controller *176*
3.1 Gestione delle sessioni *177*
3.2 Protezione pagine *178*
4. View *180*
4.1 Le librerie Standard *180*
4.1.1 I form : contenuto e layout *180*
4.1.2 Costruzione condizionale dei contenuti *181*
4.1.3 DataTable JSF *182*
4.2 Custom Components *183*
4.2.1 DateSelectComponent *184*
4.2.2 DDListComponent *187*
4.3 Funzionalità del Sistema *190*
4.3.1 Login, Registrazione e Richiesta password *190*
4.3.2 Modifica dati utente ed amministratore *192*
4.3.3 Visualizzazione e ricerca degli alimenti *193*
4.3.4 Inserimento/Modifica e Cancellazione alimenti *198*
4.3.5 Gestione Utenti *202*
4.4 Internazionalizzazione (I18N) *203*
5. Validazione *205*
5.1 Custom Validators *205*
5.1.1 EmailValidator *206*
5.1.2 SeqCifreValidator *209*
5.1.3 TelefonoValidator *210*
6. Model *212*
6.1 Classi Exception *213*
6.2 Classi di interfacciamento con il DataBase *214*
6.4.1 Package accesso *223*
6.4.2 Package ruolo *225*
6.4.3 Package alimenti *230*

6.4.8 Package mail *232*
6.5 I Backing Beans *233*
8. Navigazione *234*
9. Eccezioni *235*
10. Sicurezza *235*

Appendice A – SRS Web Application Alimenti ***239***

Riferimenti Bibliografici ***261***

Capitolo I – Introduzione

1. Il Problema Affrontato

Oggetto di questa tesi è la descrizione del processo di progettazione e sviluppo di una applicazione web per la memorizzazione, la gestione e la consultazione dei dati in una Data Warehouse riguardanti la composizione chimico-fisica degli alimenti.

Ciò che ha dato il via al progetto è stata la constatazione che, da quanto emerge dalle ricerche svolte, sono ben poche le risorse direttamente accessibili via web riguardanti la composizione degli alimenti e che siano anche in italiano.

A questo si aggiunge l'interesse sempre più sentito da parte degli utenti di conoscere la composizione degli alimenti, anche grazie ad una maggiore sensibilizzazione alla scienza fornita dai media. In quanto il tipo di alimentazione influisce sul comportamento del corpo e dell'umore; la combinazione dei cibi, può far dimagrire, combattere una malattia, rendere più attivi, combattere i virus e le allergie e risolvere alcuni problemi di estetica.

Anche gli esperti del settore, d'altro canto, cercano spesso un metodo rapido per accedere alla grande mole di informazioni oggetto delle loro ricerche, che possono ad esempio essere mirate alla costruzione di diete particolari o alla scoperta di legami tra malattie e nutrizione.

Per rendere quindi maggiormente fruibili i dati, si è deciso di realizzare un software di consultazione via web.

Si è partiti da una descrizione del pattern MVC (Model – View – Controller) e di tutti gli aspetti relativi al suo meccanismo di funzionamento, nonché ad una spiegazione delle Servlet che sono alla base del framework utilizzato.

Successivamente, viene dato ampio spazio al framework JavaServer Faces descrivendone tutte le classi che ne costituiscono l'architettura e tutte le funzionalità messe a disposizione per realizzare un'applicazione Web.

Raccolte le informazioni necessarie, è stata dapprima analizzata la Data WareHouse a disposizione e poi è stata realizzata una Web application mediante tale framework.

Tale applicazione mette a disposizione degli utenti la possibilità di usufruire dei contenuti della Data Warehouse, registrandosi in maniera appropriata al sito.

Per quanto riguarda l'amministratore, egli ha a disposizione tutte le funzionalità di gestione dell'archivio degli alimenti e degli utenti registrati.
E' stata eseguita una preventiva fase di analisi e progettazione che ha previsto la realizzazione del documento di specifica dei requisiti (SRS) e di tutti i diagrammi UML (Use Case, Class, Activity, Sequence, Statechart) oltre al modello concettuale e fisico della base di dati a disposizione. A queste fasi ha fatto seguito una prima conversione della Data WareHouse da PostgreSQL a MySQL, e successivamente una fase di implementazione che ha previsto la realizzazione della Web application con il framework JSF.
Per quanto concerne gli strumenti adottati si è fatto ampio uso del software Sybase Power Designer per lo sviluppo dei diagrammi UML, dell'ambiente IDE Eclipse ed i plug-in Exadel ed Omondo per l'implementazione ed infine del Web Container Apache Tomcat come ambiente di esecuzione dell'implementazione.

2. Stato dell'Arte

Ciò che ha dato il via al progetto è stata la constatazione che, da quanto emerge dalle ricerche svolte, sono ben poche le risorse direttamente accessibili via web riguardanti dati di composizione di alimenti e che siano anche in italiano. Dal sito dell' "Istituto Nazionale di Ricerca per gli Alimenti e la Nutrizione" (INRAN) è comunque scaricabile un applicativo per macchine Windows che permette di consultare (in italiano) le tabelle di composizione fornite dall'ente. Tale software permette anche di effettuare ricerche per alimenti contenenti particolari valori dei componenti. Si comprende comunque come questo programma non sia direttamente fruibile da parte di tutta l'utenza web.
In ambito internazionale, i migliori siti trovati sono quello del "Nutrient Data Laboratory" (NDL) statunitense e quello del "Danish Food Composition Databank". Entrambi sono in inglese e ovviamente non contengono informazioni riguardanti cibi consumati esclusivamente in Italia.
Il sito statunitense permette solo una semplice ricerca per stringa tra i nomi degli alimenti, riservando comunque a dei report cartacei (disponibili sul sito in formato pdf) la presentazione di altre informazioni.
In ogni caso i report prodotti dinamicamente dalle applicazioni web citate non contengono molte informazioni che possono essere utili agli esperti del settore, quali ad esempio: metodi utilizzati, dati statistici approfonditi, fonti, etc.

3. Definizioni Fondamentali

Prima di descrivere a fondo le motivazioni e la natura del progetto, si ritiene necessario dare alcune definizioni basilari riguardanti l'ambito applicativo di riferimento. Per maggiori informazioni si faccia riferimento alla letteratura specifica e alla bibliografia presentata.
Si tenga comunque presente che le definizioni presentate non vogliono essere universali (si veda ad esempio la definizione di "valore"), ma indicano semplicemente il senso che avranno alcuni termini nel seguito (a meno che non venga esplicitamente indicato diversamente).

Fonte (o sorgente) di dati: è inteso come un insieme di dati di composizione provenienti da una singola persona, un singolo gruppo di autori o una singola organizzazione.
Una fonte è considerata **primaria** se i suoi dati vengono direttamente inseriti nel magazzino a nome dell'ente che li ha rilasciati. Si parla invece di **fonti secondarie** per descrivere quelle fonti da cui provengono i dati delle primarie.

Alimento (cibo): nella Data Warehouse vengono considerati distinti due alimenti con la stessa descrizione, ma provenienti da fonti primarie diverse. Questo perché si vogliono in qualche modo confrontare i dati e inoltre occorrenze diverse dello stesso cibo possono avere composizioni differenti.
Si fa anche notare che a caratterizzare principalmente l'alimento è la sua composizione e non la sua descrizione in linguaggio naturale: non si può presupporre che due compilatori di due fonti diverse volessero intendere lo stesso alimento semplicemente perché ne hanno dato lo stesso nome.

Componente: un componente è una qualsiasi proprietà del cibo che sia soggetta a misurazione scientifica. In particolare un componente comprende sia i **nutrienti** che altre caratteristiche, come pH, parte edibile, etc..
Nel seguito comunque la parola nutriente verrà utilizzata anche come sinonimo di componente.
Come per gli alimenti, si assumono distinti due componenti provenienti da fonti primarie diverse, anche qualora questi abbiano descrizioni simili. Questo anche perché i metodi per ricavare la proprietà possono differire da fonte a fonte.

Valore (o composizione): quantità di un componente in un alimento e le sue proprietà statistiche.

Metodo: sistema tramite il quale viene determinato il valore di un componente in un alimento: un metodo può essere chimico, fisico, numerico, etc.

Unità di misura: unità con cui è espresso il valore di un. Per un valore che non presenti unità di misura si parla di "numero puro" o di "rapporto".
Modo di espressione della misura: quantità di alimento cui si riferisce il valore misurato, viene sempre utilizzato assieme all'unità di misura.

4. Linee Guida del Progetto

La Data Warehouse è stata costruita in modo che :

- Integri le informazioni provenienti dalle varie fonti, inserendo tutti i dati presenti nelle stesse senza cambiarne il significato.
- Mantenga i collegamenti dei dati con le fonti in maniera trasparente, deve sempre essere reperibile la provenienza di un'informazione.
- Sia gestibile semplicemente, anche attraverso dei moduli software creati ad hoc.
- Sia navigabile in maniera "elastica", con buoni collegamenti tra i frammenti di informazione.
- Contenga un supporto multilingua, i dati devono essere consultabili sia nella lingua originaria che in italiano.
- Sia accessibile via web tramite un'interfaccia amichevole.

L'applicazione web, invece, dovrà:

- essere accessibile da due tipi di utenti, un amministratore e vari utenti generici;
- essere accessibile via web tramite un'interfaccia amichevole, sia nella parte amministrativa privata che nella parte pubblica;
- essere in grado di identificare l'amministratore e autorizzarlo ad accedere alla parte amministrativa privata, vietando l'accesso a chiunque altro;
- garantire la sicurezza delle informazioni trattate nella parte amministrativa privata;
- contenere un supporto multilingua, ovvero le informazioni devono essere consultabili in lingua inglese ed in lingua italiana, predisponendo l'eventuale aggiunta di ulteriori lingue in futuro;
- essere facilmente gestibile e manutenibile.

Come si può vedere, queste semplici linee guida sono ben lungi dall'essere un rigoroso documento di specifiche: questo perché scopo della tesi è descrivere il progetto e non documentarlo approfonditamente, come invece si cerca di fare con il materiale allegato alla tesi stessa.

5. Passi Fondamentali

Nello svolgere il progetto sono stati seguiti gli step riportati di seguito (ovviamente la presentazione in serie dei passi non è indicativa di un processo di sviluppo sequenziale) :

- Valutazione delle tecnologie utilizzate per la progettazione e realizzazione dell'applicazione web: questo step comprende l'analisi dei vari metodi di sviluppo di un prodotto software per il Web e lo studio delle varie tecnologie per la progettazione e realizzazione di applicazioni web, con l'obiettivo di selezionare la tecnologia che sia più idonea alle nostre necessità;
- Selezione delle fonti da inserire nel magazzino finale.
- Reverse engineering delle fonti, gli schemi iniziali delle fonti sono stati studiati e ampiamente rimaneggiati per meglio individuare le entità coinvolte e per facilitare la successiva fase di integrazione.
- Integrazione di un database che segua tutte le linee guida mostrate in precedenza. Da tale database è stata poi ricavata la datawarehouse finale.
- Sviluppo dei moduli di feeding: sono stati progettati e implementati dei moduli automatici per l'inserimento, l'eliminazione e l'aggiornamento dei dati presenti nel database.
- Progettazione della applicazione web, in base alla metodologia progettuale selezionata.
- Realizzazione della applicazione web, in base alla tecnologia scelta.
- Testing dell'applicazione web, al fine di individuare il maggior numero possibile di malfunzionamenti e/o errori.

6. Struttura del Sistema

La figura seguente mostra la struttura di base del sistema: le frecce indicano il flusso dei dati tra i vari componenti dello stesso.
Lo schema mostra chiaramente come il modulo software automatico di feeding prelevi i dati dalle singole fonti e inserisca questi ultimi in quello che è stato chiamato "Database di Integrazione". Tale base di dati contiene già tutti i dati finali, ma in una suddivisione che facilita maggiormente l'inserimento e la consultazione.
Da questo database, soprattutto tramite l'utilizzo di viste materializzate e di indici che facilitino le ricerche, è stata creata una "Datawarehouse di Consultazione". Su questa architettura si poggiano poi i vari moduli client, che possono sia consultare le informazioni presenti nel magazzino, sia inserire dati nella database. Questo inserimento non automatico può per esempio rendersi necessario nel caso di lavori cartacei. La suddivisione tra "client per l'inserimento" e "client per la consultazione" è stata eseguita ovviamente solo a scopi illustrativi: i due moduli non devono per forza essere separati. Inoltre i client non si appoggiano direttamente sui dati, ma sono ovviamente presenti dei server, che per semplicità non sono schematizzati in figura.

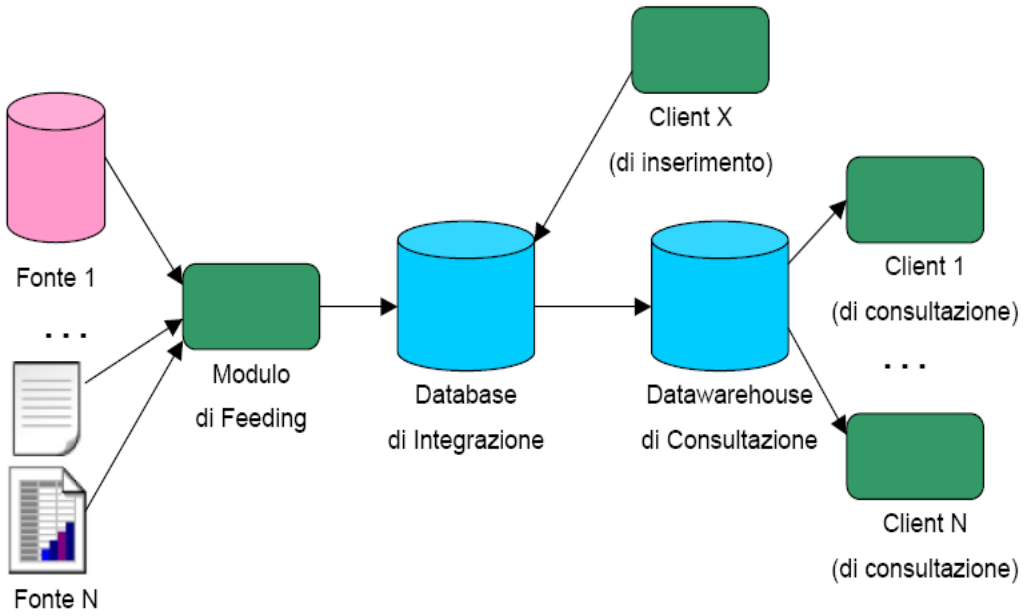


Figura 1 – Struttura del Sistema

7. Tecnologie Utilizzate

Durante tutto il processo di reverse engineering delle fonti, di integrazione dei dati e di progettazione del software è stata utilizzato il CASE PowerDesigner, uno strumento flessibile e con supporto per la stragrande maggioranza dei linguaggi di programmazione e dei DBMS presenti sul mercato.
Per la programmazione dei moduli di feeding la scelta è ricaduta sulla tecnologia di accesso ai dati ODBC e sul linguaggio di programmazione C++ (di cui sono state utilizzate solo le librerie ANSI, oltre ovviamente alle API ODBC).
Poiché il primo processo di integrazione dei dati per la creazione della data warehouse è stato eseguito utilizzando il DBMS PostgreSQL**,** si è resa necessaria in prima istanza effettuare una conversione del database in MySQL.
Vediamo ora un breve confronto dei due database, in modo da porre in rilievo le differenze e quindi motivare la nostra decisione.
Entrambi sono potenti Relational DBMS open source, e multi-piattaforma: ciò significa che possono girare sia in ambiente Linux che Windows, basati sul paradigma client/server; ma si differenziano sia per il livello di compatibilità con lo standard ANSI SQL che per le strategie scelte per implementare il "cuore" del DBMS (ovvero la parte server).
Il PostgreSQL, come il MySQL, utilizza il modello client/server; a differenza del MySQL, però, l'architettura del PostgreSQL è più semplice e meno efficiente perché incentrata sull'uso dei processi. Ogni volta che un client deve connettersi ad un database manda la sua richiesta ad un demone (chiamato "postmaster") che svolge il ruolo di supervisore: questo demone, come prima cosa, crea un nuovo processo del server SQL vero e proprio (chiamato anche "backend") e poi mette in comunicazione diretta il client con il backend. È questo modo di procedere rende il PostgreSQL meno efficiente del MySQL: infatti, dove il MySQL esegue in parallelo soltanto alcune porzioni di codice (i thread), il postmaster manda in esecuzione in un processo separato una copia completa del server SQL.
Per quanto riguarda il lato client la situazione non è migliore, perché la libreria "libpq" (che contiene le API utilizzabili dai client) non può essere utilizzata in maniera sicura da un client multi-threaded; un client single-threaded, comunque, può aprire più connessioni verso altrettanti processi backend. Le funzioni C esposte dalla libreria "libpq" sono più numerose di quelle del MySQL.

Nelle API del PostgreSQL non esiste, a differenza di quanto accade in quelle del MySQL, una funzione per eseguire una query che restituisca una riga alla volta (anziché l'intero "result set").
Il MySQL, invece, è un RDBMS multi utente e multi-threaded, costituisce oggi la soluzione ottimale per chi è in cerca di un database veloce, flessibile e soprattutto gratuito; è composto da un server chiamato "mysqld", dalla libreria "mysqlclient" (che contiene le API che i client possono utilizzare) e da alcune utilità di contorno. L'architettura del MySQL è basata sull'uso estremamente efficiente dei thread a livello di kernel (cioè quei thread gestiti direttamente dal kernel del sistema operativo): ogni qualvolta un client stabilisce una connessione con il demone "mysqld", quest'ultimo crea un thread per quella connessione; il thread, poi, resta attivo finché il client non chiude esplicitamente la connessione oppure finché la connessione non viene chiusa dal "mysqld" per time-out. I thread hanno il vantaggio di usare un limitato numero di risorse e la loro creazione/distruzione è estremamente veloce. Il MySQL, però, è multi-threaded non soltanto nella parte server, ma anche per quanto riguarda la parte client, perché la libreria "mysqlclient" è thread-safe rispetto ad una singola connessione. Questo significa che fintantoché il nostro client multi-threaded utilizza connessioni diverse per ogni thread da lui creato, ottiene dei risultati corretti; se, invece, vogliamo che il nostro client condivida una stessa connessione con più thread (di quelli da lui creati), l'uso della libreria "mysqlclient" non può più essere considerato sicuro, a meno di non osservare le seguenti indicazioni:

- due o più thread non possono mandare una query al demone "mysqld" nello stesso istante temporale;
- se i risultati di una query vengono letti una riga alla volta da un certo thread del client, nessun altro thread può utilizzare la connessione finché tutti i risultati non sono stati recuperati.

L'API C messa a disposizione dal MySQL è composta soltanto da 28 funzioni: un numero così esiguo è dovuto al fatto che tutti i comandi per manipolare le tabelle e/o i database vengono impartiti sotto forma di istruzioni SQL.
Per quanto riguarda il web server, la scelta invece non poteva che ricadere su Apache Tomcat, semplicemente il più sicuro e testato in questo genere di applicativi.
Il Tomcat è un Servlet Container compatibile con la versione 2.3 delle servlet e la 1.2 delle jsp.

Il pacchetto di istallazione, infatti, oltre ad una serie di file jar utilizzati da Tomcat, contiene il file servlet.jar necessario per compilare le applicazioni che contengono le servlet e dunque le applicazioni che usano JSF. Inoltre un altro componente fondamentale, già presente in Tomcat, è il parser XML compatibile con le Java API Parsing specification 1.1 e superiori, necessario, per poter interpretare i file web.xml e faces-config.xml dell'applicazione che andremo a scrivere.

Come già detto per la creazione dell'applicazione web è stato utilizzato il framework JSF**,** per lavorare con tale framework, bisogna preparare la nostra macchina con una serie di applicativi, alcuni fondamentali, altri molto utili per rendere il lavoro quanto più semplice possibile.

Tra i tanti ambienti di sviluppo per Java, è stato scelto Eclipse**,** si tratta di un IDE Java (Integrated Development Environment) con tanti aspetti innovativi, realizzato da IBM, ma con una licenza open source. Eclipse è un ambiente di sviluppo molto diffuso, perché oltre ad essere gratuito è leggero e raccoglie le migliori caratteristiche di tutti gli ambienti di sviluppo professionali, diventando un supporto molto valido che velocizza molto il lavoro del programmatore. La caratteristica principale di eclipse è quella di avere una architettura a plugin (motivo della sua leggerezza), il che lo rende un ambiente molto versatile e adatto alle varie esigenze. Difatti, eclipse non nasce per lo sviluppo di web application, ma consente la possibilità di installare plugin che permettono di personalizzarlo ed estenderlo notevolmente. Ad esempio, si possono installare plugin per la progettazione UML, per la connessione a Tomcat, per l'uso di JSF.

Tra questi, il più performante è sicuramente l'Exadel Studio, il quale è un potente tool free per sviluppo di web-application che estende le funzionalità di Eclipse, permettendo agli sviluppatori di utilizzare a pieno le tecniche RAD (Rapid Application Development).

Questa soluzione redditizia e semplificata include tra le caratteristiche fondamentali importanti:

- supporto per lo sviluppo di progetti con tecnologia JSF;
- scrittura del codice assistita per pagine XML e JSP;supporto per il design e l'anteprima di pagine JSP;
- flessibilità e personalizzabilità dei progetti.

8. L'architettura delle Applicazioni Web-based

Nello sviluppo delle applicazioni software, è possibile descrivere l'architettura del sistema utilizzando una architettura "a livelli" (Layered Application Architecture), la quale prevede che un sistema software sia decomposto in tre livelli nettamente distinti, che comunque abbiano la possibilità di comunicare fra loro secondo un'opportuna gerarchia. Ciascuno dei tre livelli ha un proprio ruolo ed assolve ad uno specifico compito all'interno del sistema complessivo, senza interferire con gli altri livelli ma scambiando con essi le informazioni necessarie all'esecuzione di elaborazioni anche molto complesse.
I tre livelli in questione sono i seguenti :

- *Presentation layer* : è il livello di presentazione, il cui compito è quello di interagire direttamente con l'utente del sistema, acquisire i dati di input immessi da quest'ultimo e visualizzare i risultati dell'elaborazione effettuata dal sistema stesso. Esso, in pratica, definisce la GUI (Graphic User Interface) ossia l'interfaccia grafica dell'applicazione;
- *Application processing layer* : è il livello in corrispondenza del quale si trova la "business-logic" dell'applicazione e quindi tutti i moduli software che implementano le funzionalità che il sistema mette a disposizione. In sostanza, è il centro dell'elaborazione dei dati in cui avvengono tutte le computazioni;
- *Data management layer* : è il livello che si occupa della gestione della persistenza e dell'accesso ai dati, per cui è tipicamente caratterizzato da un DBMS (DataBase Management System);

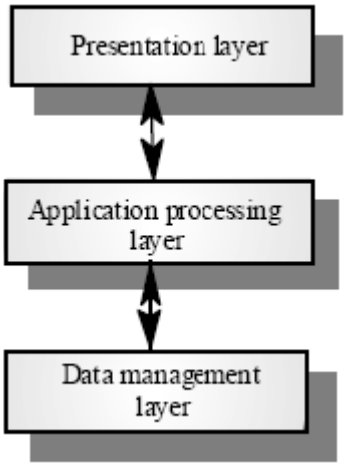


Figura 2 – Architettura Software "a livelli"

Sviluppando un'applicazione secondo questa architettura, ogni livello è indipendente dagli altri, per cui la modifica di uno di essi non ha effetto sui restanti. Tuttavia è prevista la comunicazione fra loro e lo scambio di informazioni.
Un tipico scenario di funzionamento del sistema può essere il seguente : un utente utilizza l'applicazione, interagendo direttamente con la GUI e fornisce quindi al Presentation layer, i dati su cui andrà eseguita l'elaborazione. Il Presentation layer, acquisiti i dati di input, li trasferisce all'Application processing layer che esegue su di essi una determinata computazione.
Durante l'elaborazione, la business-logic può prevedere la memorizzazione persistente dei dati oppure la necessità di acquisire ulteriori dati già memorizzati. In questo caso, c'è l'interazione con il Data managemente layer, il quale memorizza i dati che gli vengono passati dal livello superiore, oppure recupera da un Data Source (es. database) i dati richiesti e li trasmette alla business-logic. Al termine dell'elaborazione i risultati vengono passati al Presentation layer che li visualizza in una certa forma all'utente finale.
Facendo riferimento al paradigma Client-Server, notevolmente utilizzato nelle Web application ma di gran richiamo anche per applicazioni desktop, i tre livelli del sistema devono essere correttamente ripartiti anche da un punto di vista hardware.
Le principali architetture per la ripartizione sono :

- *Two-Tier* nelle due soluzioni Thin e Fat Client;
- *Three-Tier*;

L'architettura Two-Tier prevede un unico Client ed un unico Server ed i tre livelli dell'applicazione software sono distribuiti fra di essi secondo due possibili modalità :

- *Thin Client* : sul Client risiede il Presentation layer mentre sul Server gli altri due livelli (Application processing layer e Data management layer). Un vantaggio può risiedere nel fatto che una modifica alla business-logic va eseguita una sola volta sul Server, mentre lo svantaggio principale può essere caratterizzato dall'enorme carico di lavoro che deve supportare il Server stesso dato il numero elevato di Client che possono accedere ad esso;

- *Fat Client* : sul Client risiedono i primi due livelli (Presentation layer e Application processing layer) mentre sul Server soltanto il Data management layer. Il vantaggio è quello di ridurre il carico di lavoro sul Server che si occupa solo dell'accesso ai dati, delegando l'elaborazione degli stessi al Client. Lo svantaggio principale è la complessità maggiore dei Client e quindi la necessità di aggiornare ciascuno di essi nel caso in cui vengano apportate modifiche alla business-logic;

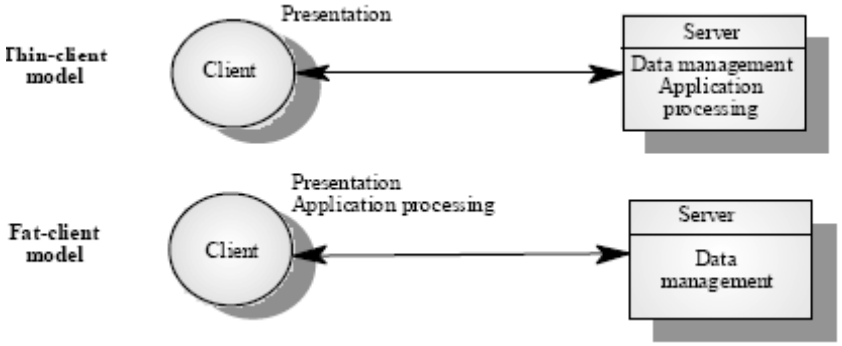


Figura 3 – Architetture Two-Tier

L'architettura Three-Tier, maggiormente utilizzata, prevede la presenza di un unico Client ed una coppia di Server. Sul Client risiede il Presentation layer e su ciascuno dei due Server sono distribuiti i due restanti livelli (Application processing layer e Data management layer). Nell'ambito di una Web application, il Client è caratterizzato da un nodo della rete sul quale è in esecuzione il browser, mentre i due Server, da un punto di vista software, sono tipicamente inglobati in un unico nodo della rete che funge da Server fisico. In particolare, sulla stessa macchina sono in esecuzione il Web Server associato all'Application proccessing layer ed il Database Server associato al Data management layer.

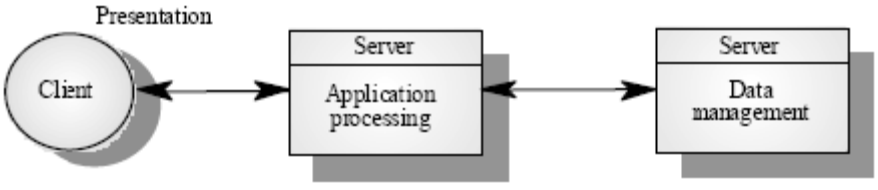


Figura 4 – Architettura Three-Tier

Un ultima considerazione, va fatta sul modello di approccio al problem-solving: la suddivisione netta che la soluzione introduce tra logiche di business, logiche di client, e logiche di presentazione consente un approccio per strati (tier) al problema, garantendo ordine nella progettazione e nello sviluppo di una soluzione, fornendo una separazione delle responsabilità, il riuso dei componenti, una migliore scalabilità, e molti altri aspetti vantaggiosi.
All'interno di questo scenario appena descritto, l'infrastruttura del JavaServer Faces si colloca all'interno del web tier. Le applicazioni JSF sono ospitate da un web conteiner e possono far uso di servizi forniti dal container stesso, quali la gestione delle richieste tramite i protocolli HTTP e HTTPS. Questo dà agli sviluppatori la libertà di concentrarsi sulla realizzazione della logica di business che risolve i problemi reali della loro applicazione, costituendo il valore aggiunto delle applicazioni web JSF.
All'interno dello strato web, il web-server gioca un ruolo fondamentale. In ascolto su un server, riceve richieste da parte del browser (client), le processa, quindi restituisce al client una entità o un eventuale codice di errore come prodotto della richiesta.

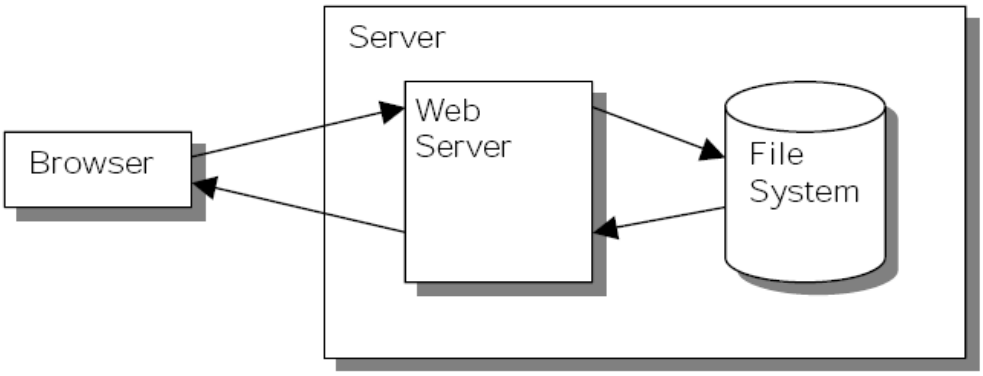


Figura 5 – Architettura con entità statiche

Le entità prodotte da un web-server possono essere entità statiche o entità dinamiche.
Una entità statica è ad esempio un file html residente sul file-system del server. Rispetto alle entità statiche, unico compito del web-server è quello di recuperare la risorsa dal file-system ed inviarla al browser che si occuperà della visualizzazione dei contenuti.

Quello che più ci interessa sono invece le entità dinamiche, ossia entità prodotte dalla esecuzione di applicazioni eseguite dal web-server su richiesta del client.
Il modello proposto nella immagine precedente ora si complica in quanto viene introdotto un nuovo grado di complessità nella architettura del sistema che, oltre a fornire accesso a risorse statiche dovrà fornire un modo per accedere a contenuti e dati memorizzati su una Base Dati, dati con i quali un utente internet potrà interagire da remoto.

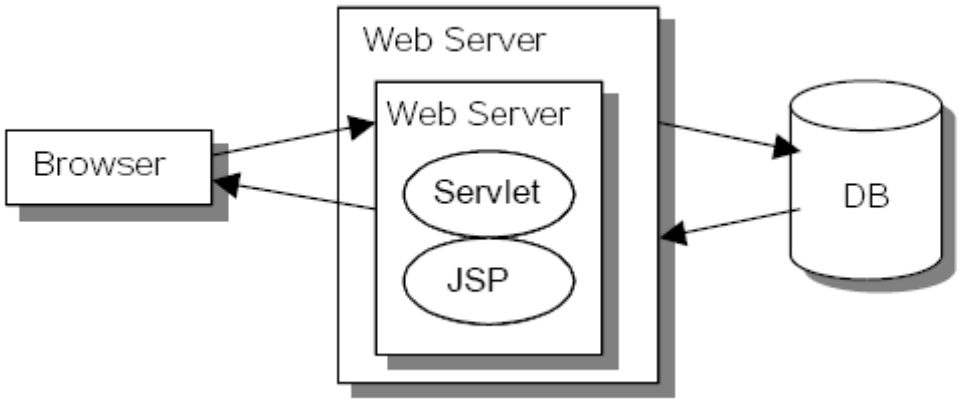


Figura 6 – Architettura con entità dinamiche

Le informazioni inviate dal web server viaggiano in rete trasportate dal protocollo HTTP, dunque diventa importante dare uno sguardo al protocollo request/response HTTP, di cui JSF fa molto uso.

9. Request/Response

HTTP è l'acronimo di HyperText Transfer Protocol (protocollo di trasferimento di un ipertesto). Usato come principale sistema per la trasmissione di informazioni sul web. Le specifiche del protocollo sono attualmente in carica al W3C (World Wide Web Consortium).
L'HTTP funziona su un meccanismo richiesta/risposta: il client apre una connessione verso un server ed esegue una richiesta, il server esegue la richiesta del client e restituisce la risposta chiedendo la connessione. Nell'uso comune il client corrisponde al browser ed il server al sito web. Vi sono quindi due tipi di messaggi HTTP: messaggi richiesta e messaggi risposta.

Il messaggio richiesta è composto di tre parti: linea di richiesta (request line), sezione header (informazioni aggiuntive), Body (corpo del messaggio).
La linea di richiesta è composta dal metodo, URI e versione del protocollo.
L'URI sta per Uniform Resource Identifier ed indica l'oggetto della richiesta (ad esempio la pagina web che si intende ottenere).
I metodi HTTP più comuni sono GET e POST. Utilizzando il metodo GET, i dati vengono appesi alla request-URI nella forma di coppie *chiave=valore* separati tra di loro dal carattere "&". La stringa seguente è un esempio di campo request-URI per una richiesta di tipo GET.

http://www.java-net.tv/servelt/Hello?nome=Fabio&cognome=Neiviller :

- "*http://www.java-net.tv*" rappresenta l'indirizzo del web-server a cui inviare la richiesta.
- "*/servlet/Hello*" rappresenta la locazione della applicazione web da eseguire.
- Il carattere "?" separa l'indirizzo dai dati e il carattere "&" separa ogni coppia chiave=valore.

Questo metodo utilizzato per default dai browser a meno di specifiche differenti, viene utilizzato nei casi in cui si richieda al server il recupero di informazioni mediante query su un database.
Il metodo POST pur producendo gli stessi effetti del precedente, utilizza una modalità di trasmissione dei dati differente impacchettando i contenuti dei campi di un form all'interno di un message-header. Rispetto al precedente, il metodo POST consente di inviare una quantità maggiore di dati ed è quindi utilizzato quando è necessario inviare al server nuovi dati affinché possano essere memorizzati all'interno della Base Dati.
Il messaggio di risposta è composto dalle seguenti tre parti: linea di stato (status line), sezione header, body (contenuto della risposta).

La linea di stato riporta un codice a tre cifre catalogato nel seguente modo:

- **2xx:** Success
- **3xx:** Redirection
- **4xx:** Client error
- **5xx:** Server error

Nel caso più comune il server risponde con un codice 200 (OK) e fornisce contenuto nella sezione body. Altri casi sicuramente incontrati da tutti sono:

- **302 Found**. La risorsa è raggiungibile con un altro URI indicato nel header Location. Di norma i browser eseguono la richiesta all'URI indicato in modo automatico senza interazione dell'utente.
- **404 Not Found**. La risorsa richiesta non è stata trovata e non se ne conosce l'ubicazione. Di solito avviene quando l'URI indicato in modo incorretto od è stato rimosso il contenuto dal server.
- **500 Internal Server Error**. Il server non è in grado di rispondere alla richiesta per un suo problema interno.

Gli header della risposta più comuni sono:

- **Server**. Indica il tipo la marca , la versione del server.
- **Content-Type**. Indica il tipo di contenuto restituito. La codifica di tali tipi (detti Media type) sono registrati presso IANA (Internet Assigned Number Authority) e sono detti tipi MIME (Multimedia Internet Message Extensions), la loro codifica è descritta nel documento RFC1521.
  I principali sono:
  text/html. Documento HTML)
  text/plaintext. Documento di testo non formattato)
  image/jpeg. Immagine di formato Jpeg

Dal momento che tutto il traffico HTTP è anonimo e *in chiaro*, sono state sviluppate diverse alternative per garantire differenti livelli di sicurezza. Ad oggi quello più usato è il protocollo HTTPS. Il meccanismo HTTPS, inventato da Netscape usa il sottostante canale cifrato a livello di trasporto mediante SSL o TLS per impedire l'intercettazione di qualsiasi parte della transazione. In pratica è un normale http inglobato in un Secure Sockets Layer (SSL), un sistema di comunicazione che garantisce la privacy quando si comunica con altre applicazioni abilitate all'SSL.
Una connessione SSL può essere stabilita tra un client e un server solo quando entrambi i sistemi sono in esecuzione in modalità SSL e hanno la capacità di autenticarsi reciprocamente.

10. Le basi del JavaServer Faces

Il framework JSF è un MVC web application framework, ovvero un framework per lo sviluppo di applicazioni web J2EE basato sul pattern Model-View-Controller. Prima di addentrarci nel mondo di JSF, per capire meglio di cosa si tratta e la sua valenza nel mondo della progettazione delle web application, diamo una breve spiegazione di alcuni concetti introdotti fino ad ora.
Un framework è una architettura generica che costituisce l'infrastruttura per lo sviluppo di applicazioni in una determinata area tecnologica, nel caso di JSF, si tratta del mondo delle web-application. In parole povere, un framework è un insieme di classi ed interfacce di base, a disposizione del programmatore, che costituiscono l'infrastruttura di una applicazione. Non bisogna però pensare, che usare un framework, equivalga quindi, ad usare una libreria di classi, quali ad esempio le classi di base del linguaggio Java. Usare una libreria, vuol dire avere a disposizione dei metodi e delle funzionalità, ma resta a nostro carico, il compito di controllare il flusso applicativo. Adottare un framework, invece, vuol dire attenersi ad una specifica architettura, in cui saranno i componenti del framework ad occuparsi del flusso applicativo ed il programmatore avrà solo il compito di estendere le classi del framework e/o implementarne delle interfacce.
Il nostro codice applicativo non è direttamente invocato dall'intervento dell'utente sul sistema ma il flusso elaborativo passa attraverso il codice del framework: sono le classi del framework che invocano il nostro codice applicativo e non viceversa come nel caso delle librerie di classi.
In genere i vantaggi dell'utilizzo di un framework vanno ben oltre gli svantaggi, anzi, quanto più il progetto è di grossi dimensioni, tanto più l'utilizzo di un framework è inevitabile. Anche se come detto, scegliere un framework, vuol dire adottare implicitamente una specifica architettura, questo non deve essere visto come vincolante, anzi, nel caso di framework validi, rappresenta uno dei maggiori vantaggi.
Utilizzare un framework maturo e già ampiamente testato significa attenersi ad una architettura che funziona e quindi significa iniziare un progetto da una base solida.
Ciò porta inoltre ad un significativo risparmio di tempo e risorse in quanto lo sviluppatore non deve più preoccuparsi di realizzare componenti infrastrutturali ma può concentrarsi esclusivamente sullo sviluppo della logica di business che poi è il valore aggiunto dell'applicazione che si scrive.

Non è raro nello sviluppo di un progetto assistere alla riscrittura di componenti di base che già esistono e che sono stati già ampiamente testati; possiamo dire che uno dei vantaggi nell'utilizzo di un framework è che si viene aiutati a non inventare tutto ex-novo come spesso purtroppo accade.
E' chiaro che tutto ciò è vero quando si fa riferimento ad un framework giunto ad uno stadio di sviluppo maturo, già adottato da molti sviluppatori e quindi già ampiamente provato "sul campo".
Nella scelta di un framework possiamo elencare alcune delle caratteristiche che sicuramente devono essere prese in considerazione:

Maturità del progetto : è sconsigliabile adottare un framework che sia in una fase iniziale di sviluppo e che sia poco adottato nella comunità degli sviluppatori e quindi poco testato sul campo in progetti reali.

Documentazione : va sempre verificato che la documentazione sia ricca e ben fatta. Questo facilita la risoluzione dei problemi che si incontrano nella realizzazione dell'applicazione e la comprensione del suo funzionamento.

Validità del disegno architetturale : proprio perché la scelta di un framework influisce sull'architettura applicativa è bene verificare che sia disegnato correttamente e quindi che siano adottati i design-pattern e le best-practises della tecnologia di riferimento.

Adozione degli standard : un framework deve essere fondato sui componenti standard della tecnologia di riferimento. Nel nostro caso sulle API che costituiscono la J2EE. Quanto più un framework impone soluzioni proprietarie, l'uso di specifici tool di sviluppo o un modello troppo indirizzato ad uno specifico caso applicativo tanto più va evitato.

Estensibilità : deve essere possibile estenderne le funzionalità per adattarlo alle proprie esigenze.

Il framework JSF rispetta i criteri sopra elencati e se utilizzato seguendo alcune principali linee guida consente di realizzare applicazioni ben strutturate, assolutamente conformi agli standard J2EE.
Servlet e JavaServer Pages, su cui si fonda JSF, rappresentano oggi lo stato dell'arte delle tecnologie web.

11. Java Servlet API

Java Servlet sono oggetti Java con proprietà particolari che vengono caricati ed eseguiti dal web server che le utilizzerà come proprie estensioni. Qualcuno le ha definite delle applet lato server. Il web server di fatto mette a disposizione delle Servlet il container che si occuperà della gestione del loro ciclo di vita, della gestione dell'ambiente all'interno delle quali le servlet girano, dei servizi di sicurezza. Il container ha anche la funzione di passare i dati dal client verso le servlet e viceversa ritornare al client i dati prodotti dalla loro esecuzione. Dal momento che una servlet è un oggetto server-side, può accedere a tutte le risorse messe a disposizione dal server per generare pagine dai contenuti dinamici come prodotto della esecuzione delle logiche di business. E' ovvio che sarà cura del programmatore implementare tali oggetti affinché gestiscano le risorse del server in modo appropriato evitando di causare danni al sistema che le ospita.
Le servlet Java sono diventate l'elemento principale per estendere e migliorare le web application usando la piattaforma Java. Forniscono un metodo component-based e indipendente dalla piattaforma, per costruire web application. Le servlet hanno soppiantato lo standard CGI per la loro efficienza e caratteristica di scalabilità.
Queste sono più efficienti del modello di thread dello standard CGI, poichè creano un solo processo e consentono a ciascuna richiesta utente di utilizzare un thread molto più leggero, che viene gestito dalla Java Virtual Machine.
Una servlet viene mappata a uno o più URL (Uniform Resource Lodato) e, quando il server riceve una richiesta ad uno degli url della servlet, viene invocato il metodo di servizio nella servlet che risponde. Dal momento che ciascuna richiesta è associata con un thread separato, thread o utenti multipli posso invocare il metodo *service()* contemporaneamente. Questa natura multithread delle servlet è una delle ragioni principali per cui sono più scalabili rispetto alle applicazioni standard. Inoltre, dal momento che sono scritte in Java, esse non risultano limitate a una sola piattaforma o ad un unico sistema operativo.
Un altro vantaggio significativo della loro natura Java è che le servlet sono in grado sfruttare l'intera serie delle API Java (application programming interfaces) tra cui JDBC (Java DataBase Connectivity).
Le servlet non vengono eseguite direttamente da un web server, ma hanno necessità di un loro contenitore, detto servlet-container.

Per le proprie servlet, gli sviluppatori sono liberi di scegliere tra i molti container disponibili. Le servlet sono portabili e possono migrare attraverso diversi container, senza il bisogno di ricompilare il codice sorgente.
Il package di base delle Servlet API è javax.servlet e contiene le classi per definire Servlet standard indipendenti dal protocollo. Tecnicamente una Servlet generica è una classe definita a partire dall'interfaccia Servlet contenuta all'interno del package *javax.servlet*. Questa interfaccia contiene i prototipi di tutti i metodi necessari alla esecuzione delle logiche di business, nonché alla gestione del ciclo di vita dell'oggetto dal momento del suo istanziamento, sino al momento della sua terminazione.
I metodi definiti in questa interfaccia devono essere supportati da tutte le servlet o possono essere ereditati attraverso la classe astratta GenericServlet che rappresenta una implementazione base di una servlet generica. Il package include inoltre una serie di interfacce che definiscono i prototipi di oggetti utilizzati per tipizzare le classi che saranno necessarie alla specializzazione della servlet generica in servlet dipendenti da un particolare protocollo.
Http servlet rappresentano una specializzazione di servlet generiche e sono specializzate per comunicare mediante protocollo http. Il package javax.servlet.http mette a disposizione una serie di definizioni di classe che rappresentano strumenti utili alla comunicazione tra client e server con questo protocollo, nonché forniscono uno strumento flessibile per accedere alle strutture definite nel protocollo, al tipo di richieste inviate ai dati trasportati.
Le interfacce ServletRequest e ServletResponse rappresentano rispettivamente richieste e risposte http.
Sebbene le servlet sono risultate essere ottime nello svolgere il loro compito fin dalla loro introduzione, si è subito capito che incorporare i risultati delle richieste all'interno di semplici pagine statiche HTML, rappresentava una forte limitazione. Il passo successivo nella progressione di sviluppo delle tecnologie web basate sulla piattaforma Java sono state le JavaServer Pages.

12. JSP - Java Server Pages

Il primo aspetto importante da sottolineare è che le JavaServer Pages sono la naturale estensione delle Java Servlet, ovvero in un certo senso, sono anche loro delle servlet.
Le pagine jsp sono documenti di testo con estensione *.jsp* contenenti una combinazione di HTML statico e tag tipo XML e di scriptlet. La possibilità di fondere codice html con codice Java senza che nessuno interferisca con l'altro consente di isolare la rappresentazione dei contenuti dinamici dalle logiche di presentazione. Il disegnatore potrà concentrarsi solo sulla impaginazione dei contenuti che saranno inseriti dal programmatore che non dovrà preoccuparsi dell'aspetto puramente grafico. I tag e gli scriptlet incapsulano la logica che genera il contenuto delle pagine. I file .jsp vengono elaborati e trasformati in file .java.
A questo punto un compilatore java compila il sorgente e crea un file .class che può andare in esecuzione in un servlet container.
Dunque è il traduttore che ha il compito di creare il file .java di creare anche la servlet a partire dalla pagina jsp.
Jsp presenta diversi vantaggi rispetto le tecnologie concorrenti:

- JSP è una specifica non un prodotto, dunque gli sviluppatori posso scegliere la loro implementazione ideale.
- Le JSP sono compilate e non interpretate, con un miglioramento delle prestazioni.
- Le pagine JSP supportano sia script che un accesso completo al linguaggio Java e possono essere estese tramite custom tag.
- Sono basate sul modello "write once run anywhere" della tecnologia Java.

13. I Patterns

La piattaforma J2EE ha costituito, sin dal suo primo apparire, una vera rivoluzione nell'ambito dello sviluppo di software. Ciò è testimoniato sia dal gran numero di applicazioni che con essa sono state sviluppate, sia dalla continua e costante evoluzione della piattaforma medesima, dato che nel corso del tempo sempre più numerose sono le tecnologie che sono andate a confluirvi.

In particolare l'architetto Christopher Alexander osservò come i suoi colleghi tendevano a risolvere i medesimi problemi, più o meno allo stesso modo. Da tale osservazione, introdusse il concetto di "design pattern", ovviamente riferito all'architettura. Tale intuizione venne applicata allo sviluppo software.
un pattern è un modello che permette di definire la "soluzione" di un "problema" specifico che si ripresenta, di volta in volta, in un "contesto" diverso.

Presenta inoltre le seguenti caratteristiche :

- il "nome" che individua il pattern e la cui importanza non è secondaria, perché rientra a far parte del vocabolario dello sviluppo software;
- la descrizione del "problema" in maniera dettagliata, con la sua struttura e le condizioni al contorno;
- la "soluzione" che descrive gli artefatti software per risolvere il problema come gli elementi che rientrano nello sviluppo, quali le classi e le relazioni fra esse, le associazioni ed i ruoli, le modalità di collaborazione tra le classi coinvolte ed infine la distribuzione delle responsabilità nella soluzione del particolare problema di design considerato;
- le "conseguenze" che descrivono i risultati che si possono ottenere dall'applicazione del pattern per spingere uno sviluppatore a farne uso;

E' ovvio che esistono numerose tipologie di design patterns, ma nell'ambito dello sviluppo delle Web application, in riferimento all'obiettivo proposto, assume un ruolo rilevante il pattern MVC (Model View Controller).

14. Il pattern MVC (Model-View-Controller)

Uno dei principali requisiti di qualsiasi applicazione web è quello di definire un modello applicativo che consenta di disaccoppiare i diversi componenti dell'applicazione in base al loro ruolo nell'architettura per ottenere vantaggi in termini di riusabilità e manutenibilità. Esempio tipico di questo problema è l'utilizzo nello sviluppo di una applicazione web J2EE di quei modelli applicativi che nella letteratura sono indicati spesso come "JSP Model 1" e "JSP Model 2", introdotti con le prime specifiche delle JavaServer Pages.

14.1 Model 1

Da sole, JavaServer Pages consentono di realizzare applicazioni web dinamiche accedendo a componenti Java contenenti logiche di business o alla base dati del sistema. In questo modello il browser accede direttamente ad una pagina JSP che riceve i dati di input, li processa utilizzando eventualmente oggetti Java, si connette alla base dati effettuando le operazioni necessarie e ritorna al client la pagina html prodotta come risultato della processazione dei dati.

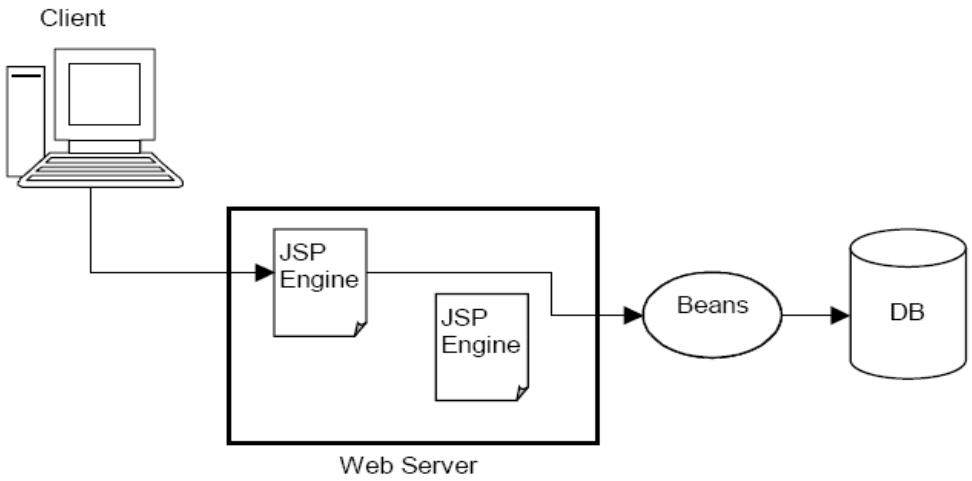


Figura 7 – Model 1

In base a questo modello l'applicazione è costruita secondo una logica "JSP centric" in base alla quale presentation , control e business logic dell'applicazione sono tutti a carico delle pagine JSP. Il web browser accede direttamente alle pagine JSP dell'applicazione che al loro interno contengono logica applicativa e logica di controllo del flusso. Quando l'applicazione cresce in complessità non è pensabile svilupparla seguendo un simile approccio.

14.2 Model 2

Il secondo modello, utilizza JavaServer Pages come strumento per sviluppare template demandando completamente ad una particolare servlet la processazione dei dati di input.

Nelle architetture Model 2 la richiesta del client viene intercettata prima da una servlet, detta controller servlet. Questa gestisce l'elaborazione iniziale della richiesta e determina quale pagina JSP deve essere visualizzata.

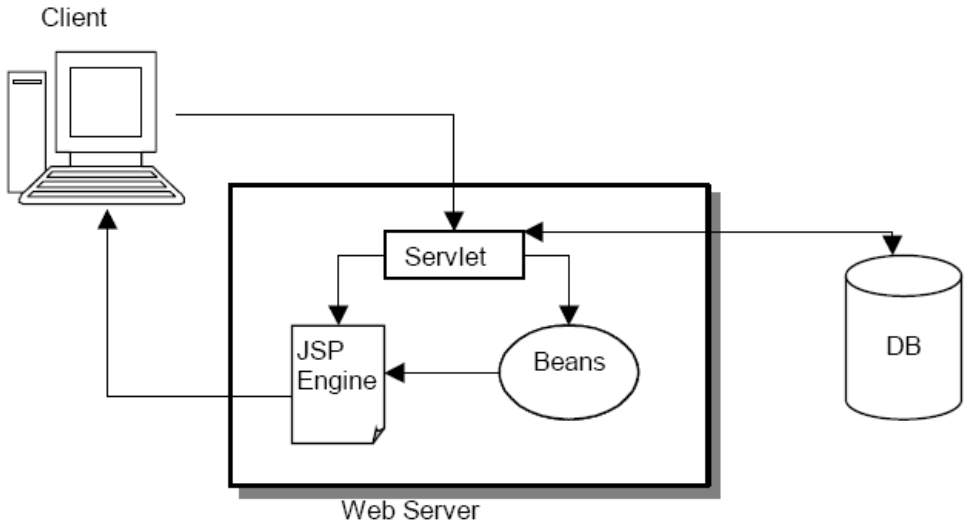


Figura 8 – Model 2

Dunque in questa architettura un client non invia mai direttamente una richiesta a una pagina jsp. Ciò permette alla servlet di effettuare una elaborazione di front-end comprendente autorizzazione ed autenticazione, logging centralizzato e internazionalizzazione. Una volta terminata l'elaborazione della richiesta, la servlet dirige la richiesta alla giusta pagina JSP. Come si vede la differenza principale tra le due architetture è la presenza di questo punto di accesso ai servizi singolo, rappresentato dalla servlet, il che incoraggia il riutilizzo e la separazione tra i vari strati dell'applicazione.

Proprio per questo il pattern Model 2 è più conosciuto come pattern MVC (Model-View-Controller) Secondo questo pattern, i componenti di un'applicazione vengono logicamente classificati a seconda che siano gestori di logiche di business e di accesso ai dati (model), gestori di visualizzazione dati e dell'input dell'utente (viewer o view) e gestori del flusso di controllo complessivo dell'applicazione (controller).

L'accoppiamento tra i vari livelli è minimo, con evidenti vantaggi in termini di riusabilità, manutenibilità, estensibilità e modularità.

In pratica, si hanno innegabili vantaggi come:

- indipendenza tra i business data (model) la logica di presentazione (view) e quella di controllo (controller);
- separazione dei ruoli e delle relative interfacce;
- viste diverse per il medesimo model;
- semplice supporto per nuove tipologie di client: bisogna scrivere la vista ed il controller appropriati riutilizzando il model esistente.

Di seguito è presentato un diagramma di interazione che evidenza le responsabilità dei tre componenti:

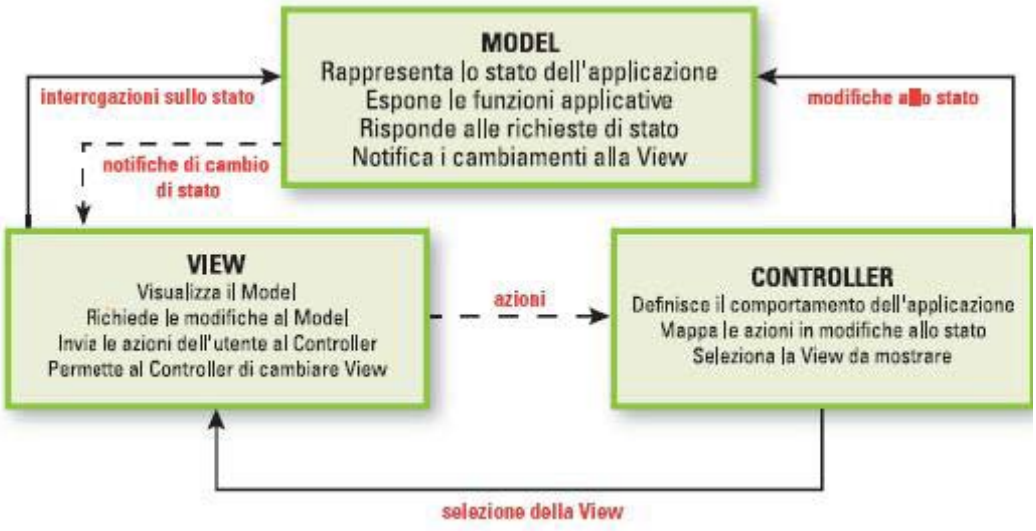


Figura 9 – MVC

Precisiamo che tale pattern non è direttamente legato alle web application, anzi, venne introdotto nel mondo del software per la costruzione di interfacce grafiche con Smalltalk-80.

14.3 MODEL

Analizzando la figura precedente, si evince che il core dell'applicazione viene implementato dal Model, che incapsulando lo stato dell'applicazione definisce i dati e le operazioni che possono essere eseguite su questi. Quindi definisce le regole di business per l'interazione con i dati, esponendo alla View ed al Controller rispettivamente le funzionalità per l'accesso e l'aggiornamento.
Per lo sviluppo del Model quindi è vivamente consigliato utilizzare le tipiche tecniche di progettazione object oriented al fine di ottenere un componente software che astragga al meglio concetti importati dal mondo reale. Il Model può inoltre avere la responsabilità di notificare ai componenti della View eventuali aggiornamenti verificatisi in seguito a richieste del Controller, al fine di permettere alle View di presentare agli occhi degli utenti dati sempre aggiornati.

14.4 VIEW

La logica di presentazione dei dati viene gestita solo e solamente dalla View. Ciò implica che questa deve fondamentalmente gestire la costruzione dell' interfaccia grafica (GUI) che rappresenta il mezzo mediante il quale gli utenti interagiranno con il sistema.
Ogni GUI può essere costituita da schermate diverse che presentano più modi di interagire con i dati dell'applicazione. Per far sì che i dati presentati siano sempre aggiornati è possibile adottare due strategie note come "push model" e "pull model". Il push model adotta il pattern Observer, registrando le View come osservatori del Model. Le View possono quindi richiedere gli aggiornamenti al Model in tempo reale grazie alla notifica di quest'ultimo. Benché questa rappresenti la strategia ideale, non è sempre applicabile. Per esempio nell'architettura J2EE se le View che vengono implementate con JSP, restituiscono GUI costituite solo da contenuti statici (HTML) e quindi non in grado di eseguire operazioni sul Model. In tali casi è possibile utilizzare il "pull Model" dove la View richiede gli aggiornamenti quando "lo ritiene opportuno". Inoltre la View delega al Controller l'esecuzione dei processi richiesti dall'utente dopo averne catturato gli input e la scelta delle eventuali schermate da presentare.

14.5 CONTROLLER

Questo componente ha la responsabilità di trasformare le interazioni dell'utente della View in azioni eseguite dal Model. Ma il Controller non rappresenta un semplice "ponte" tra View e Model. Realizzando la mappatura tra input dell'utente e processi eseguiti dal Model e selezionando la schermate della View richieste, il Controller implementa la logica di controllo dell'applicazione. Determina il modo in cui l'applicazione risponde agli input dell'utente. Esamina le richieste dei client, estrae i parametri della richiesta e li convalida, si interfaccia con lo strato di business logic dell'applicazione. Sceglie la successiva vista da fornire all'utente al termine dell'elaborazione.
Il controller è il “punto di accesso unico” di cui si parlava prima.

Capitolo II – Il framework JavaServer Faces

1. Introduzione

JavaServer Faces (JSF) è un framework per la realizzazione di Web application secondo il pattern MVC (Model – View – Controller).
L'elemento principale, che caratterizza la tecnologia JSF, è l'ampio insieme delle classi e delle relative API (Application Program Interface) che permettono di :

- definire i componenti dell'interfaccia utente (UI) e gestire il mantenimento del relativo stato;
- gestire gli eventi che si verificano sui suddetti componenti, eseguirne la conversione dei valori e la validazione degli stessi;
- specificare la navigazione fra le pagine all'interno della Web application;
- supportare l'internazionalizzazione e l'accessibilità;
- realizzare dei componenti personalizzati (Custom Components) che estendono e potenziano le funzionalità delle classi base del framework;

L'architettura complessiva si basa sull'utilizzo delle Servlet :

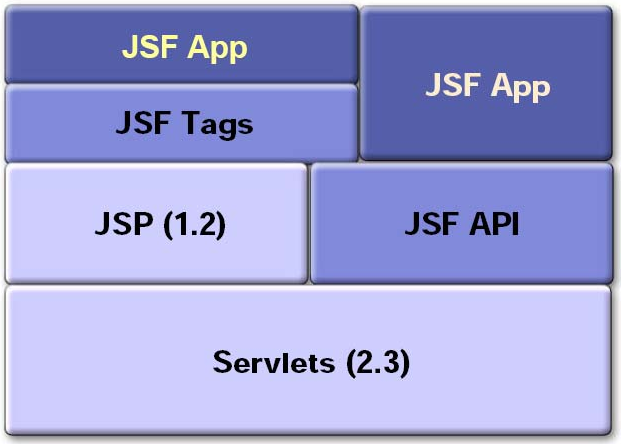


Figura 1 - Architettura JSF

I vantaggi ottenuti mediante l'utilizzo di JSF sono i seguenti :

- legare i componenti della UI con dati lato server;
- salvare e recuperare lo stato di ciascun componente della UI, attraverso il particolare ciclo di vita di una richiesta JSF (Faces request);
- costruire un'interfaccia utente con componenti riutilizzabili ed estendibili;
- gestire gli eventi che vengono generati dal lato client (client side), con oggetti e codice lato server (server side);

Lo sviluppo di una Web application con JSF, generalmente prevede i seguenti passi fondamentali :

- Creazione delle pagine JSP utilizzando i componenti della UI ed i tags delle librerie a disposizione;
- Definire la navigazione fra le pagine dell'applicazione stessa, modificando opportunamente il file di configurazione dell'applicazione "*faces-config.xml*";
- Sviluppare i backing beans che implementano le funzionalità offerte;
- Aggiungere le informazioni riguardanti i backing beans all'interno del file di configurazione dell'applicazione;

Uno dei vantaggi principali che offre la tecnologia JavaServer Faces è la separazione fra il comportamento (Application processing layer – Model) e la presentazione (Presentation layer - View) di una Web application, per favorire da un lato il lavoro di gruppo separando i compiti degli sviluppatori software e dei designers e dell'altro la manutenzione dell'applicazione.

2. Framework Models

Il framework JSF è costituito da un insieme di classi che sono raggruppate in models :

- *Execution Model*;
- *User Interface Component Model*;
- *Component Rendering Model*;
- *Conversion Model*;
- *Event and Listener Model*;
- *Validation Model*;
- *Navigation Model*;

Ad essi va aggiunto il *Backing Bean Management*, il quale permette la realizzazione dei backing beans che implementano la business-logic dell'applicazione.

2.1 Execution Model

Questo modello fa riferimento a tutto ciò che riguarda l'esecuzione dell'applicazione, quindi in particolar modo definisce le classi che costituiscono il Controller del pattern MVC.

2.1.1 FacesServlet

Rappresenta la Servlet in esecuzione nel Web Container, che riceve le richieste da parte dei client occupandosi della loro gestione ed assumendo quindi il ruolo di Controller secondo il modello MVC.

Come tutte le Servlet, secondo l'implementazione in Java, essa prevede i metodi di inizializzazione "*init()*", servizio di una richiesta "*service*()" e di distruzione "*destroy*()".

La *FacesServlet* va configurata nel Deployment Descriptor *web.xml* tipico di ogni Web Application realizzata in Java, mediante la seguente definizione :

```
<servlet>
  <servlet-name>FacesServlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Viene specificata la classe e quindi un nome da assegnare alla Servlet, e l'ordine di sequenza di start-up, nel caso in cui la Web Application preveda l'utilizzo di più Servlet per poter gestire richieste di tipo diverso.
Inoltre, la FacesServlet ha il compito di gestire le richieste per file del tipo *.jsf* oppure *.faces* e per questo motivo è necessario specificarne il mapping su di esse :

```
<servlet-mapping>
 <servlet-name>FacesServlet</servlet-name>
 <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

Con questa configurazione, ogni qualvolta giunge una richiesta di una pagina *.jsf* al Web Container, quest'ultimo passa tale richiesta alla *FacesServlet* che si occuperà di gestirla.

2.1.2 Lifecycle - PhaseListener

Ogni richiesta che arriva alla Web application ha un proprio ciclo di vita (lifecycle) che prevede l'esecuzione di una serie di fasi, fino alla determinazione della risposta. Di tali operazioni, si occupa la classe *Lifecycle* che riceve ciascuna richiesta dalla *FacesServlet* ed esegue le varie fasi, mediante il proprio metodo *execute()*.
Inoltre, ad essa è associata l'interfaccia *PhaseListener*, la quale permette di notificare l'inizio e la fine di ciascuna fase. Tale interfaccia rappresenta un punto di estensione del framework, in quanto è possibile definire alcune elaborazioni specifiche da eseguire prima o dopo una certa fase. Per fare ciò, è necessario realizzare una classe che implementi l'interfaccia *PhaseListener* ed esegua l'overriding dei metodi *beforePhase()* ed *afterPhase()*, all'interno dei quali sarà contenuto il codice da eseguire.

2.1.3 Application

All'avvio dell'applicazione, viene eseguita la lettura mediante un parsing XML, del file di configurazione e tutte le informazioni in esso contenute vengono caricate all'interno delle proprietà corrispondenti della classe *Application*.
In questo modo, durante l'esecuzione dell'applicazione è possibile accedere a queste informazioni ed eventualmente modificarle.

Le classi principali che sono legate ad essa e che vengono specificate nel file *faces-config.xml* sono le seguenti :

- *ActionListener* : per la gestione degli action events;
- *Lifecycle*, *PhaseListener* : per la gestione del ciclo di vita di ciascuna richiesta;
- *StateManager* : per il salvataggio ed il ripristino dello stato dell'applicazione fra una richiesta ed un'altra;
- *NavigationHandler* : per la gestione della navigazione tra le pagine in base alle *navigation-rules* e agli *outcome*;
- *ViewHandler* : per la gestione delle fasi di ricostruzione della vista di una pagina da uno stato precedente e di visualizzare una risposta;
- *MessageBundle* : che identifica il file *.properties* nel quale ci sono tutti i messaggi da visualizzare nell'applicazione, sulla base di un certo *Locale*, permettendo l'internazionalizzazione;

Se tali classi non vengono specificate nel file *faces-config.xml*, verranno utilizzate quelle di default. In caso contrario, lo sviluppatore ha la possibilità di personalizzare il framework, estendendo queste classi e specificando le proprie classi così ottenute nel file di configurazione.
Inoltre, essa è legata a tutte quelle classi che definiscono gli elementi costitutivi dell'applicazione, in particolare :

- *UIComponent* : che definisce ciascun componente dell'interfaccia utente insieme a tutte le sue classi derivate;
- *Validator* : che implementa un validatore per i dati di input dell'utente;
- *Converter* : che permette di eseguire la conversione dei dati;
- *MethodBinding*, *ValueBinding* : che associano i componenti ed i loro valori, ai metodi o alle proprietà dei backing beans;

2.1.4 FacesContext - ExternalContext

La classe *FacesContext* contiene tutte le informazioni che caratterizzano una richiesta e ciò che riguarda la relativa risposta da generare. Essa viene potenzialmente modificata in ciascuna fase del ciclo di vita di una richiesta, per garantire che il contesto sia sempre aggiornato.

Inoltre, è legata a tutte le altre classi che si occupano della fase di "rendering" e quindi della produzione di una risposta per una specifica richiesta :

- *UIViewRoot* : rappresenta il nodo radice della view che definisce l'albero dei componenti di una certa pagina;
- *RenderKit* : è una collezione di istanze della classe *Renderer* che si occupano di visualizzare i componenti dell'interfaccia utente, secondo una certa grafica, in relazione al dispositivo client che viene utilizzato per accedere all'applicazione, il linguaggio di markup ed il *Locale* corrente;
- *ResponseWriter* : definisce uno stream attraverso il quale è possibile produrre, direttamente in una pagina di risposta, degli elementi ed i corrispondenti attributi come per i linguaggi HTML ed XML;
- *FacesMessage* : classe che contiene un singolo messaggio associato ad un componente specifico e che va visualizzato in una pagina. Il messaggio in questione può essere tipicamente un messaggio di errore, ad esempio dovuto ad una validazione o conversione errata, ma anche di un qualsiasi altro tipo;

Infine, attraverso il metodo *getExternalContext()*, è possibile ricavare l'istanza della classe *ExternalContext*, la quale definisce il "contesto esterno", ossia tutto ciò che riguarda l'ambiente in cui viene eseguita l'applicazione. Da questa classe è possibile accedere direttamente a tutte le variabili il cui ambito di visibilità (scope) sia quello di tipo *session*, *request* ed *application* e quindi ai parametri ed agli attributi di una richiesta.

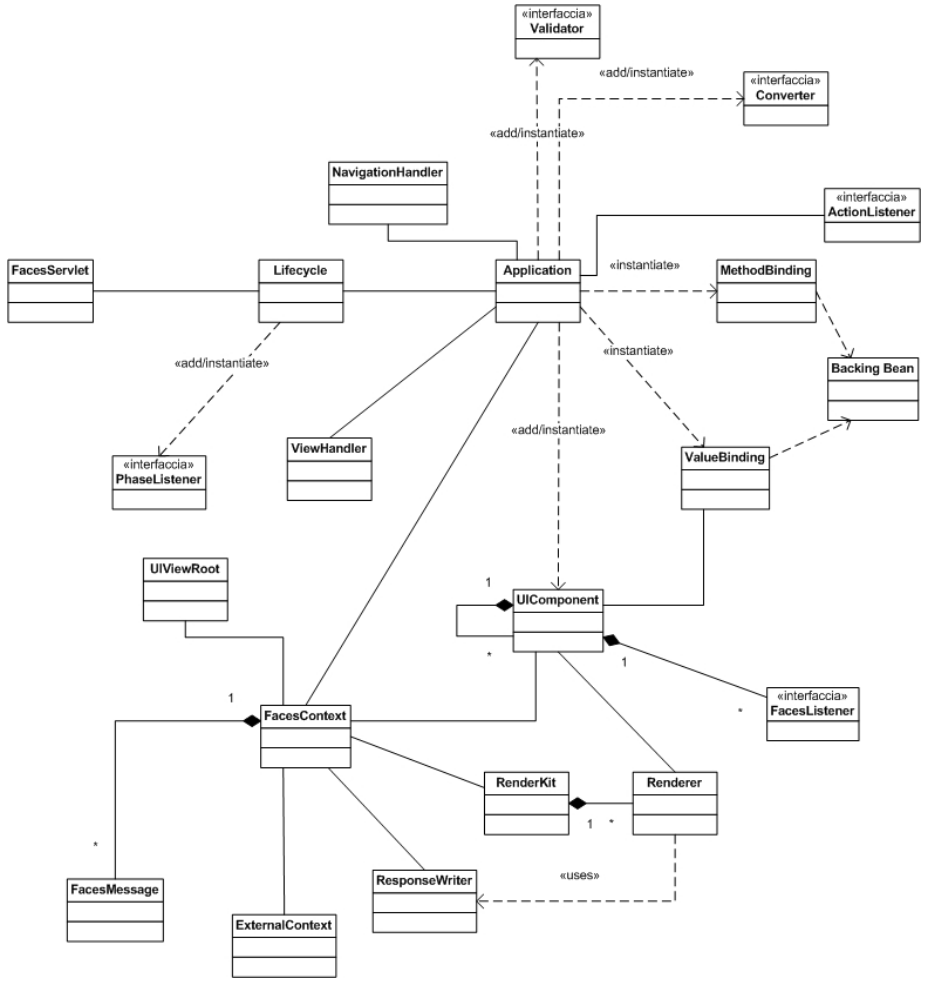


Figura 2 - Class Diagram Execution Model

2.2 User Interface Component Model

Il framework JSF permette l'estensione delle classi relative ai componenti dell'interfaccia utente, in modo da potenziare i componenti esistenti aggiungendogli nuove funzionalità.

La classe di base da cui derivano tutte le classi per i componenti standard è *UIComponentBase*, la quale definisce lo stato ed il comportamento di default di un generico componente ed a sua volta estende la classe *UIComponent*.

Di seguito sono riportate tutte le classi che descrivono i componenti previsti in JSF :

- *UIColumn* : generica colonna del componente *UIData*;
- *UICommand* : controllo che può intercettare un action event (es. click del mouse) quando viene attivato;
- *UIData* : permette di gestire un "binding" con una collection di dati per la loro visualizzazione;
- *UIForm* : permette di raggruppare più controlli di una pagina e definire, appunto, un form di invio delle informazioni contenute. E' paragonabile al tag "form" dell' HTML;
- *UIGraphic* : immagine da visualizzare in una pagina;
- *UIInput* : permette di acquisire un'informazione di input dall'utente;
- *UIMessage* : permette la visualizzazione di un messaggio anche sulla base della localizzazione geografica dell'utente (internazionalizzazione);
- *UIMessages* : come il precedente, ma permette di visualizzare un gruppo di più messaggi;
- *UIOutput* : visualizza un'informazione di output in una pagina;
- *UIPanel* : permette di raggruppare più componenti secondo un certo layout;
- *UIParameter* : usato insieme ad altri componenti per specificare una serie di parametri da passare ad essi;
- *UISelectBoolean* : controllo che permette all'utente di selezionare un valore booleano;
- *UISelectItem* : definisce una singola item di una collection di items;
- *UISelectItems* : definisce un insieme di items;
- *UISelectMany* : permette all'utente di selezionare una o più items da una collection di items;
- *UISelectOne* : permette all'utente di selezionare una sola item nell'ambito di una collection di items a disposizione;
- *UIViewRoot* : rappresenta il nodo radice dell'albero dei componenti che, secondo una gerarchia, costituiscono una pagina;

Le classi suddette, oltre ad estendere la classe base *UIComponentBase*, implementano anche una serie di interfacce che ne definiscono il comportamento:

- *ActionSource* : indica che un componente può intercettare un action–event, implementata dalla classe UICommand;
- *ValueHolder* : indica che un componente può immagazzinare un local-value così come ha la possibilità di accedere al dato corrispondente all'interno del Model, implementata dalla classe UIOutput e da UIInput;
- *EditableValueHolder* : estende *ValueHolder* aggiungendo ulteriori funzionalità ai componenti "editabili", tra cui la validazione e l'intercettazione di eventi del tipo value-change, implementata dalla classe UIInput;
- *NamingContainer* : fa in modo che ciascun componente abbia un identificativo univoco;
- *StateHolder* : segnala che un componente ha uno stato che va salvato ad ogni richiesta, implementata dalle classi UICommand,UIOutput e UIInput;
- *CustomComponent* : per la realizzazione di un componente personalizzato è necessario estendere la classe *UIComponentBase*.

Per poter introdurre un componente all'interno di una pagina, è necessario associare ad esso un tag realizzato mediante una classe nota come Tag Handler.

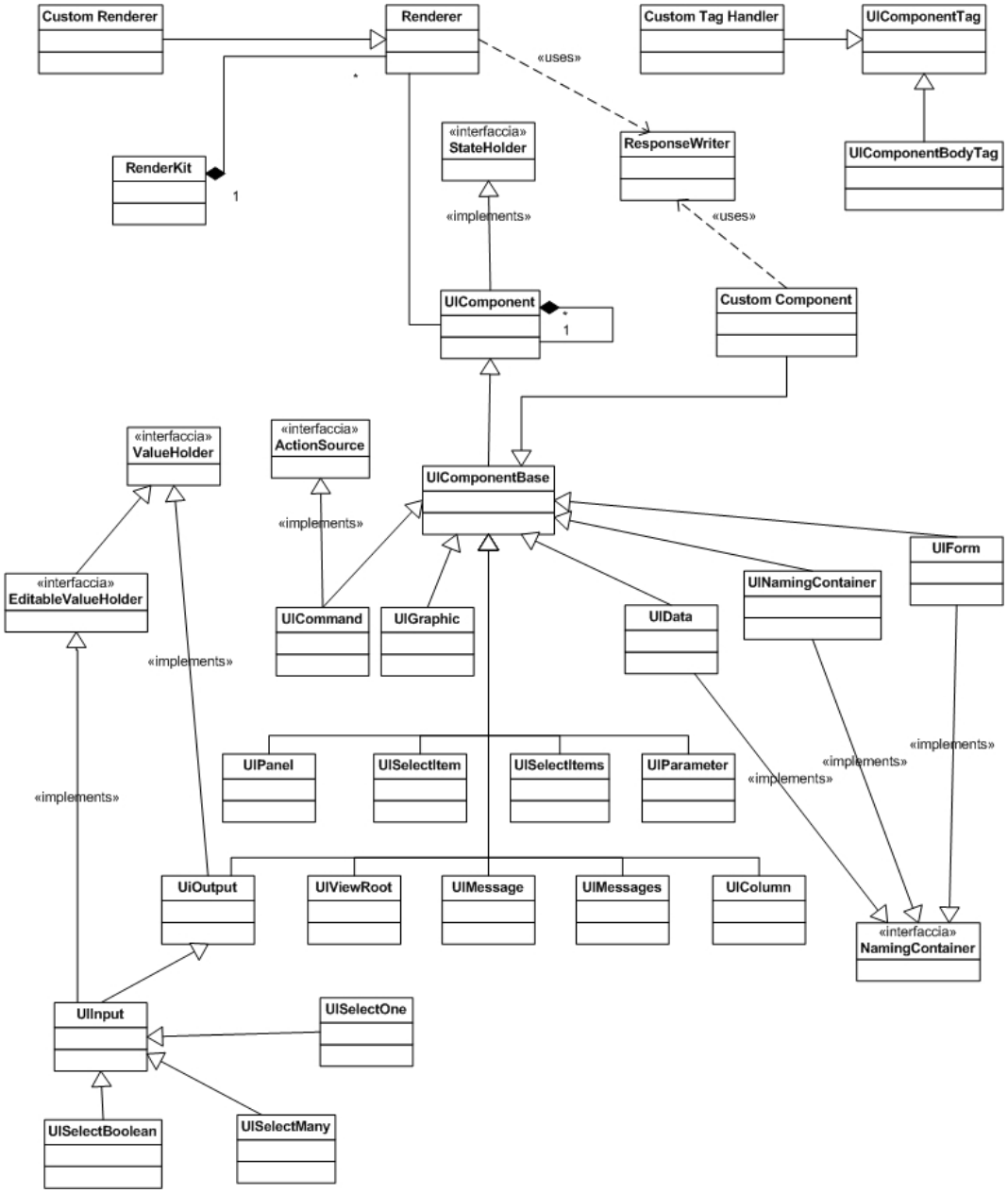


Figura 3 - Class Diagram UI Component Model / Rendering Model

2.3 Component Rendering Model

La visualizzazione dei componenti viene realizzata mediante delle altre classi specifiche note come render, questo permette di definire una sola volta il comportamento di un componente, ma possono essere realizzate diverse visualizzazioni, in relazione ai dispositivi client che possono accedere alla Web application.

La classe che permette di definire un renderer associato ad un componente è la *Renderer*. Essa converte la rappresentazione interna di un oggetto *UIComponent* in una rappresentazione grafica, attraverso uno stream di output sulla pagina.
Mediante la realizzazione di una o più classi che estendono la classe *Renderer*, si possono realizzare diversi renderers che cambiano la modalità di visualizzazione dello stesso componente in base alle necessità o al dispositivo client di accesso.
Un esempio tipico è la classe *UICommand* la cui funzionalità principale è quella di intercettare un action event e permettere un invio di informazioni, così come il passaggio da una pagina all'altra. Questo scopo può essere raggiunto attraverso un "link" oppure un "bottone". Questi ultimi hanno due tag associati per la loro visualizzazione e quindi due classi *Renderer* diverse, che ne permettono una grafica differente, pur basandosi sulla medesima classe componente (appunto *UICommand*).

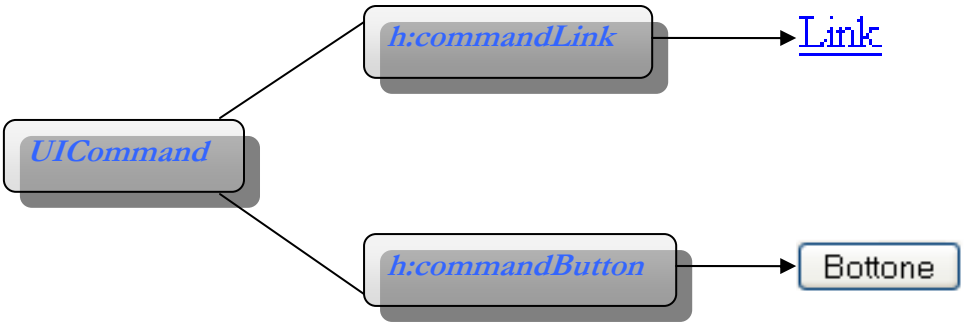


Figura 4 - Rendering UICommand

Un insieme di classi *Renderer* costituiscono un Renderer Kit associato ad uno specifico dispositivo di accesso alla Web application, il quale si basa su un determinato linguaggio di markup.

2.4 Conversion Model

In una Web application realizzata con JSF, ogni componente è associato ad un backing bean, di cui esistono due viste : la model view che rappresenta un certo tipo di dato all'interno della proprietà del backing bean e la presentation view che rappresenta la modalità secondo la quale il valore può essere letto e modificato dall'utente. i può essere letto e modificato dall'utente.

Ad esempio, un componente potrebbe prevedere l'immissione di una data da parte dell'utente. La data sarebbe rappresentata come un oggetto *java.util.Date* all'interno del backing bean (model view) e attraverso una stringa del tipo "yyyy-mm-dd" nel componente (presentation view).

L'implementazione JSF effettua automaticamente la conversione tra una vista e l'altra; in alcuni casi, è possibile che una proprietà di un backing bean, non sia di un tipo di dato standard tra quelli previsti dal linguaggio Java, ma sia un di un tipo di dato, definito dallo sviluppatore. In queste situazioni, associando il backing bean ad un componente, si rende necessaria la realizzazione dei Custom Converters.

Importanza rilevante in questo contesto è assunta dall'interfaccia Converter, che deve essere implementata da una classe per poter eseguire una conversione Object-to-String (da model view a presentation view) e quella inversa String-to-Object (da presentation view a model view), usando i metodi *getAsObject() e getAsString()*.

Ad essa è legata anche la classe *ConverterException* che definisce un'eccezione che può essere sollevata nel momento in cui, per un qualsiasi motivo, la conversione eseguita attraverso i metodi del converter non vada a buon fine. In tal caso, verrà creato automaticamente un oggetto *FacesMessage*, contenente il messaggio che segnala l'errore, che verrà memorizzato nel *FacesContext* e visualizzato all'utente.Infine, per poter associare un converter ad un componente dell'interfaccia utente, è necessario definire un tag mediante la classe *ConverterTag*.

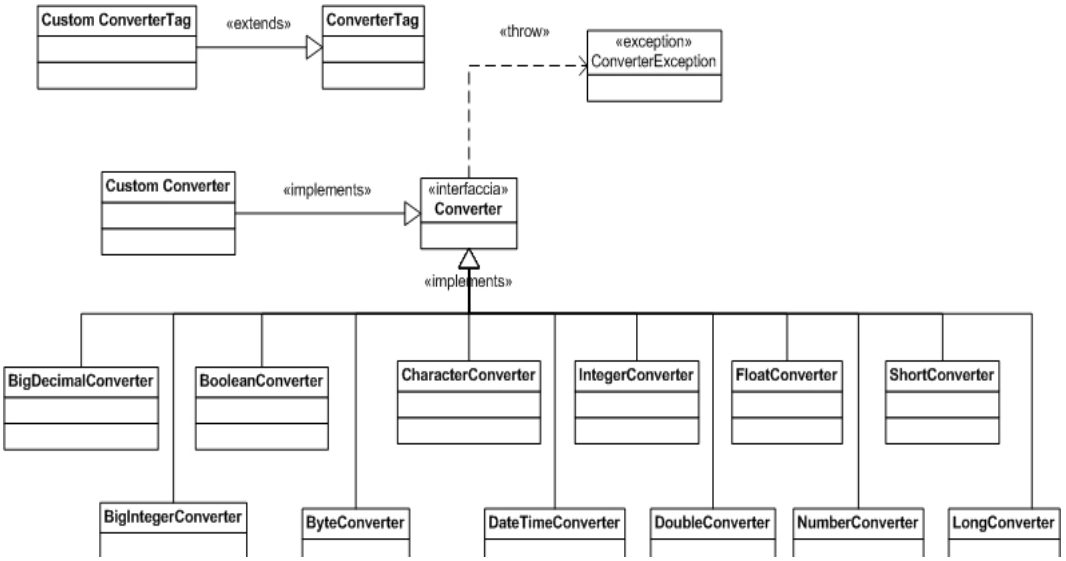


Figura 5 - Class Diagram Conversion Model

2.5 Event and Listener Model

Questo modello definisce gli eventi (events) ed i relativi gestori (listeners). Nel momento in cui l'utente, interagendo con la UI, fa scatenare l'evento sul componente, entra in gioco il listener che si occupa di gestire l'evento. La tecnologia JSF supporta tre tipi di eventi :

- *action event* : si scatena quanto l'utente attiva un componente che implementa l'interfaccia *ActionSource*, come nel caso di bottoni o collegamenti ipertestuali (link);
- *value-change event* : si verifica quando l'utente cambia il valore di un componente implementato dalla classe *UIInput* o da una delle sue sottoclassi, come il caso dei semplici campi di testo, checkbox, radio bottoni, list box, combo box e drop-down list;
- *data-model event* : si scatena quando c'è uno spostamento ad una nuova riga in un componente della classe *UIData*;

Ci sono due modalità per poter gestire un evento :

- implementare una classe listener e registrarla sul componente di interesse utilizzando gli attributi *actionListener* oppure *valueChangeListener* del tag del componente stesso;
- implementare all'interno del backing bean associato al componente, un metodo che debba gestire l'evento;

2.5.1 FacesEvent e FacesListener

E' la classe base che definisce il generico evento che si può verificare su un componente dell'interfaccia utente e memorizza tutte le informazioni che riguardano proprio quest'ultimo.
Da essa derivano due sottoclassi :

- *ActionEvent* : che definisce un action event;
- *ValueChangeEvent* : che notifica il verificarsi di un value-change event;

L'interfaccia mediante la quale è possibile realizzare una classe che abbia la capacità di gestire un generico *FacesEvent è la FacesListener*.
Da essa derivano due interfacce :

- *ActionListener* : interfaccia capace di gestire un action event, relativa alla classe *ActionEvent*;
- *ValueChangeListener* : interfaccia per la gestione di un value-change event, relativa alla classe *ValueChangeEvent*;

Le interfacce e le classi listeners in questione sono legate alla classe *AbortProcessingException*, che rappresenta l'eccezione che può essere sollevata nel momento in cui si verifichi un errore durante la gestione di un evento.

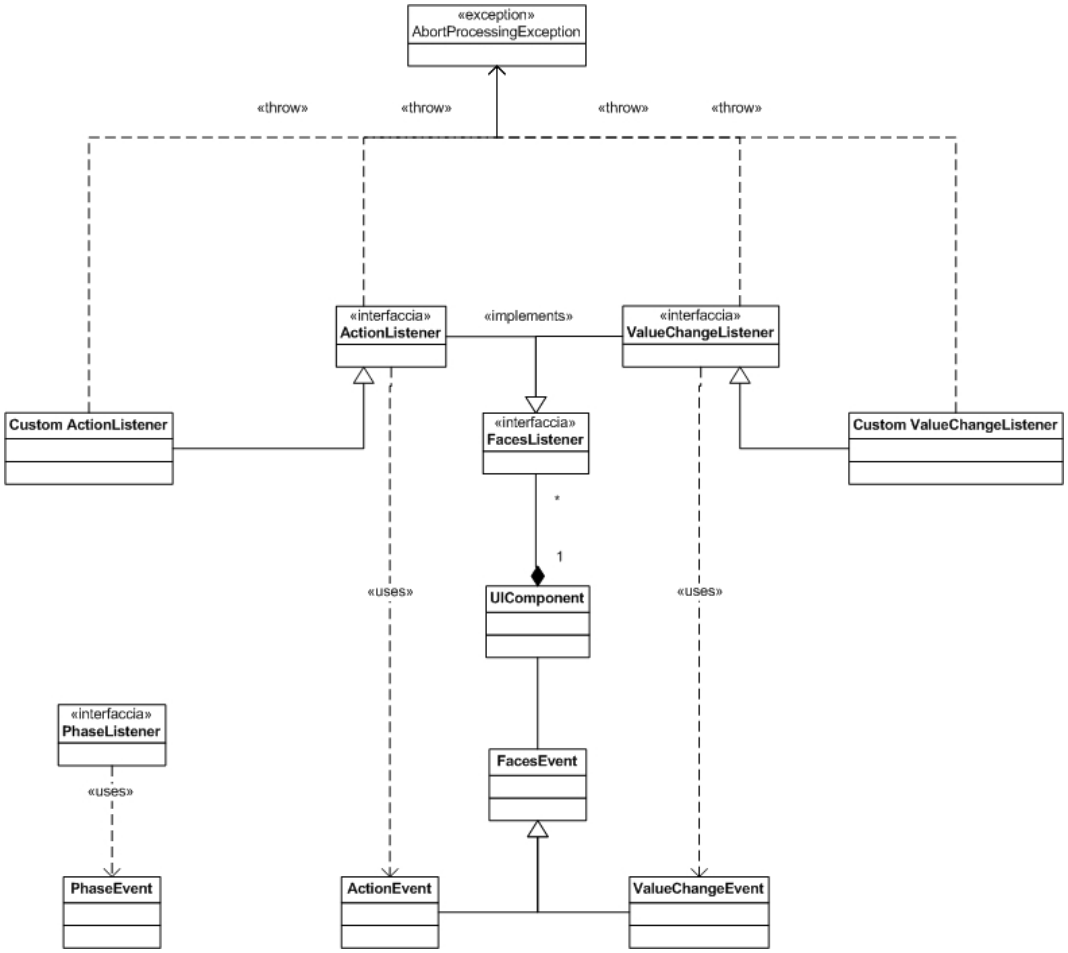


Figura 6 - Class Diagram Event and Listener Model

2.6 Validation Model

Il meccanismo della validazione viene eseguito prima che il model value corrispondente venga aggiornato ed assuma come valore proprio il local value.
Il framework JSF fornisce una serie di classi standard che possono utilizzate per eseguire delle semplici validazioni, in particolare :

- *LengthValidator* : per verificare se il valore di un componente ha una lunghezza che rientra nei limiti specificati;
- *LongRangevalidator, DoubleRangeValidator* : per verificare se il valore di tipo long o double di un componente, rientra in un range prefissato;

E' comunque possibile realizzare dei validatori personalizzati (Custom Validators), nei seguenti modi :

- realizzare una classe che implementa l'interfaccia *Validator*, registrarla all'interno dell'applicazione e definire un tag che permetta di utilizzare il validatore in una pagina JSP, associandolo ad un componente;
- implementare all'interno del backing bean associato al componente, di cui fare la validazione del valore, un metodo che la esegua;

La prima soluzione risulta quella più interessante, infatti l'interfaccia *Validator* viene implementata da una qualsiasi classe che debba eseguire delle operazioni di validazione, attraverso l'overriding dell'unico metodo *validate()*. Tale metodo può sollevare un'eccezione del tipo *ValidatorException* nel momento in cui si è verificato un problema nel processo di validazione.
Infine, viene utilizzata la classe *ValidatorTag che permette di realizzare* Tag Handler, ossia l'associazione tra validator ed il relativo tag.

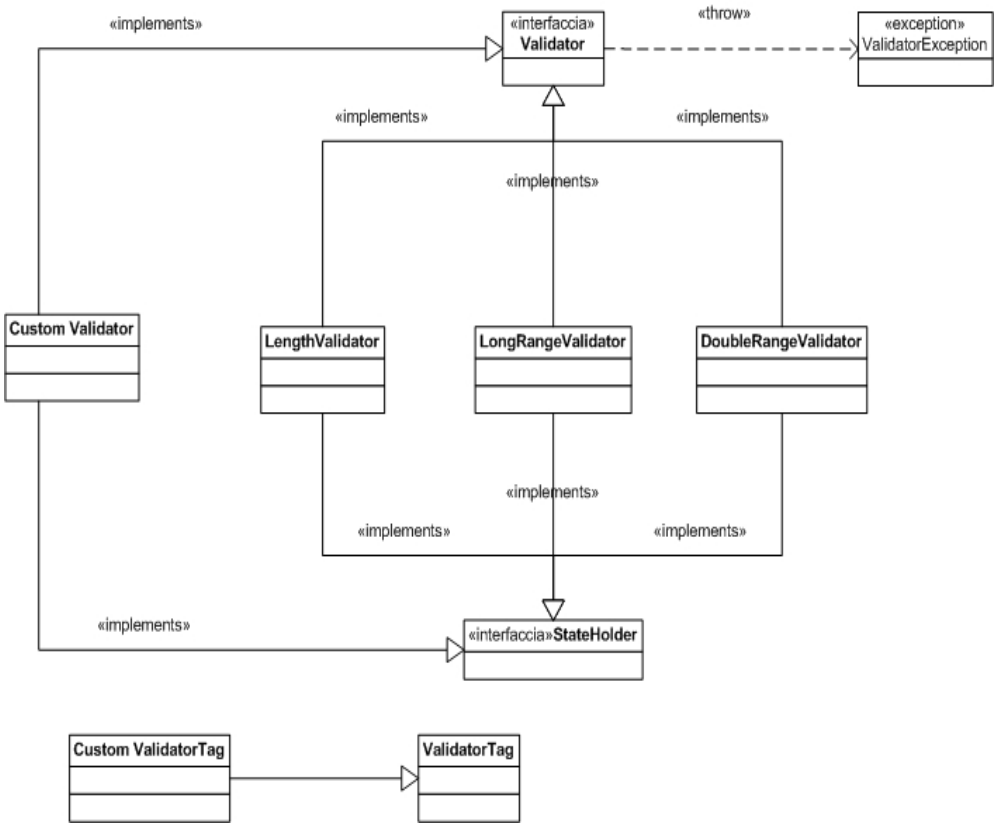


Figura 7 - Class Diagram Validation Model

2.7 Navigation Model

Questo modello permette di definire la navigazione tra le pagine della Web application mediante la definizione di una serie di regole (*navigation-rules*) e degli *outcome*.

Le regole specificano come eseguire la navigazione da una certa pagina verso un'altra, attraverso le casisistiche di navigazione (*navigation-cases*). Ciascun *navigation-case*, all'interno di una *navigation-rule*, specifica una pagina di destinazione e l'*outcome* che permette di referenziarla per proseguire la navigazione.

Gli *outcome* definisce una stringa che permette di selezionare una tra le regole ammesse e vengono definiti in corrispondenza di determinati bottoni o link.

Ad esempio, quando si clicca su un bottone oppure su di un link, viene generato un action event gestito dall'istanza *ActionListener* di default, che invoca un action-method, implementato all'interno di un backing bean. Tale metodo esegue eventualmente un'elaborazione e determina l'*outcome* che viene passato al *NavigationHandler* di default, il quale analizza le regole di navigazione, cerca l'*outcome* che gli è stato fornito ed individua la pagina successiva.

2.8 Backing Bean Management

Un'applicazione JSF prevede una serie di backing beans, che sono associati ad uno o più componenti della UI, in quanto ciascuna delle sue proprietà è legata al valore del componente stesso oppure ad una sua istanza. Inoltre, il bean può avere dei metodi che permettono di eseguire ulteriori elaborazioni sui componenti, come la validazione, la gestione degli eventi e il processo di navigazione.

Il binding tra il valore di un componente della UI oppure di una sua istanza e le proprietà di un backing bean, viene realizzato con la sintassi dell'Expression Language (EL). Nel primo caso si parla di *value-binding*, mentre nel secondo di *method-binding*.

Se si considera un componente dell'interfaccia utente, immesso in una pagina JSP mediante l'uso di un particolare tag, il suo valore sarà legato ad una proprietà di un backing bean mediante l'attributo *value*, referenziando tale proprietà mediante l'EL. In questo modo, il componente avrà un suo local value ed il corrispondente model value sarà memorizzato nella proprietà del bean.

Nel caso in cui si voglia associare l'istanza del componente e non il suo valore, ad una proprietà di un backing bean, si può utilizzare l'attributo *binding*.
Inoltre, è possibile utilizzare gli attributi *validator*, *actionListener* e *valueChangeListener* per fare riferimento ad alcuni metodi dei backing bean, per poter eseguire delle operazioni di validazione e per poter gestire gli eventi del tipo "action" e "value-change".
Ad esempio, considerando un campo di testo il cui valore deve essere trasferito nella proprietà di un backing bean, dopo essere stato validato attraverso un metodo di quest'ultimo, la sintassi del tag <*h:inputText*> è la seguente :

```
<h:inputText id="idCampoTesto"
        value="#{backingBean.proprieta}"
        validator="#{backingBean.metodoValidazione}"
        valueChangeListener="#{backingBean.metodoEvento}"/>
```

Sulla base di questa definizione, se il valore del componente subisce un cambiamento ed il form che lo contiene viene trasmesso, si scatena un value-change event che verrà gestito dal metodo del backing bean specificato.
Infine, nel momento in cui si ha a che fare con un componente che implementa l'interfaccia *ActionSource* (es. bottone o link), si può utilizzare l'attributo *action* per specificare il metodo di un backing bean che esegue una certa elaborazione e fornisce come risultato un *outcome* per proseguire la navigazione in una certa direzione.

```
<h:commandButton id="idBottone" action="#{backingBean.metodo}"/>
```

Eseguire binding fra l'istanza di un componente e la proprietà di un bean, permette al bean di modificare gli attributi del componente durante l'esecuzione, ed inoltre gli permette di istanziare un componente.
Invece, eseguire il binding fra il valore del componente e la proprietà del bean, permette a colui che ha realizzato la pagina di avere più controllo sugli attributi del componente, i backing beans non sono legati alle classi dei componenti e quindi si ha una netta separazione fra Model e View ed inoltre l'implementazione JSF può realizzare la conversione tra il valore del componente e la relativa proprietà del backing bean in maniera automatica, senza che lo sviluppatore debba realizzare un convertitore, se non in casi particolari.

I backing beans sono configurati nel file *faces-config.xml* che viene valutato all'avvio dell'applicazione, rendendo disponibili tali beans che vengono poi istanziati quando sono referenziati all'interno dei tag.

```
<managed-bean>
 <managed-bean-name>backingBean</managed-bean-name>
 <managed-bean-class>
   package.ClasseBackingBean
 </managed-bean-class>
 <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Gli ambiti di visibilità (scope) in cui sono accessibili i backing beans, sono :

- *request* : un bean è accessibile soltanto in corrispondenza di una richiesta HTTP;
- *session* : un bean è visibile durante un'intera sessione utente;
- *application* : un bean è condiviso ed accessibile fra tutti gli utenti che accedono all'applicazione;

Figura 8 - Dichiarazione Backing Beans

La presenza dei backing beans introduce in JSF, l'uso dei pattern *Inversion of Control* (IoC), noto anche come *Dependency Injection*. Tale pattern prevede che, nell'ambito della realizzazione di un software, ci sia sempre una classe nota come "container" che ha il compito di istanziare gli oggetti della business-logic e gestirne la dipendenza e lo scambio di dati fra essi. Nel framework JSF, tutto ciò accade in maniera assolutamente automatica.

3. Ciclo di vita di una pagina JavaServer Faces

Il ciclo di vita (life-cycle) di una pagina JSF, inizia quando il client richiede una pagina, successivamente l'implementazione JSF costruisce la *view*, tenendo conto dello stato degli oggetti relativo ad una richiesta precedente della pagina stessa (viene ricostruito l'ultimo stato della pagina). Il client interagisce con la pagina ed invia i dati; l'implementazione JSF esegue la validazione di questi ultimi e la conversione dei valori in tipi validi dal lato del server. Infine, vengono eseguite le funzionalità della Web application e proposti i risultati al client, nella forma della pagina HTML risultante.

Una pagina JSF è rappresentata da un albero dei componenti che costituiscono l'interfaccia utente, chiamato *view*, caratterizzato da una gerarchia ben definito.

Ad esempio, nella figura che segue, la pagina contenente il form di Login avrà :

- il nodo *root* dell'albero;
- il nodo relativo al form, legato al nodo *root*;
- all'interno del nodo del form, i nodi associati ai tre seguenti componenti : due campi di testo ed un bottone;

Ovviamente, questa è soltanto una parte della pagina, che potrebbe contenere ulteriori componenti.

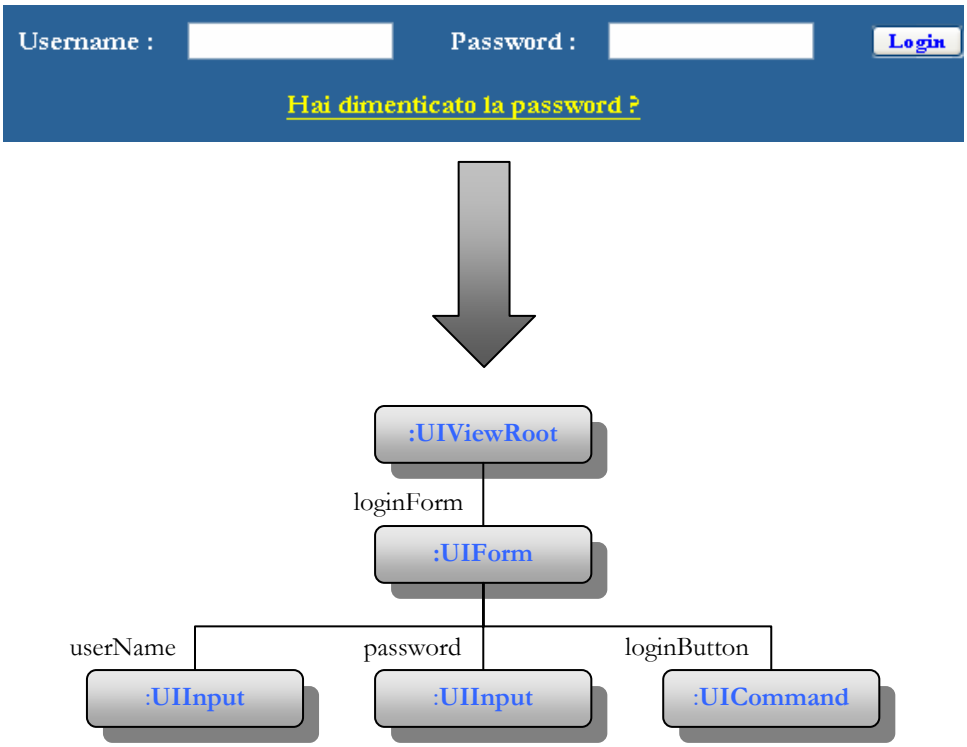


Figura 9 - Esempio struttura della view

3.1 Scenari di elaborazione di una richiesta

Un'applicazione JSF supporta due tipi di richieste (request) e due tipi di (response) :

- *Faces response* : risposta generata durante la fase di *Render Response* del ciclo di vita di una request;
- *Non-Faces response* : risposta che non è creata durante la fase di *Render Response* del ciclo di vita di una request; caso tipico è quello di una pagina JSP che non contiene componenti JSF;
- *Faces request* : una richiesta che viene inviata da una *Faces response* generata in precedenza;
- *Non-Faces request* : richiesta inviata ad un elemento come una Servlet o una pagina JSP e non direttamente ad un componente dell'albero dei componenti JSF di una pagina;

La differenziazione delle richieste e delle risposte suddetta, da luogo a tre possibili scenari del ciclo di vita di una richiesta :

Scenario 1 : Non-Faces request genera una Faces response

Un esempio tipico si verifica nel momento in cui si clicca su semplice link HTML per aprire una pagina JSF. In questo caso, entra in gioco il Controller della tecnologia JSF, ossia una Servlet della classe *FacesServlet* che accetta la richiesta e la passa alla classe che implementa il ciclo di vita della stessa per elaborarla. Quando deve essere generata la *Faces response*, l'applicazione acquisisce i riferimenti agli oggetti necessari alla *view*, memorizza quest'ultima nel *FacesContext* (contesto contenente tutte le informazioni della richiesta) ed invoca il metodo *renderResponse()*, passando alla fase *Render Response*.

Scenario 2 : Faces request genera una Non-Faces response

Talvolta, un'applicazione JSF può avere la necessità di generare una response che non contiene componenti della tecnologia JavaServer Faces. In queste situazioni, lo sviluppatore deve saltare la fase di *Render Response* invocando il metodo *responseComplete()* della classe *FacesContext*. Tale chiamata può avvenire in una delle seguenti fasi : *Apply Request Value*, *Process Validations*, *Update Model Values*.

Scenario 3 : Faces request genera una Faces response

Questo è lo scenario tipico, che segue il ciclo di vita standard di una richiesta, in cui un componente JSF invia una request ad un'applicazione JSF utilizzando la *FacesSarvlet*. Tutti i listeners, validators e converters sono invocati nella fase opportuna del ciclo di vita stesso.

3.2 Ciclo di vita Standard

Il ciclo di vita standard di una richiesta è quello dello terzo scenario, che è stato introdotto in precedenza. La tecnologia JSF permette di gestire due tipologie di richieste : richiesta iniziale (*initial request*) e *postback*. L'initial request è caratterizzata dal fatto che il client richiede per la prima volta una certa pagina e viene gestita eseguendo solo le fasi di *Restore View* e *Render Response*, mentre un postback scaturisce dall'invio di un form che è stato generato da un'initial request precedente e viene gestita eseguendo nell'ordine tutte le fasi.

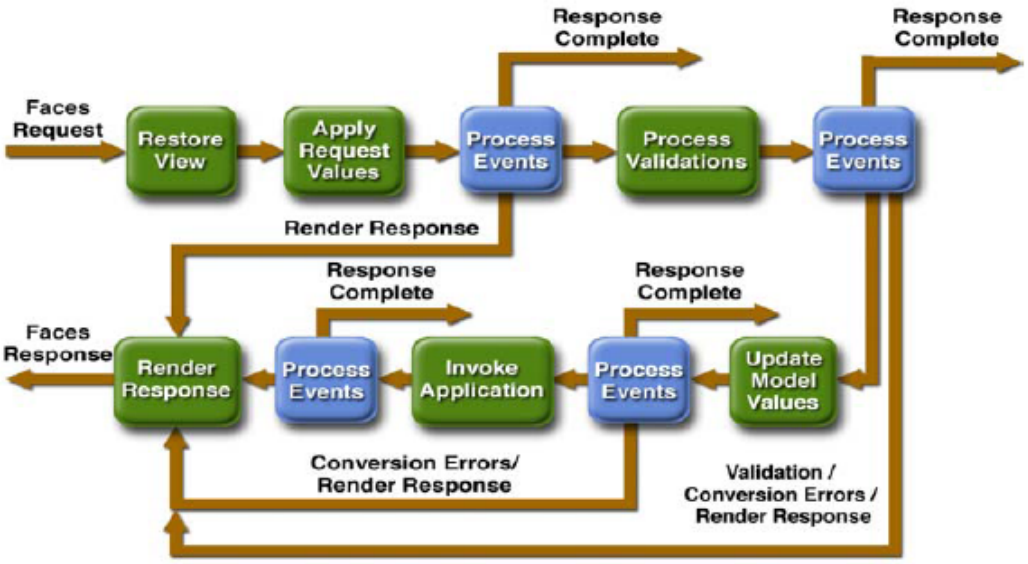


Figura 10 - Ciclo di vita Standard

3.2.1 Restore View

Rappresenta la prima fase del life-cycle e viene avviata quando una richiesta per una pagina JSF viene generata.

Durante questa fase, viene costruita la *view* della pagina (albero degli oggetti associati ai componenti della UI), vengono associati gli events handlers (listener), i validators ed i converters ai componenti ed infine la *view* viene memorizzata nel *FacesContext*, che avrà tutte le informazioni per gestire la richiesta.

Se si tratta di una initial request viene creata una *view* vuota e si passa direttamente alla fase di *Render Response*. Tale *view* vuota sarà popolata quando, ad esempio, l'utente immetterà dei dati in un form e verrà comandato un postback.

Se si tratta di un postback, la *view* precedente relativa alla pagina già esiste e viene recuperata, utilizzando le informazioni di stato dei componenti della UI che sono salvate sul client oppure sul server (per default).

3.2.2 Apply Request Values

Dopo aver recuperato l'albero dei componenti (*view*), ogni componente estrae il suo nuovo valore dai parametri della richiesta, usando il metodo *decode()*. Tale valore, noto come *submittedValue*, viene memorizzato localmente al componente stesso. Se la conversione del valore fallisce, viene memorizzato nel *FacesContext* un messaggio di errore che sarà visualizzato nella fase di *Render Response*, insieme ad eventuali altri errori di validazione successivi.

Se un metodo *decode()* oppure un event listener invoca il metodo *renderResponse()* del *FacesContext*, si salta direttamente alla fase di *Render Response*.

Se degli eventi vengono accodati in questa fase, ne verrà eseguito il broadcast verso i corrispondenti listeners, ma la loro gestione avverrà nelle fasi successive.

Se, invece, alcuni componenti della pagina hanno l'attributo *immediate* settato a *true*, allora la validazione, conversione e gestione degli eventi associati vengono processati subito e non posticipati alle fasi successive. A questo punto, se bisogna redirezionare il flusso dell'applicazione verso una risorsa che non ha componenti JSF, viene invocato il metodo *responseComplete()* del *FacesContext*.

Al termine di questa fase, i componenti sono settati ai loro nuovi valori (*submittedValues*) ed i messaggi eventuali di errore e gli eventi sono messi in coda, pronti per essere processati nelle fasi successive.

3.2.3 Process Validation

Durante questa fase, vengono eseguiti tutti i validators registrati sui componenti della *view*. In primo luogo, vengono valutati gli attributi fissati che specificano le regole di validazione e viene confrontato il valore memorizzato in ciascun componente (local-value) con le regole stesse.

Se il valore locale di un componente non è valido, viene memorizzato un messaggio di errore nel *FacesContext* e si passa alla fase di *Render Response*, per visualizzare tutti i messaggi di questo tipo, insieme ad altri eventuali messaggi dovuti ad errori di conversione nella fase precedente.

Se un metodo *validate()* oppure un event-listener invoca il metodo *renderResponse()* del *FacesContext*, si passa direttamente alla fase di *Render Response*.

A questo punto, se bisogna redirezionare il flusso dell'applicazione verso una risorsa che non ha componenti JSF, viene invocato il metodo *responseComplete()* del *FacesContext*.

Se degli eventi vengono accodati in questa fase, ne verrà eseguito il broadcast verso i corrispondenti listeners, ma la loro gestione avverrà nelle fasi successive.

3.2.4 Update Model Values

Una volta che i valori locali dei componenti sono considerati validi, è possibile percorrere l'albero dei componenti stessi ed assegnare alle proprietà dei backing beans, tali valori. Ciò vuol dire rendere i *submittedValue*, valori effettivi del modello (model-value). Ovviamente i local-values devono essere convertiti correttamente nei tipi di dati accettati dalle proprietà dei backing beans. Se si verificano degli errori, si passa direttamente alla fase di *Render Response* per visualizzarli, in maniera molto simile agli errori di validazione nelle fasi precedenti.

Se un metodo *updateModels()* oppure un event listener chiama il metodo *renderResponse()* del *FacesContext*, si salta direttamente alla fase di *Render Response*.A questo punto, se bisogna redirezionare il flusso dell'applicazione verso una risorsa che non ha componenti JSF, viene invocato il metodo *responseComplete()* del *FacesContext*.

Se degli eventi vengono accodati in questa fase, ne verrà eseguito il broadcast (smistamento) verso i corrispondenti listeners, ma la loro gestione avverrà nelle fasi successive.

3.2.5 Invoke Application

In questa fase viene gestito ogni evento sia di tipo action che di tipo value-change, come nei casi di invio di un form di click su di un link.
Se bisogna redirezionare il flusso dell'applicazione verso una risorsa che non ha componenti JSF, viene invocato il metodo *responseComplete()* del *FacesContext*.
Se la *view* è stata recuperata da una richiesta precedente ed su di un componente viene sollevato un evento, questo viene smistato verso il suo listener.

3.2.6 Render Response

Durante questa fase, l'implementazione JSF delega il compito della visualizzazione (rendering) della risposta al JSP container, ovviamente nel caso in cui si utilizzino delle pagine JSP.
Se si tratta di una initial request, i componenti della pagina vengono aggiunti alla *view*, mentre nel caso di un postback già sono presenti. In entrambi i casi, i componenti eseguono il proprio rendering sulla base dei tag che li rappresentano nella pagina JSP.
Nel caso di un postback, se nelle fasi precedenti si è verificato qualche errore, viene visualizzata comunque la pagina e se ci sono in essa dei tag *<h:message>* oppure *<h:messages>*, vengono visualizzati i messaggi di errore.
Una volta visualizzata la pagina, viene salvato lo stato della response, ossia lo stato dei componenti, sul client oppure sul server, in modo che per un'eventuale successiva request potrà essere eseguita la fase di *Restore View*.

4. JSF Expression Language

Il framework JSF utilizza l'Expression Language (EL) mediante il quale è possibile definire delle espressioni, per realizzare il value-binding ed il method-bindig tra i componenti dell'interfaccia utente e le proprietà oppure i metodi dei backing beans. In queste espressioni viene specificato il nome del bean e la proprietà oppure il metodo a cui fare riferimento, mediante la tipica "dot notation".

#{backingBean.[proprieta | metodo]}

Inoltre, esso fornisce una serie di operatori aritmetici e logici per definire espressioni molto più complesse, magari utilizzando come operandi le proprietà dei backing beans a cui si fa riferimento.
Sono altresì disponibili i seguenti oggetti impliciti (implicit object) ai quali si può fare riferimento da una qualsiasi pagina JSP :

- *applicationScope* : oggetto Map con varibiabili e beans che si trovano nell'ambito di visibilità di tipo *application*;
- *requestScope* : oggetto Map con varibiabili e beans che si trovano nell'ambito di visibilità di tipo *request*;
- *sessionScope* : oggetto Map con varibiabili e beans che si trovano nell'ambito di visibilità di tipo *session*;
- *cookie* : oggetto Map contenente i cookie associati alla richiesta;
- *facesContext* : istanza della classe *FacesContext*;
- *header* : oggetto Map contenente gli header HTTP della richiesta corrente;
- *param* : oggetto Map che ha al suo interno i parametri della richiesta;
- *view* : oggetto che rappresenta il nodo root della *view*;

5. Espandibilità del framework

La tecnologia JavaServer Faces offre allo sviluppatore la possibilità di estendere completamente le classi del framework. Nella maggior parte dei casi, per quanto riguarda l'Execution Model, vengono utilizzate sempre le classi di default a meno di particolari esigenze. La potenzialità di personalizzare il framework viene sfruttata soprattutto per quanto riguarda la realizzazione di :

- *Custom Converter*;
- *Event Listeners*;
- *Custom Validator*;
- *Custom UI Components*;

5.1 Custom Converter

Il framework JavaServer Faces esegue automaticamente tutte le conversioni necessarie dei valori dei componenti della UI nelle corrispondenti proprietà dei backing beans, in alcuni casi, può nascere l'esigenza di realizzare un convertitore personalizzato realizzando una classe che implementi l'interfaccia *Converter*.
Tale classe dovrà inoltre eseguire l'overriding dei due metodi seguenti :

- *getAsObject()* : riceve il valore del componente e lo converte nel tipo di dato della proprietà corrispondente del backing bean associato;
- *getAsString()* : riceve il valore della proprietà del backing bean e lo converte in stringa per la visualizzazione nella UI;

```
public Object getAsObject(FacesContext context,
                          UIComponent component,
                          String value) {
...
}

public String getAsString(FacesContext context,
                          UIComponent component,
                          Object value) {
...
}
```

Durante la fase di *Apply Request Values*, quando l'utente ha immesso dei valori nei componenti (es. riempiendo un campo di testo), viene invocato il metodo

decode() del componente oppure del renderer corrispondente, che si occupa di acquisire tale valore. Ovviamente, è in questo metodo che viene invocato *getAsObject()*.
Viceversa, durante la fase di *Render Response*, quando bisogna visualizzare nell'interfaccia utente il valore di un componente, vengono invocati i metodi di encoding del componente stesso oppure del renderer associato, che a loro volta invocano *getAsString()*.
Infine, per rendere disponibile il converter nell'applicazione, bisogna registrarlo nel file *faces-config.xml*. Inoltre, affinché lo si possa associare ad un componente, è necessario usare l'attributo *converter* del componente stesso oppure il tag <*f:converter*> della Core Library di JSF.

```
<f:converter converterId="idConverter"/>
```

5.2 *Registrare un Custom Converter*

La registrazione di un Custom Converter viene realizzata all'interno del file di configurazione, mediante l'elemento <*converter*>. Quest'ultimo, a sua volta, prevede ulteriori elementi al suo interno, per specificare le seguenti informazioni:

- identificativo univoco del converter per poterlo referenziare nel tag di un componente della UI che ne fa uso (<*converter-id*>);
- la classe che implementa il converter (<*converter-class*>);

```
<converter>
 <converter-id>idConverter</converter-id>
 <converter-class>package.ClasseConverter</converter-class>
</converter>
```

5.3 *Event Listener*

Come visto in precedenza, l'implementazione JSF permette di gestire due tipologie di eventi che si possono verificare sui componenti della UI :

- action event;
- value-change event;

Tale gestione, può essere eseguita in due modi diversi :

- realizzare una classe listener che implementi l'interfaccia *ActionListener* oppure *ValueChangeListener*;
- realizzare, all'interno di un certo backing bean, un metodo che sia capace di gestire l'evento;

5.3.1 Implementazione di un ActionListener/ValueChangeListener

Per poter gestire un action event che può essere sollevato da un componente, è possibile sviluppare una classe che implementa l'interfaccia *ActionListener*. All'interno di essa, sarà eseguito l'overriding del metodo *processAction()*, il quale prevede come parametro di ingresso un oggetto di tipo *ActionEvent* che contiene tutte le informazioni sull'evento e sul componente che l'ha generato.
Ovviamente, all'interno di questo metodo, sarà necessario scrivere il codice per la gestione dell'evento, in quanto il metodo sarà immediatamente invocato, nel momento in cui l'evento sarà intercettato.

```
public void processAction(ActionEvent event)
    throws AbortProcessingException {
...
}
```

Una volta realizzata la classe, essa sarà associata ad un componente utilizzando il tag <*f:actionListener*>.

```
<f:actionListener type="classeActionListener"/>
```

Per poter gestire un value-change event per un certo componente, è possibile realizzare una classe che implementa l'interfaccia *ValueChangeListener*. Tale classe deve semplicemente eseguire l'overriding del metodo *processValueChange()*, il quale ha un parametro di ingresso di tipo *ValueChangeEvent* che contiene le informazioni sull'evento che si è verificato. Tale metodo viene invocato nel momento in cui l'evento si verifica e quindi conterrà il codice per la sua gestione.

```
public void processValueChange(ValueChangeEvent event)
        throws AbortProcessingException {
...
}
```

Infine, per poter registrare la classe listener sul componente, si può utilizzare il tag <*f:valueChangeListener*> della Core Library di JSF.

```
<f:valueChangeListener type="classeValueChangeListener"/>
```

5.3.2 Backing Bean Method Listener

Una possibile strada alternativa alla realizzazione delle classi listener, è lo sviluppo di alcuni metodi all'interno dei backing beans che abbiano la capacità di gestire gli eventi.

Nel caso in cui bisogna gestire un action event, è possibile realizzare un metodo che non ha parametri di uscite ed ha come unico parametro di ingresso, un oggetto di tipo *ActionEvent* con le informazioni dell'evento intercettato.

Tale metodo potrà essere associato al componente, utilizzando l'attributo *actionListener* del tag che rappresenta il componente stesso all'interno di una pagina.

```
<h:commandLink actionListener="#{backingBean.metodoListener}"/>
```

Se invece abbiamo a che fare con un value-change event, bisogna realizzare un metodo che ha come parametro di ingresso, un oggetto del tipo *ValueChangeEvent* e che sarà associato al componente, mediante l'attributo *valueChangeListener* del medesimo tag.

```
<h:inputText
    valueChangeListener="#{backingBean.metodoListener}"/>
```

5.4 Custom Validator

Per eseguire la validazione dei dati, JSF mette a disposizione una serie di validatori standard, ma nel caso in cui questi ultimi non siano sufficienti, è possibile realizzarne di propri.

Le modalità possibili sono le seguenti :

- realizzare una classe che implementa l'interfaccia *Validator*;
- sviluppare un metodo all'interno di un backing bean, che si occupi della validazione;

5.4.1 Implementazione di un Validator

Nel caso in cui la scelta sia quella di realizzare una classe che implementa l'interfaccia *Validator*, si deve poi poter associare tale validatore ad un qualsiasi componente che ne possa fare uso. Le possibili soluzioni sono le seguenti :

- se il validator prevede degli attributi e si vuole dare la possibilità al *Page author* di assegnare ad essi dei valori specifici, bisogna realizzare un custom tag mediante il quale è possibile immettere il validatore all'interno della pagina. Tale tag avrà quindi degli attributi accessibili al realizzatore della pagina, secondo una modalità XML-like;
- in alcuni casi si può preferire non realizzare un tag proprietario del validator. In queste situazioni, per associare il validatore ad un componente, si deve utilizzare il tag <*f:validator*> della Core Library , specificando la classe del validatore, ed uno o più tag <*f:attribute*> per specificarne gli attributi;
- un ultimo caso particolare è quello in cui si realizza un Custom Component e gli si vuole associare un validatore integrato. Ciò vuol dire che quest'ultimo verrà associato al componente durante l'esecuzione, mediante il metodo *addValidator()*, e non ci sarà la possibilità per il *Page author* di farlo manualmente all'interno di una pagina. Questa soluzione "integrata" fornisce un componente potenziato di un validator integrato, ma non da flessibilità al realizzatore della pagina di poter utlizzare il validatore al di fuori del componente;

Durante la validazione, si potrebbe verificare un errore ed in questo caso, sarebbe utile visualizzare un messaggio all'utente. E' possibile creare i messaggi durante l'esecuzione mediante la classe *FacesMessage*, memorizzandoli nel *FacesContext*. In generale, si preferisce sollevare un'eccezione del tipo *ValidatorException*, nel metodo *validate()* della classe , con il messaggio di errore da visualizzare.

Nel caso più generale possibile, i passi per poter realizzare un validatore sono i seguenti :

- realizzare la classe che implementa l'interfaccia *Validator*;
- realizzare un Tag Handler;
- definire un tag associato al validatore, all'interno di un file TLD (Tag Library Descriptor);

5.4.1.1 Class Validator

La classe che implementa l'interfaccia *Validator* deve tipicamente avere una serie di proprietà, che non sono altro che gli attributi del validatore stesso. Ovviamente, deve essere anche dotata dei metodi di accesso a tali proprietà (Accessor Method), nella forma *get*NomeProprieta*() set*NomeProprieta*()*. Inoltre, bisogna eseguire l'overriding del metodo *validate()* che dovrà contenere il codice per eseguire la validazione.

```
public void validate(FacesContext context,
                     UIComponent toValidate,
                     Object value)
              throws ValidatorException {
  ...
}
```

5.4.1.2 Tag Handler

Per poter associare un tag al validatore, è necessario realizzare un Tag Handler, che altro non è che una classe che estende la classe di base del framework *ValidatorTag*. Tale classe deve avere le stesse proprietà della classe validator che saranno gli attributi del tag.

Le caratteristiche principali della classe sono le seguenti :

- i metodi di accesso alle proprietà;
- esegue l'overriding del metodo *createValidator()*, all'interno del quale viene creata un'istanza del validatore e ne vengono inizializzate le proprietà sulla base dei valori che il page author ha assegnato agli attributi del tag;

```
protected Validator createValidator() throws JspException {
  ...
}
```

5.4.1.3 Tag Library Descriptor

Per poter associare il validatore ad un componente all'interno di una pagina, è necessario realizzare un tag che lo rappresenti. Fondamentalmente, il tag non fa altro che rappresentare il Tag Handler, il quale avrà poi il compito di utilizzare la classe validator. Per la descrizione dei tag, vengono utilizzati i file TLD (Tag Library Descriptor) all'interno dei quali, i tag stessi, vengono elencati con i relativi attributi, sfruttando una sintassi XML-like.

```
<tag>
 <name>nomeValidator</name>
 <tag-class>package.ClasseValidatorTag</tag-class>
 <body-content>JSP</body-content>
 <attribute>
  <name>attributo</name>
  <required>true</required>
  <type>package.ClasseTipoAttributo</type>
 </attribute>
 ...
</tag>
```

5.4.2 Backing Bean Method Validator

La validazione del valore di un componente può essere eseguita anche realizzando, all'interno di un backing bean, un metodo che riceve come parametri di ingresso : l'oggetto *FacesContext* dell'applicazione, il componente del cui valore farne la validazione ed infine il valore stesso. Tale metodo non ha parametri di uscita ed ha la possibilità di generare dei messaggi di errore, nel caso in cui la validazione non andasse a buon fine.

```
public void metodoValidate(FacesContext context,
                           UIComponent toValidate,
                           Object value) {
  ...
}
```

Per fare in modo che il metodo venga eseguito, è necessario utilizzare l'attributo *validator* del componente il cui valore deve essere soggetto a validazione.

```
<h:inputText id="idCampoTesto" value="#{backingBean.proprieta}"
      validator="#{backingBean.metodoValidate}"/>
```

5.4.3 Registrare un Custom Validator

Se l'Application developer ha realizzato dei Custom Validators per eseguire operazioni di validazione, è necessario registrarli all'interno del file di configurazione mediante l'elemento <*validator*>, che conterrà ulteriori tag per specificare le seguenti informazioni :

- identificativo univoco del validator per referenziarlo all'interno della classe che realizza il Tag Handler (<*validator-id*>);
- la classe che implementa il validator (<*validator-class*>);
- gli attributi previsti dal validator, con il relativo nome ed il tipo (<*attribute*>);

Le prime due informazioni sono necessarie, mentre l'ultima è facoltativa considerando che gli attributi del validator sono definiti nella classe che lo implementa e nel suo Tag Handler.

```
<validator>
 <validator-id>idValidator</validator-id>
 <validator-class>package.ClasseValidator</validator-class>
</validator>
```

5.5 Custom UI Components

La tecnologia JSF mette a disposizione una serie di componenti standard per la realizzazione dell'interfaccia utente, ma è comunque possibile estendere le funzionalità di questi ultimi o addirittura creare di nuovi, con delle funzionalità non ancora esistenti. E' inoltre possibile, creare uno o più renderer per poter visualizzare ciascun componente in maniera diversa su client di tipo differente. Ciò vuol dire che il comportamento del componente viene definito una sola volta, mentre la sua visualizzazione viene delegata ai renderer che possono essere molteplici. Ciò non toglie, che un componente può gestire la visualizzazione in maniera autonoma, non facendo uso dei renderer.

In generale, la classe che realizza un componente, ne definisce lo stato ed il comportamento, il quale include : convertire i valori del componente attraverso i convertitori , accodare gli eventi, eseguire la validazione facendo uso dei validatori ed altre funzionalità.

C'è bisogno di creare un Custom Component, in queste situazioni :

- aggiungere un nuovo comportamento ad un componente standard, ad esempio un nuovo tipo di evento;
- aggregare più componenti per costituirne uno solo con un unico comportamento;
- se si ha bisogno di un componente supportato da un client HTML ma non implementato da JSF;
- realizzare un componente che possa essere utilizzato da un client non HTML, quando magari il componente stesso esiste per un client HTML;

Non c'è la necessità di realizzare un Custom Component in queste situazioni :

- se bisogna manipolare i dati di un componente o aggiungervi semplici funzionalità. In questo caso, basta creare dei metodi nel backing bean, associato ad esso;
- bisogna convertire il valore di un componente in un dato non supportato dal suo renderer. In tale situazione, conviene adottare un convertitore standard o realizzarne uno personalizzato;
- se bisogna eseguire la validazione sul valore del componente, conviene utilizzare un validatore standard o realizzarne uno;
- se c'è la necessità di registrare degli event listeners su un componente. In tal caso, si adottano le tecniche per la realizzazione degli event listeners;

Quando si crea un Custom Component, bisogna essere sicuri che la classe sia capace di eseguire le seguenti operazioni :

- *Deconding* : prelevare, dai parametri della request, il valore del componente che è stato immesso dall'utente ed associarlo al local value;
- *Encoding* : eseguire il rendering del componente secondo un certo linguaggio di markup, quale può essere HTML , XML , WML in relazione al tipo di client;

JSF supporta due modelli di programmazione per il deconding e l'encoding :

- *Direct implementation* : la stessa classe del componente implementa il decoding e l'encoding;
- *Delegated implementation* : la classe del componente delega l'implementazione del decoding e dell'encoding ad un renderer;

Con il secondo modello, si ha il vantaggio di poter definire il componente una sola volta (stato e comportamento) ma di avere più renderer associati ad esso, che ne permettano la visualizzazione in forme diverse a secondo del device client (browser web , cellulare,...).
Nel caso di un Custom Component, si rende necessaria la realizzazione di un tag che permetta di introdurre il componente in una pagina, operazione opzionale, invece, nel caso in cui si realizzi un validator e quasi mai utilizzata nel caso di un converter. In generale, per poter realizzare un Custom Component, si seguono i seguenti passi :

- sviluppare una classe che rappresenti il componente ed estenda la classe base del framework *UIComponentBase* oppure una delle sue derivate;
- realizzare un Tag Handler;
- eventualmente, sviluppare uno o più renderer estendendo la classe *Renderer*;
- infine, descrivere il tag all'interno di un file TLD;

5.5.1 Registrare un Custom Component

Oltre alla registrazione di un Custom Renderer, si rende ovviamente necessaria anche la registrazione del Custom Component a cui esso è associato, mediante l'elemento <*component*>. Per ciascun componente, vengono specificate le seguenti informazioni :

- il tipo del componente (<*component-type*>);
- la classe che implementa il componente (<*component-class*>);
- le proprietà del componente (<*property*>), per ciascuna delle quali è definito il nome (<*property-name*>) e la classe (<*property-class*>);

Tipicamente, le prime due informazioni sono necessarie, mentre le successive sono facoltative, in quanto tali proprietà vengono comunque definite all'interno della classe che implementa il componente.

```
<component>
 <component-type>TipoComponente</component-type>
 <component-class>package.ClasseComponente</component-class>
</component>
```

5.5.2 Class Component

Tutti i componenti standard estendono le classi *UIComponentBase* oppure *UIComponent*. Quando si vuole creare un Custom Component, bisogna estendere una di queste ultime se lo si vuole realizzare partendo da zero. E' possibile invece, poter sfruttare le caratteristiche di un componente standard derivando dalla sua classe. Ad esempio se bisogna creare un menù editabile, non conviene estendere la classe *UIComponentBase* ma la classe *UISelectOne*, in modo da sfruttarne il suo comportamento e doverne aggiungere solo le funzionalità necessarie.

Ovviamente, il componente sarà utilizzato all'interno di una pagina ed in generale sarà associato ad un backing bean, che conterrà in una sua proprietà il valore del componente (value-binding) e potrebbe avere dei metodi referenziati dal componente stesso (method-binding).

Il primo passo per la realizzazione della classe del Custom Component è quello di eseguire l'overriding del metodo *getFamily()* che restituisce la famiglia di appartenenza del componente, registrata anche all'interno del file *faces-config.xml*.

```
public final static String COMPONENT_TYPE ="TipoComponente";
public final static String COMPONENT_FAMILY ="FamigliaComponente";
...
...
public String getFamily() {
  return COMPONENT_FAMILY;
}
...
```

```
<component>
 <component-type>TipoComponente</component-type>
 <component-class>package.ClasseComponente</component-class>
</component>
```

Nel caso in cui il componente gestisca in maniera autonoma il rendering, deve occuparsi sia della fase di decoding che di encoding. Per quanto riguarda la fase di deconding, basta eseguire l'overriding del metodo *decode()* che viene invocato nella fase di *Apply Values Change* del ciclo di vita di una richiesta. Per quanto concerne l'encoding, bisogna eseguire l'overriding dei metodi *encodeBegin()*, *encodeChildre()* ed *encodeEnd()*.

C'è da precisare che, nella maggior parte dei casi, basta realizzare soltanto il primo metodo; i due successivi vengono usati soltanto se il componente ha dei componenti figli al suo interno.

```
...
public void decode(FacesContext context) {
 ...
}
...
...
public void encodeBegin(FacesContext context) throws IOException {
 ...
 ResponseWriter writer = context.getResponseWriter();
 writer.startElement("nomeElemento", this);
 writer.writeAttribute("nomeAttributo", valore,"nomeAttributo");
 ...
 writer.write(body);
 writer.endElement("nomeElemento");
 ...
}
```

All'interno di ciascuno di questi metodi, si fa uso di un oggetto *ResponseWriter*, che permette di scrivere direttamente su una pagina, mediante uno stream di output, in un linguaggio XML-like. Si osserva che il metodo *startElement()* permette di scrivere sullo stream il tag di apertura, usando una o più volte *writeAttribute()* è possibile scrivere gli attributi ed i corrispondenti valori, con il metodo *write()* si scrive il body del tag ed infine, usando *endElement()* si scrive il tag di chiusura. In una pagina JSP, il risultato di queste operazioni sarà una cosa del tipo seguente :

```
<libreria:nomeElemento nomeAttributo="[valore]" />
[body]
</libreria:nomeElemento>
```

Se il rendering viene delegato ad un renderer, i metodi di decoding e di encoding non vengono realizzati, ma viene eseguito l'overriding del metodo *getRendererType()* che restituisce l'identificativo del renderer adottato.

5.5.3 Tag Handler

La classe Tag Handler associata con un componente, estende la classe base del framework *UIComponentTag* e guida la fase *Render Response* del ciclo di vita di una richiesta. La prima operazione eseguita è quella di recuperare il tipo di componente associato al tag mediante il metodo *getComponentType()*.

```
public String getComponentType() {
  return ClasseComponente.COMPONENT_TYPE;
}
```

Successivamente, mediante il metodo *setProperties()*, vengono inizializzate le proprietà del componente con i valori degli attributi del tag che sono stati specificati nella pagina JSP in cui compare il tag stesso.

```
protected void setProperties(UIComponent component) {
  ...
  componente.setNomeAttributo(this.nomeAttributo);
  ...
  ...
}
```

Infine, tale classe, mediante il metodo *getRendererType()*, restituisce il tipo di renderer adottato, se ne è associato uno al componente, in modo da permettere all'implementazione JSF di eseguire il deconding e l'encoding di quest'ultimo.

```
public String getRendererType() {
  return ClasseRenderer.RENDERER_TYPE;
}
```

Ovviamente, gli attributi di un tag possono avere come valore, non una stringa, ma un'espressione nel linguaggio EL, per poter eseguire le operazioni di value-binding o method-bindig e quindi introdurre dei riferimenti alle proprietà di un backing bean oppure ai suoi metodi.

Per ogni attributo che accetta una espressione EL, all'interno del metodo *setPropterties()*, per settare l'attributo, è necessario utilizzare le classi *MethodBinding* e *ValueBinding*. L'oggetto *MethodBinding* serve per valutare un'espressione relativa ad un metodo di un backing bean, mentre l'oggetto *ValueBinding* permette di valutare un'espressione relativa ad una delle sue proprietà.E' raccomandabile che ogni Tag Handler abbia un metodo *release()* per rilasciare le risorse allocate, una volta che siano state utilizzate.

5.5.4 Tag Library Descriptor

Per poter utilizzare un componente all'interno di una pagina, è necessario realizzare un tag che lo rappresenti. Fondamentalmente, il tag non fa altro che rappresentare il Tag Handler, il quale avrà poi il compito di utilizzare la classe del componente. Per la descrizione dei tag, vengono utilizzati i file TLD (Tag LIbrary Descriptor) all'interno dei quali, i tag stessi, vengono elencati con i relativi attributi, sfruttando una sintassi XML-like.

```
<tag>
 <name>nomeComponente</name>
 <tag-class>package.ClasseComponente</tag-class>
 <body-content>JSP</body-content>
 <attribute>
  <name>attributo</name>
  <type>package.classeTipoAttributo</type>
 </attribute>
 ...
</tag>
```

5.5.5 Classe Renderer

Nel caso in cui il componente non gestisca il decoding e l'encoding in maniera autonoma, è necessario realizzare un renderer al quale delegare queste operazioni.

La classe che implementa il renderer estende la classe base *Renderer* e deve eseguire l'overriding dei metodi *decode()*, *encodeBegin()*, *encodeChildren()* ed *encodeEnd()* per portare a termine il suo compito.

```
public void decode(FacesContext context, UIComponent component) {
 ...
}
...
...
public void encodeBegin(FacesContext context,
            UIComponent component) throws IOException {
 ...
}
```

Rispetto alla signature degli stessi metodi all'interno della classe che realizza il componente, questi ultimi hanno come parametro di ingresso anche l'istanza del componente da renderizzare.

5.5.6 Registrare un Custom Renderer in un Renderer Kit

Un Renderer Kit definisce un insieme di classi *Renderer* che vengono utilizzate per poter eseguire la visualizzazione di uno o più componenti della UI, in maniera diversa in base al dispositivo client con cui si accede alla Web application.

Nel caso in cui l'*Application developer* crea un Custom Renderer, per la visualizzazione di un certo componente, è necessario che venga assegnato ad un Renderer Kit, mediante la registrazione nel file di configurazione.

L'elemento XML adottato è <*renderer-kit*>, il quale prevede al suo interno uno o più tag <*renderer*>, ciascuno dei quali contiene le informazioni seguenti relative ad uno specifico renderer :

- la famiglia del componente della UI a cui è associato il renderer (<*component-family*>);
- il tipo del renderer (<*renderer-type*>), che viene utilizzato dal Tag Handler associato al componente, per determinare quale renderer applicare su di esso;
- la classe che implementa il renderer (<*renderer-class*>);
- gli attributi eventuali del renderer (<*attribute*>) con il proprio nome (<*attribute-name*>) e la classe (<*attribute-class*>);

In generale, non è obbligatorio specificare il Renderer Kit di appartenenza di un Custom Renderer; in questo caso, esso sarà assegnato a quello di default HTML, fornito con l'implementazione JSF.

```
<render-kit>
 <renderer>
  <component-family>FamigliaComponente</component-family>
  <renderer-type>TipoRenderer</renderer-type>
  <renderer-class>package.ClasseRenderer</renderer-class>
 </renderer>
 ...
 ...
</render-kit>
```

6. Sicurezza

Per quanto riguarda la sicurezza, un'applicazione realizzata con JavaServerFaces può utilizzare il protocollo HTTPS per il trasferimento delle pagine e delle informazioni, in modo da garantire la crittografia dei dati. La gestione della sicurezza tramite l'SSL (Secure Socket Layer) è completamente indipendente dal framework ed è praticamente la medesima di una qualsiasi applicazione Web realizzata in Java. In sostanza, si fa affidamento ai meccanismi del Web Container sottostante, all'interno del quale va abilitato il supporto per le connessioni basate su SSL. Una volta effettuata questa operazioni, nell'ambito dell'applicazione è necessario modificare opportunamente il Deployment Descriptor *web.xml*, per poter specificare quali pagine dovranno utilizzare il protocollo HTTPS.
Per questo scopo, si utilizza il tag <*security-constraint*>, all'interno del quale si specificano le risorse Web da trasmettere tramite SSL (tag <*web-resource-collection*>) ed il tipo di trasporto da utilizzare (tag <*transport-guarantee*>) che può essere di tre tipi :

- *NONE* : utilizza HTTP;
- *INTEGRAL* , *CONFIDENTIAL* : utilizza HTTPS;

```
<security-constraint>
 <display-name>SSL Constraint</display-name>
 <web-resource-collection>
  <web-resource-name>
    Automatic SLL Forwarding
  </web-resource-name>
  <url-pattern>[pagina.jsp]</url-pattern>
  <http-method>GET</http-method>
  <http-method>PUT</http-method>
  <http-method>POST</http-method>
  <http-method>DELETE</http-method>
 </web-resource-collection>
 <user-data-constraint>
  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
 </user-data-constraint>
</security-constraint>
```

Questo tipo di approccio evidenzia che la sicurezza è strettamente legata all'architettura sulla quale è in esecuzione l'applicazione e non dal framework con cui è stata realizzata.

7. Configurazione dell'applicazione

In generale, nell'ambito di una Web application realizzata con JSF, i compiti principali dell' *Application architect* sono tipicamente i seguenti :

- registrare gli oggetti di back-end in modo che siano accessibili da un qualsiasi punto dell'applicazione;
- configurare i backing beans (managed beans) in modo che siano istanziati in maniera corretta quando le pagine fanno riferimento ad essi;
- definire la navigazione tra le pagine dell'applicazione;
- eseguire il "packaging" dell'applicazione, includendo tutte le pagine, gli oggetti e gli altri file, in modo che possa essere distribuita in un qualsiasi web container;

La tecnologia JavaServer Faces mette a disposizione una modalità di configurazione dell'applicazione che è completamente portabile, mediante un documento XML tipicamente chiamato *faces-config.xml*.
In realtà, nel caso di applicazioni di grandi dimensioni, l'applicazione può prevedere anche più di un file di configurazione, ciascuno dei quali permette di configurare una parte delle funzionalità dell'applicazione stessa.
E' da ricordare, inoltre, che l'*Application developer* può sempre accedere direttamente da codice al file *faces-config.xml*, grazie all'utilizzo della classe *Application*. Infatti, all'avvio dell'applicazione, un parser XML legge le informazioni del file di configurazione e le carica nell'unica istanza della classe *Application* prevista.

7.1 Configurazione dei Backing Beans

Per istanziare i backing beans che vengono utilizzati in un'applicazione JSF ed assegnarli ad un certo ambito di visibilità (scope), è possibile utilizzare la corrispondente funzionalità offerta nel file di configurazione XML. In questo modo, quando una pagina fa riferimento ad un backing bean, l'implementazione JSF lo inizializza in maniera opportuna in accordo alle informazioni contenute nel file *faces-config.xml*.

Attraverso tale funzionalità, si possono ottenere i seguenti vantaggi :

- registrare i beans in un unico file centralizzato, in modo da essere disponibili nell'ambito di tutta l'applicazione o comunque essere istanziati solo nel momento in cui sono necessari;
- personalizzare le proprietà dei beans, direttamente nel file di configurazione senza scrivere del codice aggiuntivo in Java;
- è possibile assegnare dei valori iniziali alle proprietà di un bean, nel momento in cui viene creato;

Per registrare un backing bean nel file di configurazione, si utilizza l'elemento XML <*managed-bean*>, all'interno del quale si utilizzano ulteriori tag per specificare :

- il nome da assegnare al bean, per poterlo referenziare all'interno delle pagine (<*managed-bean-name*>);
- la classe che implementa il bean (<*managed-bean-class*>);
- l'ambito di visiblità (scope) all'interno dell'applicazione (<*managed-bean-scope*>);
- le relative proprietà con eventuali valori iniziali (<*managed-property*>);

Le prime tre informazioni sono strettamente necessarie, mentre l'ultima è facoltativa, considerando che le proprietà del bean vengono comunque definite nella corrispondente classe Java. Per quanto riguarda l'ambito di visibilità, esso può essere del tipo : *request*, *session*, *application* e *none* (in questo caso il bean viene istanziato ogni volta che deve essere utilizzato).

```
<managed-bean>
 <managed-bean-name>backingBean</managed-bean-name>
 <managed-bean-class>
   package.ClasseBackingBean
 </managed-bean-class>
 <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

7.2 Localizzazione, Internazionalizzazione e Messaggi

Una delle caratteristiche offerte dalla tecnologia JSF è l'Internazionalizzazione (I18N), che permette di visualizzare il testo della Web application, in un linguaggio diverso in base alla localizzazione geografica dell'utilizzatore. Per fare ciò è necessario creare uno o più Resource Bundle, ciascuno dei quali contiene i messaggi da visualizzare in un certo linguaggio, tra quelli supportati dall'applicazione. In generale, i Resource Bundles non sono altro che dei file con estensione "properties", aventi tutti lo stesso nome più il suffisso relativo al linguaggio a cui ciascuno di essi fa riferimento.

Per rendere disponibile questa funzionalità all'applicazione, è necessario registrare i Resource Bundle all'interno del file *faces-config.xml* mediante l'elemento <*message-bundle*>. Ad esso, vanno poi aggiunti il default *Locale* e tutti i restanti *Locale* supportati dall'applicazione, facendo uso di un solo elemento <*default-locale*> ed uno o più <*supported-locale*>.

```
<application>
 <locale-config>
  <default-locale>it</default-locale>
  <supported-locale>en</supported-locale>
 </locale-config>
 <message-bundle>ApplicationResources</message-bundle>
</application>
```

7.3 Configurare le Navigation Rules

Come è stato detto nel trattare il Navigation Model, la navigazione all'interno dell'applicazione è definita da un insieme di regole (*navigation-rules*) che permettono di determinare quale sia la pagina successiva, nel momento in cui si clicca su un bottone oppure su di un link. Tali regole sono configurate all'interno del file *faces-config.xml*.

Ogni regola di navigazione specifica in quale direzione muoversi attraverso le pagine, a partire da una certa pagina. Una volta individuata la regola, sulla base della pagina attualmente visualizzata, si determina la pagina successiva sulla base di un *outcome*, ossia una stringa che viene restituita da un metodo di un backing bean, associato al componente (bottone o link) su cui si è cliccato.

Ciascuna regola viene definita all'interno del file di configurazione, mediante l'elemento <*navigation-rule*>, il quale contiene al suo interno ulteriori tag per specificare le seguenti informazioni :

- la pagina dalla quale si parte per la navigazione applicando tale regola (<*from-view-id*>);
- le possibili pagine di destinazione, individuate attraverso dei "casi" di navigazione (<*navigation-case*>), ciascuno dei quali specifica al suo interno : l'*outcome* (<*from-outcome*>) oppure una action (<*from-action*>) e la pagina stessa di destinazione (<*to-view-id*>);

```
<navigation-rule>
 <from-view-id>/pagina.jsp</from-view-id>
 <navigation-case>
  <from-outcome>nomeOutcome</from-outcome>
  <to-view-id>/paginaDestinazione.jsp</to-view-id>
 </navigation-case>
 ...
</navigation-rule>
```

Capitolo III – Data WareHouse

1. Ricerca delle Fonti

Per la progettazione della data warehouse come primo passo sono state cercate via web le possibili fonti da cui trarre i dati per il magazzino: esistono molteplici siti sia gratuiti che a pagamento.
La ricerca si è mossa quasi esclusivamente su siti "ufficiali", cioè siti di organizzazioni nazionali o internazionali. Questo allo scopo di ottenere dati iniziali che comunque presentino una buona qualità.
I siti gratuiti visitati sono i seguenti :

- Sito dell'Istituto Nazionale di Ricerca per gli Alimenti e la Nutrizione. http://www.inran.it/
- Sito dell'Istituto Europeo di Oncologia. http://www.ieo.it
- Sito del Nutrient Data Laboratory (NDL), facente capo all'United States Department of Agriculture (USDA). http://www.nal.usda.gov/fnic/foodcomp
- Sito dell'ISA (Istituto di Scienze dell'Alimentazione del CNR) dedicata ai prodotti caseari campani. http://www.isa.cnr.it/PRODTIP/
- Sito dell' "International Network of Food Data Systems (INFOODS)", organizzazione della FAO il cui scopo è ottenere dati di composizione alimentari adeguati e affidabili. http://www.fao.org/infoods/data_en.stm
- Sito del Danish Food Composition Databank danese. http://www.foodcomp.dk/fcdb_aboutfooddata.htm
- Sito di Latinfoods, la "Rete Latinoamericana di composizione degli alimenti", costruita con gli auspici della FAO . http://www.inta.cl/latinfoods/default.htm
- Sito del db brasiliano, che sembra avere pochi dati ed in formato html. http://www.fcf.usp.br/tabela/tbcamenu.php
- Sito del database finlandese di composizione: è completamente in finlandese. http://www.ktl.fi/fineli/

Trai i siti a Pagamento invece ci sono :

- Sito della Banca Dati Svizzera di Composizione degli Alimenti http://food.ethz.ch/swifd/CHNWDB_it/HOME/home_it.htm
- Sito della Royal Society of Chemistry (UK), http://www.rsc.org/index.htm
- Sito del "Souci-Fachmann-Kraut Online-Database", database http://www.sfk-online.net/cgi-bin/start.mysql?language=english

2. Fonti Selezionate

Tra le fonti menzionate ne sono state selezionate quattro. La progettazione dei moduli di feeding ha comunque avuto tra gli obiettivi primari quello di garantire una facile estensione del numero di sorgenti di dati.
Le fonti selezionate sono quelle INRAN, IEO, NDL-USDA e ISA-CNR : sono tutte istituzioni autorevoli, cosa che da sola garantisce una buona qualità delle informazioni, ed inoltre forniscono gratuitamente i propri dati.
Le prime due fonti sono state scelte perché contengono informazioni sui cibi effettivamente consumati in Italia e quindi sono di sicuro interesse in ambito nazionale.
La sorgente USDA è stata scelta perché è una vera e propria miniera di informazioni e contiene comunque molti alimenti che non sono troppo differenti da quelli consumati nel nostro paese.
Infine i dati ISA riguardanti i prodotti caseari campani sono stati selezionati in quanto caratterizzano una interessante realtà locale.
Tutte le fonti, tranne quella USDA (che è tutta in inglese), sono in buona parte in italiano. Una delle linee guida del progetto è quella di fornire un supporto per tradurre le parti esclusivamente in inglese in lingua italiana e viceversa. Ovviamente il lavoro di traduzione vero e proprio esula dagli scopi di questa tesi e deve comunque essere coadiuvato dalla presenza di un esperto del settore.

3. Fonte INRAN

3.1 Il Database

La base di dati (scaricata dal sito http://www.inran.it/) è in formato Access 97 e viene distribuita con un semplice programma di consultazione (entrambi gratuiti): il file Access sembra essere funzionale al programma, nel senso che molte tabelle contengono dati il cui scopo è l'utilizzo da parte del sw allegato; la base di dati è protetta da password.
Il database ospita informazioni nutrizionali su 790 alimenti consumati in italia. Le informazioni contenute in questa base di dati sono in italiano.

3.2 Il Software di Consultazione

Il software di interrogazione allegato al database INRAN è molto semplice da utilizzare e permette di consultare la tabella di un singolo alimento. Si può altresì fare una ricerca degli alimenti del db (o di una sola categoria) che contengono maggiormente un certo nutriente. L'applicativo permette di consultare, oltre al db Access, anche delle pagine html statiche con ulteriori informazioni.

3.3 Stato Iniziale del Database

Il db iniziale è più che altro un insieme di tabelle scollegate (non esistono vincoli di integrità referenziale) e fortemente denormalizzate. Si fa notare inoltre come tutti i campi siano di tipo testuale.

3.4 Tabelle Escluse dal Processo di Reverse Engineering

Le tabelle non utilizzate dal processo di reverse engineering sono i cataloghi di sistema e le tabelle:

- "Sezioni" (lista delle sezioni del software allegato).
- "Appendici" (elenco delle appendici del sw allegato).
- "TableDict" e "DataDict" (tabelle di metadatazione che non sono però quelle di sistema di Access, presenti probabilmente a causa di una migrazione da qualche altro DBMS).

E' da notare comunque che le tabelle del punto 3 sono state utilizzate per ricavare i nomi degli attributi del modello concettuale ristrutturato e le descrizioni dei nutrienti e delle unità di misura.

3.5 Considerazioni sull'Uso del Database Come Fonte per il Datawarehousing

Il db considerato è fornito con poca documentazione e quindi le considerazioni che seguono sono state tratte principalmente da un'analisi del database stesso e del suo software.

La base di dati è aggiornata al 2000 (e legata al sw di consultazione): questo è indice che gli aggiornamenti hanno frequenza bassa. Da ciò si evince una maggiore possibilità di cambiamenti radicali dello schema tra un aggiornamento e l'altro e quindi il modulo di feeding automatico potrebbe dover essere modificato per due versioni successive del database.

I dati sono forniti con poche indicazioni riguardo alle fonti secondarie: nella documentazione allegata si legge comunque che i dati riportati sono per il 70% dati originali ottenuti da studi programmati ad hoc e per il rimanente 30% provenienti da una accurata selezione bibliografica prevalentemente italiana, le analisi sono state condotte nell'Unità di Chimica degli Alimenti dell'INRAN.

Un aspetto sicuramente positivo del database è che contiene dati su cibi effettivamente consumati in Italia, cosa importante per un suo utilizzo per ricerche a livello nazionale.

3.6 Fonti di Dati INRAN Non Presenti nel Database

Sul sito dell'INRAN sono presenti degli articoli in formato pdf che presentano anche tabelle di composizione alimentare che potrebbero essere integrate col db, qualora non fossero già presenti in esso, con l'aiuto di un esperto.

Sempre sul sito è inoltre presente un file .doc con un errata corrige che è stato ovviamente integrato con i dati della base di dati.

Il programma di utilizzo del database fa uso anche di pagine html statiche: le informazioni presenti in molte di esse possono essere perfettamente integrate con quelle già presenti nel db.

3.7 Ristrutturazione dello Schema Concettuale

Essendo i codici di alcune colonne del diagramma iniziale poco significativi, sono stati sostituiti con le etichette ricavate dalla tabella di metadatazione "DataDict", al fine di rendere maggiormente leggibili gli schemi finali (esempio ALI_DESC = Alimento, AMLIMI = Aminoacido Limitante, ecc)
Questa operazione di aggiunta di nuovi nomi si è resa necessaria anche per la scarsa documentazione che accompagna il database.
Lo schema ER vero e proprio è nel diagramma "INRAN Final Conceptual", mentre il diagramma "Campi Eliminati" riporta la lista degli attributi dello schema concettuale iniziale che, a causa del processo di normalizzazione, non hanno un corrispettivo in nessun attributo del diagramma concettuale finale.
La normalizzazione si è resa necessaria sia perché nello schema iniziale erano presenti concetti mischiati tra di loro (es. componente e alimento), con le ovvie anomalie che questo comporta, sia per facilitare la successiva operazione di integrazione con gli altri database.
Le nuove entità inserite sono “Nutrienti”, che contiene informazioni generiche sui componenti, e “Note Nutriente”, che invece contiene delle note riguardanti il particolare valore di nutriente in un alimento.
Entrambe le entità aggiunte erano presenti come colonne nelle tabelle iniziali: nello schema “Campi Eliminati” vi sono due tabelle (“Nomi Nutrienti” e “Nomi Note Nutrienti”) che contengono le colonne rimosse dal diagramma iniziale e che sono occorrenze particolari delle due nuove entità create.
Sono state poi ovviamente costruite delle relazioni che legano tutte le entità e una di queste,”Composizione”, presenta anche l’attributo “Quantità”, che ovviamente denota l’ammontare di nutriente nell’alimento.
In fase di normalizzazione si è reso necessario creare anche degli identificativi per le nuove entità (“Nutrienti” e “Note nutriente”).
Si fa notare ancora che il campo "Unità di misura" dice l'unità (es. g, mg) con cui è espresso il nutriente, mentre il campo "Modo di espressione della misura" dice la quantità di alimento rispetto alla quale si riferisce la misura (es. 100g di parte edibile, 100g di proteine, etc.).
Tutti gli attributi inseriti ex novo non hanno definito un tipo (sono “<UNDEF>”), in quanto questa informazione verrà aggiunta in fase di integrazione nel solo diagramma di “unione” degli schemi di tutte le fonti.

3.7.1 Schema "INRAN Final Conceptual"

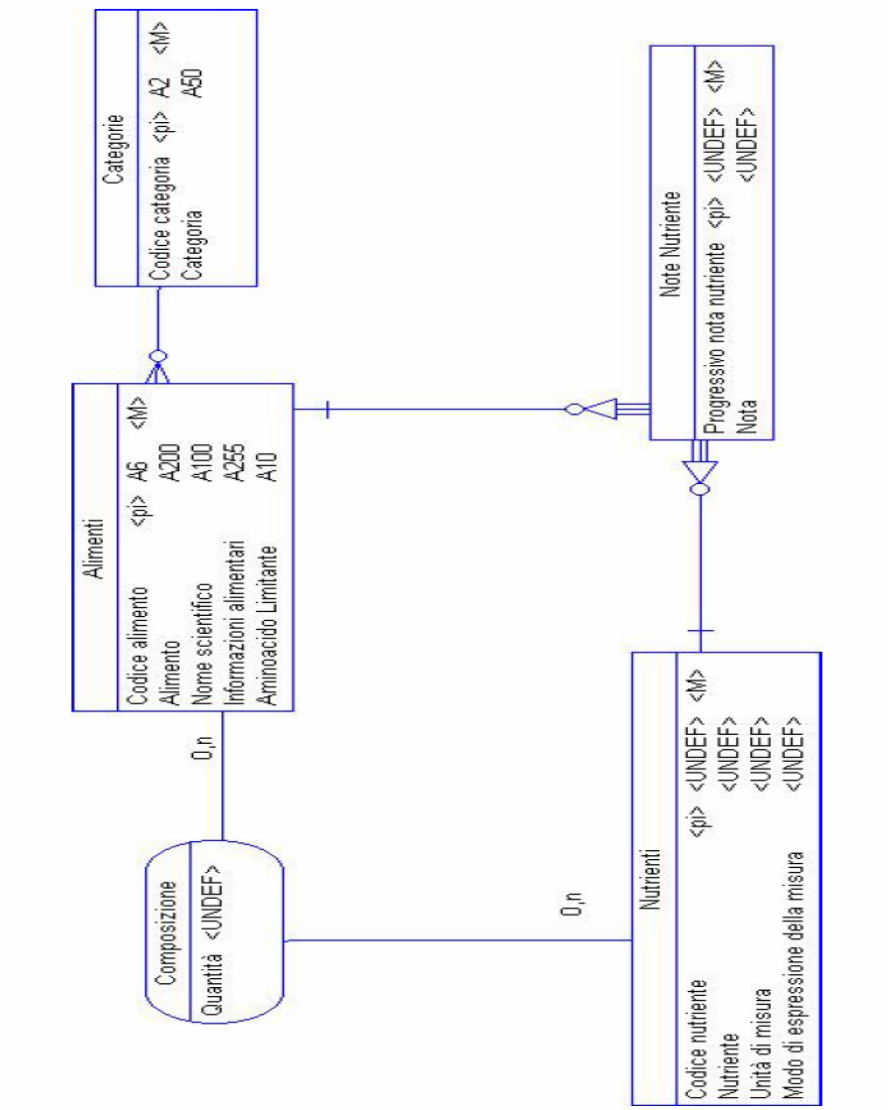


Figura 1 - INRAN Final Conceptual

3.8 Ulteriori Vincoli

Oltre ai vincoli visibili nello schema ristrutturato, ne sono stati individuati altri, che sono:

- "Parte edibile (%)" <= 100
- "Energia in kJ" = 4.184 * "Energia in kcal"
- "Calorie da Proteine (%)" + "Calorie da Lipidi (%)" + "Calorie da Carboidrati (%)" + "Calorie da Alcool (%)" = 100
- "Acqua(g)"+"Proteine(g)"+"Lipidi(g)"+"Carboidrati disponibili(g)"<=100
- "Amido (g)" + "Zuccheri solubili (g)" <= "Carboidrati disponibili (g)"
- "Fibra insolubile (g)" + "Fibra solubile (g)" <= "Fibra totale (g)"

- “Rapporto Polinsaturi/Saturi” = “Polinsaturi totali(g)” / “Saturi totali (g)”
- “Monoinsaturi totali(g)” + “Polinsaturi totali (g)” + “Saturi totali(g)”<= “Lipidi (g)”
- “C4:0-C10:0” + “C12:0” + “C14:0” + “C16:0” + “C18:0” + “C20:0” + “C22:0” <= “Saturi totali (g)”
- “C14:1” + “C16:1” + “C18:1”+“C20:1”+“C22:1”<=“Monoinsaturi totali (g)”
- “C18:2”+“C18:3”+“C20:4” + “C20:5”+ “C22:6” <= “Polinsaturi totali (g)”

Molti di questi vincoli non vengono comunque implementati nel sistema finale sia perché lo renderebbero più lento, sia perché a causa delle approssimazioni alcuni vincoli potrebbero perdere di validità.

3.9 Analisi delle Ridondanze

Nello schema finale sono presenti alcune istanze dell’entità “Nutrienti” che possono sembrare ridondanti, come ad esempio "Calorie da proteine (%)", "Calorie da lipidi (%)", etc.

Non essendo comunque in possesso degli elementi necessari per stabilire se ciò accade per ogni possibile istanza dell’entità, si è deciso di non eliminare nel database finale di integrazione tali dati.

4. Fonte USDA

4.1 Il Database

La base di dati è in due formati: Access 2000 e ASCII, ed è accessibile dal sito http://www.nal.usda.gov/fnic/foodcomp.

Il database ha due semplici programmi di consultazione (entrambi gratuiti): un applicativo desktop per i sistemi operativi Windows o per personal digital assistant (PDA) e uno web based. E' presente una buona documentazione sia della base di dati che del sw allegato.

Il db (denominato “National Nutrient Database for Standard Reference, Release 16”) contiene informazioni per 125 componenti alimentari e 6661 alimenti consumati negli Stati Uniti.

Le informazioni presenti sono esclusivamente in inglese.

4.2 Il Software di Consultazione

Il software di consultazione web based, disponibile on line allo stesso sito del db, permette di fare una veloce ricerca per stringa tra i nomi degli alimenti e di visualizzarne in seguito la scheda di composizione.

Il programma di consultazione in ambiente Windows (e PDA) permette di fare la stessa ricerca per stringa, ma consente anche in seguito di restringere i risultati della ricerca scegliendo la sola categoria di cui si vogliono visualizzare i dati.

4.3 Stato Iniziale del Database

Nel diagramma seguente riportiamo lo schema concettuale iniziale del database, che si presenta già con una buona struttura.

La figura rappresenta, come al solito, lo schema concettuale ottenuto dal reverse engineering automatico fatto con il CASE PowerDesigner.

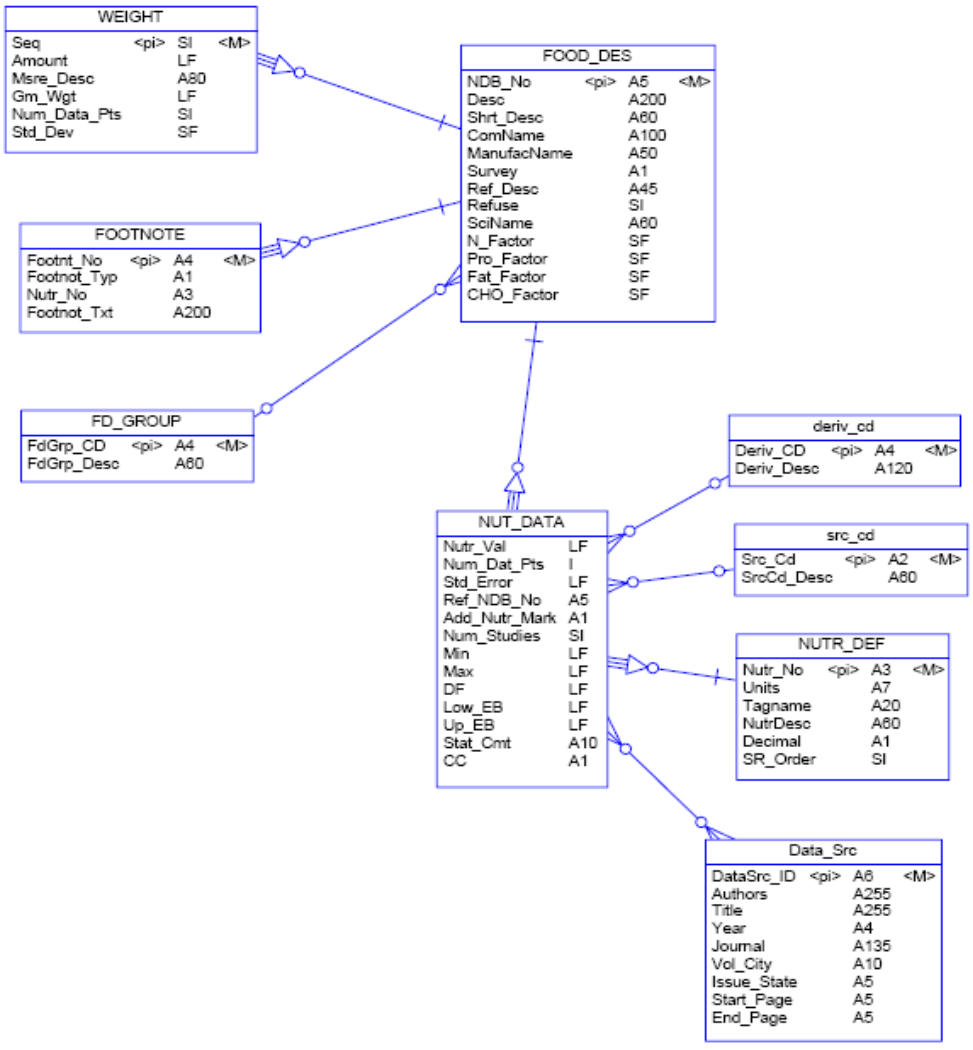


Figura 2 – Stato Iniziale Database della Fonte USDA

4.4 Tabelle Escluse dal Processo di Reverse Engineering

A parte le tabelle di sistema, l'unica altra tabella esclusa dal processo di reverse engineering (e quindi non presente nello schema precedente) è "ABBREV", che contiene, per ogni alimento, un sunto delle informazione più importanti presenti nel database (come in una vista materializzata).

4.5 Considerazioni sull'Uso del Database Come Fonte per il Datawarehousing

La base di dati è aggiornata annualmente e vengono forniti ad ogni aggiornamento dei file "differenziali" in formato ASCII per le tabelle più importanti: quindi siamo di fronte ad una discreta fonte di datawarehousing, almeno dal punto di vista degli aggiornamenti. Una nota negativa è però data dal fatto che, saltuariamente, con gli aumenti di versione cambia anche lo schema del db e non ci è dato sapere se questi cambianti si ripercuotono anche nei file ASCII di aggiornamento.

Servirebbe il parere di un esperto e forse maggiore documentazione per valutare la qualità dei dati (anche se per molti dati sono fornite anche le fonti secondarie e il metodo di derivazione del valore). Si ritiene comunque che, data l'autorevolezza della fonte, i dati siano di buona qualità.

Un aspetto negativo del database è che contiene dati su cibi consumati negli Stati Uniti, anche se comunque molti di questi alimenti (seppur con differenze a volte significative nei componenti) sono consumati anche in Italia.

4.6 Fonti di Dati USDA-NDL Non Presenti nel Database

Sul sito USDA-NDL si possono scaricare dei report, in formato pdf, fatti sulla base di dati: tali dati, essendo in realtà delle informazioni di riepilogo di quelli presenti nel database, sono ritenuti di scarso interesse.

Sempre sul sito è invece reperibile altro materiale compositivo riguardante studi più specifici su altri componenti alimentari e sui fattori di ritenzione degli stessi. Tali dati potrebbero essere integrati col db grazie anche all'ausilio di un esperto.

4.7 Ristrutturazione dello Schema Concettuale

Si fa notare che, sebbene i nomi degli attributi degli schemi non siano abbastanza significativi (sono stati lasciati i codici dei campi della base di dati), un'ampia spiegazione di tali nomi può essere trovata nella documentazione allegata al database.

Lo schema ER vero e proprio è nel diagramma "USDA Final Conceptual".

Durante la ristrutturazione sono state introdotte due nuove entità: “NUT_FOOTNOTE” e “Stat_Cmt”. La prima contiene delle note riguardanti uno specifico valore di un componente in un alimento, ed è un’entità che è stata divisa dalle note riguardanti il singolo alimento, che restano invece presenti nell’entità “FOOTNOTE”.

L’entità “Stat_Cmt” contiene invece dei commenti statistici sul valore: questi commenti erano presenti solo nella documentazione allegata al database, mentre in quest’ultimo c’era solo un attributo di “NUT_DATA” contenente un elenco, separato da virgole, dei codici dei commenti che si applicano al dato valore.

In fase di normalizzazione si è reso necessario creare degli identificativi per le nuove entità (NUT_FOOTNOTE e Stat_Cmt).

Altra modifica allo schema è stata quella di ricavare dai campi "doppi" (nel senso che, a seconda del contesto, contengono una tra due informazioni completamente differenti) “Vol_City” e “Issue_State” dell’entità “Data_Src” altri quattro campi: “Volume”, “City”, “Issue” e “State”. Questo aumenta di molto i valori NULL, ma rende sicuramente più leggibile lo schema.

Si noti anche che l’attributo "Footnt_Typ" dell’entità “FOOTNOTE” era a tre valori: “D” indicava nota sull'alimento, “M” indicava nota sulla misura dell'alimento, “N” indicava nota sul nutriente. Il valore “N” è stato eliminato dallo schema ristrutturato perché le note sull'alimento e sulla misura sono rimaste in FOOTNOTE, mentre quelle sul nutriente sono state spostate in NUT_FOOTNOTE.

mentre il diagramma "Campi Eliminati" riporta la lista degli attributi dello schema concettuale iniziale che, a causa del processo di ristrutturazione, non hanno un corrispettivo in nessun attributo del diagramma concettuale finale.

In "Campi Eliminati" sono presenti gli attributi dello schema concettuale iniziale, che, a causa del processo di ristrutturazione, non hanno un corrispettivo in nessun attributo del diagramma concettuale finale. Tra questi segnaliamo :

- "Survey": campo flag che indica se l'alimento parteciperà o meno al sondaggio "National Health and Nutrition Examination Survey: What We Eat in America". Eliminato perché di scarso interesse e perché ridondante (un alimento partecipa al sondaggio se ha non nulli determinati nutrienti).
- "Refuse": percentuale di parte non edibile dell'alimento. Eliminato perché è meglio individuarlo come un'occorrenza particolare dell'entità "NUT_DEF".
- "Stat_Cmt" e "FOOTNOTE.Nutr_No": eliminati perché sono stati individuati come entità a sé stanti.
- "SR_Order": campo utilizzato per ordinare i record dei nutrienti nello stesso ordine nei vari report pubblicati. Eliminato perché application dependent.
- "CC": Confidence Code, dovrebbe indicare la qualità del dato. Eliminato perché ha tutti valori NULL e perché sembra poter essere calcolato in base ad altri campi (come numero di data point, metodo, etc.).
- "Nutr_no": eliminato da FOOTNOTE perché ora c'è un'entità a parte (NUT_FOOTNOTE) per le note sui nutrienti.

Come sempre, tutti gli attributi inseriti ex novo non hanno definito un tipo (sono "<UNDEF>"), in quanto questa informazione verrà aggiunta in fase di integrazione nel solo diagramma di "unione" degli schemi di tutte le fonti.

4.7.1 Schema "USDA Final Conceptual"

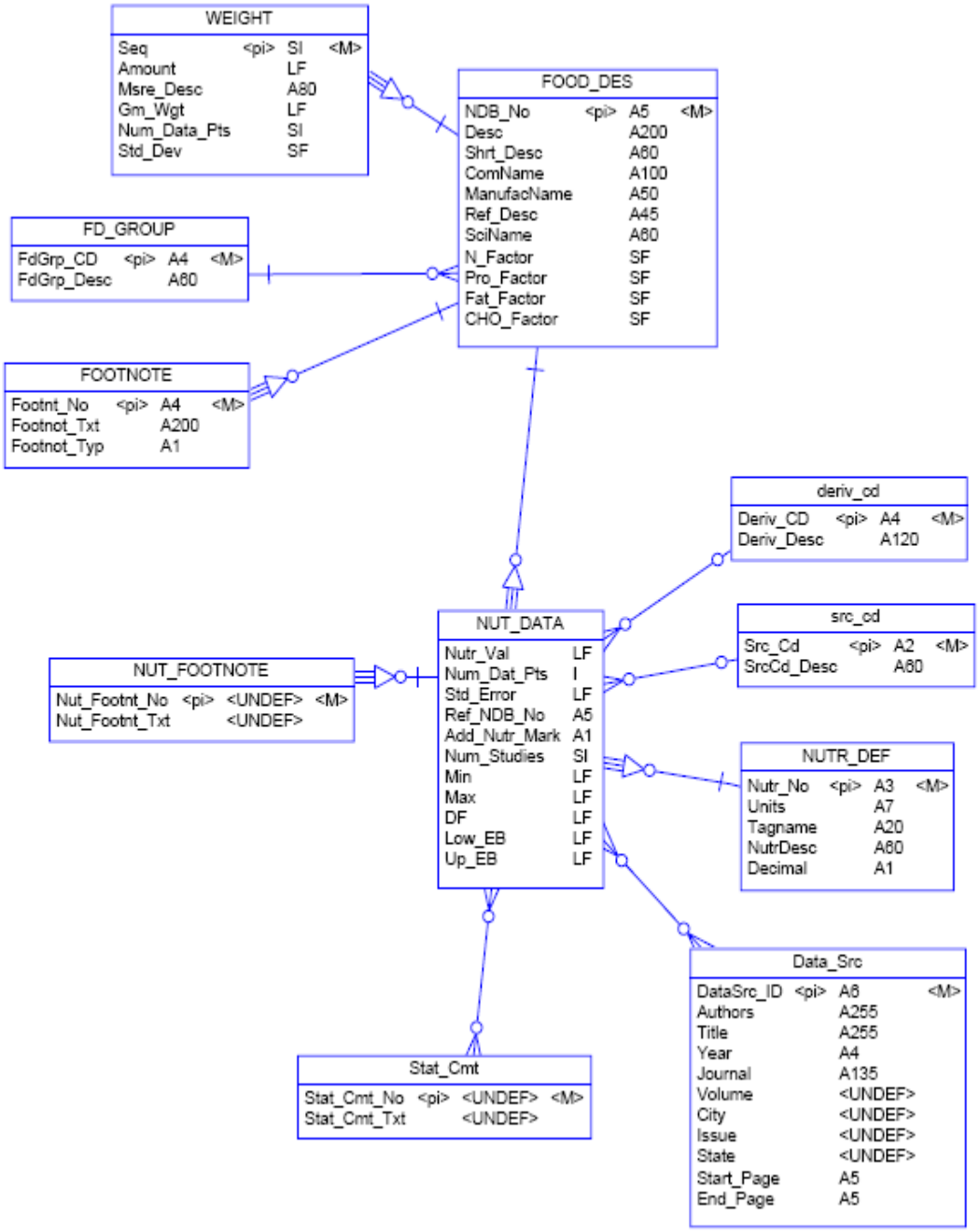


Figura 3 – USDA Final Conceptual

4.8 Problemi di Ristrutturazione Risolti Parzialmente

Gli attributi di testo "Deriv_Desc" e "SrcCd_Desc", che sono campi descriventi le metodologie utilizzate per ricavare un valore, fanno riferimento ai codici presenti in "Src_Cd".

Si dovrebbero “esplodere" tali codici presenti nei campi descrittivi (bisognerebbe cioè sostituire ai “Src_Cd” le “Src_Cd_Desc” corrispondenti), così da non dover dipendere da tali codici.

4.9 Ulteriori Vincoli

Un primo vincolo, non presentato nello schema finale solo per motivi di leggibilità del diagramma stesso, è rappresentato dal fatto che l’attributo “Ref_NDB_No” dell’entità “NUT_DATA” deve avere, se diverso da NULL, un corrispettivo nel campo “NDB_No” di “FOOD_DES”.
Esistono poi vincoli simili a quelli di INRAN e IEO tra i nutrienti (si vedano i paragrafi “Ulteriori Vincoli” relativi alle due fonti), che però vanno trovati e verificati con un esperto. Si fa comunque presente che nella documentazione USDA è scritto che, a causa delle approssimazioni, molti vincoli in cui dovrebbe comparire il simbolo di “minore o uguale” a volte non vengono rispettati.

4.10 Analisi delle Ridondanze

Nello schema finale non sembrano esserci dati ridondanti, anche se non è ben chiaro se il campo “Shrt_Desc” dell’entità “FOOD_DES” sia ricavato o meno in modo automatico dal campo “Desc” della stessa entità, cosa che potrebbe essere attuata utilizzando la lista di abbreviazioni presente nella documentazione allegata alla base di dati.

5 Fonte IEO

5.1 Il Database

La base di dati utilizzata nella fase di reverse engineering (scaricata dal sito http://www.ieo.it/) è in realtà un libro (dal titolo “Banca Dati di Composizione degli Alimenti per Studi Epidemiologici in Italia”) in formato pdf.
In questa fase è stata utilizzata la versione “cartacea” e non il file ASCII corrispondente per il semplice motivo che quest’ultima si è resa disponibile solo in seguito. Comunque, a parte delle lievi differenze illustrate nei paragrafi successivi, le due versioni sono pressoché coincidenti. La base di dati non sembra possedere programmi di consultazione.

Nella documentazione allegata (e sul sito web) si legge che il libro è stato pubblicato nel 1998 e contiene informazioni su 778 alimenti e 37 componenti alimentari: gli alimenti inclusi sono per lo più alimenti semplici, crudi e quelli più frequentemente consumati dalla popolazione italiana; inoltre, si apprende che la banca dati può essere definita "compilativa", nel senso che "i dati inseriti sono stati ricavati da fonti preesistenti e non da analisi eseguite ad hoc".

Sebbene una delle fonti principali del database siano le tabelle dell'INN (Istituto Nazionale della Nutrizione, oggi INRAN), cioè una delle altre sorgenti selezionate per far parte della base di dati finale, i dati IEO possiedono comunque caratteristiche molto interessanti, ad esempio la presenza di componenti assenti in INRAN e la piccola percentuale di valori NULL.

Le informazioni presenti nella banca dati sono in italiano e in buona parte anche in inglese (nel file ASCII però sono solo in italiano).

5.2 Stato Iniziale del Database

Come si evince dalla figura riportata di seguito, il "cuore" della banca dati è un insieme di tabelle di composizione, una per ogni alimento.

Composizione per 100g di parte edibile

AGNELLO [OVIS AGNUS] **codice 1060**

Categoria merceologica 10080

alimento interamente sostituito nota INN: pronto da cuocere, lo scarto e' costituito dalle ossa del busto

Componenti alimentari, unità	Valori	Fonte	Codice	Note e valori originali
Parte edibile, g	83	H		
Acqua, g	70.1	H		
Proteine totali, g	20.8	H		
Proteine animali, g	20.8	H		
Proteine vegetali, g	0.0	H		
Lipidi totali, g	8.8	H		
Lipidi animali, g	8.8	H		
Lipidi vegetali, g	0.0	H		
Acidi grassi:				
Saturi totali, g	4.15	02	APP	RIPR riferito a agnello (nota INN: valori medi relativi al totale della parte edibile) lip=30.5g saturi=14.4g 18:1=10.89g mono=11.26g 18:2=0.71g 18:3=0.71g altri PUFA=0 PUFA=1.42g
Acido oleico, g	3.14	02	APP	
Monoinsaturi totali, g	3.25	02	APP	
Acido linoleico, g	0.20	02	APP	
Acido linolenico, g	0.20	02	APP	
Altri polinsaturi, g	0.00	02	APP	
Polinsaturi totali, g	0.41	02	APP	
Colesterolo, mg	71	02	APP	riferito a Agnello parte edibile totale cruda
Glucidi Disponibili, g	0.0	H		
Amido, g	0.0	H		
Glucidi solubili, g	0.0	H		
Fibra alimentare, g	0.0	H		
Alcool, g	0.0	H		
Energia, kcal	162	H		
Energia, kJ	678	86		
Minerali:				
Ferro, mg	1.6	H		
Calcio, mg	7	H		
Sodio, mg	88	H		
Potassio, mg	350	H		
Fosforo, mg	190	H		
Zinco, mg	3.3	T	96	per tutti i nutrienti fonte T cod 96 ALIMENTO SIMILE Lamb, average, trimmed lean, raw acqua=70.6g, prot=20.2g, lip=8.3g
Vitamine:				
Tiamina, mg	0.14	H		
Riboflavina, mg	0.28	H		
Niacina, mg	6.0	H		
Vitamina C, mg	0	H		

Figura 4 – Stato Iniziale Database della Fonte IEO

5.3 Considerazioni sull'Uso del Database Come Fonte per il Datawarehousing

Come si deduce dalla documentazione, siamo di fronte ad una base di dati "statica", nel senso che non sembrano previsti ulteriori aggiornamenti. Servirebbe comunque il parere di un esperto per valutare la qualità dei dati, che al momento è ritenuta buona semplicemente basandosi sull'autorevolezza della fonte.

Anche questo database, al pari di quello INRAN, contiene dati su cibi effettivamente consumati in Italia, cosa importante per un suo utilizzo per ricerche a livello nazionale.

5.4 Fonti di Dati IEO Non Presenti nel Database

Sul sito IEO è presente un file .pdf con un errata corrige: i dati nel file ASCII hanno già apportate queste modifiche.

Oltre ai dati contenuti nelle tabelle di composizione, ci sono molti dati nelle appendici che possono essere perfettamente integrati col database. Ad esempio, in appendice sono presenti le descrizioni in inglese degli alimenti: tali informazioni sono assenti nel file ASCII e dovrebbero quindi essere inserite manualmente in un secondo momento.

5.5 Creazione dello Schema Concettuale

Di seguito riportiamo il diagramma concettuale finale, creato sulla base dell'esame di svariate tabelle di composizione del libro.

Lo schema ER è denominato "IEO Final Conceptual": in fase di creazione dello schema, i dati delle schede presenti nella versione pdf sono stati calati nella struttura relazionale mostrata nella figura successiva. La corrispondenza tra i campi delle schede e quelli del modello relazionale è abbastanza immediata e viene data una spiegazione solo per gli attributi il cui nome potrebbe non essere auto-esplicativo.

Innanzitutto si è reso necessario creare degli identificativi per le entità che non avevano codici nella fonte ("Nutrienti" e "Categorie").

Si nota inoltre che l'entità "Categorie non è altro che una classificazione meno specifica rispetto a quella fornita da "Categorie Merceologiche".

L'attributo "Codice Originale" di "Composizione" non è altro che l'informazione presente nella colonna "Codice" delle schede originali: indica il codice dell'alimento da cui è stato ricavato il valore in questione.

Altro attributo che merita un commento è "Nota" dell'entità "Alimenti": contiene il testo presente sotto la categoria merceologica nelle tabelle.
Infine è da notare che tutti gli attributi non hanno definito il tipo (sono "<UNDEF>"), in quanto questa informazione, ovviamente non presente nella fonte, verrà aggiunta in fase di integrazione nel solo diagramma di "unione" degli schemi di tutte le fonti.

5.5.1 Schema "IEO Final Conceptual"

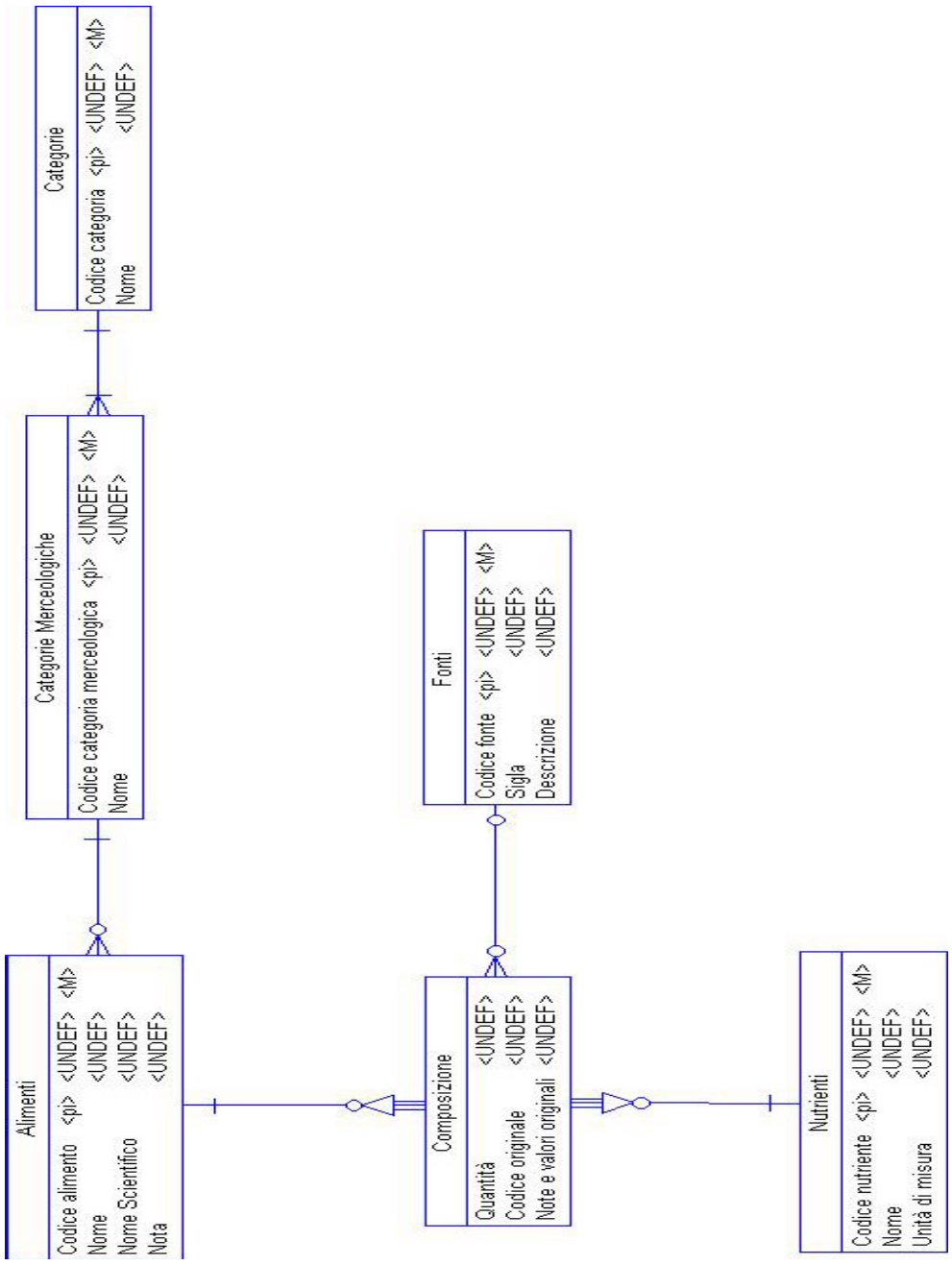


Figura 5 – IEO Final Conceptual

5.6 Informazioni Non Presenti nel File ASCII

Nel file ASCII utilizzato per fare il feeding della datawarehouse finale non sono presenti tutti gli attributi visti in precedenza. In particolare mancano l'attributo "Nota" di "Alimento" e i campi "Codice originale" e "Note e valori originali" di "Composizione".

Queste informazioni non sono state quindi inserite in modo automatico nella datawarehouse, anche se comunque sono state tenute in conto nella creazione dello schema della stessa, in quanto potrebbero essere aggiunte in un secondo momento.

5.7 Ulteriori Vincoli

I vincoli che seguono vengono riportati senza alcuna modifica alla scrittura originale:

- Somma degli acidi grassi saturi, monoinsaturi e polinsaturi minore dei lipidi totali.
- Acido oleico minore o uguale ai monoinsaturi totali.
- Somma di acido linoleico, linolenico e degli altri acidi grassi polinsaturi minore o uguale ai polinsaturi totali.
- Somma dei glucidi solubili e amido uguale ai glucidi disponibili.
- Somma delle proteine (o lipidi) vegetali ed animali, uguali alle proteine (o lipidi) totali.
- Retinolo equivalente uguale alla somma del retinolo più un sesto del ß-carotene.

I vincoli seguenti sono invece abbastanza banali e sono stati ricavati da una semplice analisi delle schede:

- "Parte edibile, g" <= 100.
- "Energia, kJ" = 4.184 * "Energia, kcal".
- "Acqua, g" + "Proteine totali, g" + "Lipidi totali, g" + "Glucidi disponibili, g" <= 100.

Come sempre, non tutti questi vincoli vengono in realtà implementati, principalmente per motivi di prestazioni. Trattandosi di una datawarehouse, infatti, risulta molto più conveniente gestire questi vincoli con controlli una tantum.

5.8 Analisi delle Ridondanze

L'unico dato che sembra essere ridondante è il nutriente "Retinolo equivalente" che, come si evince dal paragrafo "Ulteriori vincoli" (e ovviamente dalla documentazione allegata alle schede), è calcolato in base ad altri nutrienti.

6. Fonte ISA-CNR

6.1 Il Database

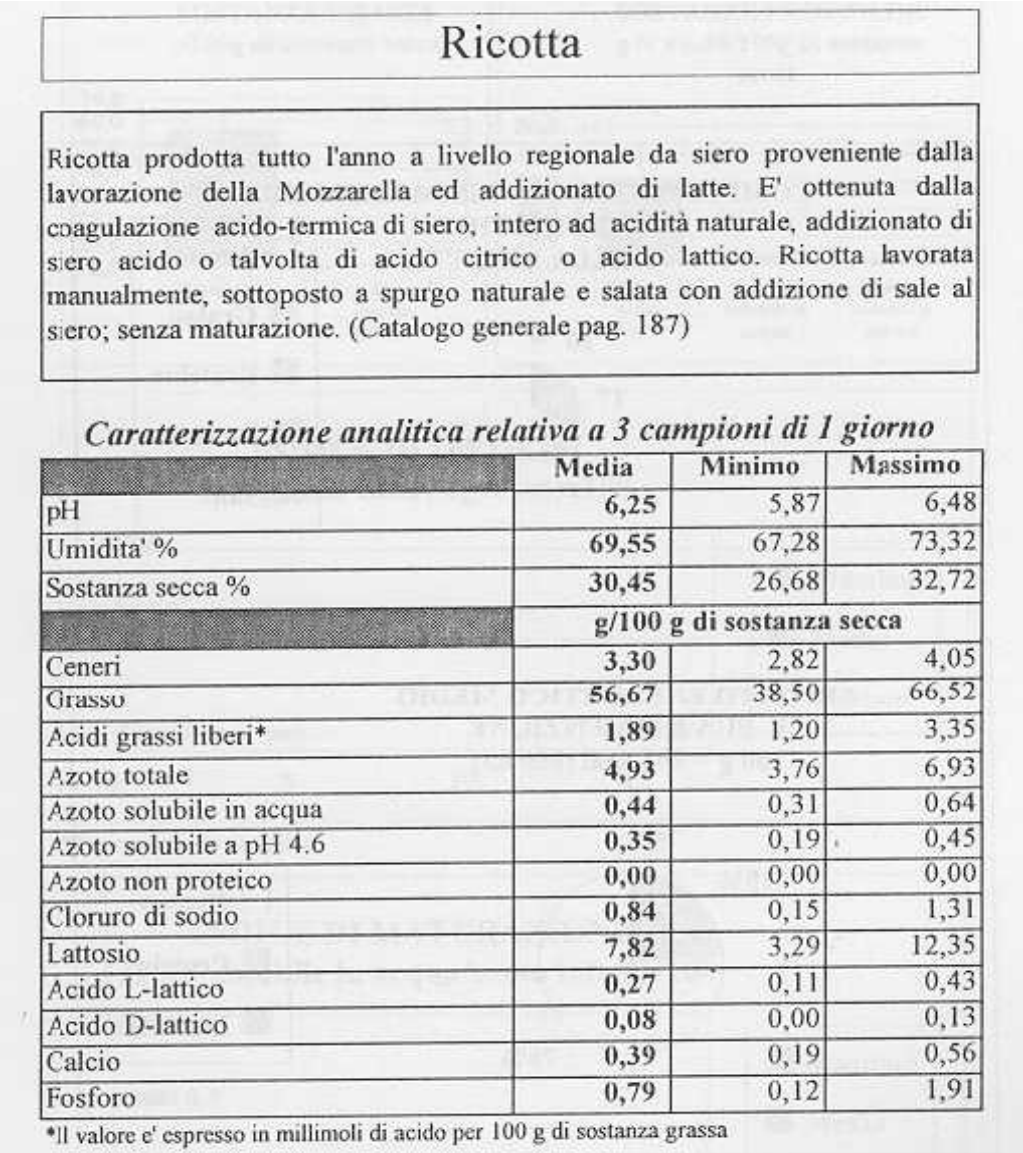
La base di dati (scaricata dal sito http://www.isa.cnr.it/PRODTIP/) è in realtà una ricerca cartacea di cui viene distribuita gratuitamente una scansione in immagini jpg: non è presente alcun programma di ricerca, eccettuato il sito tramite il quale si possono richiamare le suddette immagini.

Nel database sono presenti informazioni nutrizionali su prodotti caseari campani: sono disponibili sia varie descrizioni, sia quattro pagine di composizione chimica. Di queste ultime le prime tre sono di composizione nutrizionale, mentre la quarta contiene il ciclo di produzione dell'alimento. Nel seguito saranno quindi considerati solo i dati presenti nelle tre pagine in questione.

Le informazioni presenti sono esclusivamente in italiano.

6.2 Stato Iniziale del Database

Nell'immagine successiva si vede un esempio di scheda di un singolo alimento. Da tale lavoro cartaceo è stato ricavato lo schema ER ristrutturato.



Ricotta

Ricotta prodotta tutto l'anno a livello regionale da siero proveniente dalla lavorazione della Mozzarella ed addizionato di latte. E' ottenuta dalla coagulazione acido-termica di siero, intero ad acidità naturale, addizionato di siero acido o talvolta di acido citrico o acido lattico. Ricotta lavorata manualmente, sottoposto a spurgo naturale e salata con addizione di sale al siero; senza maturazione. (Catalogo generale pag. 187)

Caratterizzazione analitica relativa a 3 campioni di 1 giorno

	Media	Minimo	Massimo
pH	6,25	5,87	6,48
Umidita' %	69,55	67,28	73,32
Sostanza secca %	30,45	26,68	32,72
	g/100 g di sostanza secca		
Ceneri	3,30	2,82	4,05
Grasso	56,67	38,50	66,52
Acidi grassi liberi*	1,89	1,20	3,35
Azoto totale	4,93	3,76	6,93
Azoto solubile in acqua	0,44	0,31	0,64
Azoto solubile a pH 4.6	0,35	0,19	0,45
Azoto non proteico	0,00	0,00	0,00
Cloruro di sodio	0,84	0,15	1,31
Lattosio	7,82	3,29	12,35
Acido L-lattico	0,27	0,11	0,43
Acido D-lattico	0,08	0,00	0,13
Calcio	0,39	0,19	0,56
Fosforo	0,79	0,12	1,91

*Il valore e' espresso in millimoli di acido per 100 g di sostanza grassa

Figura 6 – Stato Iniziale Database della Fonte ISA-CNR

6.3 Considerazioni sull'Uso del Database Come Fonte per il Datawarehousing

I dati sono presentati come un lavoro esclusivamente cartaceo e non sembrano essere previsti aggiornamenti periodici. Quindi non ha senso costruire un modulo di feeding automatico per tale sorgente.

Servirebbe il parere di un esperto (e forse maggiore documentazione) per valutare la qualità dei dati. Si ritiene comunque che, data l'autorevolezza della fonte, i dati siano di buona qualità.

Un aspetto certamente positivo del database è che contiene dati su cibi effettivamente prodotti in Campania, caratterizzando così un'importante realtà locale, gli alimenti considerati vengono comunque ampiamente consumati anche a livello nazionale, cosa che dà alla fonte ancora maggior rilievo.

6.4 Creazione dello Schema Concettuale

Di seguito riportiamo il diagramma concettuale ristrutturato, creato sulla base dell'analisi delle schede cartacee di diversi alimenti.

Lo schema ER è denominato "ISA-CNR Final Conceptual": durante la fase di creazione di quest'ultimo, i dati delle immagini prelevate dal sito sono stati calati nella struttura relazionale mostrata nella figura successiva.

L'entità "Alimento" contiene informazioni prelevate dalla prima immagine (o pagina) relativa a ogni cibo: l'attributo "Nome alimento" è il titolo della prima pagina, mentre "Nota" è il testo immediatamente successivo. I campi "Numero campioni" ed "Età campioni" contengono poi i dati presenti nel rigo immediatamente precedente alla tabella compositiva di pagina uno.

In fase di normalizzazione si è reso poi necessario creare degli identificativi per tutte le entità create ("Alimenti" e "Nutrienti").

L'entità "Nutrienti" contiene informazioni generiche (valide cioè per qualsiasi alimento), il campo "Unità di misura" dice l'unità (es. "g", "millimoli") con cui è espresso il nutriente, mentre il campo "Modo di espressione della misura" dice la quantità di alimento rispetto alla quale si riferisce la misura (es. "100g di parte edibile", "100g di sostanza secca", etc.).

L'associazione "Composizione" contiene invece informazioni sullo specifico valore: oltre alla media, sono presenti anche due campi per il minimo e il massimo, che però non sono riempiti per tutti i valori, ma solo per quelli della prima pagina.

Infine è da notare che, anche in questo caso, tutti gli attributi non hanno definito il tipo (sono "<UNDEF>"), in quanto questa informazione, ovviamente non presente nella fonte, verrà aggiunta in fase di integrazione nel solo diagramma di "unione" degli schemi di tutte le fonti.

6.4.1 Schema "ISA-CNR Final Conceptual"

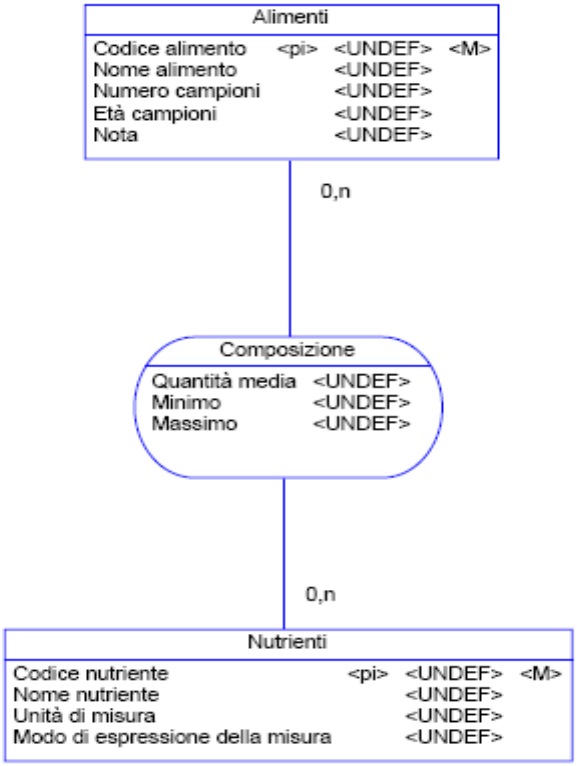


Figura 7 – ISA-CNR Final Conceptual

6.5 Analisi delle Ridondanze

Nello schema finale sono presenti alcune istanze dell'entità "Nutrienti" che possono sembrare ridondanti, come ad esempio le percentuali di calorie (da proteine, da grasso, etc.), le sostanze proteiche, l'indice di lipolisi e l'indice di maturazione.

Non essendo comunque in possesso degli elementi necessari per stabilire se ciò accade per ogni possibile istanza dell'entità, si è deciso di non eliminare nel database finale di integrazione tali dati.

7. Il DataBase di Integrazione

7.1 Le Entità Fondamentali

Dall'analisi degli schemi ristrutturati relativi alle quattro fonti (quelli che nei capitoli precedenti sono stati denominati "NOME_FONTE Final Conceptual") si ricava facilmente che le entità principali in gioco sono "Alimento", "Componente" e "Valore".

Affianco a queste va ovviamente inserita una entità creata appositamente per tenere traccia della sorgente del singolo dato: "Fonte primaria".

Insieme, questi quattro pezzi di informazione esprimono il concetto basilare dello schema, che si può così brevemente riassumere: "L'alimento X, secondo la fonte primaria Y, ha un valore Z del componente W". In prima battuta infatti le restanti informazioni (es. categorie, note, etc.) possono essere considerate come semplice metadatazione di una delle suddette entità fondamentali.

Si noti poi che i nomi che vengono dati alle entità in questo schema integrato sono stati assegnati seguendo le definizioni al capitolo uno, mentre nei singoli schemi ristrutturati delle fonti tali nomi possono differire. Questo perché si è cercato, in fase di reverse engineering, di non discostarsi troppo dalle denominazioni degli schemi originali e/o della documentazione allegata agli stessi.

Per attuare le doverose corrispondenze tra lo schema di questo capitolo e quelli dei precedenti si tenga quindi in conto che quello che qui viene chiamato "Componente" è sinonimo di "Nutriente" ("NUT_DEF" in USDA), mentre "Valore" ha potuto assumere in precedenza il nome "Composizione" ("NUT_DATA" in USDA).

Per non generare confusione, nel grafico vengono presentati i soli attributi essenziali. Questo diagramma è una semplice base su cui viene poi costruito il ben più complesso schema finale.

Nella figura si possono notare anche le associazioni tra le entità: due di queste ("Origine alimento" e "Origine Componente") contengono anche degli attributi, che indicano rispettivamente il codice che avevano nella fonte primaria l'alimento e il componente.

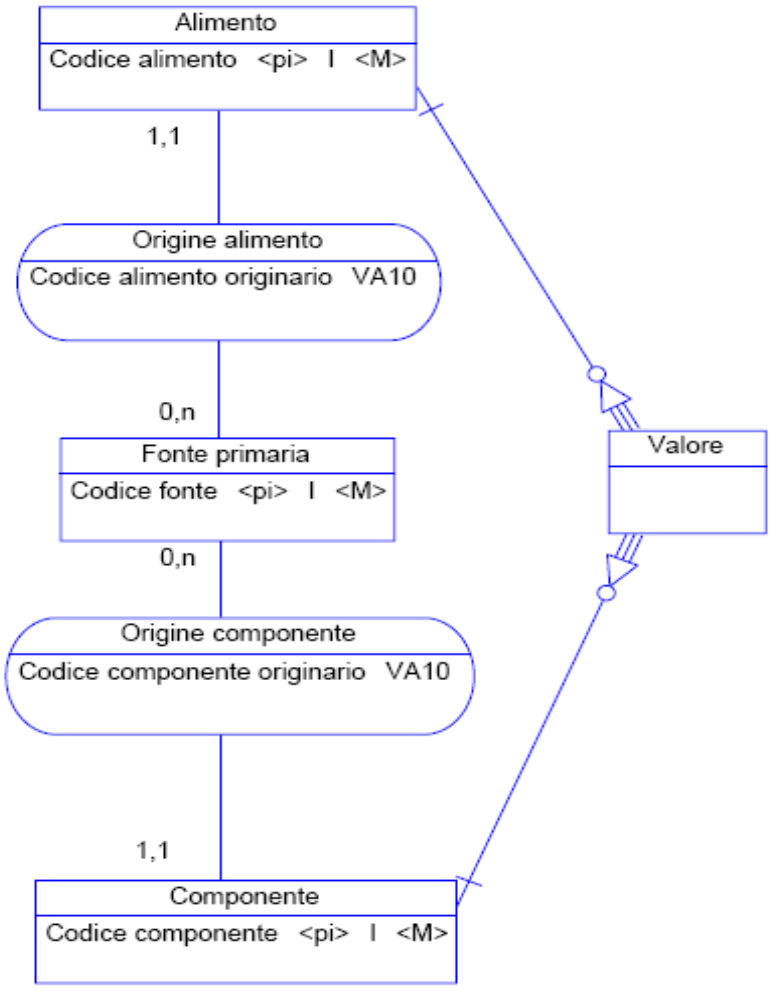


Figura 8 – Entità Fondamentali

7.2 Le Altre Entità

Sono state poi aggiunte tutte le entità atte ad ospitare le informazioni presenti nei vari schemi ristrutturati delle singole fonti.

La figura seguente mostra molto bene come ognuna di queste ulteriori informazioni vada a specificare meglio i dati di una o più entità "fondamentali". A differenza di queste ultime, presenti in tutte le fonti, non tutte le altre entità contengono informazioni provenienti da tutte le sorgenti di dati selezionate: molte anzi provengono dalla sola fonte USDA.

Come già anticipato nelle linee guida presentate nel capitolo uno, si è cercato di non perdere nessuna informazione riguardo ai database iniziali, pur sapendo che questo comporterà attributi con molti valori NULL.

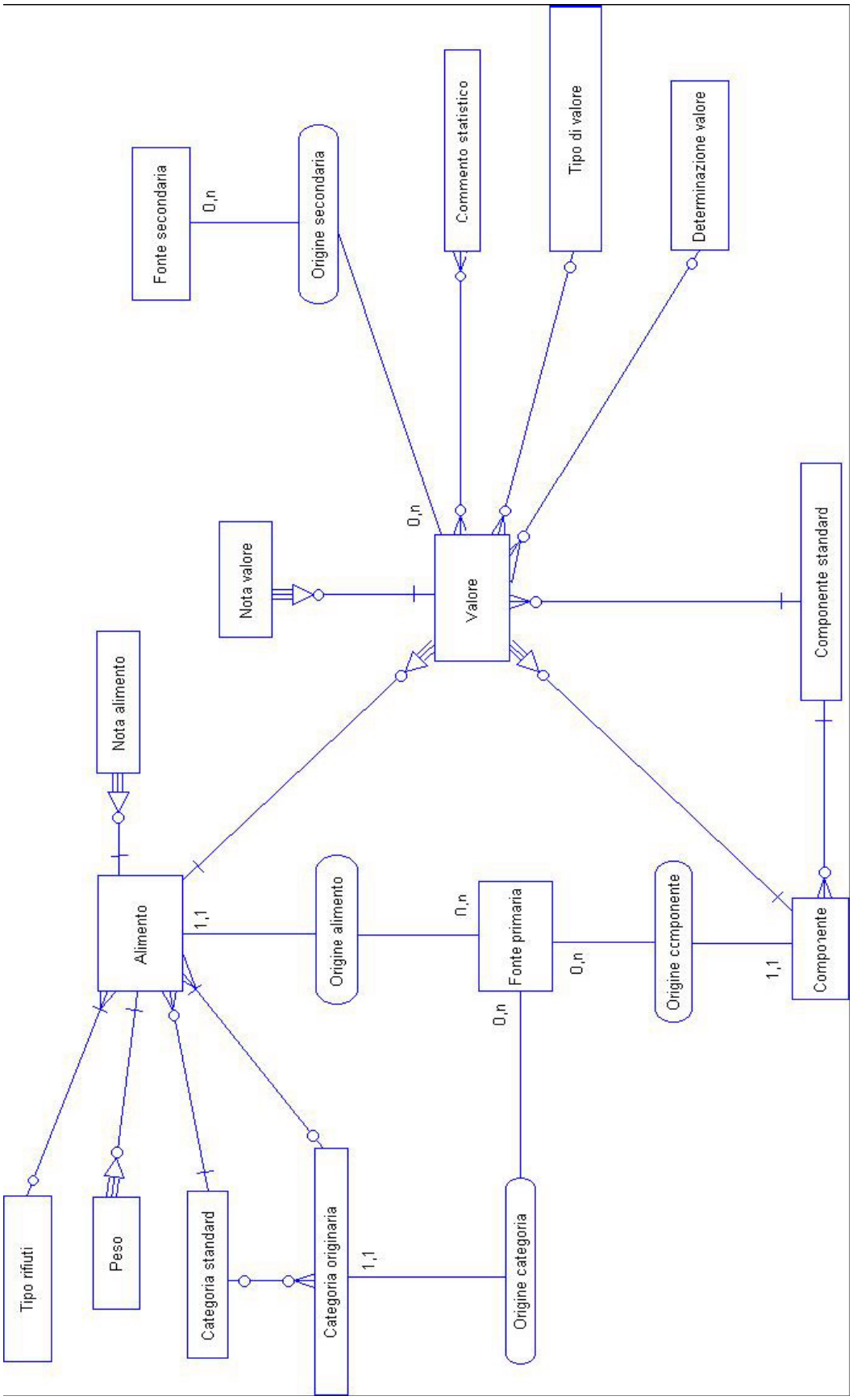


Figura 9 – Altre Entità

7.3 Le Entità "Standard"

Occorre notare inoltre che, nella figura precedente, sono state introdotte due entità i cui dati non provengono da nessuna fonte: "Categoria standard" e "Componente standard". La presenza di questi due nuovi concetti è dovuta alla necessità di confrontare i dati delle singole fonti.

Ci si ricorderà infatti che ogni fonte porta con sé i propri componenti e le proprie categorie di alimenti: questi vengono quindi visti come diversi anche quando in realtà esprimono informazioni molto simili (es. un software che consulta il database assume come due componenti distinti le proteine provenienti da due fonti differenti). Per permettere i confronti sono quindi state introdotte delle semplici etichette aggiuntive nella forma delle due entità "standard" (es. i componenti "proteine" di due fonti diverse sono entrambi associati allo stesso componente standard).

Gli standard utilizzati sono la classificazione dei cibi "Eurocode 99/2" (solo i gruppi principali) e la denominazione dei componenti del vocabolario standardizzato "COST Action 99 - EUROFOODS.

Bisogna far notare che, nell'ambito nutrizionale, non esistono standard universalmente adottati e quindi quelli utilizzati sono stati scelti principalmente per la loro flessibilità e facilità di utilizzo: un'alternativa al vocabolario EUROFOODS sarebbero stati infatti i cosiddetti tagname INFOODS, che però risultano molto meno pratici da gestire.

In ogni caso si tenga presente che le corrispondenze tra componenti originari e standard (e tra categorie originarie e standard) sono del tutto sperimentali, in quando non sono state ancora validate da un esperto. Le suddette correlazioni sono comunque facilmente modificabili e si incoraggiano anzi gli utilizzatori dei dati a segnalare prontamente qualsiasi errore presente.

Data la provvisorietà di queste informazioni, esse non vanno assolutamente utilizzate per scambiare informazioni tra questo database e altri. Qualsiasi software che si appoggi sulla base di dati dovrebbe limitare l'utilizzo degli "standard" al solo loro scopo iniziale, cioè quello di semplice confronto (sperimentale) tra dati di fonti diverse.

Va detto poi che il vocabolario EUROFOODS è stato anche opportunamente allargato per comprendere alcuni componenti presenti nelle fonti ma non nello standard.

7.4 Il Supporto Multilingua

Sono state infine aggiunte molte entità (praticamente tutte quelle il cui nome inizia per "Descrizione") per fornire un supporto multilingua (inizialmente solo italiano e inglese).

Ogni volta che ci si è trovati di fronte ad un testo che si voleva presente nel database in più lingue, si è scorporato quest'ultimo in un'ulteriore entità che, oltre ad un identificativo univoco, porti anche l'indicazione della lingua della descrizione in questione.

Data la somiglianza tra le varie entità "descrizione", queste sono state organizzate in una gerarchia, di cui comunque l'entità padre (denominata "Descrizione generica") non viene effettivamente tradotta in una tabella dello schema logico derivato dallo schema ER.

Nelle due figure che seguono viene quindi presentata la struttura finale del database: nella prima vi è lo schema ER principale e nella seconda la semplice gerarchia delle descrizioni.

Si fa infine presente che, durante tutto il processo di creazione dello schema, si è cercato di tener conto delle raccomandazioni EUROFOODS, a loro volta basate su varie raccomandazioni INFOODS.

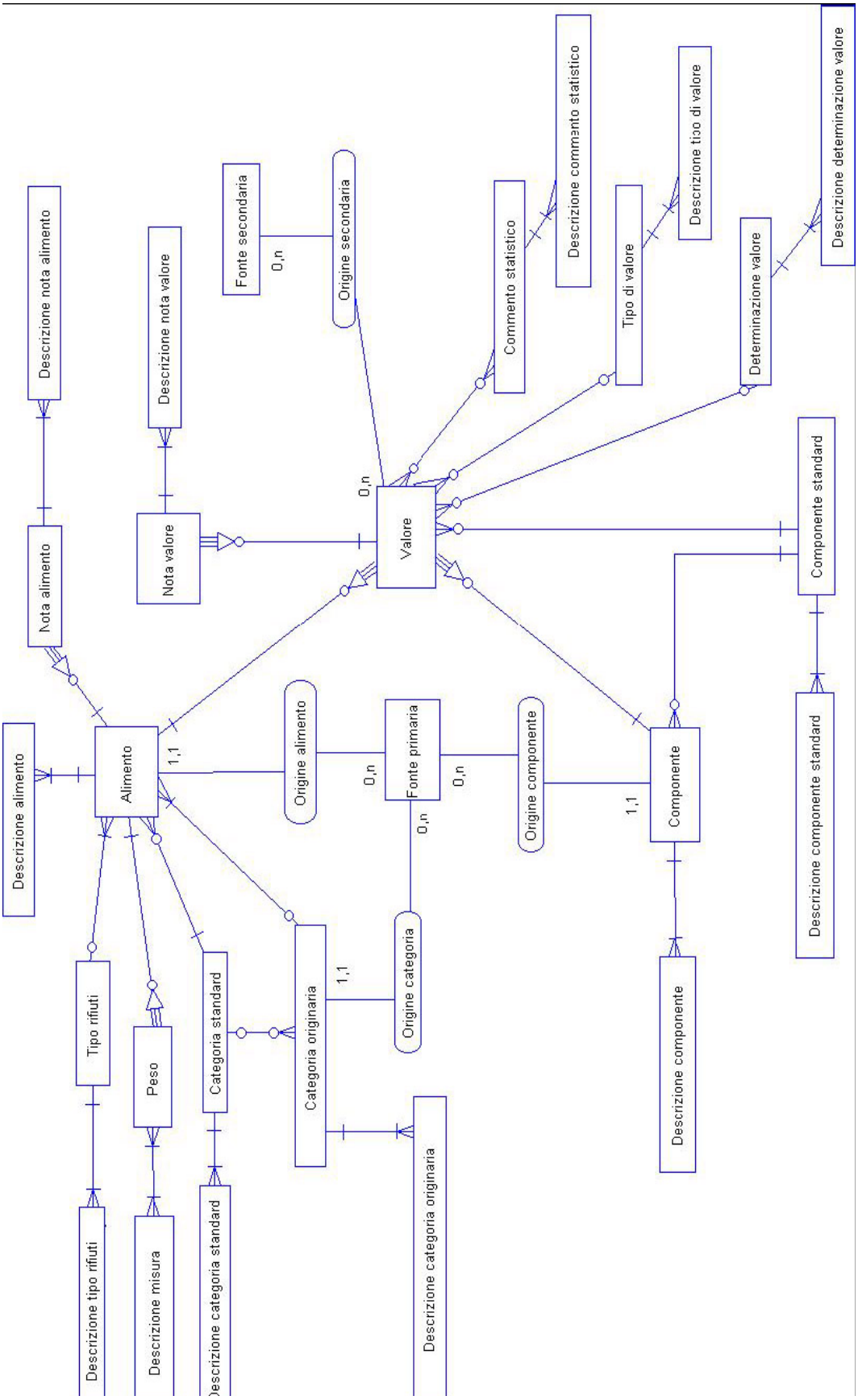


Figura 10 – Schema ER Principale

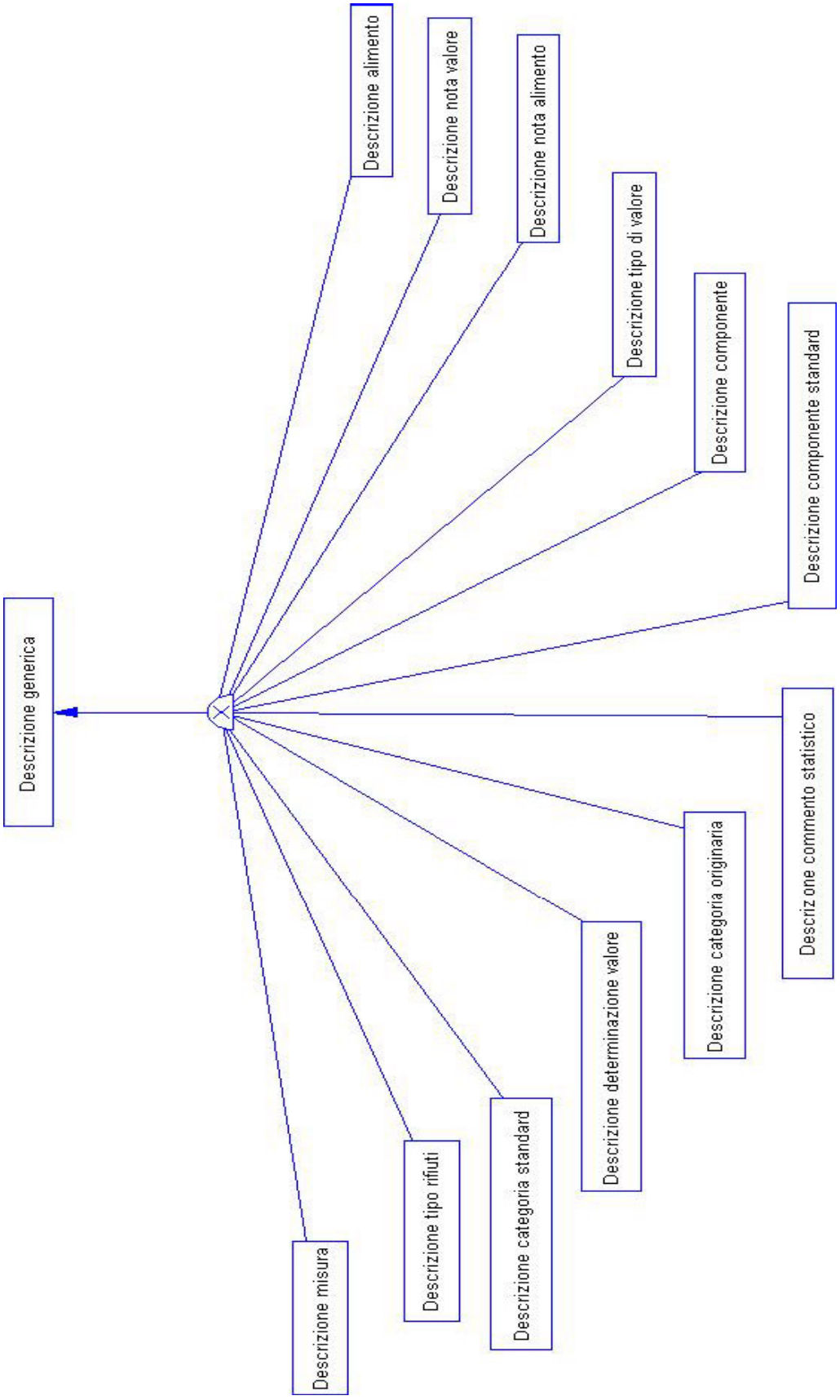


Figura 11 – Correlazione Descrizioni

7.5 Dizionario dei Dati

Presentiamo ora delle piccole schede delle singole entità. Per ognuna si dà una breve descrizione e, laddove necessario, delle spiegazioni aggiuntive sugli attributi.

7.5.1 Alimento

Alimento			
Codice alimento	<pi>	I	<M>
Marca		VA100	
Nome scientifico		VA150	
N Factor		SF	
Pro Factor		SF	
Fat Factor		SF	
CHO Factor		SF	
Aminoacido limitante		VA15	

Figura 12 – Alimento

Contiene informazioni generali sull'alimento:

- Marca: nome del produttore dell'alimento.
- N Factor: fattore utilizzato per convertire l'azoto in proteine.
- Pro Factor: fattore utilizzato per calcolare le calorie da proteine.
- Fat Factor: fattore utilizzato per calcolare le calorie da grassi.
- CHO Factor: fattore utilizzato per calcolare le calorie da carboidrati.
- Aminoacido limitante: campo presente nella sola fonte INRAN e che non è ben documentato.

7.5.2 Categoria originaria

Categoria originaria			
Codice categoria originaria	<pi>	I	<M>
Categoria standard corrispondente		I	

Figura 13 – Categoria Originaria

Contiene informazioni sulla categoria utilizzata nella fonte primaria:

- Codice categoria originaria: codice utilizzato in questo database per identificare univocamente la categoria originaria.
- Categoria standard corrispondente: eventuale categoria standard in cui la categoria originaria in esame è inclusa.

7.5.3 Categoria standard

Categoria standard			
Codice categoria standard	<pi>	I	<M>

Figura 14 – Categoria Standard

7.5.4 Commento statistico

Commento statistico			
Codice commento statistico	<pi>	I	<M>
Codice originario commento statistico		I	

Figura 15 – Commento Statistico

Informazioni sui commenti statistici riguardanti i valori.

Tale entità proviene al momento dalla sola fonte.

- Codice originario commento statistico: codice che identifica il commento nella fonte primaria.

7.5.5 Componente

Componente			
Codice componente	<pi>	I	<M>
Unità di misura		VA10	<M>
Modo di espressione misura		VA10	<M>
Cifre decimali		SI	
Tagname		VA20	

Figura 16 – Componente

Informazioni sul componente preso dalla fonte primaria:

- Unità di misura: unità con cui è espressa la misura (es. g, mg, etc.).
- Modo di espressione misura: quantità di cibo rispetto alla quale si riferisce la misura (es. 100g di parte edibile, 100g di sostanza secca, etc.).
- Cifre decimali: numero di cifre decimali alle quali è arrotondato il valore del nutriente. Al momento è disponibile per la sola fonte USDA.
- Tagname: tagname utilizzato dall' “International Network of Food Data Systems” (INFOODS): un abbreviazione univoca sviluppata da INFOODS per aiutare nello scambio di dati. Al momento è presente per i soli componenti della fonte USDA ed è riportato così come era nella fonte.

7.5.6 Componente standard

Componente standard			
Sigla componente standard	<pi>	VA12	<M>

Figura 17 – Componente Standard

Contiene le sigle dei componenti del vocabolario standardizzato "COST Action 99 - EUROFOODS". Sono stati aggiunte delle sigle (e le corrispondenti descrizioni) per i componenti presenti nelle fonti ma non nel vocabolario.

7.5.7 Determinazione valore

Determinazione valore			
Codice determinazione valore	<pi>	I	<M>
Codice originario determinazione valore		VA5	

Figura 18 – Determinazione Valore

Informazioni su come il valore è stato ricavato (al momento proviene solo da USDA e corrisponde al suo “Data Derivation Code File”):

- Codice originario determinazione valore: codice utilizzato nella fonte primaria (“DERIV_CD” in USDA).

7.5.8 Fonte primaria

Fonte primaria			
Codice fonte	<pi>	I	<M>
Descrizione fonte		VA150	<M>
Ente		VA150	
Sigla ente		VA20	
Indirizzo web		VA100	
Data pubblicazione fonte		VA10	

Figura 19 – Fonte Primaria

Informazioni sulle fonti da cui vengono direttamente presi i dati inseriti nel database:

- Descrizione fonte: denominazione della fonte. Non è vista come entità a parte perché questa informazione viene riportata come è presentata nella fonte stessa, quindi solo in lingua originale.
- Ente: ente fornitore dei dati.
- Sigla ente: eventuale acronimo dell’ente.

- Indirizzo web: url da cui si possono reperire i dati originali e/o informazioni aggiuntive su di essi.
- Data pubblicazione fonte: data della pubblicazione delle fonte da parte dell'organizzazione competente.

7.5.9 Nota valore

Nota valore			
Progressivo nota valore	<pi>	I	<M>
Codice originario nota valore		VA5	

Figura 20 – Nota Valore

Informazioni sulla nota relativa al valore.

- Codice originario nota valore: codice (o numero di sequenza) della nota nella fonte primaria.

7.5.10 Fonte secondaria

Fonte secondaria			
Codice fonte secondaria	<pi>	I	<M>
Codice originario fonte secondaria		VA10	
Sigla originaria fonte secondaria		VA20	
Autori		VA254	
Titolo		VA254	<M>
Editore		VA50	
Anno		SI	
Nome rivista		VA100	
Numero rivista		VA10	
Volume		VA10	
Città		VA20	
Stato		VA20	
Pagina iniziale		SI	
Pagina finale		SI	

Figura 21 – Fonte Secondaria

Informazioni sulle fonti secondarie, cioè le fonti utilizzate dalle fonti primarie per compilare i rispettivi database:

- Codice originario fonte secondaria: codice utilizzato nella fonte primaria per designare univocamente la fonte secondaria.
- Sigla originaria fonte secondaria: sigla utilizzata nella fonte primaria per descrivere la fonte secondaria. Il campo è presente solo nei pdf IEO e quindi al momento non è riempito.
- Autori: autori del documento o organizzazione sponsorizzante lo stesso.
- Anno: anno di pubblicazione del documento.
- Volume: volume del libro o dell'articolo di giornale (in inglese: volume).
- Numero rivista: numero di rivista in cui compare l'articolo (in inglese: issue).
- Città: città dove risiede l'organizzazione sponsorizzante (o l'editore).
- Stato: stato dove risiede l'organizzazione sponsorizzante (o l'editore).

7.5.11 Tipo rifiuti

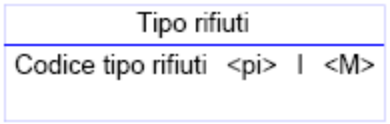


Figura 22 – Fonte Secondaria

Informazioni sul tipo di scarti (parte non edibile) dell'alimento (es. “ossa”, “semi”, etc.). Al momento questa entità è solo USDA.

7.5.12 Nota alimento

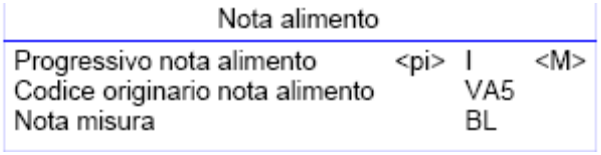


Figura 23 – Fonte Secondaria

Informazioni sulla nota relativa all'alimento.

- Codice originario nota alimento: codice (o numero di sequenza) della nota nella fonte primaria.
- Nota misura: campo booleano che, se alto, indica che la nota si riferisce ad una misura dell'alimento (si veda l'entità “Peso”). Campo solo USDA.

7.5.13 Peso

Peso			
Progressivo peso	<pi>	NO	<M>
Progressivo peso originario		VA5	
Quantità pesata		SF	<M>
Peso in grammi		SF	<M>
No of data points		I	
Standard Deviation		SF	

Figura 24 – Peso Alimento

Informazioni sul peso in grammi di misure comuni (cioè utilizzate nella vita quotidiana, come ad esempio "una tazza" o "un cucchiaio") dell'alimento. Le misure sono fatte considerando la sola parte edibile del cibo in questione.
Questa entità al momento proviene solo da USDA.

- Progressivo peso originario: progressivo (o codice) utilizzato nella fonte primaria per designare univocamente il peso.
- Quantità pesata: modificatore della quantità pesata (es. se è “0,5” e la misura è "tazza", allora è stato misurato il peso del contenuto di mezza tazza).
- Peso in grammi: media del peso in grammi risultante dalla misura della quantità comune (es. peso del latte presente in una tazza).
- No of data points: numero n di osservazioni che hanno contribuito al calcolo della media.
- Standard deviation: deviazione standard della media.

7.5.14 Valore

Valore		
Valore	SF	<M>
Tracce	BL	
Standard error	SF	
Min	SF	
Max	SF	
No of data points	I	
Number of studies	I	
Degrees of freedom	SF	
Lower 95% error bound	SF	
Upper 95% error bound	SF	
Alimento di riferimento	I	
Nutriente arricchito	BL	
Età data points(ore)	I	

Figura 25 – Componente dell'Alimento

Informazioni sul valore di un dato componente in un dato alimento:

- Valore: quantità media del componente presente nell'alimento.
- Tracce: flag che indica, nel caso in cui sia alto, che il componente è presente in tracce. (campo solo INRAN e IEO)
- Standard error: errore standard della media. (campo solo USDA)
- Min: valore minimo osservato. (campo solo USDA e ISA-CNR)
- Max: valore massimo osservato. (campo solo USDA e ISA-CNR)
- No of data points: numero n di osservazioni del cibo utilizzate per calcolare i valori statistici. (campo solo USDA e ISA-CNR)
- Number of studies: numero di studi analitici adoperati per calcolare la media. (campo solo USDA)
- Degrees of freedom: Numero di valori che sono liberi di variare dopo che si sono fissate certe restrizioni sui dati. (campo solo USDA)
- Lower 95% error bound: valore al di sopra del quale ci si aspetta ricada la media, con livello di confidenza del 95%. (campo solo USDA)
- Upper 95% error bound: valore al di sotto del quale ci si aspetta ricada la media, con livello di confidenza del 95%. (campo solo USDA)
- Alimento di riferimento: codice dell'alimento che è stato usato per imputare il valore del componente all'alimento. (campo solo USDA)
- Nutriente arricchito: flag che, se alto, indica che il nutriente in questione è stato in parte o totalmente aggiunto artificialmente. (campo solo USDA)
- Età data points (ore): età (in ore) dei data points considerati. (campo solo ISA-CNR).

7.5.15 Tipo di valore

Tipo di valore			
Codice tipo di valore	<pi>	I	<M>
Codice originario tipo di valore		VA3	

Figura 26 – Tipo di Valore

Informazioni sul tipo di valore(solo da USDA) e corrisponde al "Source Code File"

- Codice originario tipo di valore: codice utilizzato nella fonte primaria ("SRC_CD" in USDA).

7.5.16 Descrizione generica

Descrizione generica		
Lingua	A2	<M>

Figura 27 – Descrizione Generica

Descrizione generica: tutte le altre descrizioni derivano gli attributi di questa entità, che comunque non ha un corrispettivo nello schema logico.

- Lingua: sigla ISO 639:1998 della lingua in cui è espressa la descrizione.

7.5.17 Descrizione alimento

Descrizione alimento			
Progressivo descrizione alimento	<pi>	NO	<M>
Descrizione alimento		VA250	<M>
Descrizione breve		VA70	
Nomi comuni		VA200	

Figura 28 – Descrizione Alimento

Contiene le denominazioni dell'alimento:

- Descrizione breve: descrizione con al più 70 caratteri; coincide con la descrizione normale se quest'ultima è minore di 70 caratteri (al momento proviene solo da USDA).
- Nomi comuni: altri nomi comunemente usati per descrivere l'alimento.

7.5.18 Altre Descrizioni

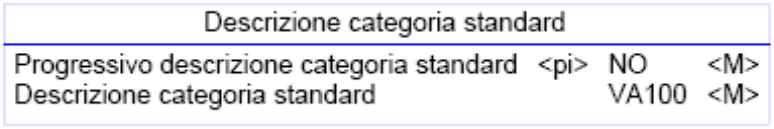


Figura 29 – Altre Descrizioni

Tutte le altre entità descrittive contengono un campo identificativo (dal nome "Progressivo descrizione NOME_ENTITA'") e un campo con la descrizione vera e propria (denominato "Descrizione NOME_ENTITA'").

7.6 Informazioni sulle Singole Associazioni

Le schede mostrate in questo paragrafo riguardano le sole associazioni che presentano degli attributi.

7.6.1 Origine alimento

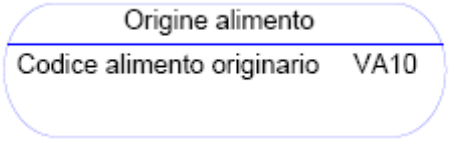


Figura 30 – Origine dell'Alimento

Lega le entità "Fonte primaria" e "Alimento":

- Codice alimento originario: codice utilizzato nella fonte primaria per identificare univocamente l'alimento.

7.6.2 Origine categoria

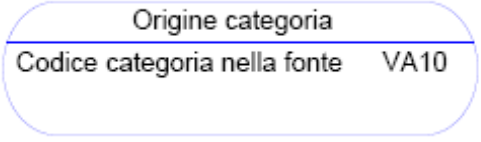


Figura 31 – Origine Categoria

Lega le entità "Fonte primaria" e "Categoria originaria":

- Codice categoria nella fonte: codice utilizzato nella fonte primaria per identificare univocamente la categoria.

7.6.3 Origine componente

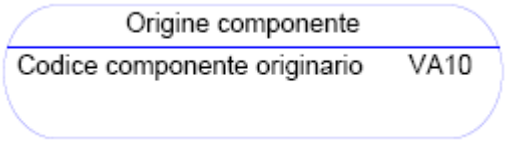


Figura 32 – Origine Componente

Lega le entità “Fonte primaria” e “Componente”:

- Codice componente originario: codice utilizzato nella fonte primaria per determinare univocamente il componente.

7.6.4 Origine secondaria

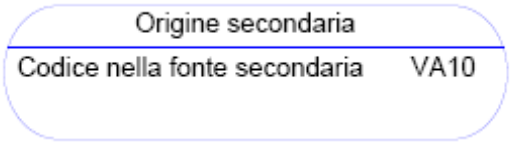


Figura 33 – Origine Secondaria

Lega le entità “Valore” e “Fonte secondaria”:

- Codice nella fonte secondaria: codice che identifica univocamente nella fonte secondaria l'alimento utilizzato per determinare il valore in questione.

7.6.5 Assegnazione del Tipo agli Attributi

Sono stati stabiliti poi i tipi degli attributi, basandosi su quelli già presenti negli schemi da integrare e seguendo le semplici regole illustrate di seguito:

- I campi testuali sono stati tutti tradotti in stringhe a lunghezza variabile con lunghezza massima maggiore o uguale dell'originale.
- Tutti i campi floating point sono stati portati a singola precisione, questo perché non c'è bisogno dell'enorme quantità di cifre significative fornita dalla precisione doppia.
- I nuovi identificativi sono stati creati come interi lunghi, tranne quelli delle entità "Descrizione", che sono stati creati come numeri progressivi aumentati automaticamente (cioè sono di tipo “serial number”).

7.7 Ulteriori Vincoli

Oltre ai vincoli visibili nello schema ER e nelle schede delle singole entità, sono presenti (e implementati) i seguenti vincoli:

- L’attributo “Lingua” di “Descrizione generica” può assumere solo un set definito di valori, presi dallo standard ISO 639:1998 "Code for the representation of names of languages". Al momento l’insieme di valori validi comprende “it” e “en”, rispettivamente per italiano e inglese.
- Nell’entità “Valore”, se il campo booleano “Tracce” è vero, allora l’attributo “Valore” deve essere zero o NULL.
- Tutte le entità sono legate alla rispettiva entità “Descrizione” con una associazione di cardinalità (1,n): n in questo caso è fissato ed è il numero di lingue supportate nella base di dati.
- Il campo “Alimento di riferimento” di "Valore" deve avere, se diverso da NULL, un corrispettivo in "Codice alimento" dell’entità “Alimento”. Questo vincolo non è presente nello schema per non generare confusione.
- Il campo “Categoria standard corrispondente” dell’entità "Categoria originaria" deve avere, se non NULL, un corrispettivo in “Codice categoria standard” di "Categoria standard".

- “Unità di misura”, presente nell’entità “Componente”, può assumere solo un set di valori presi dal vocabolario delle raccomandazioni EUROFOODS, più eventuali necessarie integrazioni. L’insieme di valori possibili è: “g” (grammi); “mg” (milligrammi); “ug” (microgrammi); “mmol” (millimoli); “R” (rapporto o numero puro); “kcal” (chilocalorie); “kJ” (chiloJoule); “IU” (International Units, aggiunto rispetto allo standard).
- “Modo di espressione misura” di “Componente” può assumere solo un insieme di valori presi dal vocabolario EUROFOODS, più eventuali necessarie integrazioni. I valori utilizzati al momento sono:
    - “W” (che significa “per 100g di parte edibile”);
    - "T" (=”per 100g di cibo totale”);
    - "P" (“per 100g di proteine”, aggiunto rispetto allo standard);
    - "D" (=”per 100g di sostanza secca”);
    - "G" (“per 100g di sostanza grassa”, aggiunto allo standard).

7.8 Diagrammi Completi

Vengono ora presentati, a scopo di riepilogo, lo schema ER completo “Final Conceptual” e il corrispettivo schema logico. Quest’ultimo è stato ottenuto dalla traduzione automatica effettuata dal CASE, specificando che il DBMS utilizzato è PostgreSQL.

Entrambi i diagrammi occupano due pagine ciascuno e quindi si è fatto utilizzo di sinonimi grafici: la stessa entità compare due volte per mostrare i collegamenti tra le due pagine.

Da notare anche che nello schema logico “Final Physical” i tipi non sono quelli dello standard SQL, ma vengono usati degli alias tipici di PostgreSQL (es. “INT4” al posto di “INTEGER”).

7.8.1 Schema "Final Conceptual"

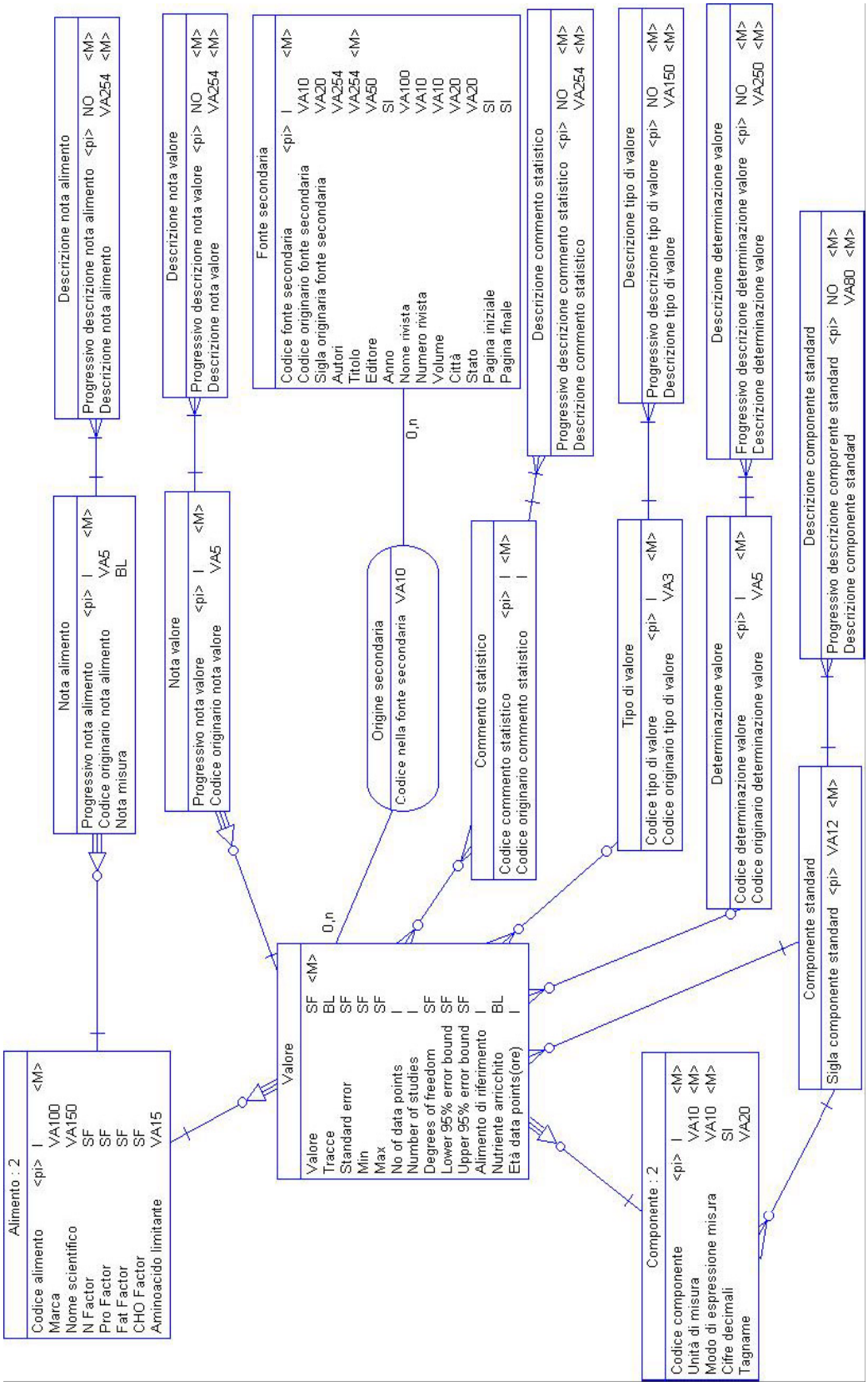


Figura 34 – Final Conceptual

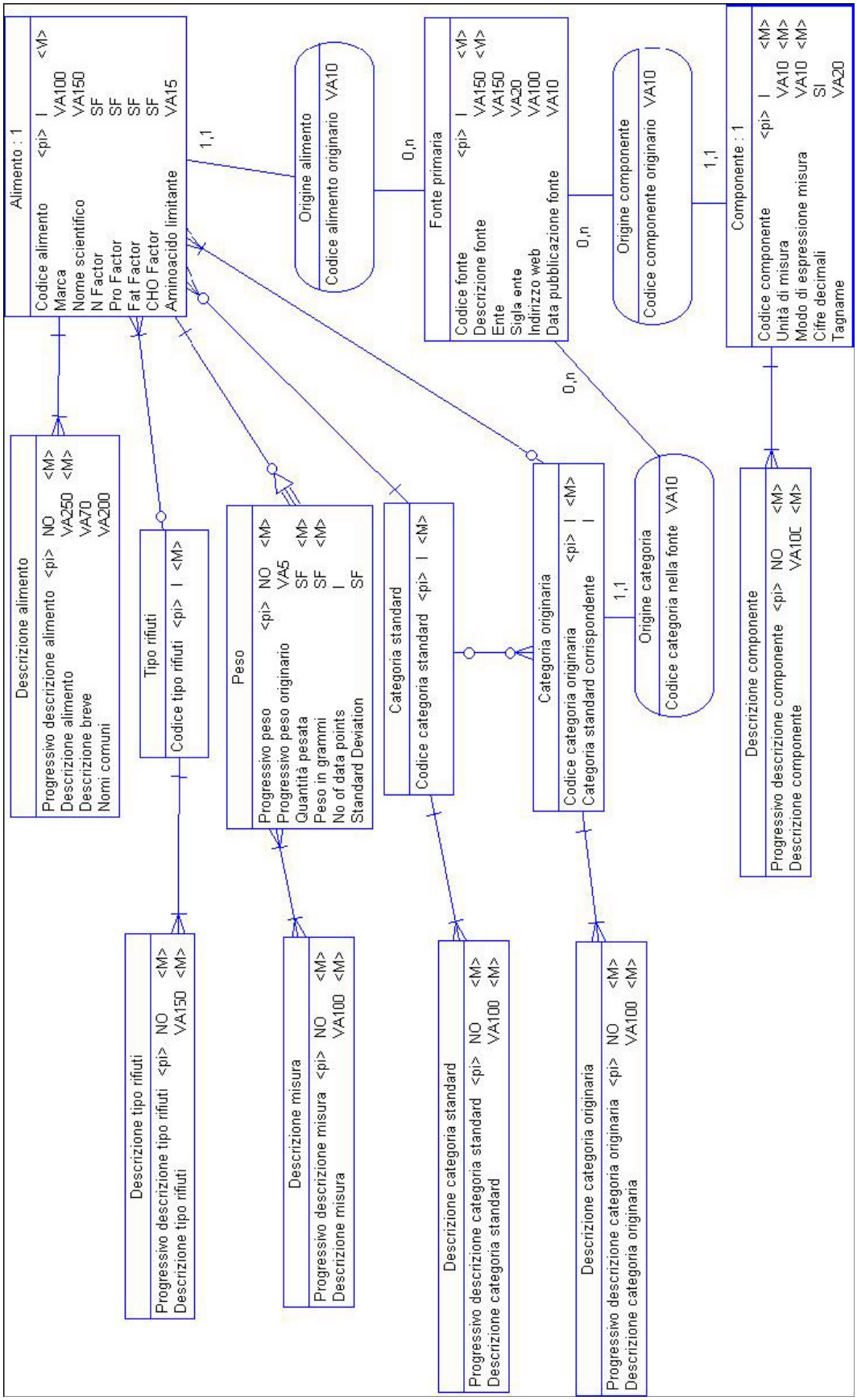


Figura 35 – Final Conceptual

7.8.2 Schema "Final Physical"

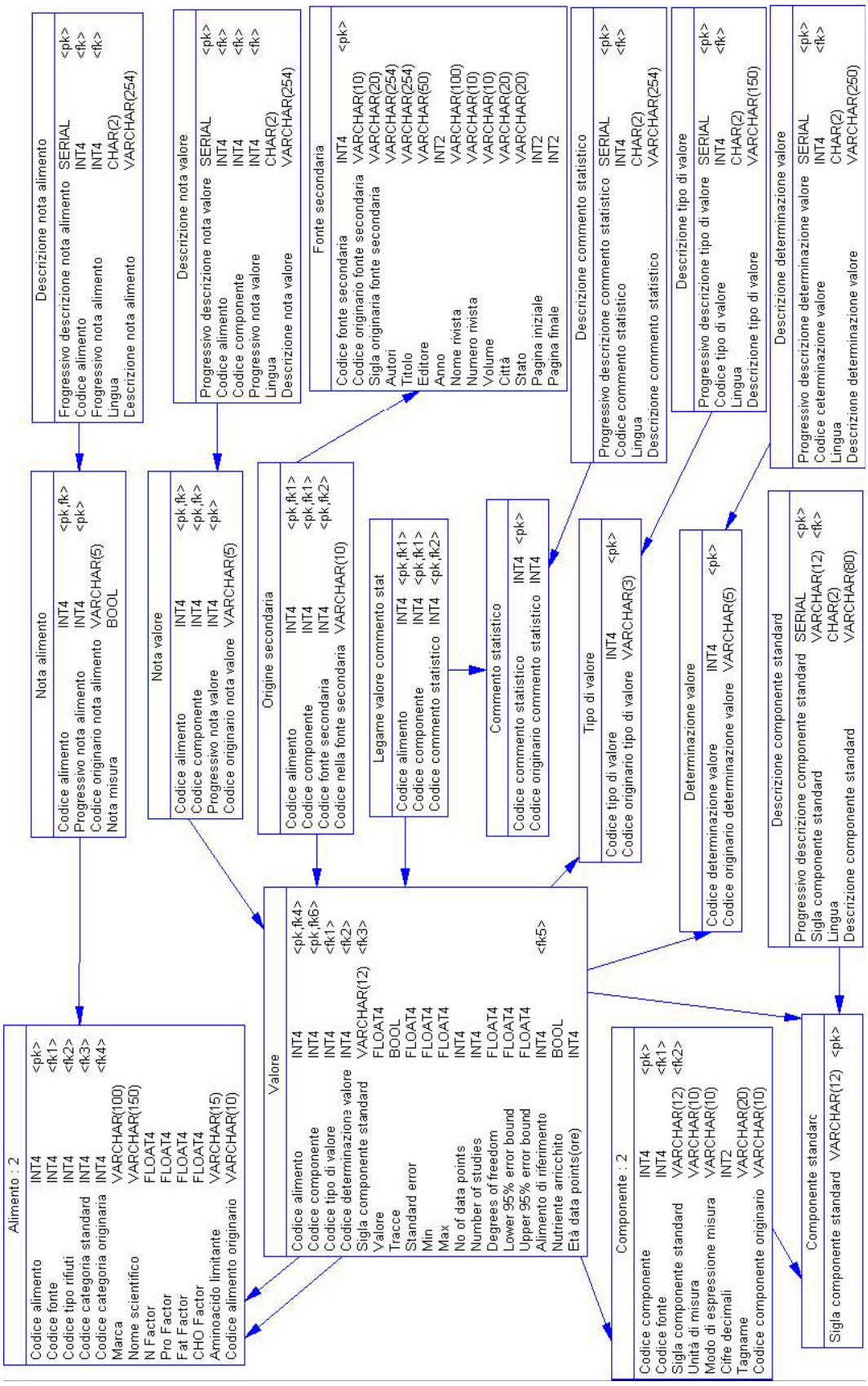


Figura 36 – Final Physical

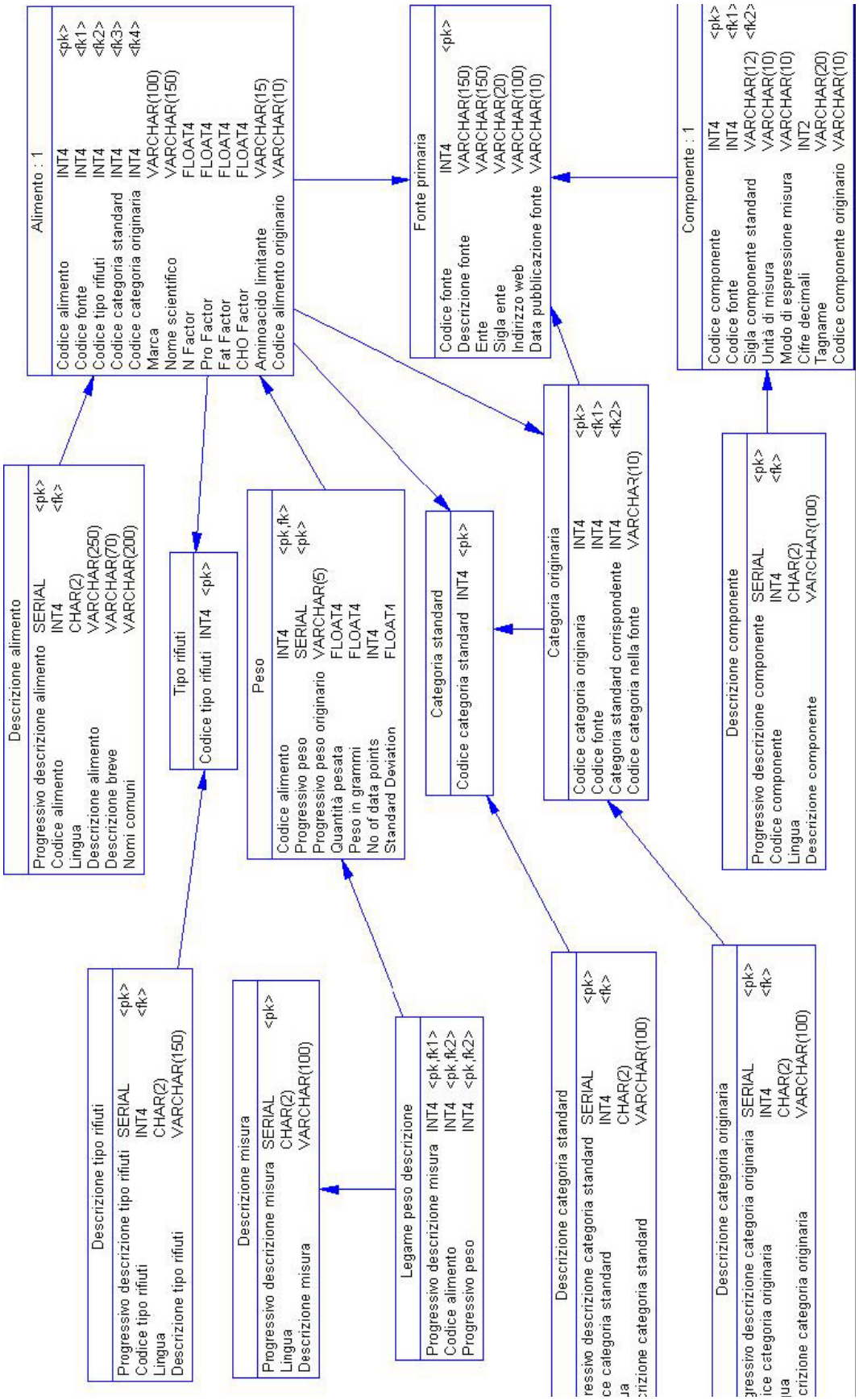


Figura 37 – Final Physical

8. I Moduli di Feeding

8.1 Nota Iniziale

Prima di iniziare la disamina del progetto, si ricorda che vengono illustrati gli aspetti essenziali alla comprensione dello stesso, senza approfondire le tecnologie utilizzate e senza documentare in maniera pedante tutto. Per maggiori delucidazioni si veda la bibliografia fornita e il materiale allegato alla tesi.

Si fa anche presente che nel seguito ci si riferirà a quello che finora è stato chiamato "database di integrazione" anche col termine datawarehouse. Questo perché i dati presenti in entrambi sono gli stessi e quindi, in senso lato, il modulo di feeding alimenta anche il magazzino di dati ricavato dal database di integrazione.

8.2 I Requisiti Software

Il modulo software, denominato semplicemente "Feeder", è progettato per assolvere alle operazioni di gestione della datawarehouse che siano automatizzabili e batch, cioè non richiedano la presenza di un operatore durante il loro completamento.

L'utente dell'applicativo è il Data Base Administrator (DBA), quindi ci si rivolge ad un'utenza esperta, cui siano ben chiare le conseguenze di certe azioni critiche.

Le operazioni implementate dal modulo sono fondamentalmente tre:

- Inserimento dei dati da una fonte al database.
- Aggiornamento dei dati da una fonte al database.
- Eliminazione dei dati relativi ad una singola fonte dal database.

Le operazioni suddette vanno svolte in maniera atomica, nel senso che, ad esempio, anche un errore su una singola tupla deve far annullare l'intero inserimento di una fonte. Questo per non avere incongruenze all'interno della datawarehouse: è facile immaginare le conseguenze di una diversa scelta progettuale.

Inoltre, trattandosi spesso di grossi lavori batch da svolgere una tantum, il tempo di esecuzione può oscillare tra qualche minuto a poche ore. Ovviamente tale tempo sarà sempre proporzionale alle dimensioni dei dati in gioco (es. un'operazione di inserimento di pochi megabyte di dati non può impiegare delle ore!).

Altro importante requisito è che il software permetta una facile introduzione di ulteriori sorgenti di dati: deve essere possibile riusare il codice e scrivere per ogni fonte solo le linee necessarie all'estrazione dei dati dalla specifica sorgente.
Per meglio illustrare quanto detto finora, presentiamo un semplice use case diagram in UML

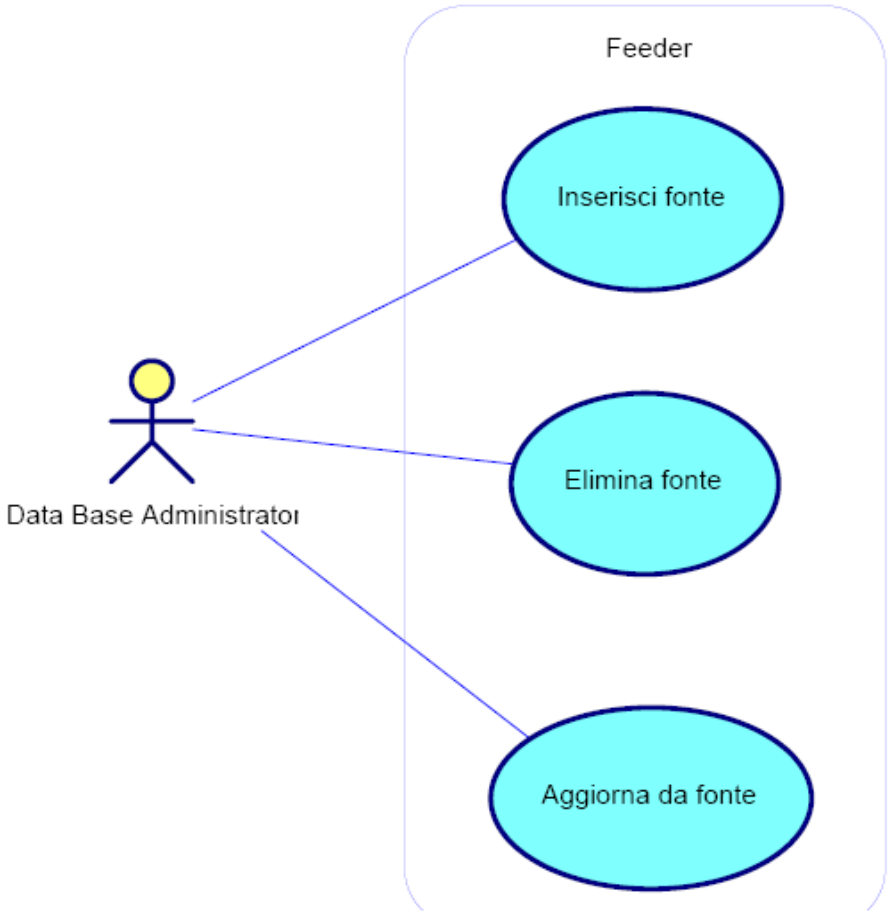


Figura 38 – Use Case Feeder

Presentiamo quindi una semplice descrizione dei singoli casi d'uso

8.2.1 Inserisci fonte

Tramite questa funzionalità l'utente effettua un primo inserimento dei dati relativi ad una fonte. Le informazioni non introducibili con un inserimento automatico, ma indispensabili a quest'ultimo, devono essere ovviamente immesse prima dal DBA, ad esempio tramite l'utilizzo di script (così che la procedura risulti comunque ripetibile).
Si noti che tra i dati inseriti dall'utente è presente lo schema della fonte: il DBA deve cioè specificare se la fonte è INRAN, USDA, etc.

8.2.2 Elimina fonte

Questa operazione permette all'utente di eliminare tutti i dati facenti riferimento ad una singola fonte, quindi anche quelli introdotti dopo il primo inserimento automatico.

Si capisce dunque come questa procedura, come spesso accade per le eliminazioni, sia estremamente rischiosa e vada adoperata raramente, soprattutto vista la natura poco dinamica del database di integrazione.

L'andamento tipico di questa operazione è simile a quello visto per l'inserimento, con la sola differenza che ora l'utente non deve inserire i dati di accesso alla fonte, in quanto l'operazione si svolge esclusivamente sulla datawarehouse.

8.2.3 Aggiorna da fonte

Dopo il primo inserimento, possono avvenire degli aggiornamenti successivi qualora la sorgente di dati fornisca i file "differenziali": tra le fonti selezionate solo USDA garantisce questo tipo di servizio.

Anche questa operazione ha un andamento simile a quello dell'inserimento.

8.3 La Struttura delle Classi

8.3.1 Un Ulteriore Sguardo alle Tecnologie

Si è deciso di progettare e implementare i moduli di feeding utilizzando metodologie di progettazione e di programmazione ad oggetti.

Per l'implementazione si è scelto il linguaggio C++, corredato delle API ODBC: questa combinazione è facilmente portabile in ambiente Windows ed inoltre garantisce efficienza.

Il programma si interfaccia alla datawarehouse tramite ODBC, cosa che garantisce indipendenza dal DBMS: per cambiare quest'ultimo, infatti, l'utente deve semplicemente creare uno script di generazione del database (con PowerDesigner) per il nuovo DBMS e dare al programma un Data Source Name (DSN) diverso.

Si fa presente sin da ora che la struttura a oggetti non è stata utilizzata per creare un incapsulamento delle API ODBC: il programma è scritto, per motivi di semplicità e di efficienza, in C, utilizzando le estensioni ad oggetti solo per le funzionalità proprie dell'applicazione.

8.3.2 La Gerarchia Principale

Data la natura relativamente semplice del problema, si è deciso di adottare una gerarchia unica, senza particolari collaborazioni, che viene mostrata nel class diagram seguente.

Nella figura, per motivi di chiarezza, non vengono mostrati né i parametri né il valore ritornato dai singoli metodi, che saranno comunque illustrati in seguito.

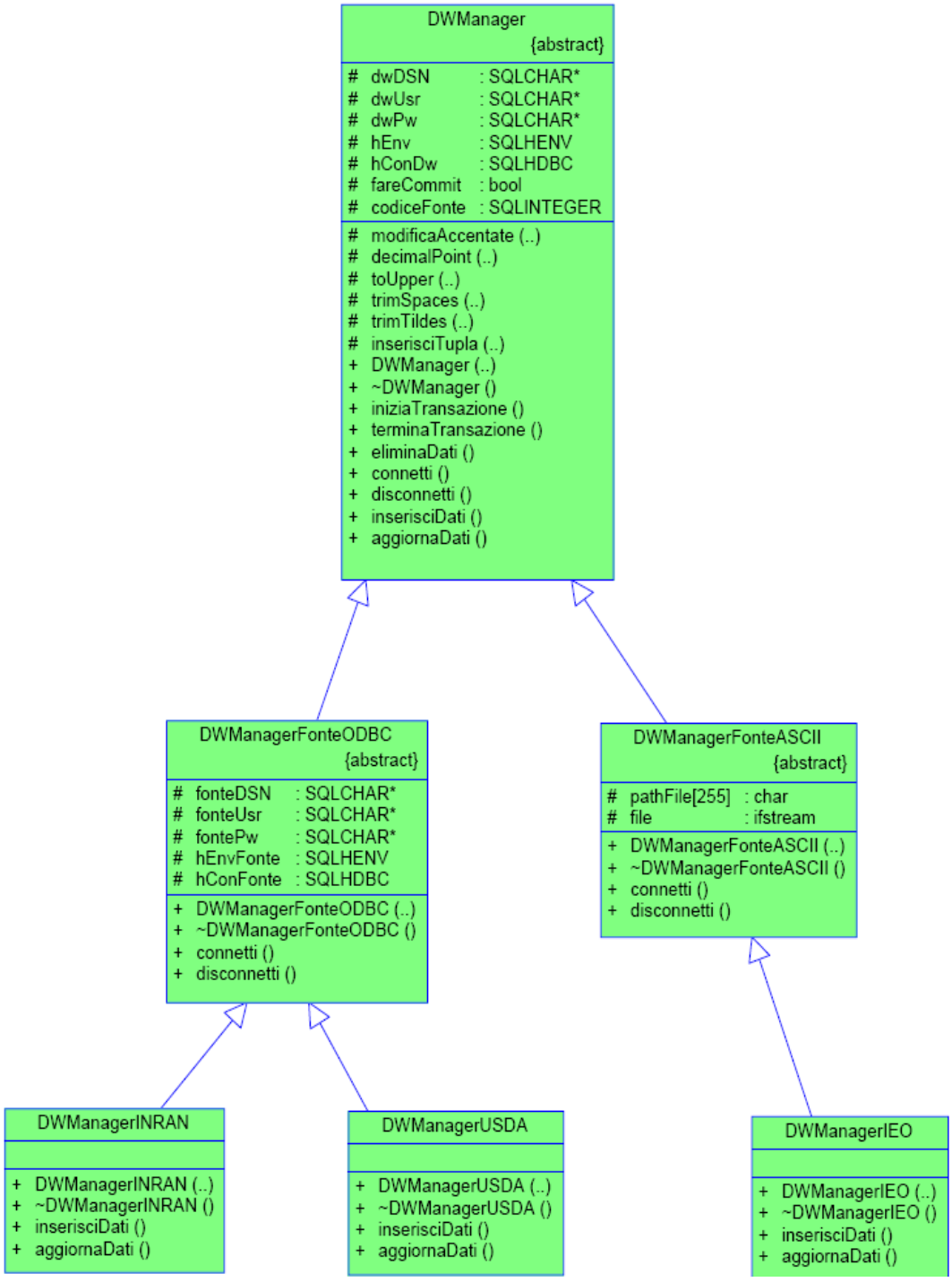


Figura 39 – Class Diagram Feeder

Scopo primario della gerarchia è ovviamente la gestione della datawarehouse, cioè l'implementazione delle regole alla base dei casi d'uso. Per fare ciò queste classi devono ovviamente collaborare con un interfaccia utente, che ha il semplice compito di richiedere e fornire i dati al DBA.
Viste nel loro insieme, le classi della figura precedente possono essere concettualmente raggruppate in un package denominato "Data Warehouse Management".

8.3.3 La Gerarchia dei Buffer

Oltre all'interfaccia utente, l'unico altro elemento dell'applicazione è un ulteriore package, contenente delle classi con semplice funzione di storage delle tuple (sono cioè dei veri e propri buffer).
La relazione di dipendenza tra i due package è mostrata nel diagramma seguente:

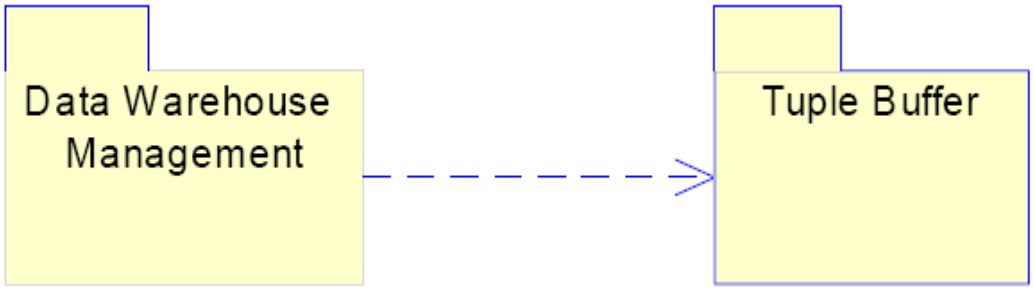


Figura 40 – Gerarchia dei Buffer

La dipendenza è dovuta essenzialmente al fatto che le classi nel primo package istanziano e utilizzano i buffer presenti nel secondo.
Il contenuto del package "Tuple Buffer" è illustrato nel class diagram seguente, dove si può notare una classe base astratta ("Tupla") e le classi derivate atte ad ospitare una tupla delle varie tabelle del database di integrazione.
Come sempre, in figura mancano molti particolari, omessi per non rendere confuso il diagramma: per ora si vuole semplicemente mostrare il modo in cui i vari elementi si relazionano tra di loro.

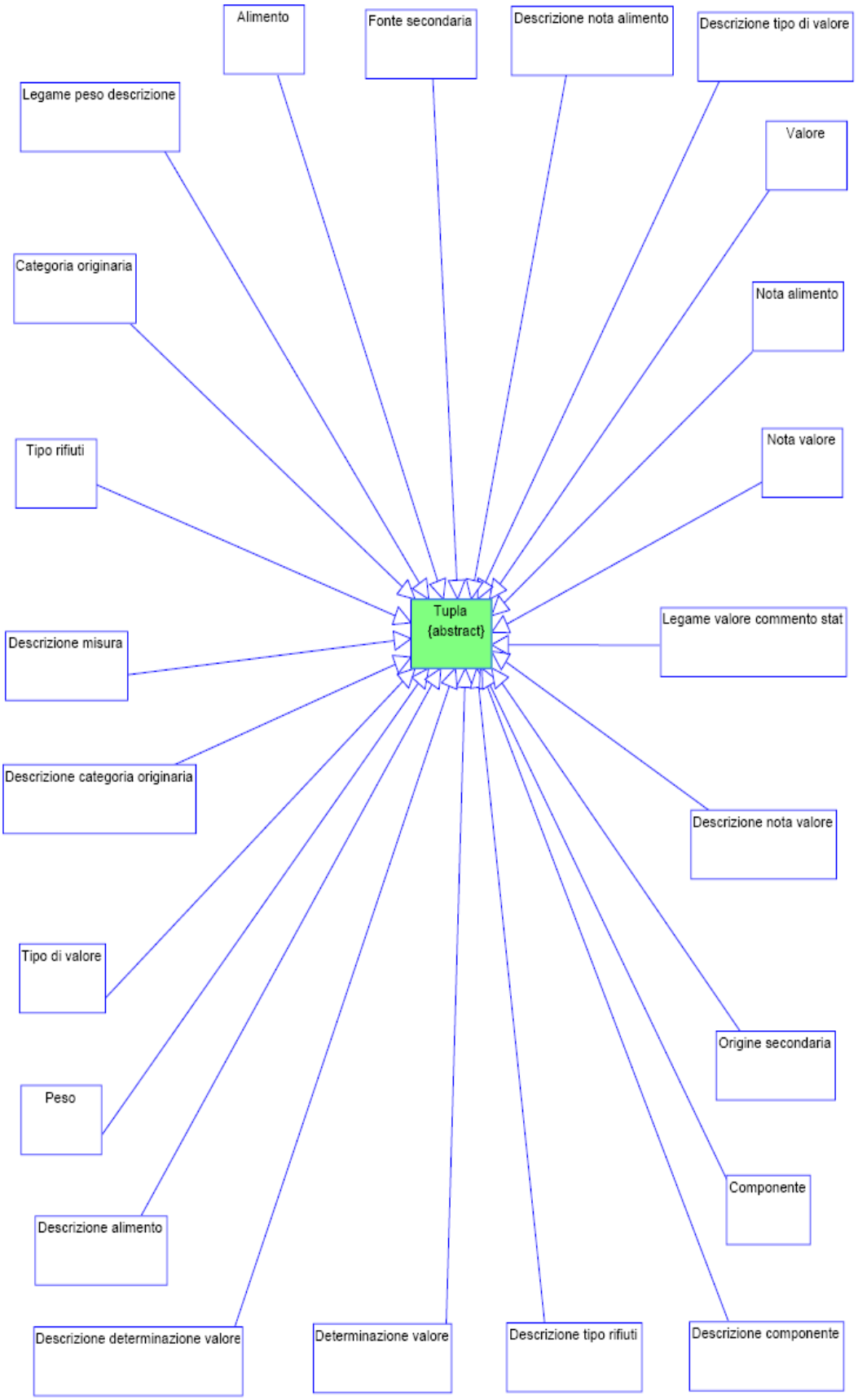


Figura 41 – Classe Tupla

8.4 Informazioni sulle Singole Classi

Presentiamo ora delle schede sulle varie classi, illustrando ovviamente dove necessario anche le relazioni che le legano con le altre. La trattazione parte ovviamente dal package "Data Warehouse Management", che costituisce l'essenza dell'applicazione.

8.4.1 DWManager

Questa classe ha la fondamentale responsabilità di fornire alle classi da essa derivate tutti i metodi per operare sulla datawarehouse (tramite ODBC). Tale classe cioè ha piena conoscenza della struttura della datawarehouse, ma ignora completamente invece lo schema delle fonti.
Questa classe inoltre impone, attraverso la definizione di alcuni metodi virtuali puri, la semplice interfaccia cui tutte le classi derivate devono conformarsi: si tratta dunque di una classe astratta.
Attributi:

- dwDSN, dwUsr, dwPw: immagazzinano rispettivamente DSN, user name e password della sorgente di dati ODBC corrispondente alla datawarehouse.
- hEnv: handle dell'environment ODBC all'interno del quale vengono definite le connessioni alla datawarehouse.
- hConDw: handle della connessione ODBC alla datawarehouse.
- fareCommit: campo booleano che indica lo stato corrente della transazione: se è vero la transazione va terminata con commit, se è falso con rollback.
- codiceFonte: codice della fonte nella datawarehouse.

Metodi:

- DWManager(SQLCHAR* DSNdw, SQLCHAR* Utentedw, SQLCHAR* Pwdw): il costruttore inizializza gli attributi dwDSN, dwUsr e dwPw con i parametri forniti dall'utente. Inoltre in tale metodo fareCommit viene settato a falso.
- connetti(): istanzia gli handle hEnv e hConDw e connette alla datawarehouse.
- disconnetti(): disconnette dalla datawarehouse e dealloca gli handle dell'ambiente e della connessione.

- iniziaTransazione(): comincia una transazione sulla connessione alla datawarehouse.
- terminaTransazione(): conclude la transazione sulla datawarehouse, facendo commit o rollback a seconda del flag fareCommit.
- inserisciDati(): funzione virtuale pura il cui corpo deve essere definito nelle classi derivate: questo metodo ha infatti il compito di inserire i dati relativi alla fonte nella datawarehouse, e quindi richiede la conoscenza dello schema della specifica fonte. Per raggiungere lo scopo vengono utilizzati i buffer del package "Tuple Buffer" e il metodo protetto inserisciTupla().
- aggiornaDati(): funzione virtuale pura il cui corpo deve essere definito nelle classi derivate: questo metodo ha infatti il compito di aggiornare i dati relativi alla fonte nella datawarehouse, e quindi richiede la conoscenza dello schema della specifica fonte. Per raggiungere lo scopo vengono utilizzati i buffer del package "Tuple Buffer" e il metodo protetto inserisciTupla().
- eliminaDati(): questo metodo elimina tutti i dati relativi alla fonte. E' implementato in DWManager perché effettua operazioni esclusivamente sulla datawarehouse.
- inserisciTupla(): metodo che accetta come parametro un puntatore alla classe astratta Tupla: all'interno, tramite l'utilizzo del supporto Run Time Type Identification (RTTI) del C++, si stabilisce quale è il reale aspetto del buffer e si inserisce la tupla nella corrispettiva tabella. Alla prima invocazione prepara le query, che verranno poi semplicemente eseguite con i parametri inviati nel buffer di volta in volta. I rimanenti metodi lavorano tutti sulla stringa passata loro come parametro e compiono operazioni di formattazione del testo sui buffer prima che questi vengano inseriti nella datawarehouse.
- toUpper(char* stringa): porta tutte le lettere della stringa da minuscole a maiuscole. Nella datawarehouse infatti, per velocizzare le ricerche, tutti caratteri sono memorizzati come maiuscoli.
- modificaAccentate(char* stringa): sostituisce tutte le occorrenze di lettere accentate (es. " è ") con la corrispettiva lettera minuscola più l'apostrofo (es. " e' "). Se si vogliono caratteri tutti maiuscoli, va utilizzata prima di toUpper().

- decimalPoint(char* stringa): sostituisce tutte le virgole in una stringa con dei punti. E' adoperata qualora la fonte fornisca valori numerici in campi di testo e usi la virgola come separatore delle cifre decimali: molti dei driver ODBC adoperati, infatti, richiedono che si usi il punto.
- trimSpaces(char* stringa): elimina gli spazi presenti all'inizio e alla fine di una stringa.
- trimTildes(char* stringa): elimina i tutti i caratteri tilde (cioè "~") presenti all'inizio e alla fine di una stringa. Viene al momento impiegata nel processamento dei soli file di aggiornamento USDA, in cui i campi di testo iniziano e terminano appunto con delle tilde.

8.4.2 DWManagerFonteODBC

Questa classe fornisce i metodi per effettuare la connessione alle fonti utilizzando ODBC. Per far ciò vengono semplicemente aggiunti degli attributi e ridefiniti dei metodi della classe base.

Poiché tale classe eredita ma non implementa i metodi virtuali puri del padre, anche essa è astratta.

Attributi:

- fonteDSN, fonteUsr, fontePw: immagazzinano rispettivamente DSN, user name e password della sorgente di dati ODBC cui corrisponde la fonte.
- hEnvFonte: handle dell'environment ODBC all'interno del quale vengono definite le connessioni alla fonte.
- hConFonte: handle della connessione ODBC alla fonte.

Metodi:

- DWManagerFonteODBC(SQLCHAR* DSNfonte, SQLCHAR* UtenteFonte, SQLCHAR* PwFonte, SQLCHAR* DSNdw, SQLCHAR* Utentedw, SQLCHAR* Pwdw): questo costruttore assegna il valore iniziale agli attributi fonteDSN, fonteUsr e fontePw adoperando i corrispettivi parametri forniti dall'utente. Gli altri parametri vengono invece passati al costruttore della classe base.
- connetti(): ridefinizione di connetti() della classe padre. Invoca il metodo corrispondente della classe base (per connettersi alla datawarehouse). Dopo, se il DSN della fonte è diverso dalla stringa vuota, istanzia gli handle hEnvFonte e hConFonte e si connette alla fonte.

- disconnetti(): ridefinizione di disconnetti() della classe padre. Invoca il metodo corrispondente della classe base (per disconnettersi dalla datawarehouse). Dopo si disconnette dalla fonte e dealloca gli handle hEnvFonte e hConFonte.

8.4.3 DWManagerFonteASCII

Questa classe fornisce i metodi per effettuare la connessione alle fonti i cui dati risiedono in semplici file di testo ASCII. Per far ciò vengono aggiunti degli attributi e ridefiniti dei metodi della classe base.

Per usufruire di file ASCII come fonti si può adoperare anche la classe DWManagerFonteODBC, in quanto è possibile connettersi ad un file ASCII con campi delimitati da un separatore utilizzando per l'appunto ODBC. Per file con campi non delimitati si hanno due possibilità: o si trasforma il file, inserendo dei separatori, o si utilizza questa classe.

Poiché tale classe eredita ma non implementa i metodi virtuali puri del padre, anche essa è astratta.

Attributi:

- pathFile: stringa contenente il percorso che denota la posizione del file ASCII, compreso il nome del file stesso. Questo path non può essere più lungo di 254 caratteri e non deve includere spazi.
- file: oggetto di tipo ifstream attraverso il quale viene gestito lo stream di input dal file.

Metodi:

- connetti(): ridefinizione di connetti() della classe padre. Invoca il metodo corrispondente della classe base (per connettersi alla datawarehouse). Dopo, se la path del file è diversa dalla stringa vuota, lo apre e lo lega all'attributo file.
- disconnetti(): ridefinizione di disconnetti() della classe padre. Invoca il metodo corrispondente della classe base (per disconnettersi dalla datawarehouse). Dopo chiude il file.

8.4.4 DWManagerINRAN, DWManagerUSDA, DWManagerIEO

Queste classi, che rappresentano le foglie della gerarchia, sono a conoscenza sia dello schema della fonte sia di quello della datawarehouse. Grazie a ciò esse possono implementare la semplice interfaccia definita in DWManager e assolvere quindi allo scopo primario della gerarchia: alimentare il database.

Va comunque detto che le implementazioni di aggiornaDati() di DWManagerINRAN e DWManagerIEO sono vuote. Questo è dovuto al fatto che tali fonti non solo non forniscono periodicamente dei file di aggiornamento, ma inoltre non sembrano garantire la persistenza dello schema tra due aggiornamenti: tutto ciò rende ovviamente un inutile esercizio di programmazione l'implementare i rispettivi metodi aggiornaDati().

Metodi:

- DWManagerINRAN(SQLCHAR* DSNfonte, SQLCHAR* UtenteFonte, SQLCHAR* PwFonte, SQLCHAR* DSNdw, SQLCHAR* Utentedw, SQLCHAR* Pwdw): il costruttore passa i parametri al costruttore della classe base e scrive in codiceFonte il codice che identifica la specifica fonte nella datawarehouse.
- inserisciDati(): questa funzione preleva i dati dalla fonte, copiandoli nei buffer del package "Tuple Buffer", e li inserisce nella datawarehouse tramite inserisciTupla(). Tale metodo viene utilizzata per un primo inserimento dei dati da una fonte: per ulteriori introduzioni si veda aggiornaDati().
- aggiornaDati(): questo metodo inserisce i nuovi dati come accade per inserisciDati() e effettua anche le eventuali cancellazioni e aggiornamenti dei dati preesistenti.

8.4.5 Tupla

Questa banale classe astratta serve sia a realizzare il vincolo della realtà che indica che tutti i buffer sono accomunati dal contenere una tupla, sia a fornire un ulteriore tipo alle classi dei buffer. Infatti il tipo "Tupla" viene usato nei metodi che accolgono come parametro il generico buffer per poi identificarne a run time l'effettivo tipo specifico (es. "Alimento", etc.).

L'utilizzo di tale classe è dettato da questioni di stile più che di praticità: si è semplicemente deciso di utilizzare Tupla* piuttosto di un generico void*.

8.4.6 Le Classi Buffer

Queste classi ospitano ognuna una tupla di una tabella diversa della datawarehouse. I loro campi corrispondono alle colonne del database ed infatti sono queste classi sono state ricavate dalla traduzione automatica, effettuata dal CASE, dello schema "Final Physical" del capitolo precedente. Ogni attributo ha poi associato un campo indicatore tipico di tutti i buffer ODBC.

Essendo l'unico scopo di queste classi quello di funzionare da buffer, è stato ritenuto inutile utilizzare metodi di get e set e si è quindi deciso di mantenere pubblici i vari attributi.

L'unico metodo di tali classi è il costruttore, che semplicemente inizializza a NULL i vari campi, scrivendo SQL_NULL_DATA nell'indicatore corrispondente.

8.5 L'Interfaccia Utente

Scopo dell'interfaccia utente è richiedere i dati, richiamare i metodi del package "Data Warehouse Management" e infine presentare i risultati all'utente.

Questi semplici compiti possono essere realizzati sia da un'interfaccia grafica che testuale: trattandosi comunque di un applicativo per utenti esperti (il DBA), si è deciso inizialmente di costruire un semplice software testuale. Per avere informazioni aggiuntive su questa interfaccia si veda anche l'Appendice A.

Nulla vieta comunque di legare al package menzionato un'interfaccia di tipo grafico, magari implementata con filosofia di progettazione e programmazione ad oggetti.

8.6 Aggiunta di Altre Fonti

Uno dei requisiti più importanti cui il software risponde è quello dell'estensione ad ulteriori sorgenti di dati.

Per venire incontro a questa esigenza, la gerarchia del package "Data Warehouse Management" è estendibile in due punti: innanzitutto si può aggiungere un ulteriore classe derivata da DWManagerFonteODBC o da DWManagerFonteASCII. Questa classe deve ridefinire semplicemente le due funzioni inserisciDati() e aggiornaDati(), utilizzando tra l'altro i metodi protetti messi a disposizione dalle classi base.

Oltre a ciò, si può estendere la gerarchia aggiungendo una classe derivata da "DWManager": si può infatti voler accedere alla fonte in modo differente da ODBC o dallo strema di input per file del C (ad esempio, si potrebbe decidere di utilizzare le API C di un particolare DBMS, cosa possibile anche per PostgreSQL). Questa estensione "al secondo livello" della gerarchia dovrebbe comunque uniformarsi alle classi già create: bisognerebbe cioè che questa nuova classe ridefinisca semplicemente i metodi connetti() e disconnetti(), fornendo nel contempo alle classi derivate degli attributi che facciano da handle per la fonte. Il restante comportamento (la definizione di inserisciDati() e aggiornaDati()) è invece compito del terzo livello della gerarchia e differisce ovviamente da fonte a fonte.

8.7 Architettura del Sistema

A completamento di quanto detto finora, si mostra di seguito un deployment diagram con una possibile disposizione dei componenti tra le varie unità di calcolo. Le relazioni tra i componenti non vengono illustrate in quanto dovrebbero risultare chiare alla luce della discussione precedente.

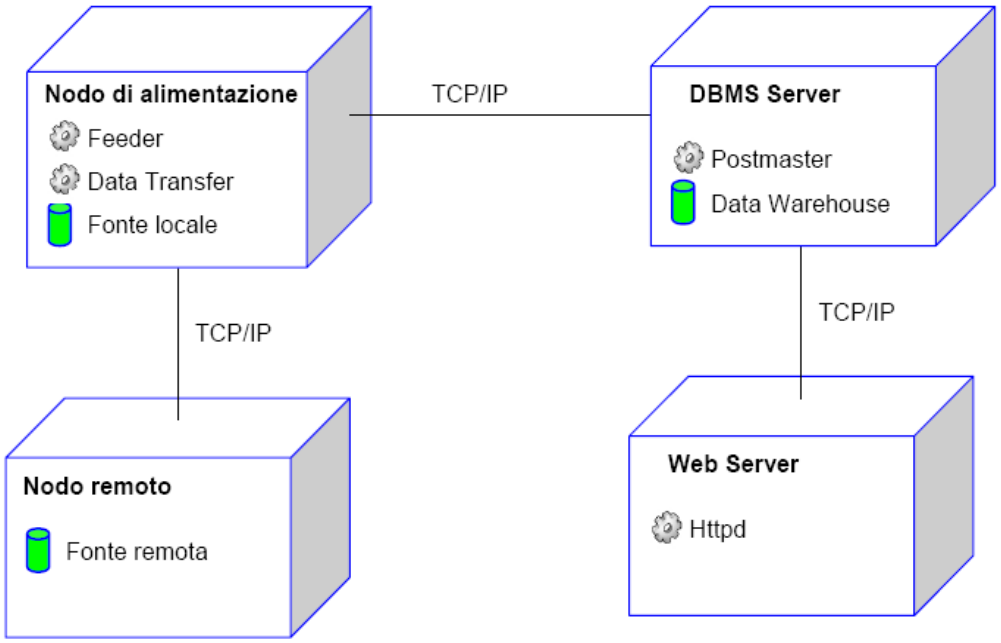


Figura 42 – Architettura del Sistema

Nel diagramma precedente non compaiono i client, che ovviamente sono connessi (sempre tramite una rete TCP/IP) con il web server, che nell'esempio mostrato è Apache.

Capitolo IV – Case Study : Analisi e Progettazione

1. Introduzione

È stata realizzata un'applicazione Web che permetta agli utilizzatori di usufruire dei contenuti della Data WareHouse, mettendo a disposizione degli utenti e dell'amministratore del sistema una serie di funzionalità definite dalle specifiche assegnate.
La Web application è stata realizzata con l'utilizzo del framework JavaServer Faces, usufruendo, di tutte le potenzialità messe a disposizione da tale framework.
Le fasi di sviluppo hanno seguito il tipico ciclo di vita del software secondo il modello "a cascata", *Waterfall Model*, e possono essere riassunte di seguito :

- studio di fattibilità;
- analisi e specifica dei requisiti;
- progettazione;
- codifica ed implementazione;
- testing;
- messa in esercizio;
- manutenzione;

Lo studio di fattibilità, in questo caso, è stato immediatamente superato, in quanto non c'erano da fare considerazioni riguardo le possibili soluzioni da adottare, le risorse finanziarie ed umane a disposizione, tenendo conto che sono stati fissati a priori gli strumenti da utilizzare.
L'Analisi e la specifica dei requisiti ha previsto la realizzazione del Documento di Specifica dei Requisiti Software (SRS – Software Requirements Specification), elencando tutte le funzionalità fornite dal sistema software sia all'utente che all'amministratore e considerando per ciascuna di esse, una breve descrizione, gli input previsti, le elaborazioni eseguite ed i possibili output prodotti.

Nella fase di progettazione, sono stati realizzati i principali diagrammi UML (Unified Modelling Language) previsti per lo sviluppo, quali :

- Class Diagram;
- Use Case Diagram;
- Sequence Diagram;
- Activity Diagram;
- Statechart Diagram;

Lo sviluppo ha poi previsto la fase di codifica, realizzando l'applicazione con il framework JSF.
La fase di testing ha permesso la correzione di errori e problemi rilevati durante il funzionamento del software.
Infine, non sono state previste delle vere e proprie fasi di messa in esercizio e manutenzione.
Lo strumento software di tipo CASE, che è stato utilizzato per la realizzazione di tutti i diagrammi della fase di progettazione, è il noto Sybase Power Designer.

2. Requisiti

Il case study (caso di studio) preso in esame, prevede lo sviluppo di una Web application che permette all'utenza in generale, di usufruire dei contenuti della Data WareHouse.
Gli utilizzatori possono avere permessi di accesso diversi e quindi funzionalità differenziate, sulla base della distinzione dei seguenti ruoli :

- utente;
- amministratore;

Per ciascuno di essi, le funzionalità accessibili nell'applicazione sono descritte di seguito.

Utente :

- registrazione che permette di effettuare ricerche sulla Data WareHouse, dopo che un utente si è registrato al sito, non può ancora effettuare l'operazione di accesso, in quanto resterà in attesa che l'amministratore gli fornirà l'accesso;
- accesso all'archivio alimenti, con possibilità di fissare diversi criteri di ricerca degli alimenti e di visionarne le informazioni;
- gestione dei propri dati personali per un eventuale aggiornamento;
- richiesta di una mail, contenente username e password di accesso, in caso di dimenticanze;
- operazioni di login e logout;

Amministratore :

- gestione completa dell'archivio della Data WareHouse, con la possibilità di inserire, modificare e cancellare gli alimenti;
- accesso all'archivio della Data WareHouse, con possibilità di fissare diversi criteri di ricerca degli alimenti e di visionarne le informazioni;
- gestione degli utenti registrati, con la possibilità di visionarne i dati ed eseguire le operazioni di cancellazione e sblocco, qualora un utente si sia appena registrato ed è in attesa della convalida da parte dell'amministratore;
- gestione dei propri dati di accesso per un eventuale aggiornamento;
- operazioni di login e logout;

Sulla base di tali requisiti è stato redatto il Documento di Specifica dei Requisiti Software (SRS), di fondamentale importanza per le fasi successive di sviluppo.

3. Progettazione

La fase di progettazione ha previsto la realizzazione dei principali diagrammi UML, in relazione soprattutto all'analisi del software e non tanto alla sua corrispondente implementazione. Essi costituiscono un'astrazione di alto livello del sistema e sono completamente indipendenti da quelli che possono essere gli strumenti utilizzati nella codifica.

3.1 Use Case Diagrams

I diagrammi dei casi d'uso (Use Case Diagrams) costituiscono uno strumento utile per catturare il comportamento esterno del sistema da sviluppare, senza dover specificare come tale comportamento debba essere realizzato; il sistema è visto come una scatola nera (black-box). Essi forniscono una descrizione dei "modi" in cui il sistema potrà essere utilizzato, ossia ciò che l'utente può fare su di esso e come quest'ultimo risponde alle sollecitazioni. La realizzazione dei diagrammi ha previsto un approccio top-down, partendo dallo scenario di carattere generale, in cui le modalità d'uso logicamente correlate sono accorpate, fino ad una decomposizione in cui sono descritti i casi di utilizzo non ulteriormente scomponibili.

3.1.1 Generale

Facendo riferimento ai requisiti assegnati, è stato definito un Use Case Diagram Generale, che si pone al livello di astrazione più elevato del sistema, mettendo in evidenza tutte quelle che sono le modalità di utilizzo dello stesso, da parte dell'utente e dell'amministratore. Questi ultimi rappresentano gli attori che possono accedere alle funzionalità offerte dal sistema, di cui alcune sono condivise ossia accessibili da entrambi, mentre altre sono prerogativa esclusiva di uno dei due ruoli.

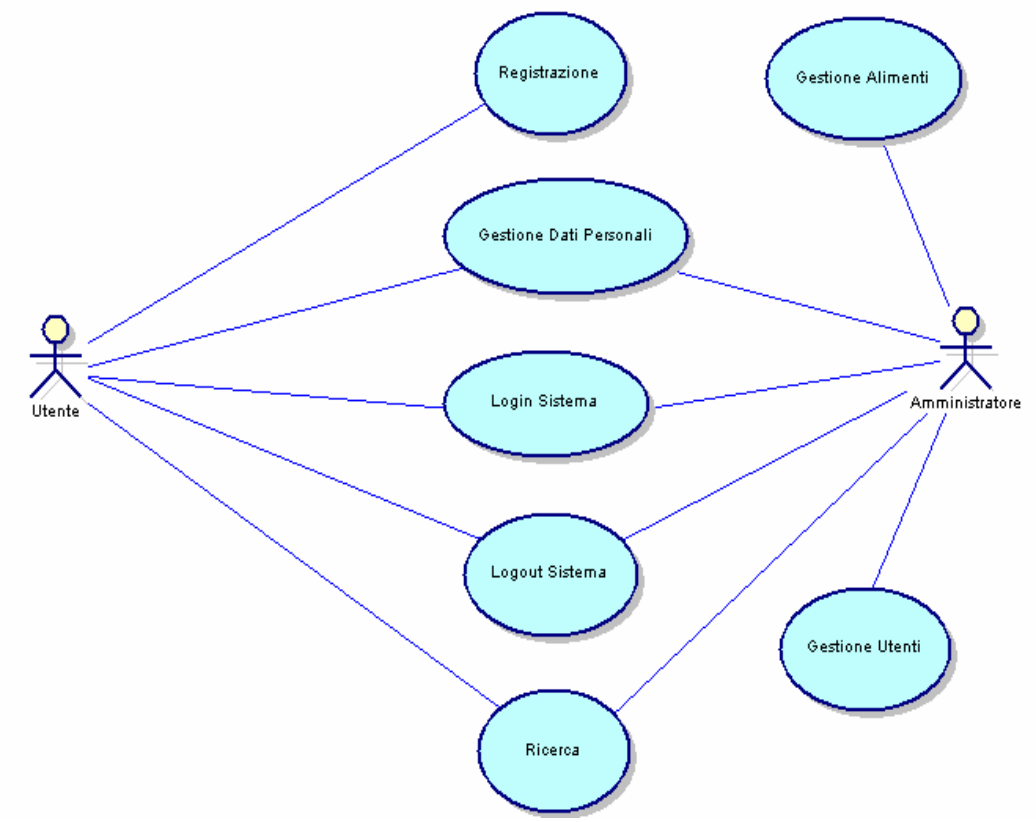


Figura 1 - Use Case Diagram Generale

A partire da questo diagramma, è possibile effettuare una decomposizione scendendo ad un livello di astrazione minore, in cui si prendono in considerazione nel dettaglio le modalità di utilizzo del sistema.
In particolare, le modalità accessibili esclusivamente dall'utente sono le seguenti :

- Registrazione;
- Ricerca;

3.1.3 Registrazione

La modalità di "Registrazione" può essere considerata atomica e non ulteriormente decomponibile, facendo riferimento alla procedura eseguita dall'utente per registrarsi ed usufruire dei servizi della Web application.

3.1.4 Caso d'uso comune:i Ricerca

Questa operazione può essere effettuata sia dall'utente che dall'amministratore, ed è stata realizzata in modo da poter semplificare al massimo la ricerca di un determinato alimento.

Infatti sono state realizzate tre tipi di ricerca diverse :

- Ricerca per Alimento;
- Ricerca per Componente;
- Ricerca per Categoria;

Dopo aver effettuare la ricerca è possibile visualizzare l'elenco degli alimenti che corrispondono ai criteri impostati nella ricerca, ed infine è possibile visualizzare le informazioni di un singolo alimento.

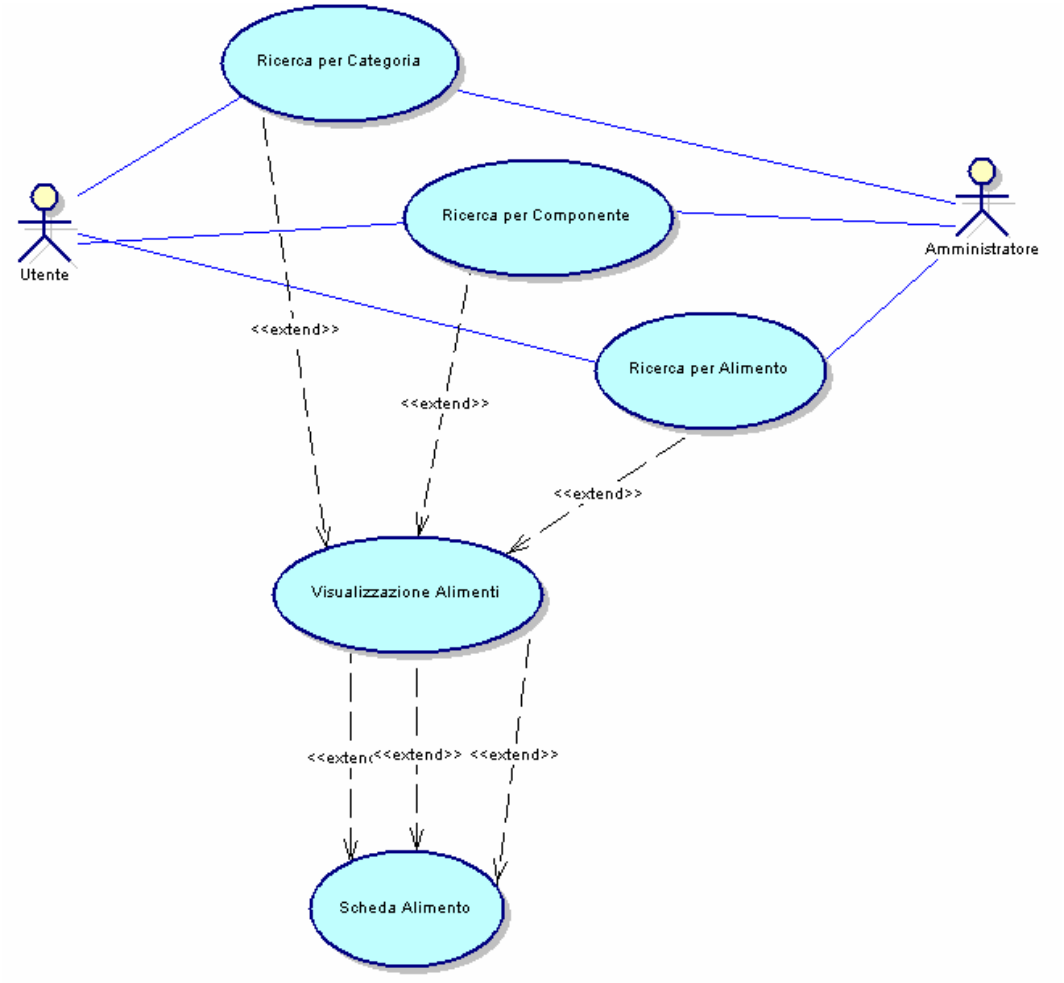


Figura 2 - Use Case Diagram Ricerca

3.1.5 Gestione Utenti

Considerando nuovamente il diagramma dei casi d'uso Generale, si evince che ci sono le modalità "Gestione Utenti" e "Gestione Alimenti" che sono accessibili solo ed esclusivamente dall'amministratore.

In particolare, "Gestione Utenti" comprende alcune azioni basilari che possono essere eseguite sugli utenti registrati, in particolare la visualizzazione di un elenco di questi ultimi, la cancellazione, la visualizzazione dei dati anagrafici di un singolo utente e l'operazione di sblocco.

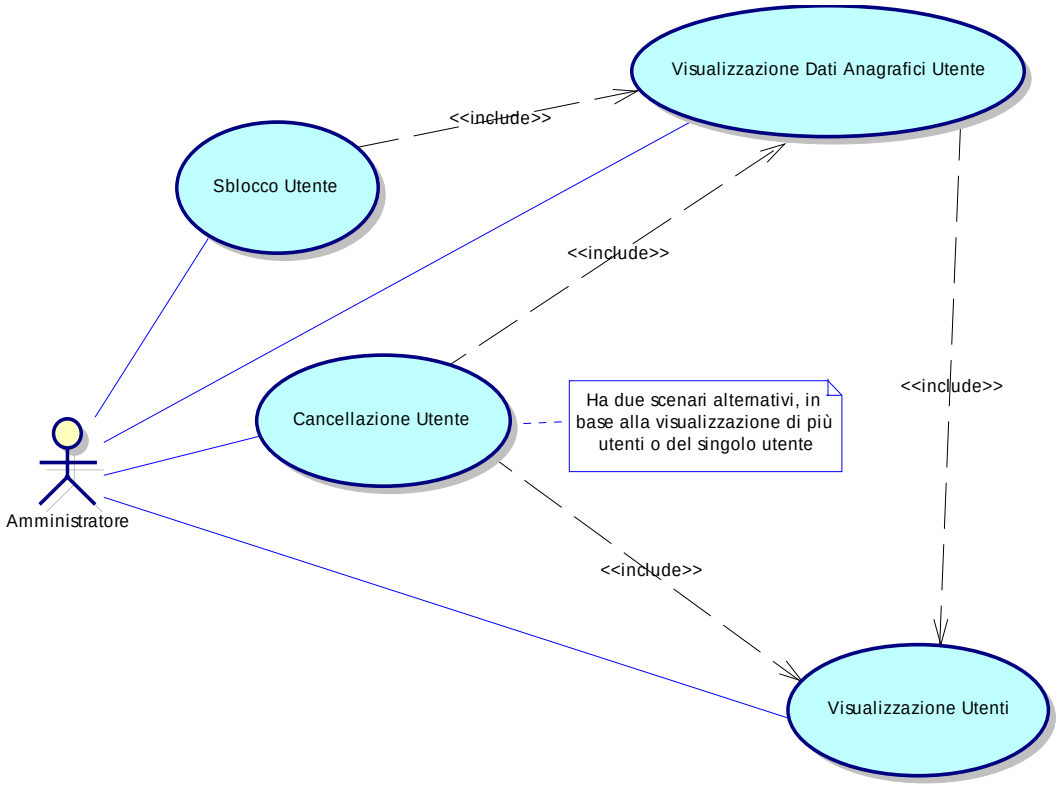


Figura 3 - Use Case Diagram Gestione Utenti

3.1.6 Gestione Alimenti

La modalità d'uso più complessa è certamente la "Gestione Alimenti", nell'ambito della quale, l'amministratore esegue tutte le operazioni di gestione dell'archivio del sistema contenente gli alimenti. In particolare, le operazioni possibili sono le seguenti :

- visualizzazione degli alimenti in base a dei criteri di ricerca opportunamente fissati;
- visualizzazione e modifica delle informazioni di un alimento;
- inserimento di un nuovo alimento in archivio;
- cancellazione di un alimento dall'archivio;

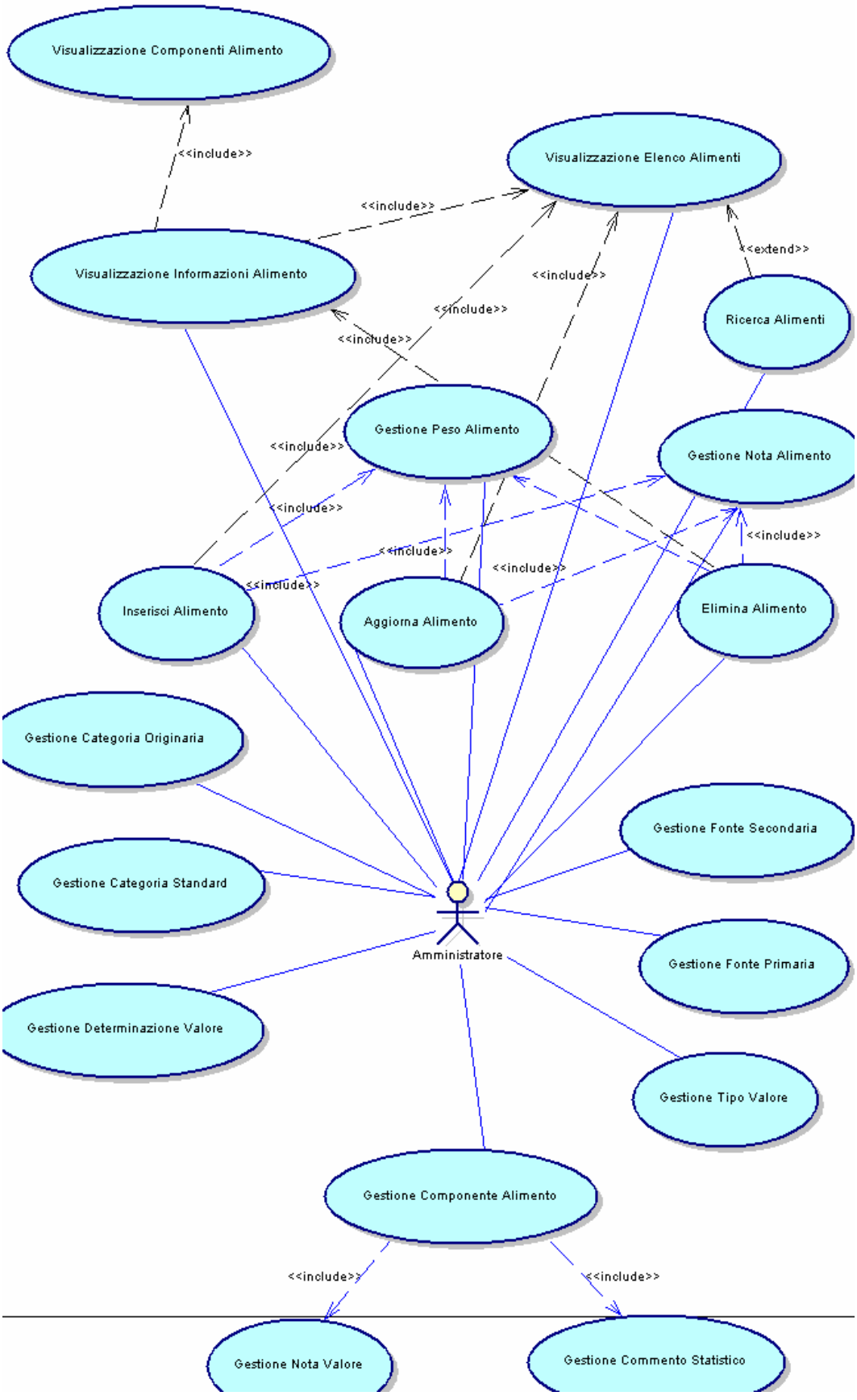


Figura 4 - Use Case Diagram Gestione Alimenti

3.1.7 Casi d'uso comuni : Login, Logout, Gestione Dati Personali

Infine, le modalità di utilizzo del sistema che sono comuni all'utente ed all'amministratore, riguardano il "Login Sistema" ed il "Logout Sistema" che possono essere considerate atomiche e la "Gestione Dati Personali" che comprende le operazioni di gestione dei propri dati di registrazione e di accesso per entrambi i ruoli. In particolare, l'amministratore ha la possibilità di modificare: la propria Password, la propria email ed il relativo SMTP. Per quanto riguarda l'utente, può modificare i suoi dati di accesso nonché i propri dati anagrafici. Un modo di utilizzo ulteriore disponibile soltanto all'utente, riguarda la possibilità di richiedere una mail contenente Username e Password di accesso in caso di dimenticanza.

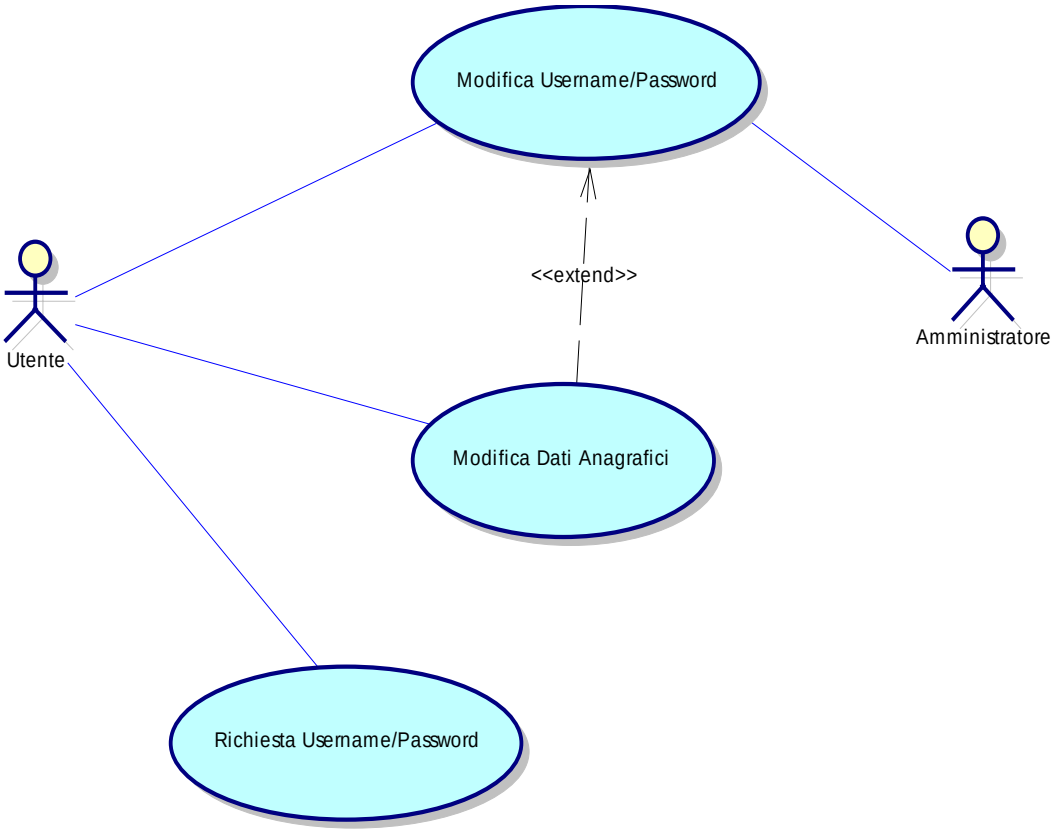


Figura 5 - Use Case Diagram Gestione Dati Personali

3.2 Class Diagram

Mediante il Class Diagram è possibile definire tutte quelle che sono le entità caratteristiche del sistema software e le relazioni che ci sono tra di esse. Ciascuna entità è ovviamente modellabile attraverso il concetto di classe, che ne definisce le caratteristiche (i dati) ed il comportamento (le operazioni).

Ovviamente, anche il Class Diagram può essere realizzato a diversi livelli di astrazione, passando da un livello elevato, definito di seguito, ad un livello più basso e strettamente legato alla fase di implementazione e quindi relazionato al framework ed al linguaggio utilizzato.

Attraverso questo diagramma è possibile modellare la vista statica del sistema, tenendo anche conto di quelle che sono le funzionalità offerte da quest'ultimo e rese disponibili attraverso le entità che lo compongono.

Sulla base delle specifiche definite per l'applicazione, il sistema prevede le seguenti entità e le relative classi che le modellano :

- *Ruolo* : definisce un generico utilizzatore del sistema, da distinguere tra utente ed amministratore;
- *Utente* : generico utente registrato che accede al sistema;
- *Amministratore* : amministratore del sistema;
- *Accesso* : classe astratta che contiene le informazioni relative all'accesso al sistema da parte di un utilizzatore qualsiasi;
- *AccessoUtente* : informazioni di accesso di un utente;
- *AccessoAmministratore* : informazioni di accesso dell'amministratore;
- *Alimento* : contiene le informazioni relative ad un singolo alimento presente nell'archivio;
- *CategoriaStandard :* informazioni relative alle categorie standard presenti nell'archivio (utilizzate per effettuare ricerche contemporanee su più fonti);
- *Determinazione Valore : informazioni relative alle determinazioni valori degli alimenti presenti nell'archivio;*
- *Nota Alimento : informazioni riguardanti le note di un alimento presenti nell'archivio;*
- *Tipo di Valore : informazioni riguardanti il tipo di valore di un alimento presente nell'archivio;*
- *Peso Alimento : informazioni riguardanti il peso di un alimento presente nell'archivio;*

- *Categoria Originaria : informazioni relative alle categorie originarie degli alimenti presenti nell'archivio;*
- *Fonte primaria : informazioni sulle fonti presenti nell'archivio;*
- *Origine Secondaria : informazioni sulle fonti secondarie dei componenti di un alimento presente nell'archivio;*
- *Fonte Secondaria : informazioni relative alle fonti secondarie presenti nell'archivio;*
- *Note Valore : informazioni sulle note dei componenti di un alimento presente nell'archivio;*
- *Componente Alimento : informazioni sui componenti di un alimento presente nell'archivio;*
- *Commento Statistico : informazioni sui commenti statistici memorizzati nell'archivio;*
- *Legame Valore Commento Stat : informazioni sui commenti di un componente presente nell'archivio;*
- *Componente : informazioni sui componenti memorizzati nell'archivio;*
- *Componente Standard : informazioni sui componenti standard memorizzati nell'archivio;*

Il diagramma definito di seguito si pone ad un livello di astrazione intermedio, nell'ambito del quale i membri di ciascuna classe (campi e metodi), fanno riferimento al linguaggio utilizzato nell'implementazione, ossia il Java. Ovviamente, astraendosi da quest'ultimo, è facile osservare che la struttura del sistema che ne scaturisce è completamente indipendente dall'implementazione.
In riferimento alle dipendenze che ci sono tra le varie classi, si possono fare le seguenti osservazioni :

- le classi *AccessoUtente* ed *AccessoAmministratore* costituiscono una specializzazione della classe *Accesso*, più da un punto di vista concettuale che pratico;
- la classe *Ruolo* generalizza le classi *Utente* ed *Amministratore*, in quanto queste ultime definiscono un particolare tipo di utilizzatore del sistema, l'uno diverso dall'altro ma con alcune caratteristiche comuni;

Le altre tipologie di relazioni previste sono auto-esplicative, in quanto esprimono il legame tra un'entità e le altre, sulla base delle specifiche e delle funzionalità del sistema.

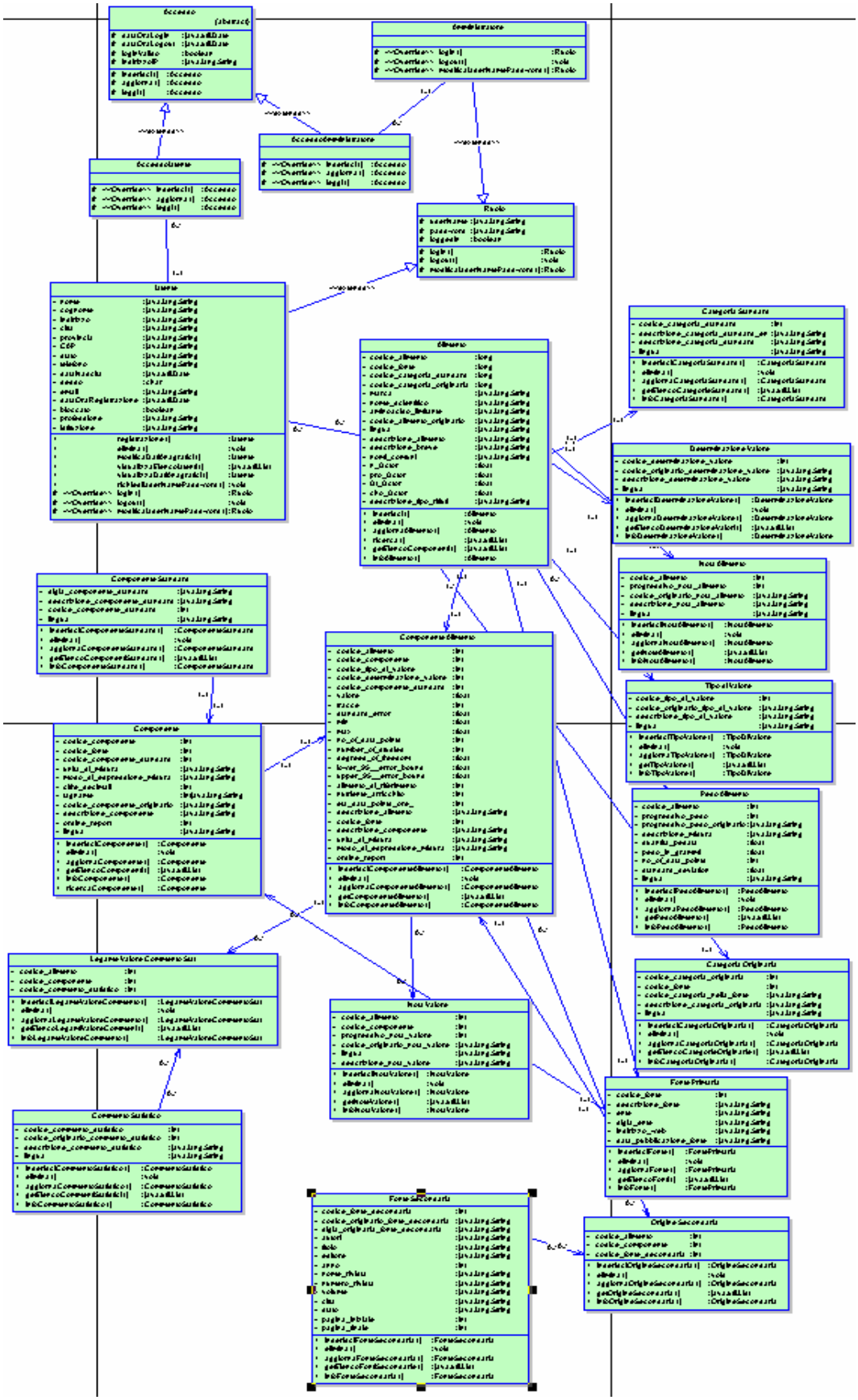


Figura 6 - Class Diagram

3.3 Activity Diagrams – Sequence Diagrams

Facendo riferimento a quelle che sono le funzionalità offerte dal sistema e le modalità di utilizzo di quest'ultimo, attraverso gli Activity Diagrams è possibile modellare le azioni necessarie per compiere un'attività ed il flusso di controllo tra di esse. Ciascuna "activity" rappresenta un'azione che viene eseguita dal sistema oppure dall'utilizzatore e può essere atomica oppure scomposta in più "action" elementari. Nella gestione del flusso di controllo, è possibile definire esecuzioni parallele di operazioni e la loro sincronizzazione, nonché esecuzioni condizionali. Inoltre, attraverso lo strumento dell' "object flow" è possibile specificare quali sono gli oggetti del sistema che entrano in gioco per ciascuna azione eseguita e quale sia il loro stato.

Invece, attraverso i Sequence Diagrams è possibile descrivere la vista dinamica del sistema, enfatizzando le interazioni dovute allo scambio di messaggi tra gli oggetti e il loro ordinamento temporale. Ciascun diagramma evidenzia il modo in cui uno scenario, ossia uno specifico percorso in un caso d'uso, viene risolto dalla collaborazione tra un insieme di oggetti. Generalmente, però, al posto di realizzare un diagramma per ciascuno scenario, si preferisce definire un unico diagramma all'interno del quale vengono descritte le situazioni alternative, facendo uso dei numeri di sequenza sui messaggi.

E' da osservare che nell'ambito dei Sequence Diagrams sono previsti degli oggetti di tipo "Interfaccia", che definiscono le parti del sistema che interagiscono con l'utente, ossia la cosiddetta UI (User Interface). In questa fase, non viene specificata la tipologia di interfaccia utente, in modo da poter sfruttare i diagrammi realizzati anche con implementazioni diverse. E' ovvio che, nel caso di una Web application, l'interfaccia utente sia caratterizzata da pagine Web e dai relativi form che le compongono.

E' stato definito un Activity Diagram di carattere generale, che si pone ad un elevato livello di astrazione e che descrive tutte le possibili azioni ed il relativo flusso, che possono essere eseguite da un utente non registrato, da un utente registrato ed ovviamente dall'amministratore.

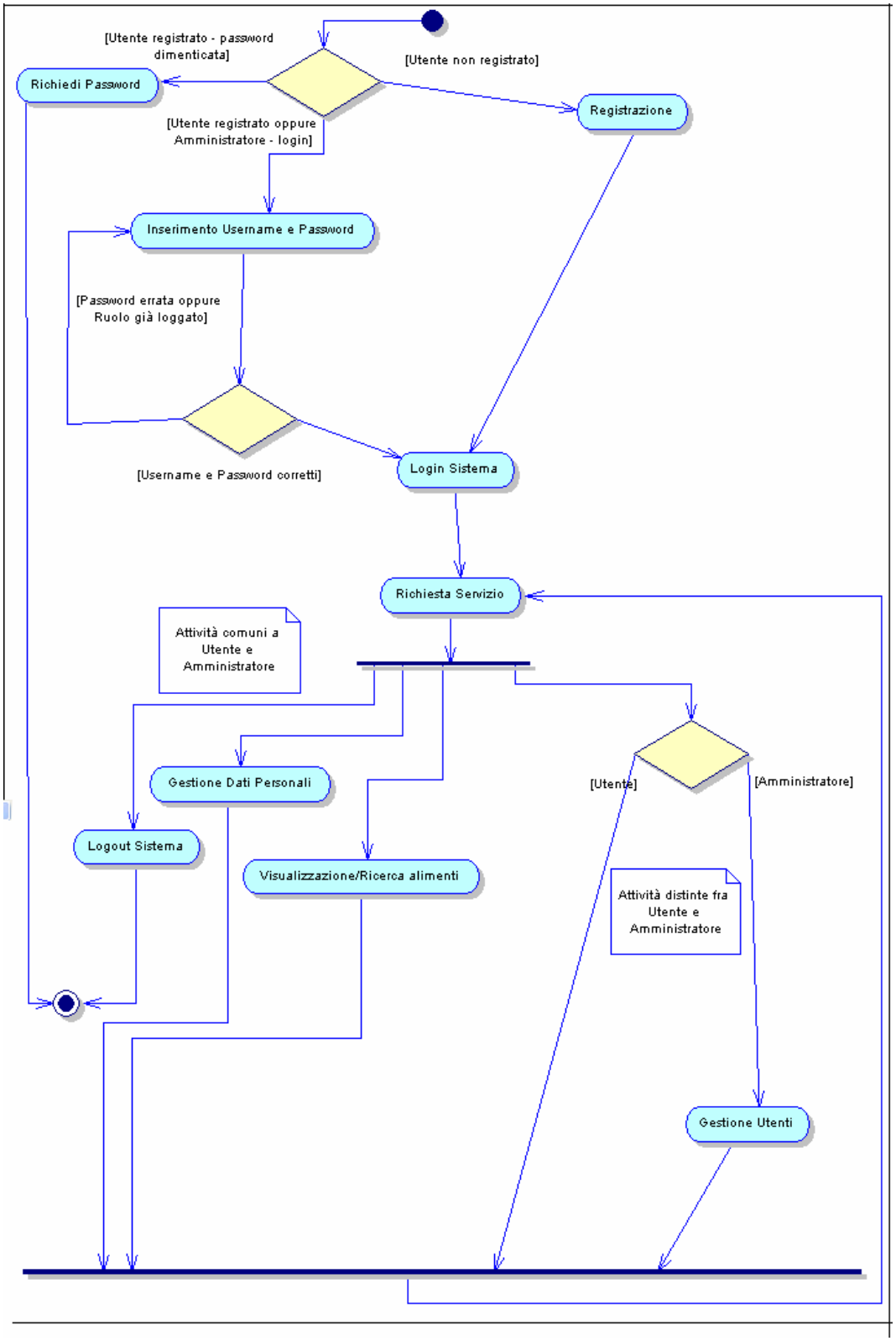


Figura 7 - Activity Diagram Generale

A partire da esso, è possibile scendere ad un livello di dettaglio maggiore, specificando la sequenza delle azioni per ogni funzionalità del sistema.

3.3.1 Login

Per quanto riguarda il Login dell'utente e dell'amministratore, le azioni da eseguire sono praticamente le stesse a meno degli oggetti che entrano in gioco.
Una volta inseriti Username e Password, il sistema ne effettua il controllo della correttezza, valutando in primo luogo se lo Username esiste e successivamente verificando la validità della Password. Ovviamente, a colui che tenta di accedere, viene sempre dato un messaggio generico di errore, senza mai specificare quale dei due parametri sia errato.
Per evitare tentativi di accesso non autorizzati, dopo che l'utente si è registrato, il sistema blocca tale account, inviando un email all'amministratore informandolo che è stata effettuata una nuova registrazione, e finché l'amministratore non sbloccherà tale utente, l'account resterà bloccato.
Ovviamente, nel caso in cui Username e Password siano corretti, si valuta comunque se è attivo un blocco sull'account; nel caso di login consentito, si esegue l'aggiornamento delle informazioni di accesso.
La monitorizzazione e registrazione degli accessi, validi o non validi, relativi agli utilizzatori della Web application, può essere considerato uno strumento mediante il quale sia possibile eseguire delle statistiche o comunque ricercare eventuali frodi e tentativi di accesso non legittimi al sistema.
Nell'ambito del Sequence Diagram, è possibile osservare lo scenario di "Username errata" e quello di "Username corretta", nel quale si possono verificare le seguenti situazioni distinte :

- Password errata ;
- Password corretta ma l'utilizzatore è bloccato;
- Utilizzatore già loggato;
- Login consentito;

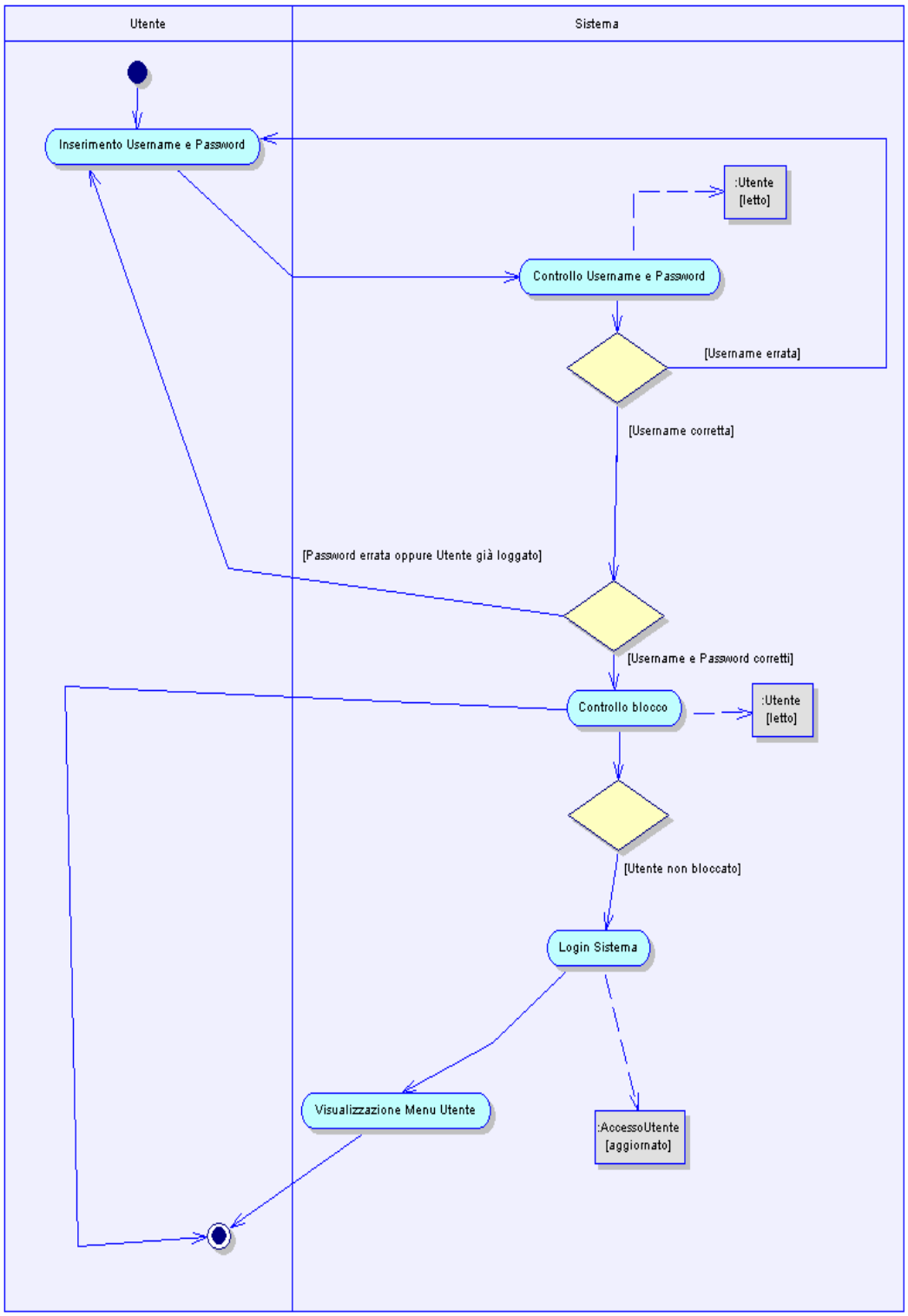


Figura 8 - Activity Diagram Login Utente

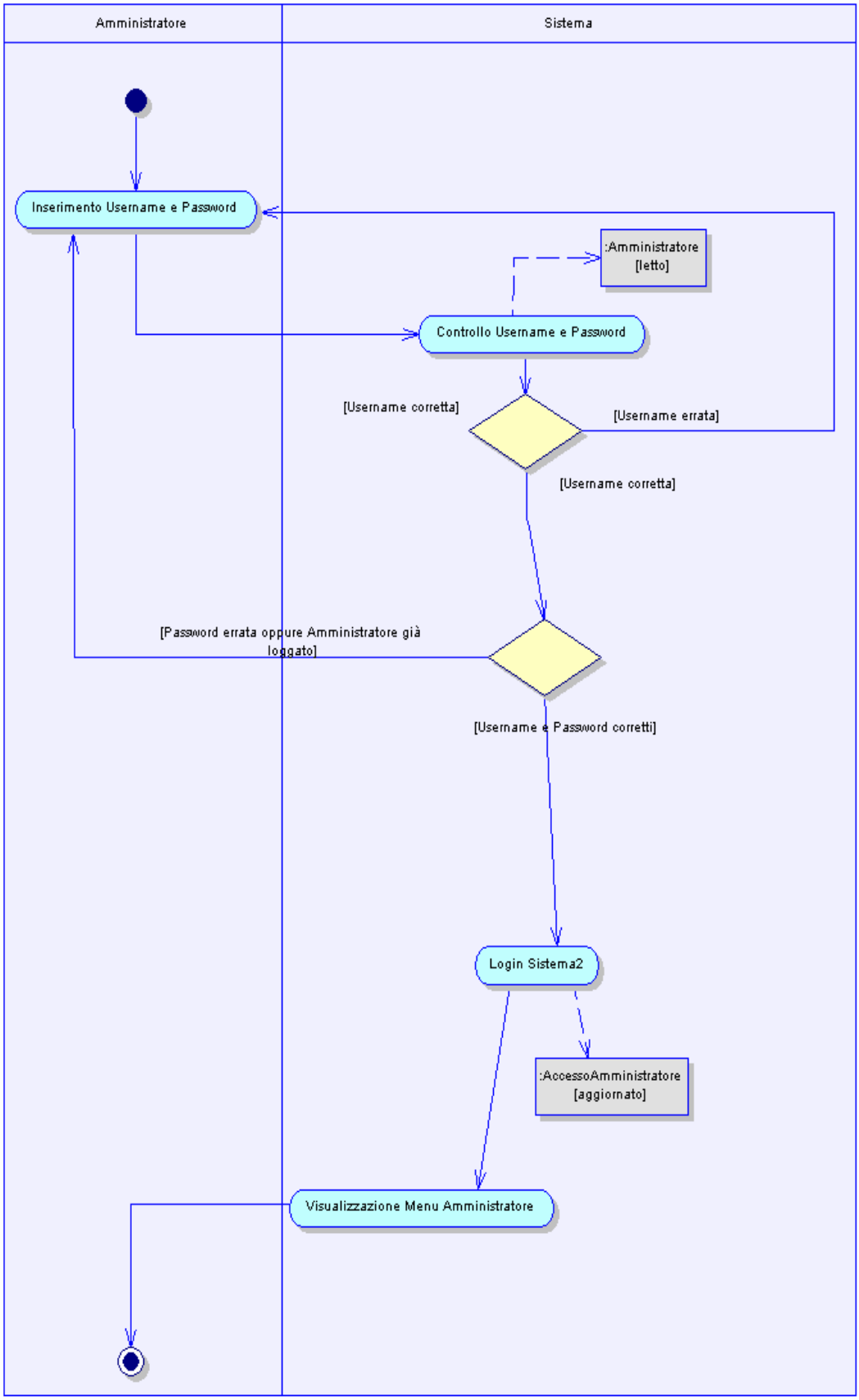


Figura 9 - Activity Diagram Login Amministratore

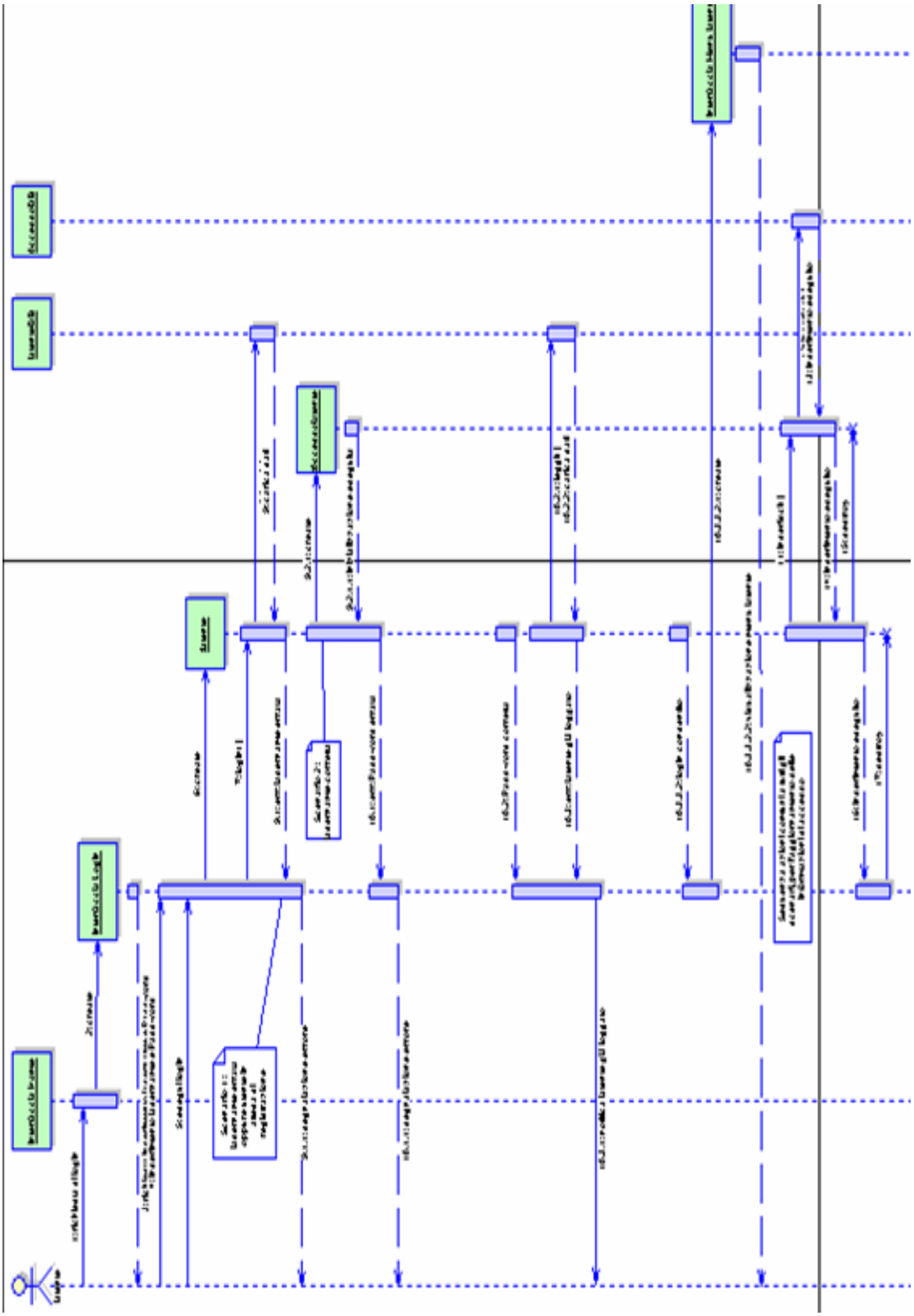


Figura 10 - Sequence Diagram Login Utente

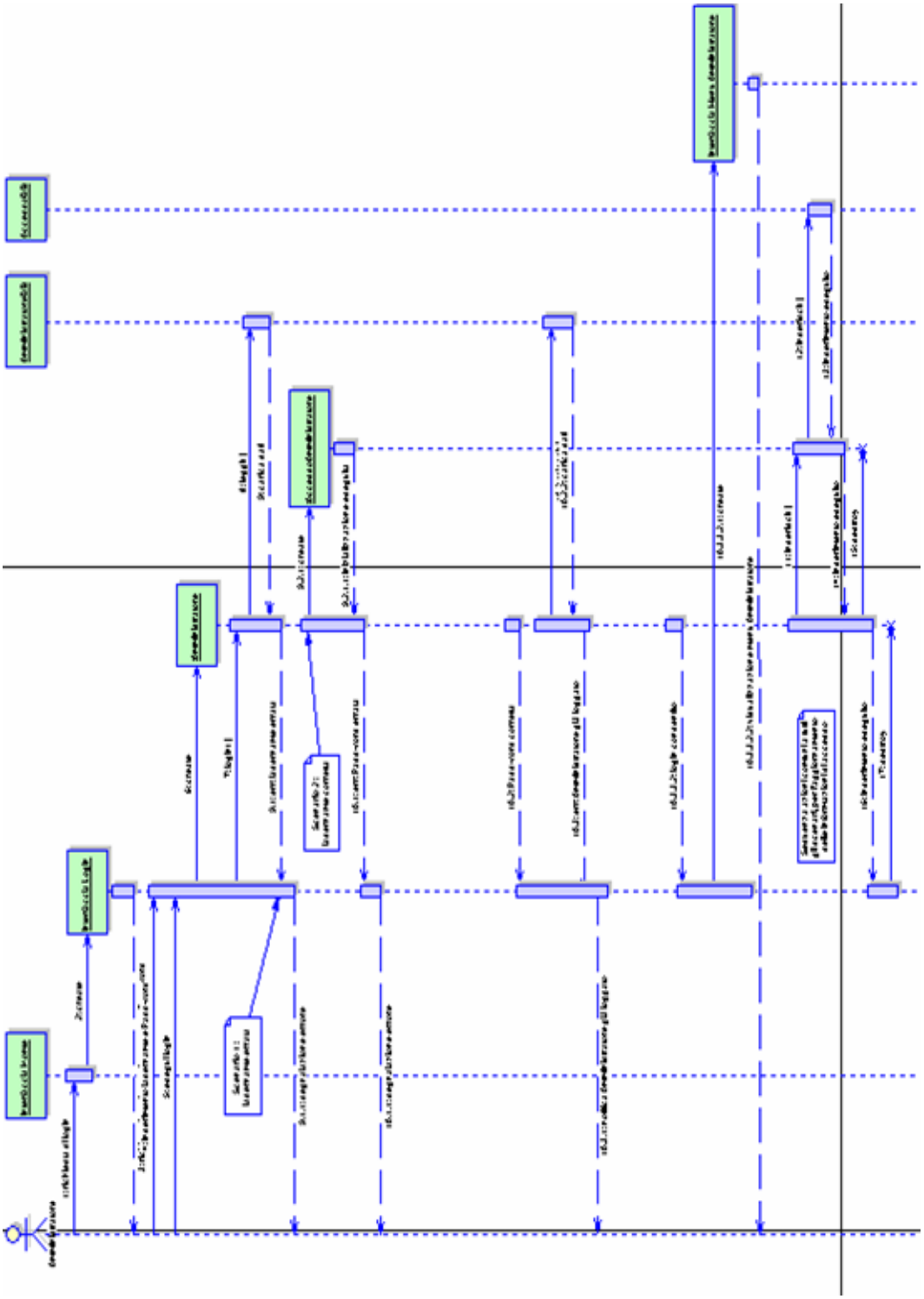


Figura 11 - Sequence Diagram Login Amministratore

3.3.2 Logout

Anche la funzionalità Logout prevede le medesime azioni per l'utente e l'amministratore, a meno degli oggetti che vengono utilizzati per le operazioni necessarie.

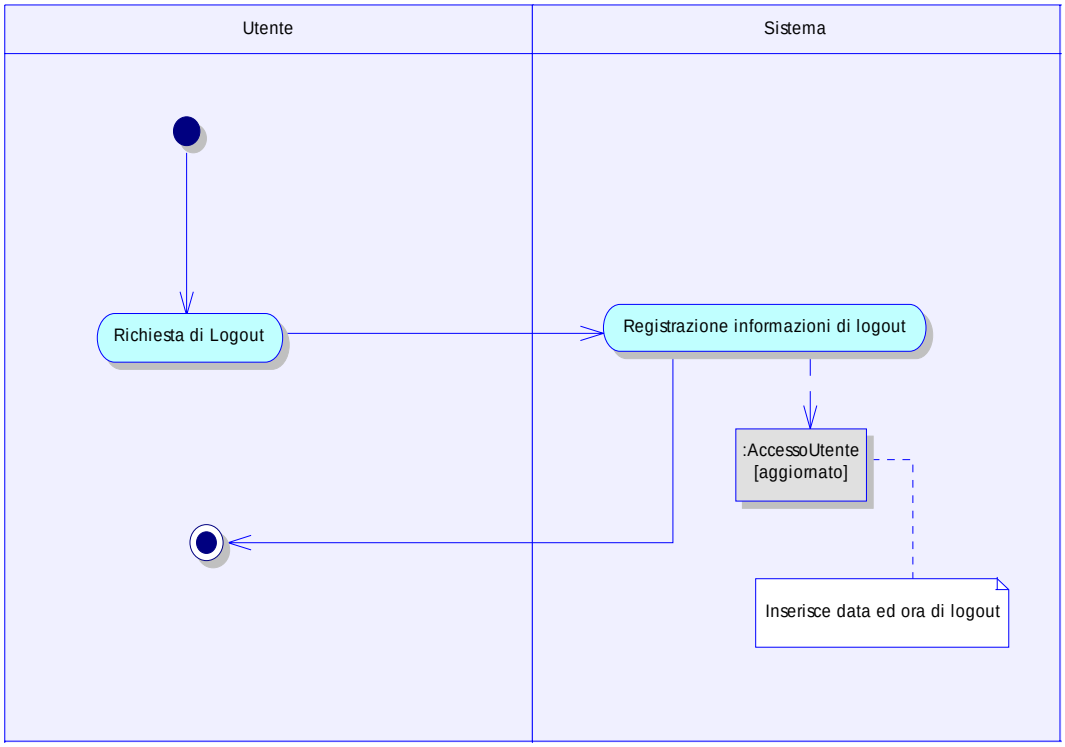


Figura 12 - Activity Diagram Logout Utente

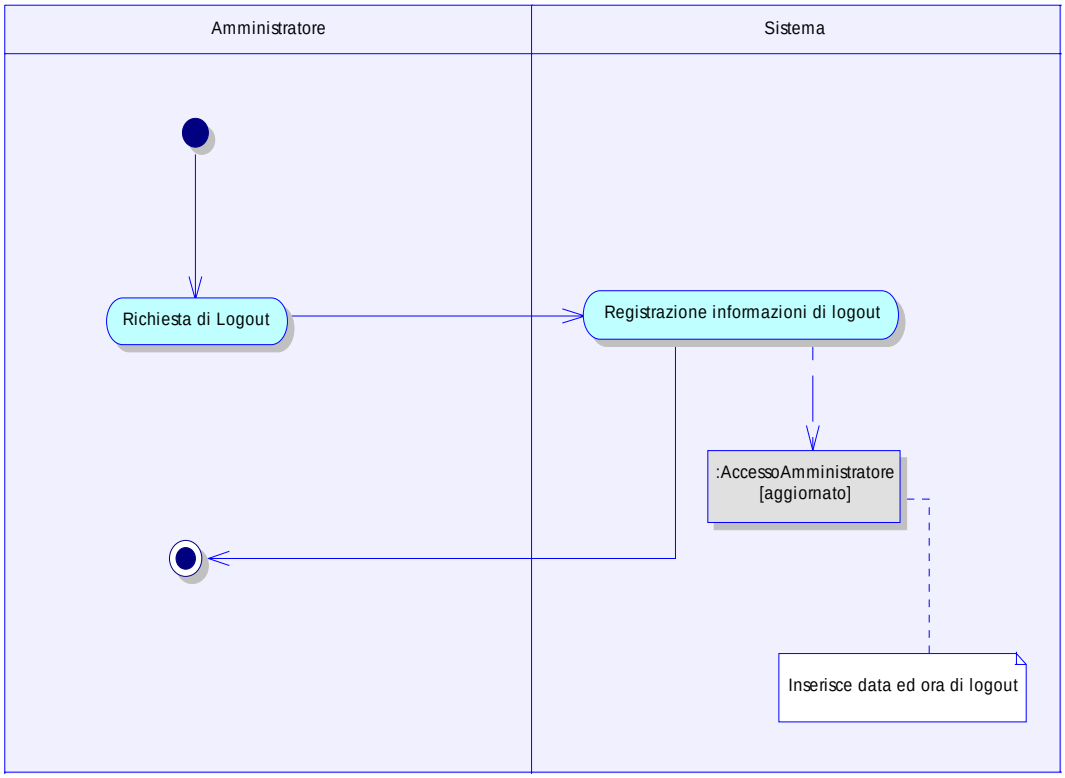


Figura 13 - Activity Diagram Logout Amministratore

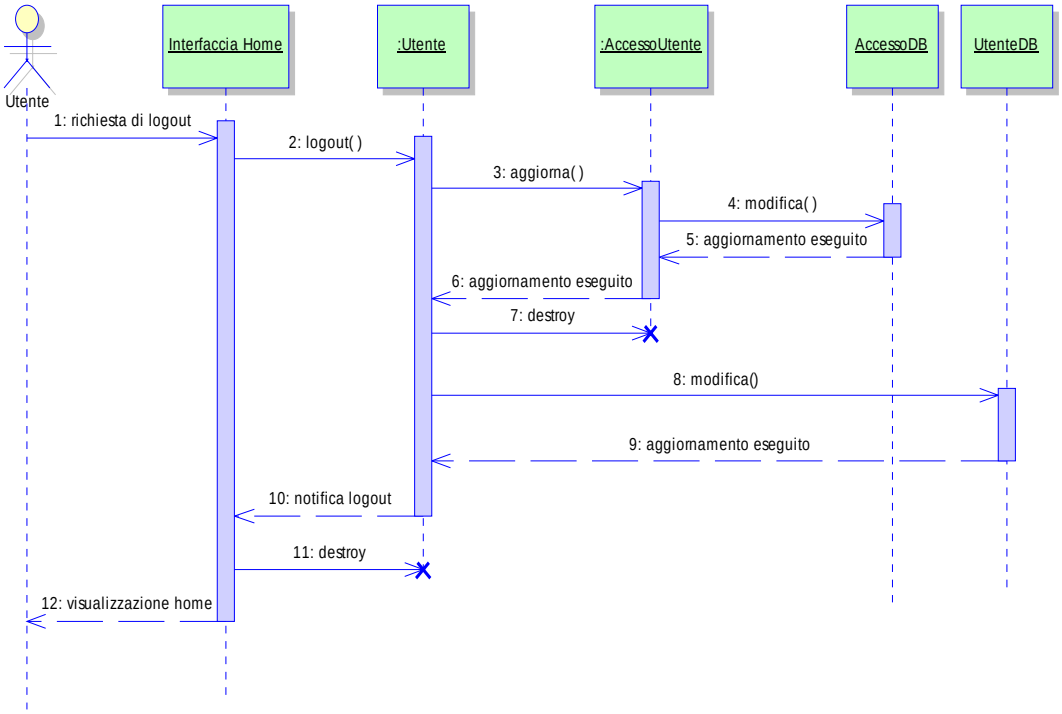


Figura 24 - Sequence Diagram Logout Utente

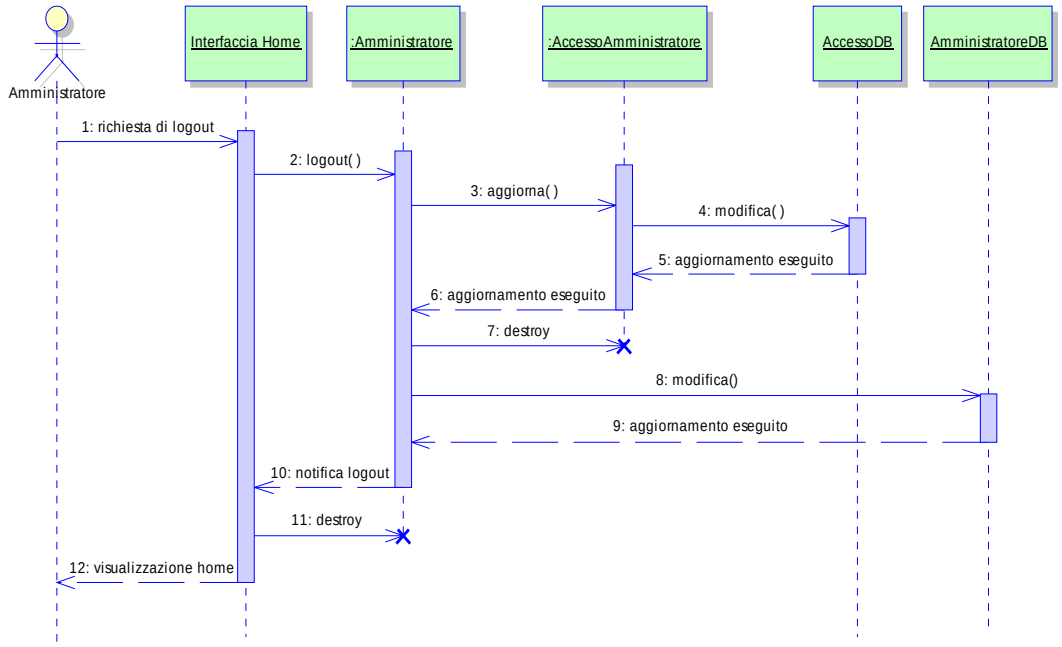


Figura 35 - Sequence Diagram Logout Amministratore

3.3.3 Registrazione

La registrazione dell'utente, per poter usufruire dei servizi della Web application, prevede l'immissione da parte dell'utente: dei dati di accesso, Username e Password, oltre che dell'indirizzo email e dei propri dati anagrafici.
Ovviamente, sono previste tutte le operazioni di validazione dei dati, necessarie a garantire che i dati immessi dall'utente rispettino i formati attesi dall'applicazione.
Al termine della registrazione, viene bloccato tale account e viene inviata un email all'amministratore contenente la richiesta di registrazione di quell'utente; tale account resterà bloccato finché l'amministratore non provvederà a sbloccarlo, inviando in questo caso un email all'utente indicandogli che l'accesso è stato convalidato.
Per quanto riguarda il Sequence Diagram, si evidenzia il possibile scenario relativo alla registrazione dell'utente.

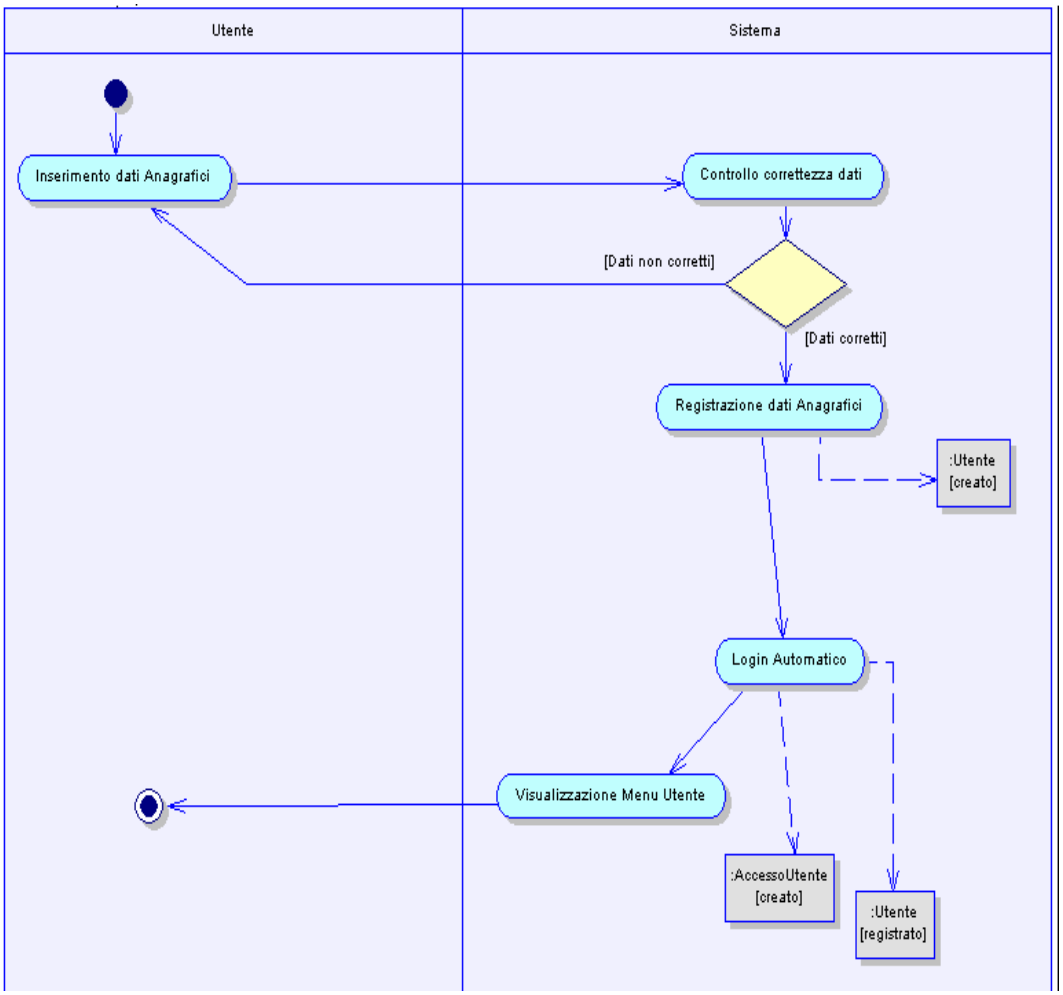


Figura 46 - Activity Diagram Registrazione

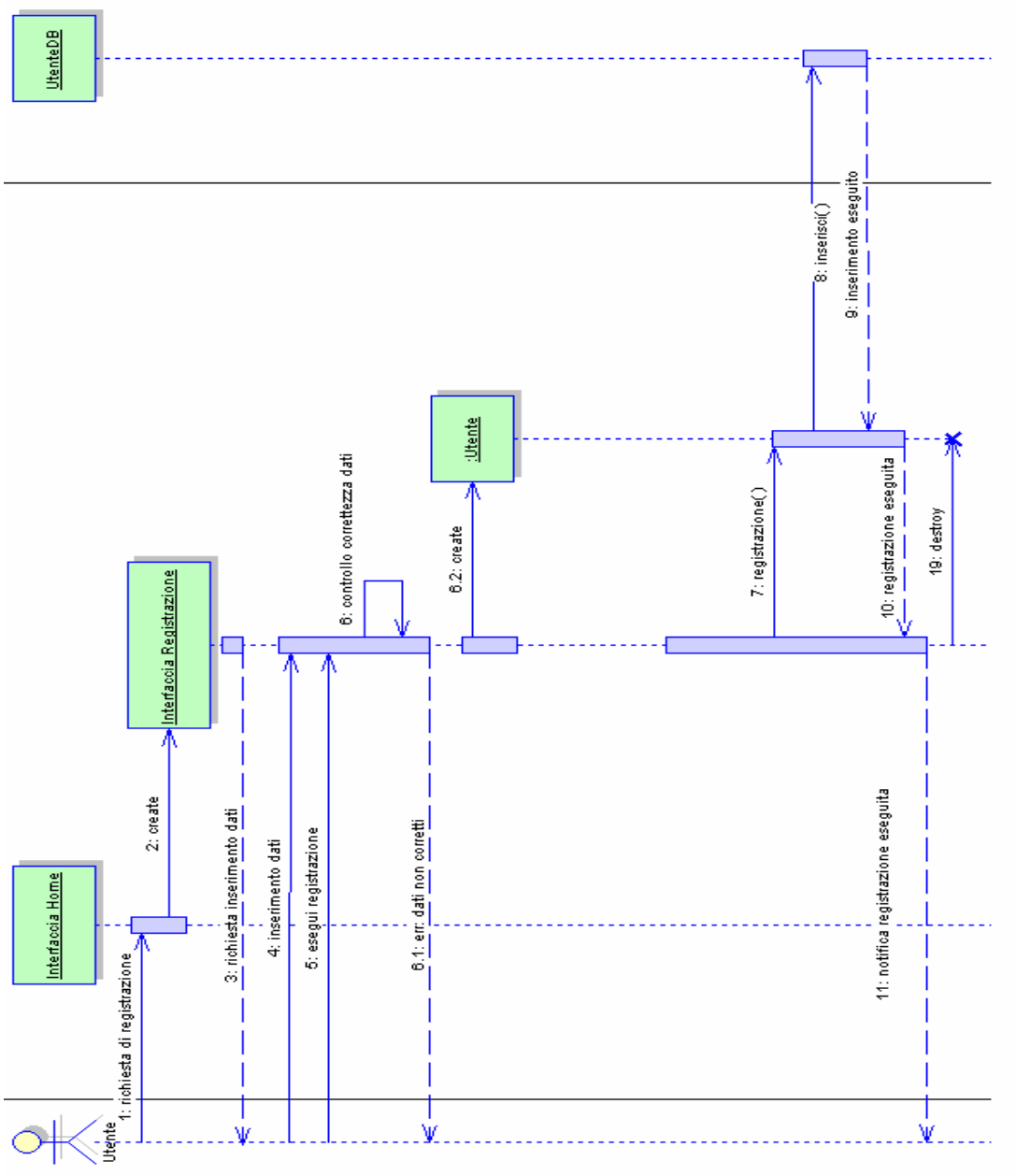


Figura 57 - Sequence Diagram Registrazione

3.3.4 Richiesta Username / Password

Nel caso in cui un utente registrato abbia perso oppure dimenticato le informazioni relative all'accesso, è possibile utilizzare la funzionalità del sistema che permette di inviare una mail contenente tali informazioni.

L'utente deve fornire al sistema l'indirizzo email con cui ha effettuato la registrazione, dopodiché il sistema stesso eseguirà il controllo di correttezza del formato dell'indirizzo e verifica che quest'ultimo sia correttamente registrato.

In caso di esito positivo, l'utente riceverà una mail con tutte le informazioni richieste.

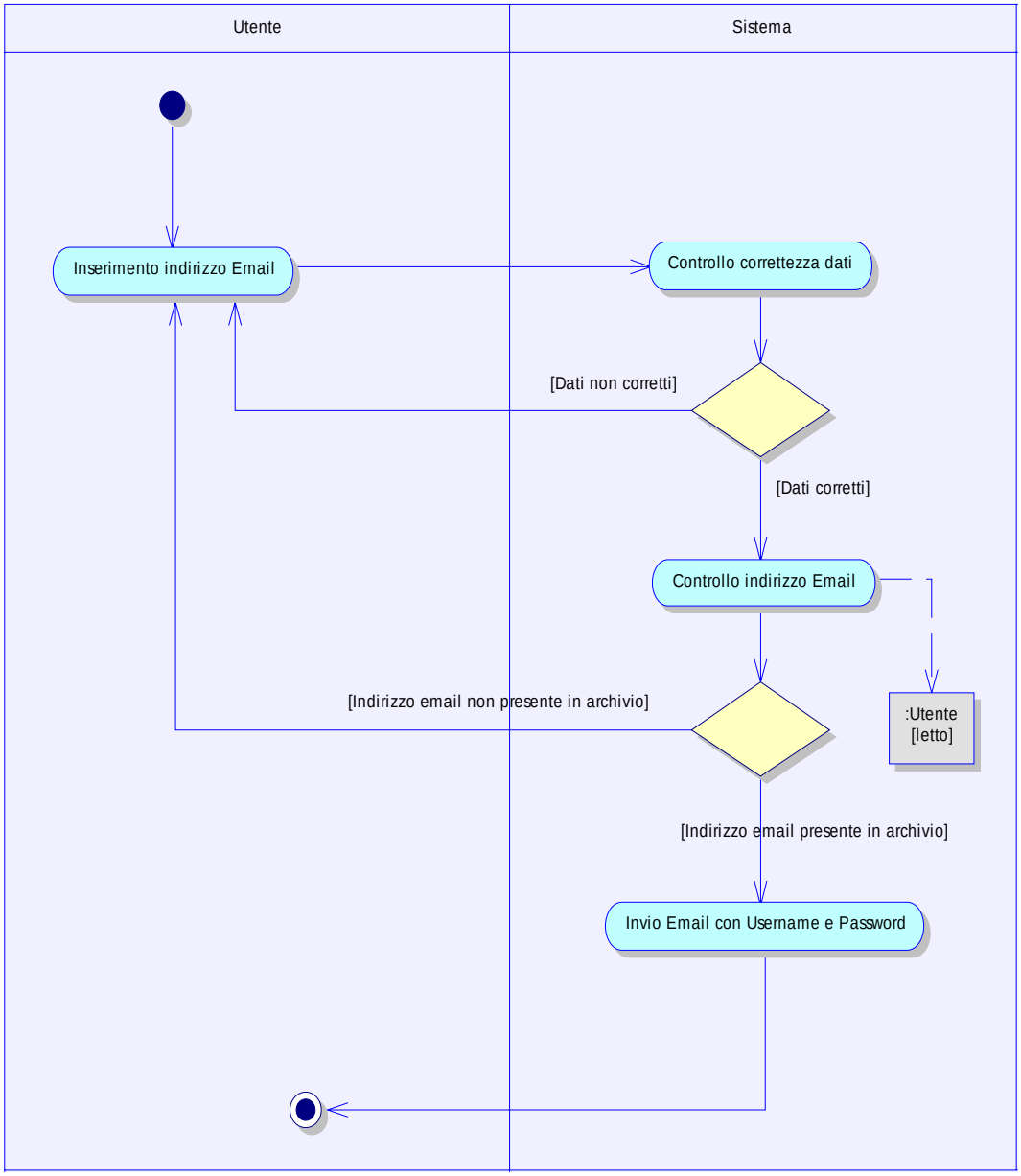


Figura 18 - Activity Diagram Richiesta Username/Password

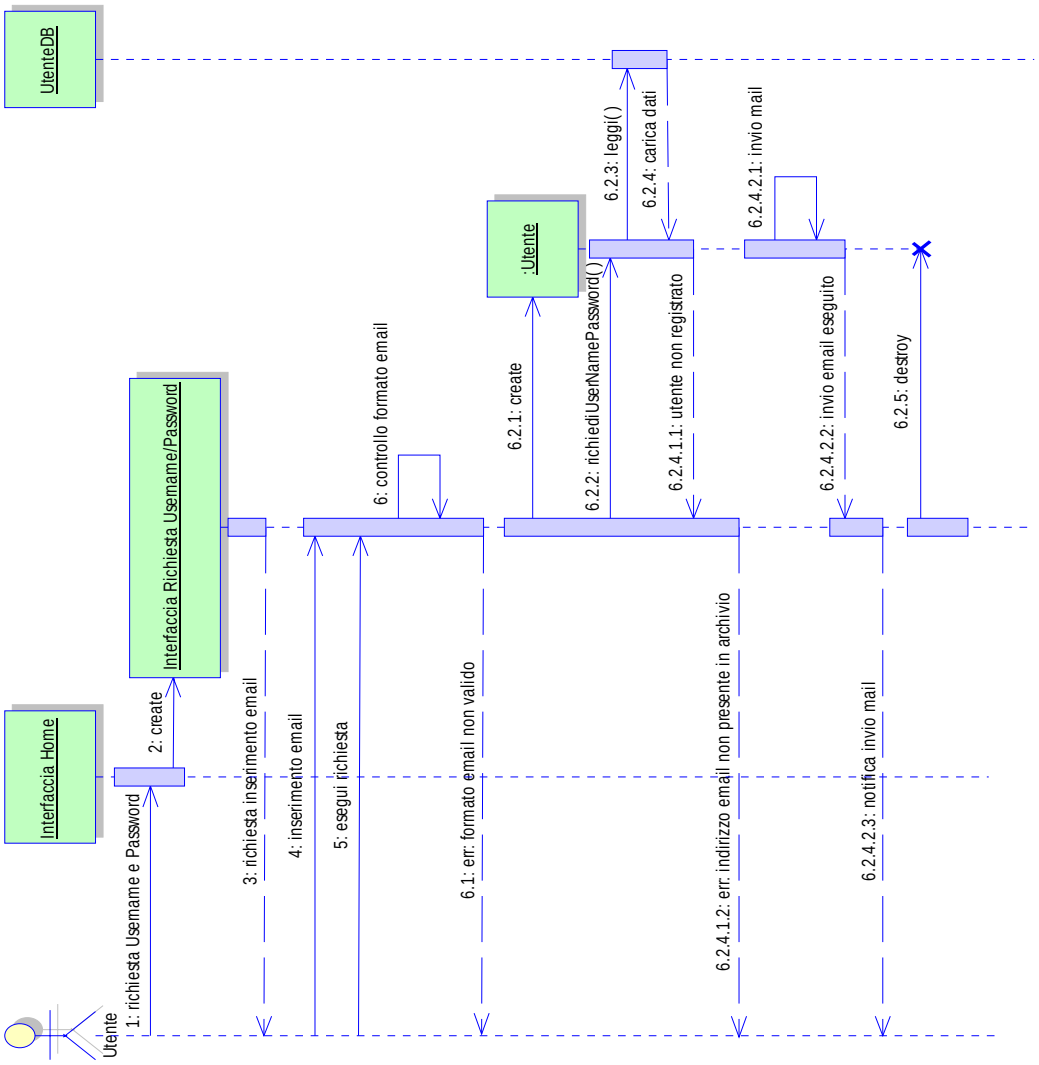


Figura 19 - Sequence Diagram Richiesta Username/Password

3.3.5 Visualizzazione e Ricerca Alimenti

Per poter accedere all'archivio degli alimenti e visualizzarne le informazioni, il sistema mette a disposizioni l'operazione di ricerca sulla base di criteri impostati relativi alla descrizione dell'alimento, alla categoria, ed ai componenti.

Gli Activity Diagrams che descrivono queste funzionalità sono due, in quanto fanno riferimento alla richiesta da parte dell'utilizzatore ed alle operazioni eseguite dal sistema. Si è preferita una scomposizione in questi termini, poiché tali diagrammi diventano componenti dei diagrammi che descrivono funzionalità più complesse che prevedono la visualizzazione degli alimenti. In questo modo si riesce a modellare il sistema mediante livelli di astrazione successivi.

Il Sequence Diagram, invece, evidenzia il possibile scenario sulla base del quale l'utilizzatore può richiedere la visualizzazione degli alimenti presenti nell'archivio.

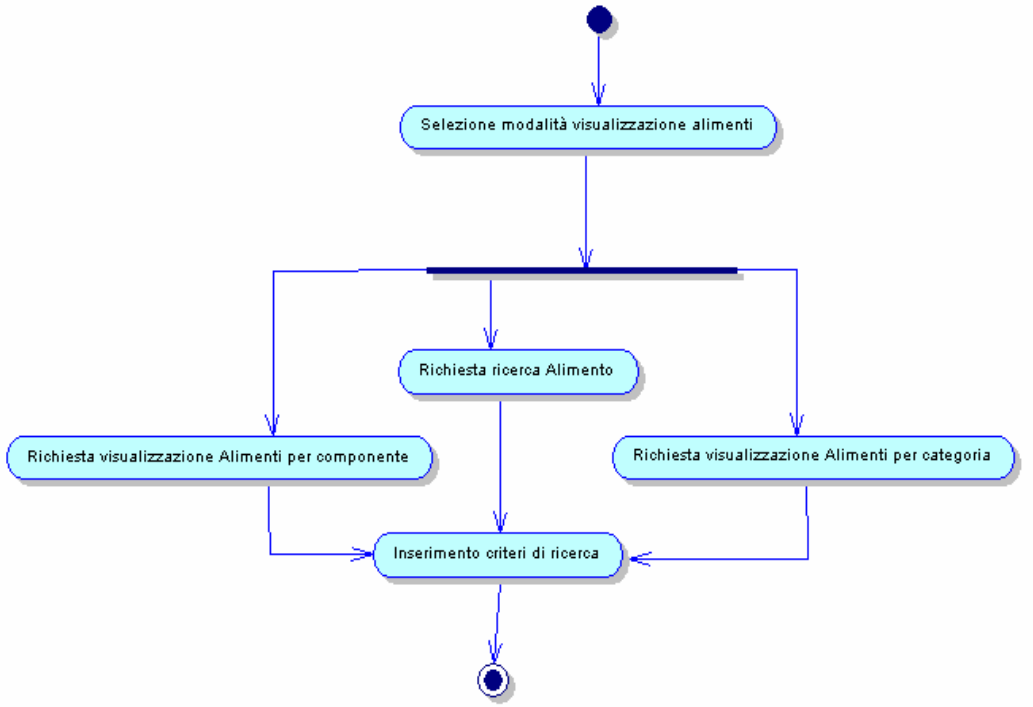


Figura 20 - Activity Diagram Richiesta Visualizzazione

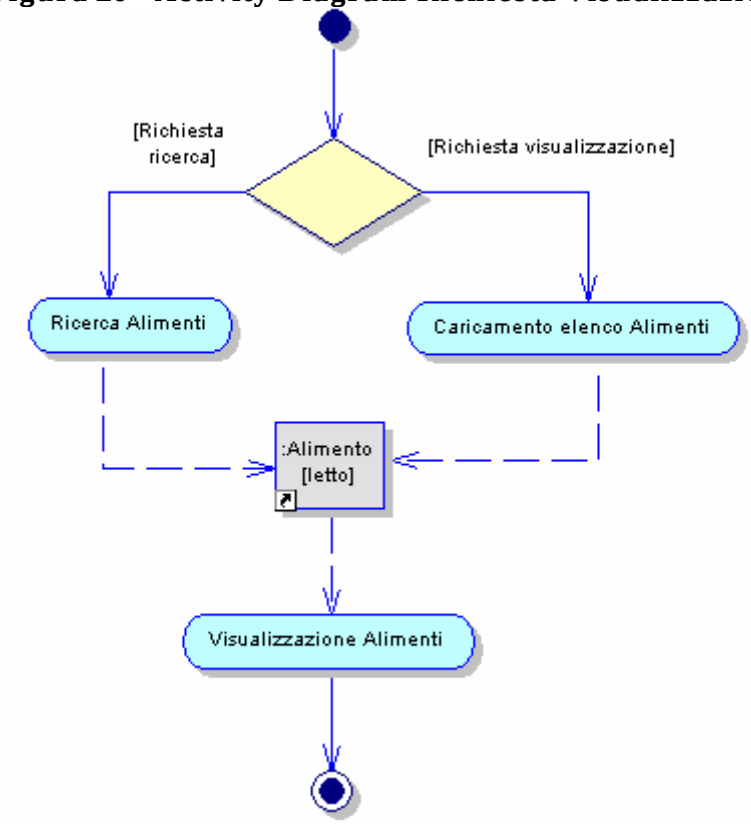


Figura 61 - Activity Diagram Ricerca Alimenti

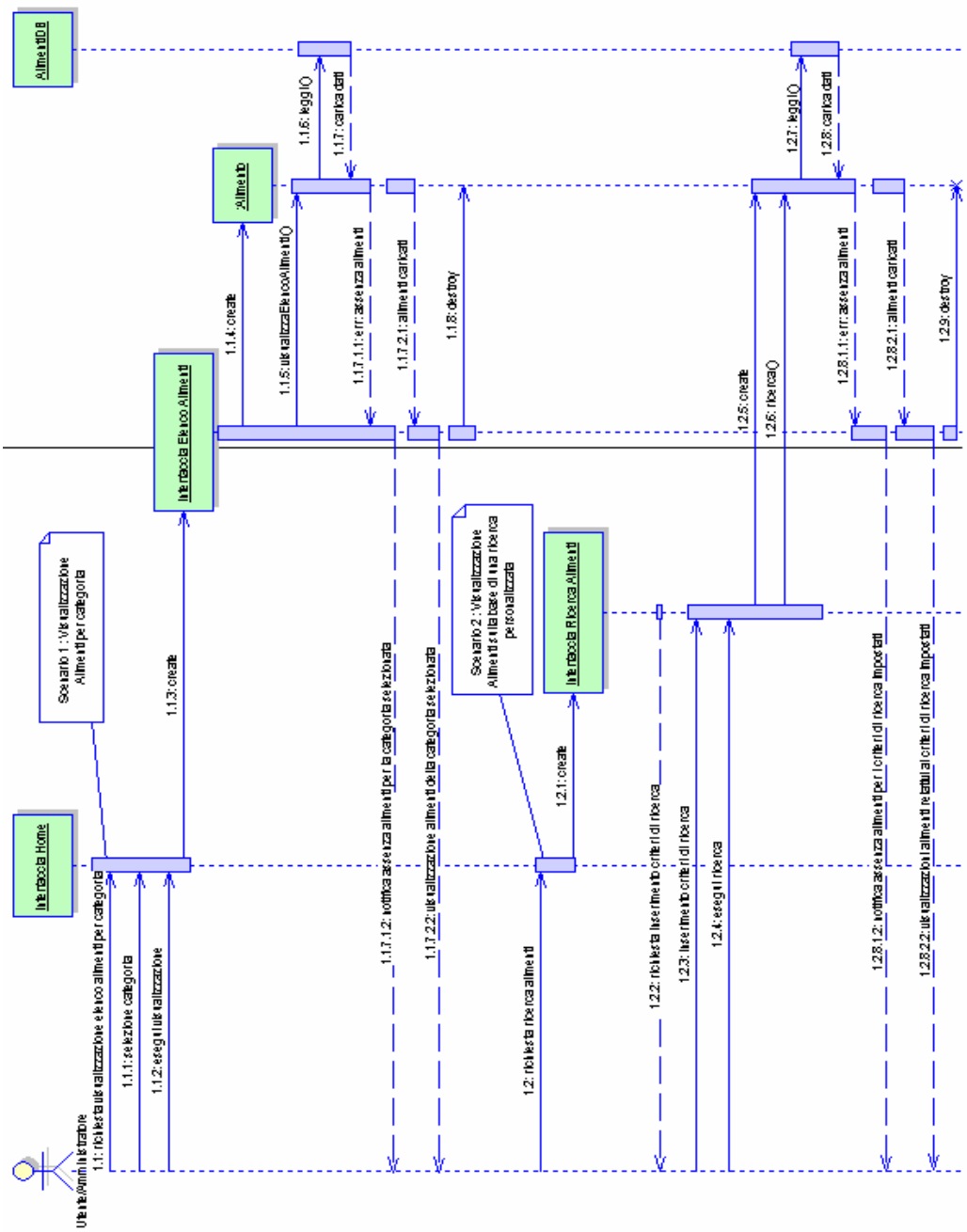


Figura 22 - Sequence Diagram Visualizzazione e ricerca Alimenti

3.3.6 Inserimento, Modifica ed Eliminazione Alimento

Nell'ambito della gestione dell'archivio degli alimenti, l'amministratore ha ovviamente la possibilità di inserire un nuovo alimento. Le informazioni richieste dal sistema riguardano la descrizione dell'alimento, la fonte, il codice dell'alimento originario, il nome scientifico, il nome comune, marca, aminoacido limitante, la descrizione del tipo di rifiuti, la descrizione breve, la lingua dell'alimento, la categoria originaria, la categoria standard, le note dell'alimento,

il peso dell'alimento, Fattore per la Conversione dell'Azoto in Proteine, Fattore per il Calcolo delle Calorie da Proteine. Fattore per il Calcolo delle Calorie da Grassi, Fattore per il Calcolo delle Calorie da Carboidrati; mentre per i componenti che lo costituiscono, le informazioni richieste dal sistema sono : la descrizione del componente, il valore, le tracce, l'unità di misura, il modo di espressione della misura, la sigla del componente standard, il minimo, il massimo, lo standard error, il numero di data points per ore, il lower 95% error bound, l'upper 95% error bound, il number of studies, degresss of freedom, nutriente arricchito, l'alimento di riferimento, la determinazione del valore, il tipo di valore, la nota del valore, la fonte secondaria ed i commenti statistici.

Le informazioni suddette non sono tutte strettamente necessarie, se non la descrizione dell'alimento, la fonte, il codice dell'alimento originario per quanto riguarda l'alimento; invece per il componente dell'alimento le informazioni necessarie sono la descrizione del componente, il valore e le tracce.

Il Sequence Diagram evidenzia il caricamento del singolo alimento attraverso il seguente scenario.

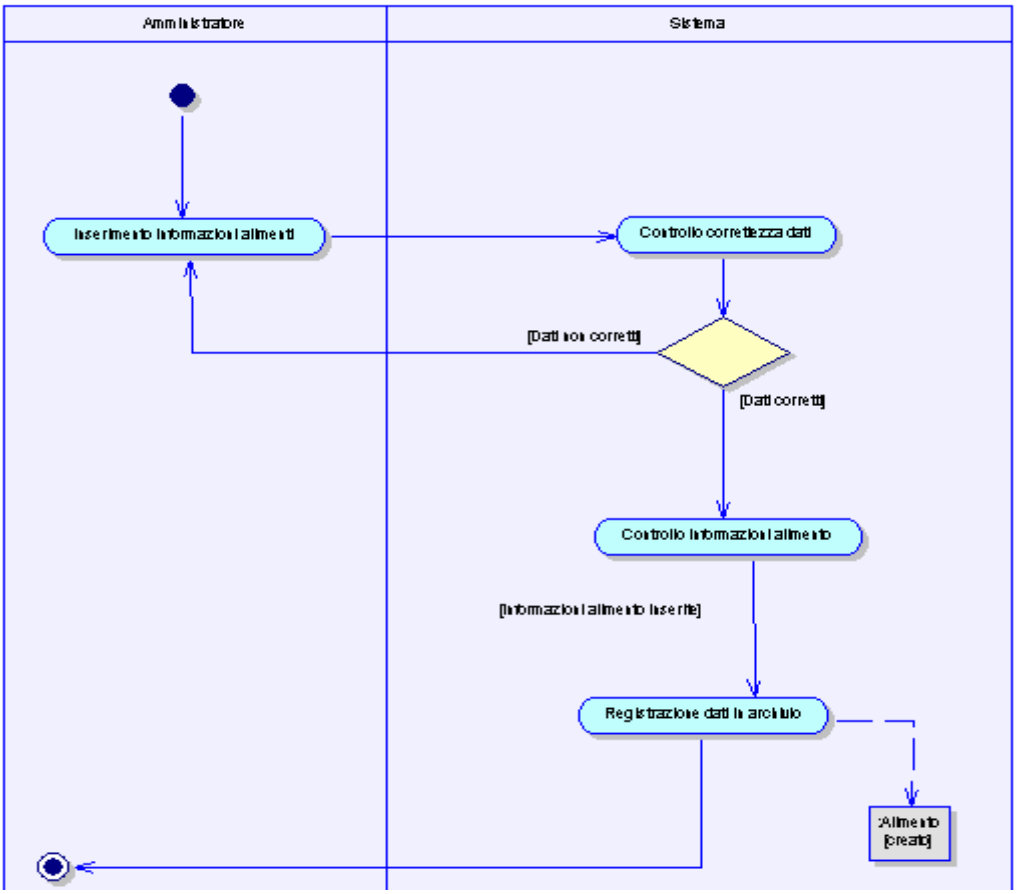


Figura 23 - Activity Diagram Inserimento Alimento in archivio

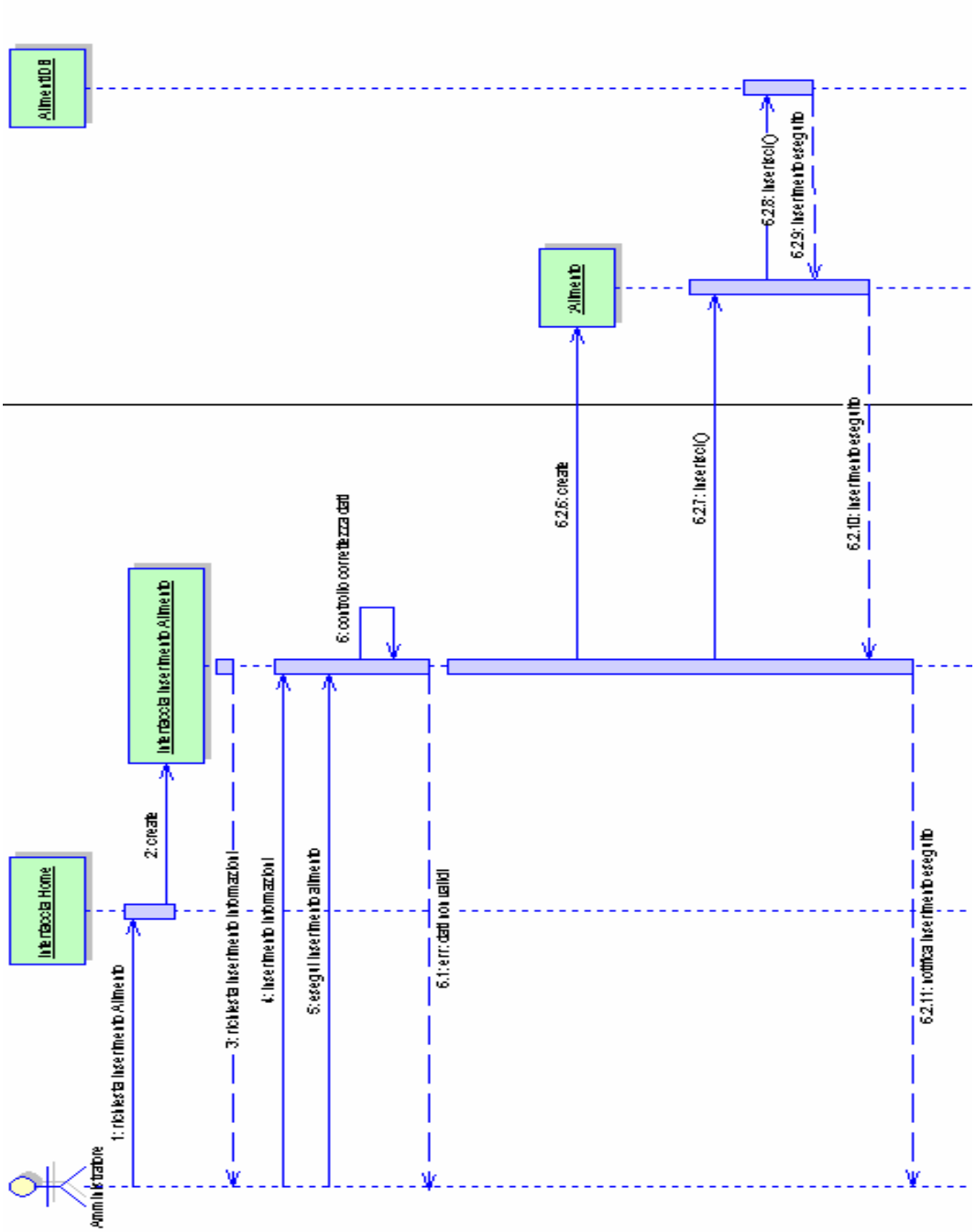


Figura 24 - Sequence Diagram Inserimento alimento in archivio

Una volta che un alimento è stato inserito, è ovviamente disponibile una funzionalità mediante la quale l'amministratore può visualizzare e modificare le informazioni dello stesso.

La visualizzazione dei dati prevede un Activity Diagram che diventa componente in quello di modifica, considerando che la funzionalità di visualizzazione è utilizzata in molteplici situazioni.

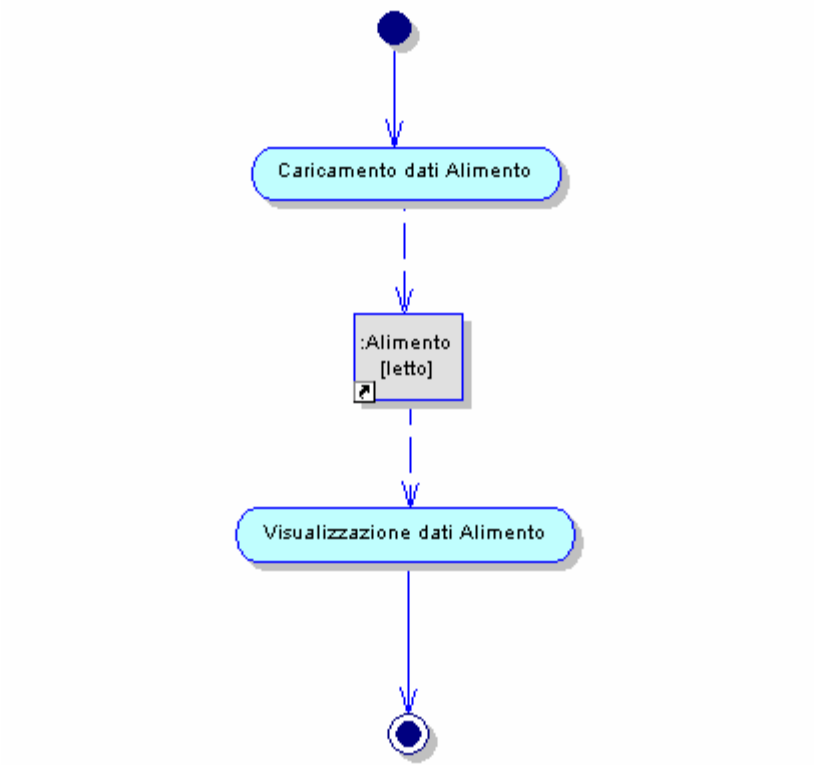


Figura 25 - Activity Diagram Visualizzazione info alimento

Inoltre, l'Activity Diagram relativo alla funzionalità di modifica sfrutta i diagrammi di visualizzazione e ricerca degli alimenti, tra i quali selezionare un alimento di cui modificarne le informazioni.

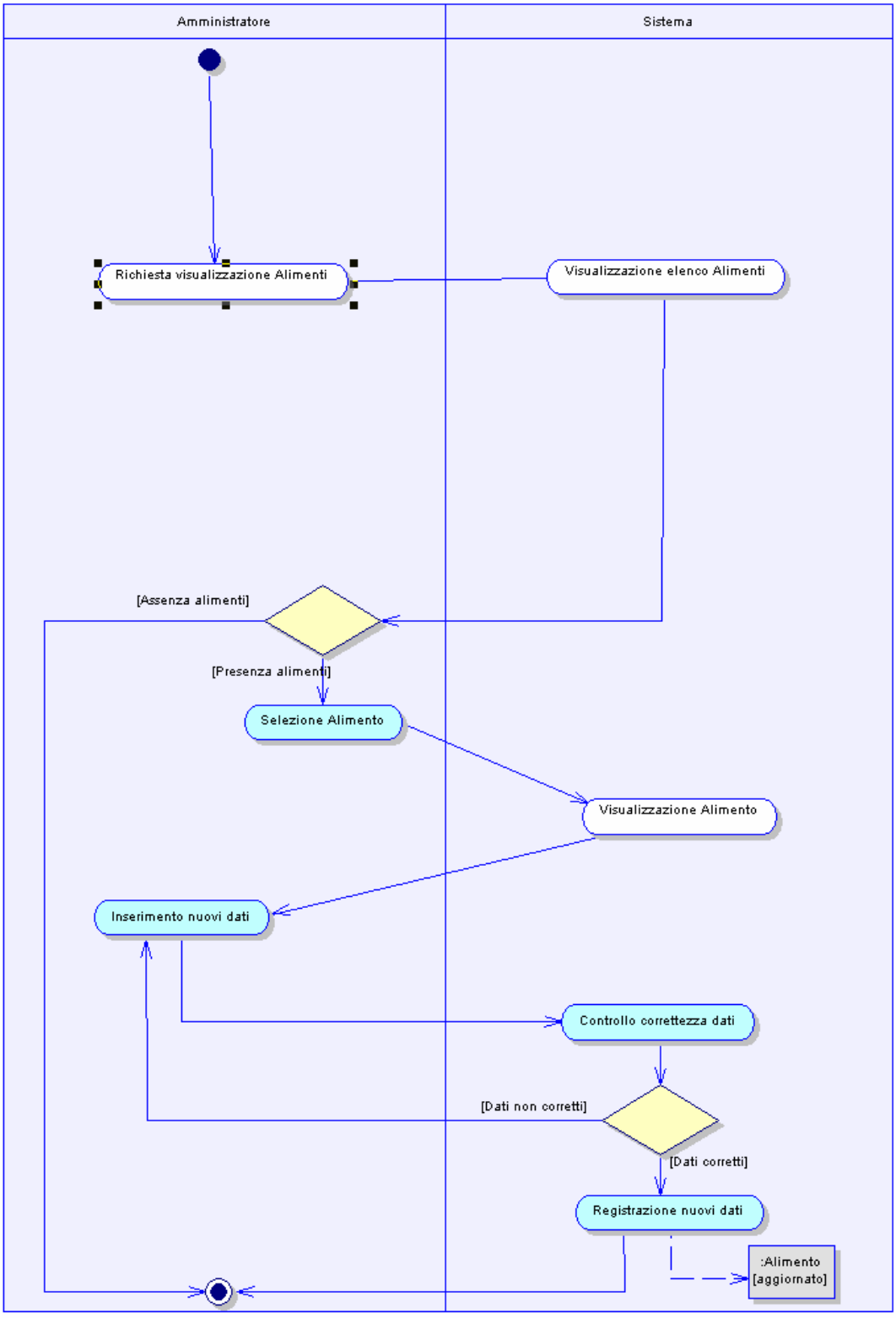


Figura 26 - Activity Diagram Modifica alimento

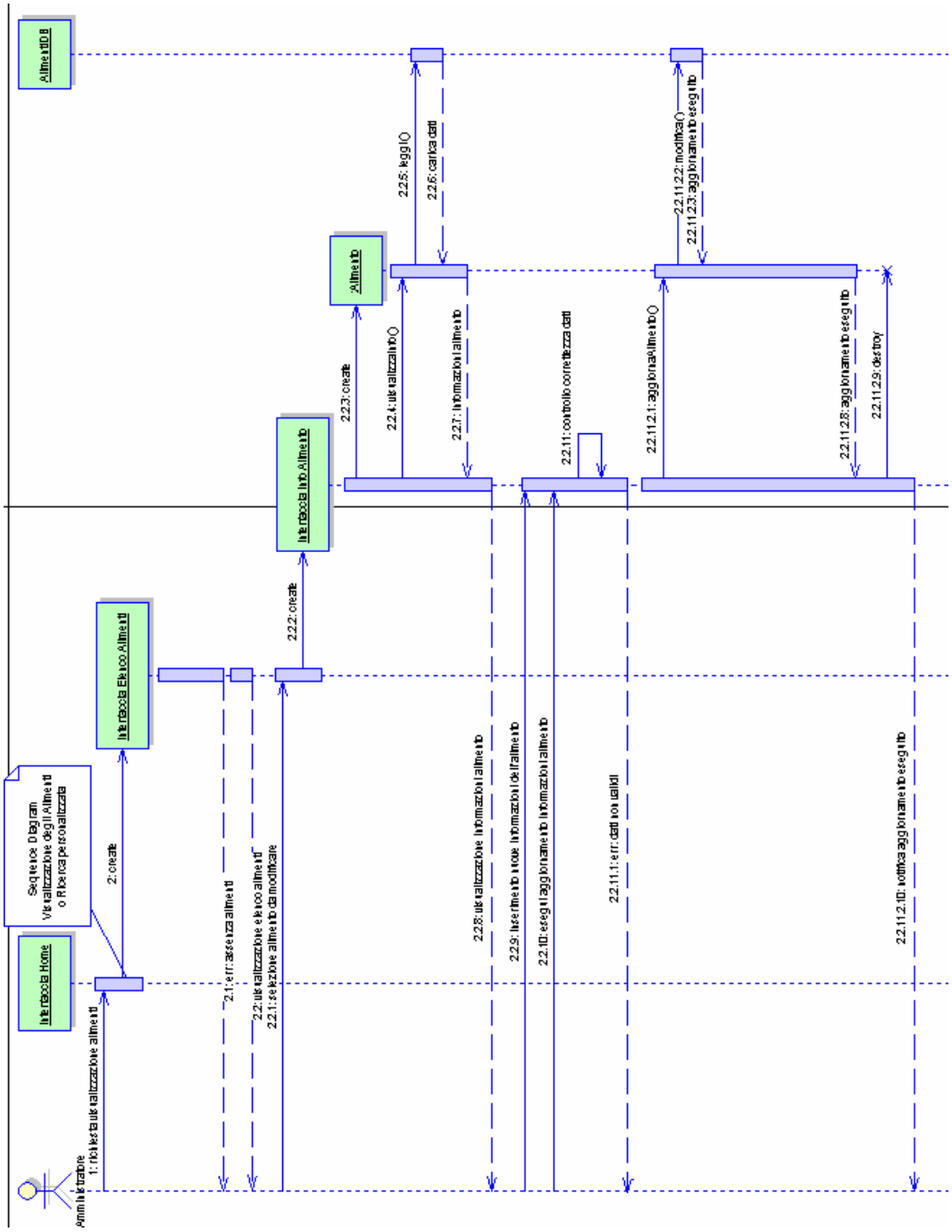


Figura 27 - Sequence Diagram Modifica alimento

L'ultima funzionalità fondamentale per la gestione dell'archivio permette l'eliminazione di un alimento. Tale operazione è prevista solo nell'interfaccia di visualizzazione delle informazioni dello stesso, tale modalità è descritta attraverso il corrispondente scenario all'interno del Sequence Diagram.

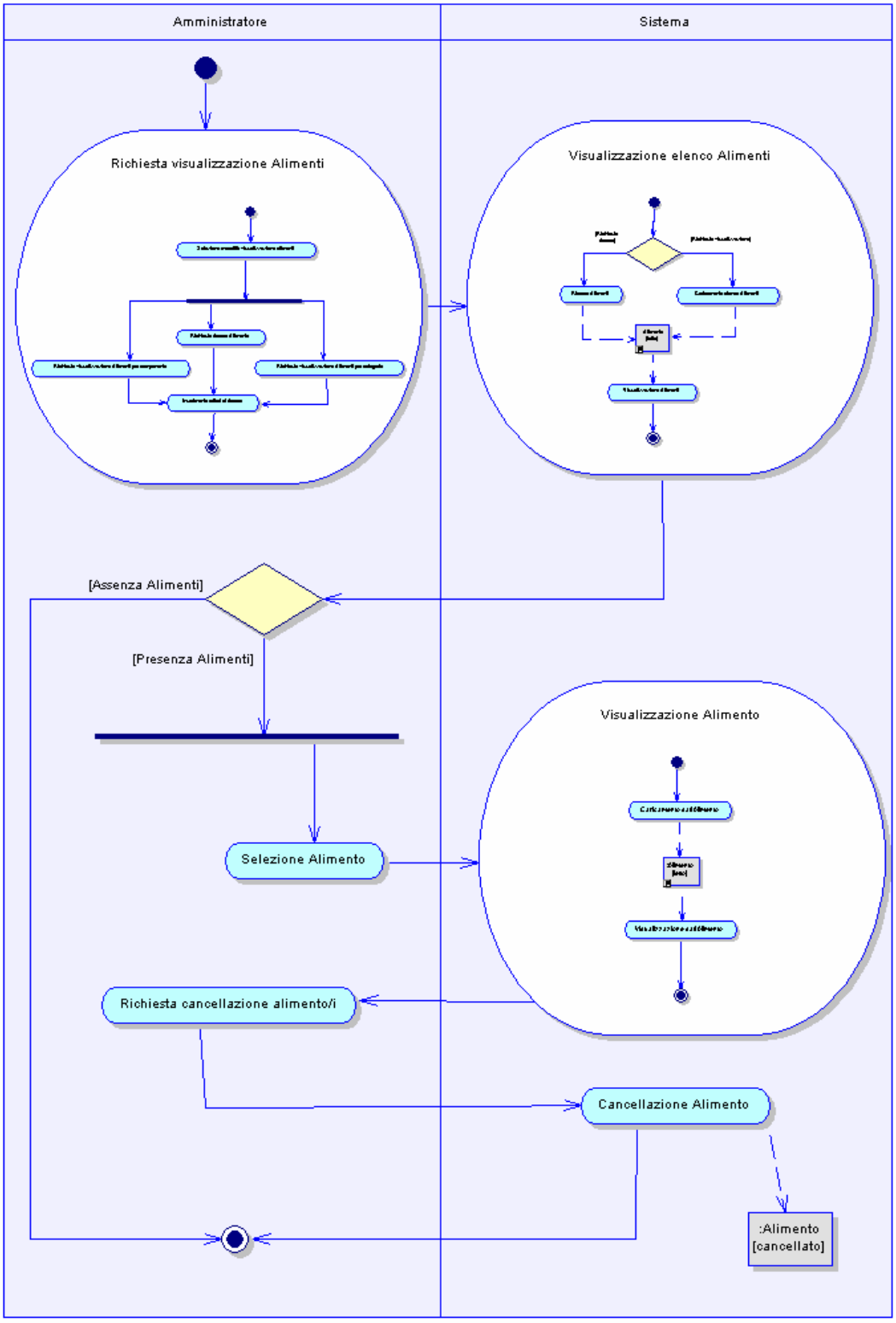


Figura 28 - Activity Diagram Eliminazione alimento dall'archivio

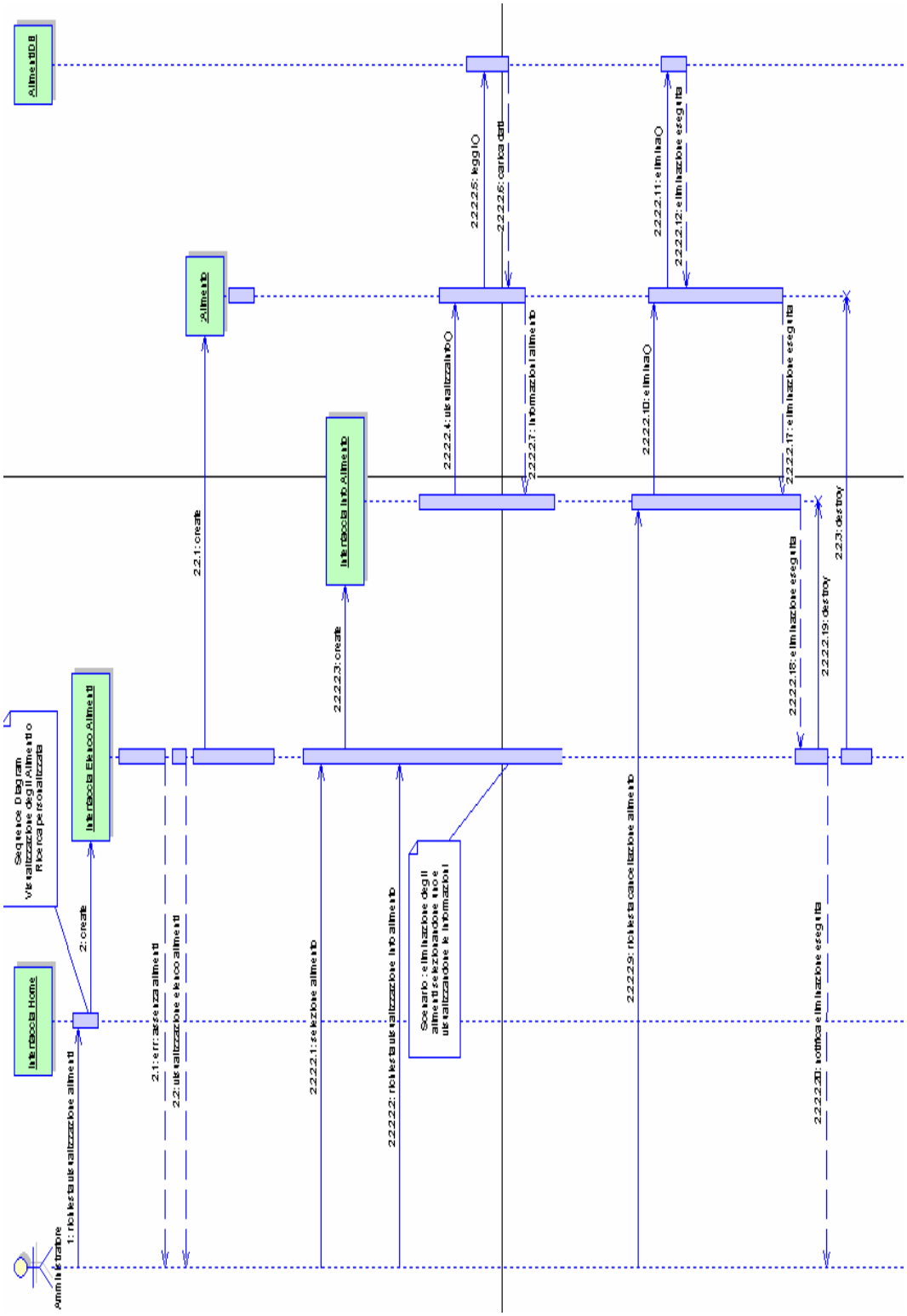


Figura 29 - Sequence Diagram Eliminazione alimento dall'archivio

3.3.7 Modifica Dati Personali

L'utente e l'amministratore hanno entrambi l'opportunità di modificare i propri dati di accesso. In particolare, l'amministratore ha esclusivamente la possibilità di modificare la password l'email ed il relativo SMTP, in quanto non ci sono ulteriori informazioni ad esso associate. Viceversa, l'utente avrà la possibilità di modificare la propria password, username e indirizzo email, oltre a tutte le informazioni anagrafiche.

In entrambi i casi, il sistema visualizza i dati attuali, permettendo all'utilizzatore di modificarne i valori e confermarne l'aggiornamento.

A questo punto, l'applicazione esegue in primo luogo un controllo di correttezza sui dati, per verificare che siano del formato atteso ed in particolare chiede all'utilizzatore, utente o amministratore che sia, di inserire una conferma della password scelta. Ovviamente, viene verificato che la password scelta e la sua conferma coincidano, altrimenti non si consente la modifica dei propri dati.

La funzionalità di modifica dei dati, può essere considerata del tutto simile, senza alcuna distinzione, tra l'utente e l'amministratore, tenendo esclusivamente conto dei limiti di aggiornamento che può eseguire il secondo.

Si è preferito comunque sviluppare gli Activity e Sequence Diagram in maniera separata, considerando che gli oggetti in gioco sono sostanzialmente diversi, poiché fanno riferimento a due utilizzatori differenti.

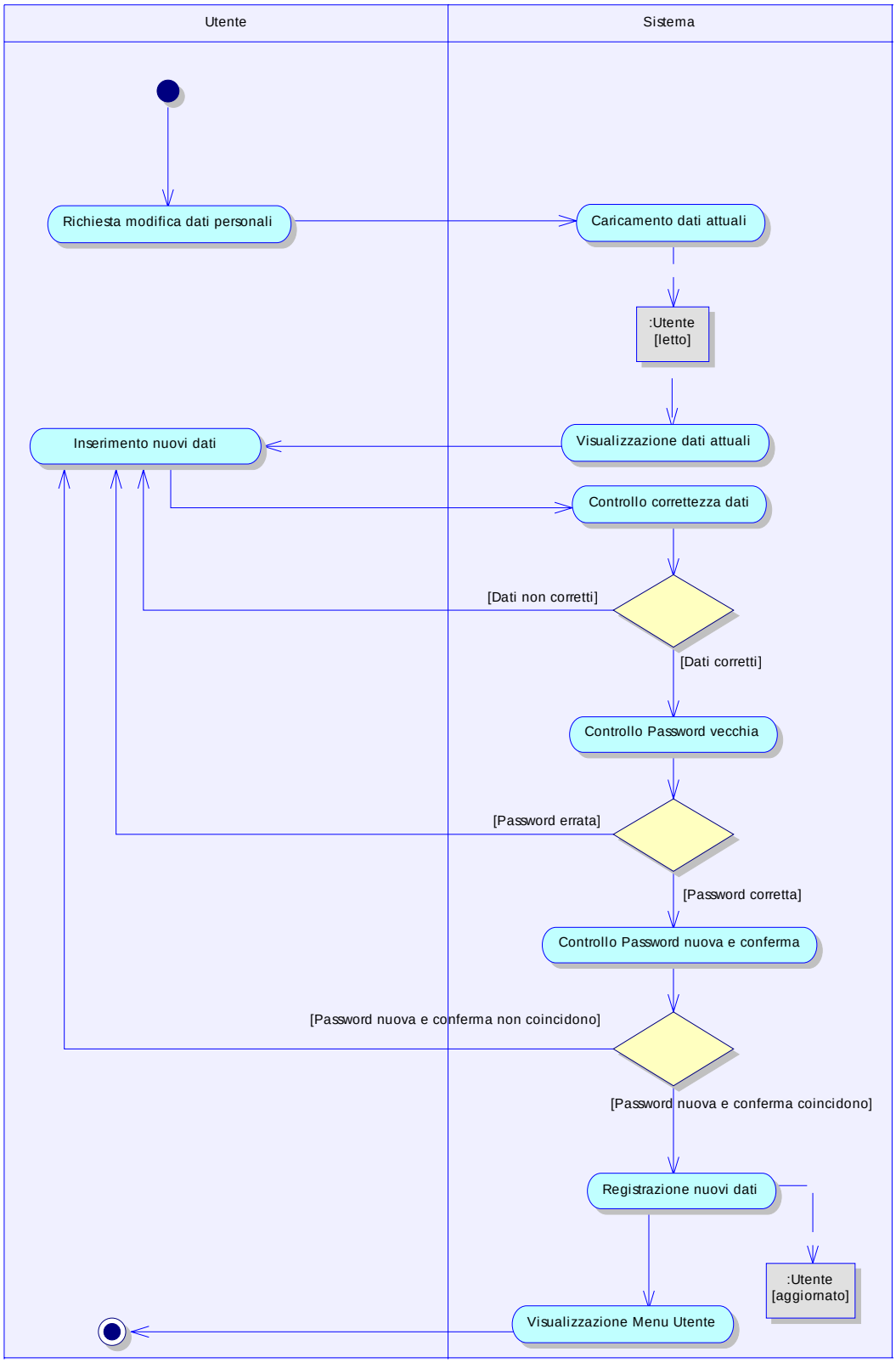


Figura 30 - Activity Diagram Modifica dati personali Utente

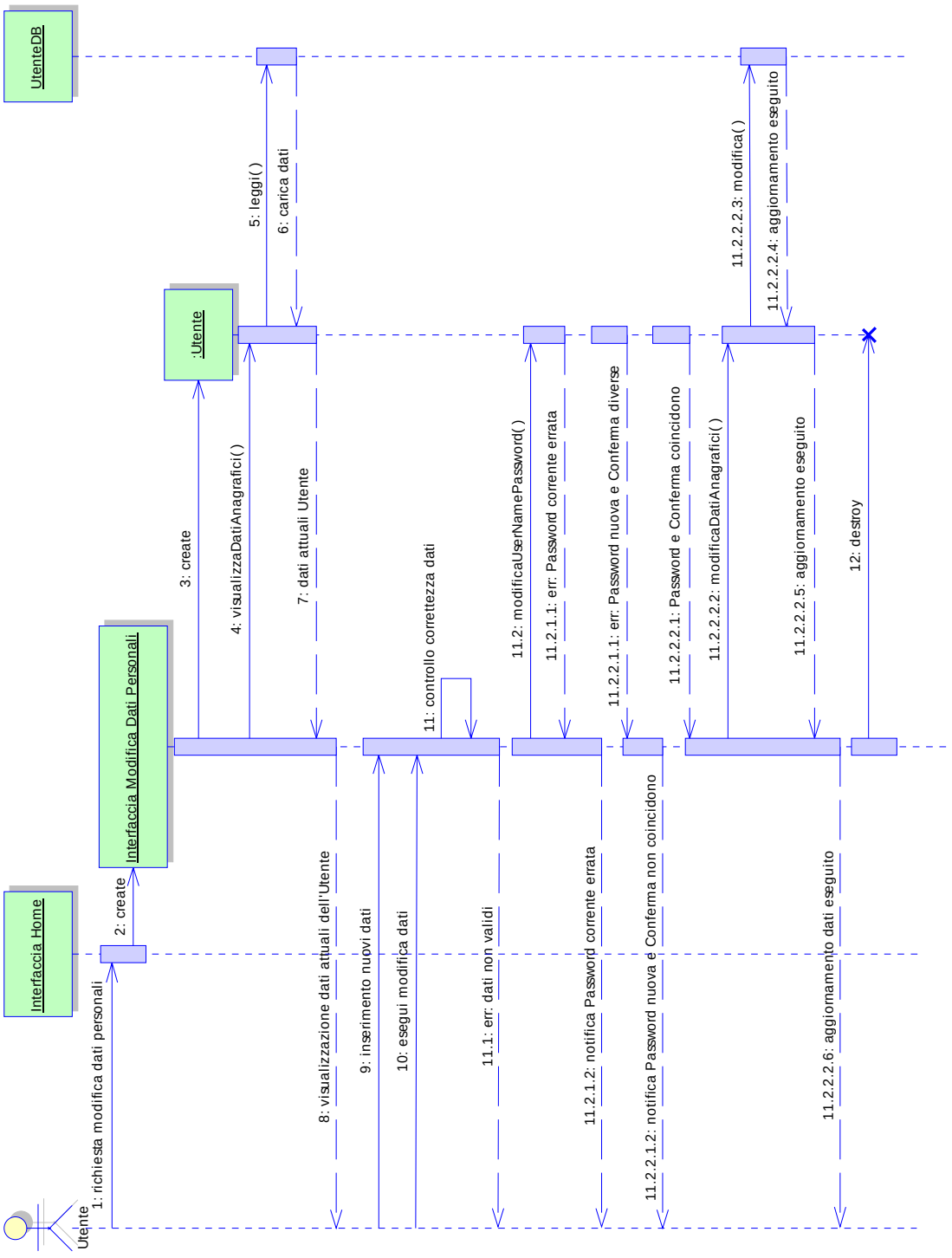


Figura 31 - Sequence Diagram Modifica dati personali Utente

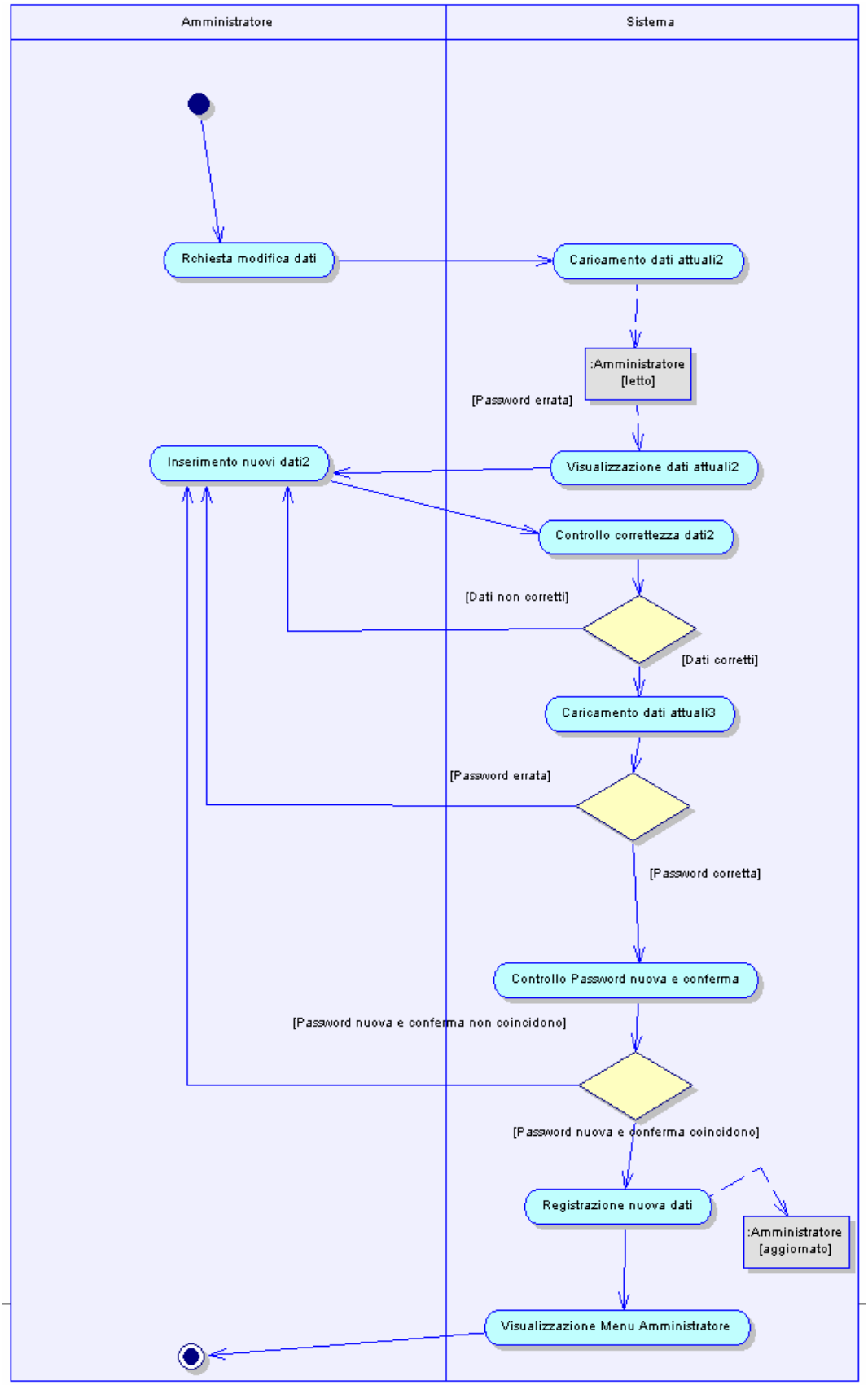


Figura 32 - Activity Diagram Modifica dati Amministratore

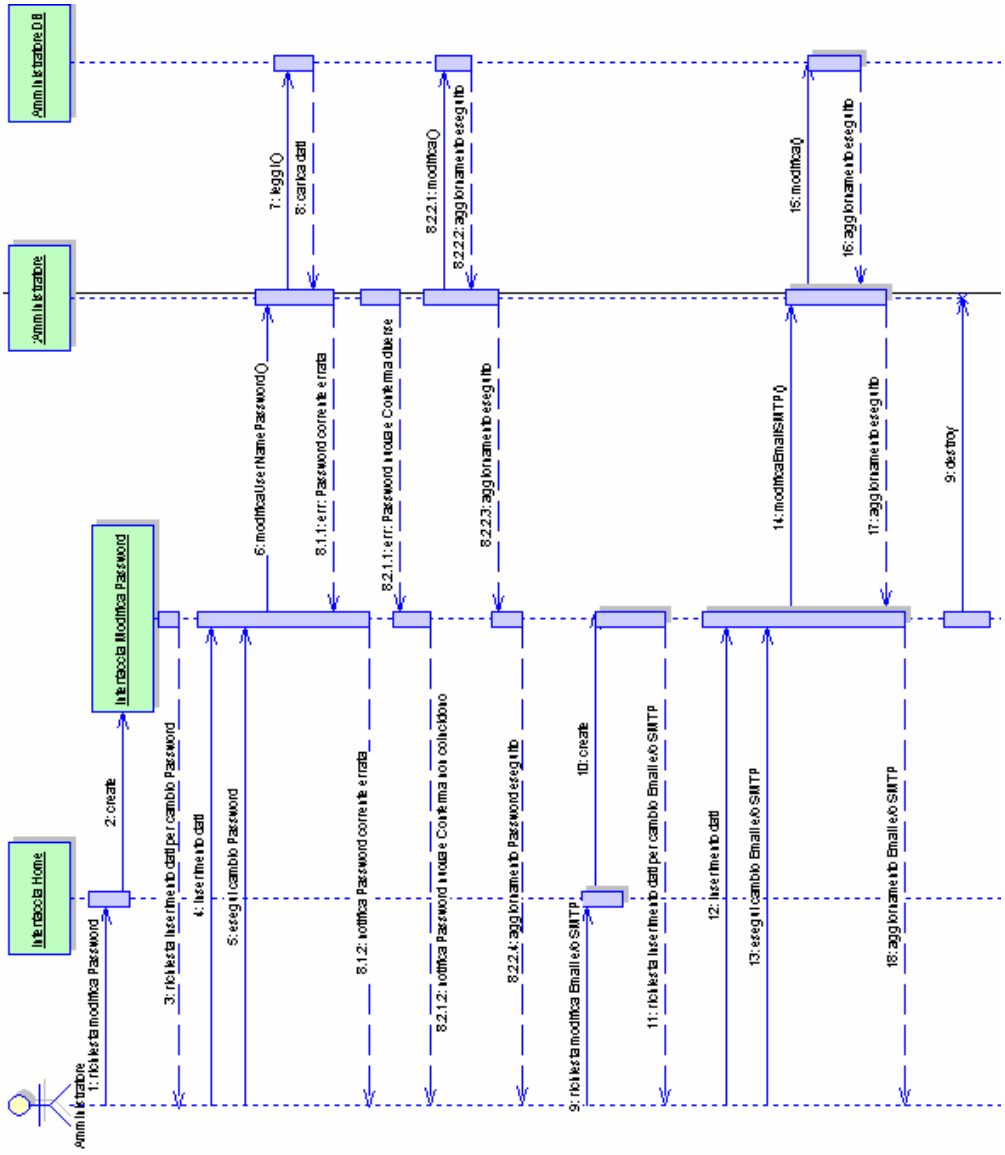


Figura 33 - Sequence Diagram Modifica dati Amministratore

3.3.8 Gestione Utenti

L'amministratore ha accesso ad una particolare sezione della Web application, mediante la quale può effettuare delle semplici operazioni relative alla gestione degli utenti.

La prima funzionalità disponibile è quella che permette la visualizzazione dell'elenco degli utenti registrati, con la possibilità di selezionare uno o più utenti e di eseguirne la cancellazione dal sistema.

A partire da tale elenco, è inoltre possibile selezionare uno degli utenti per poterne visualizzare le informazioni di registrazione, ovviamente esclusa la password di accesso al sistema.

In corrispondenza dell'interfaccia di visualizzazione dei dati del singolo utente, l'amministratore può eseguire due semplici operazioni :

- eliminare l'utente;
- sbloccare l'utente, qualora quest'ultimo risulti bloccato in virtù del fatto che si sia appena registrato ed è in attesa della convalida dell'accesso;

Il Sequence Diagram evidenzia i due scenari possibili per la cancellazione di uno o più utenti, nonché la funzionalità di sblocco.

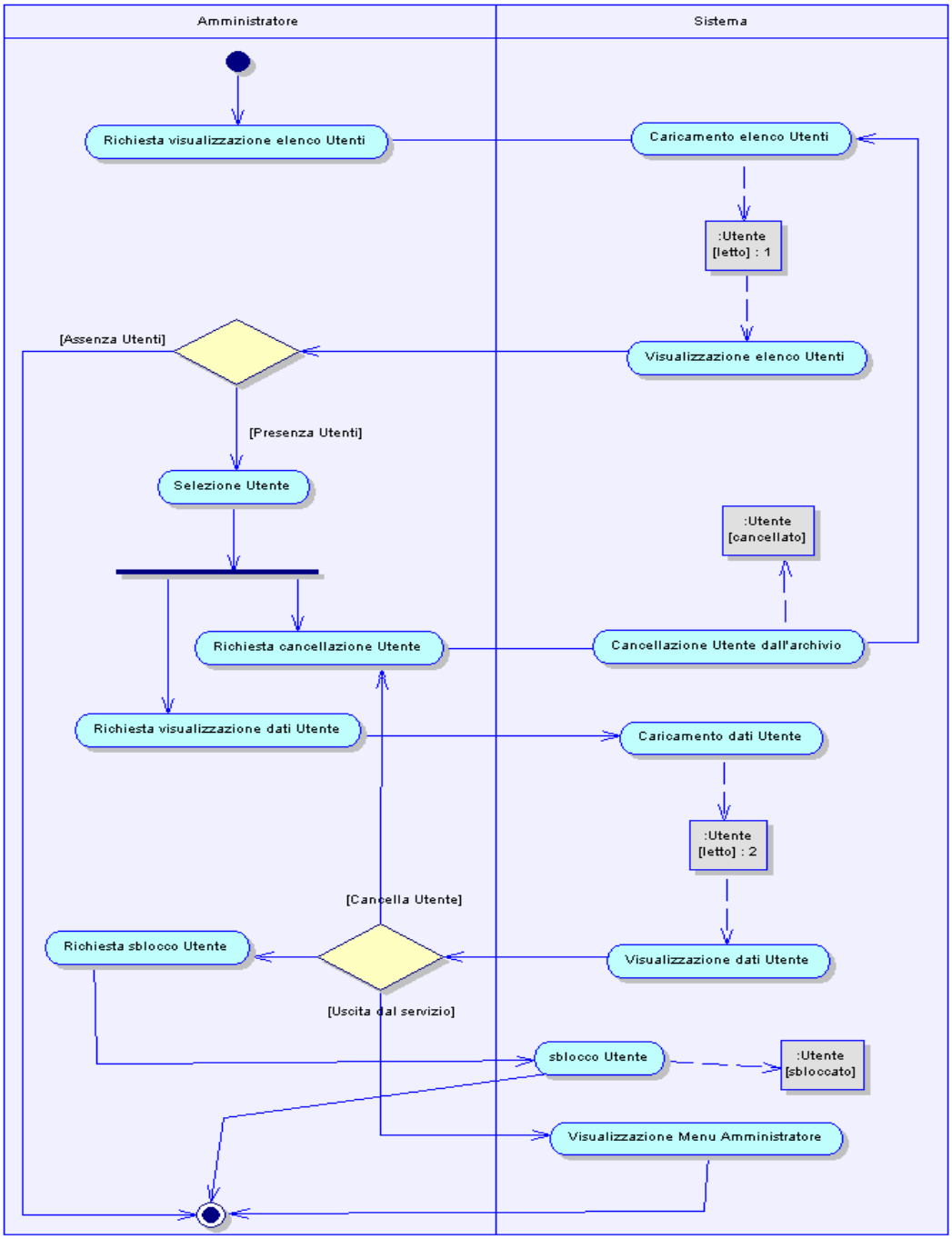


Figura 34 - Activity Diagram Gestione Utenti

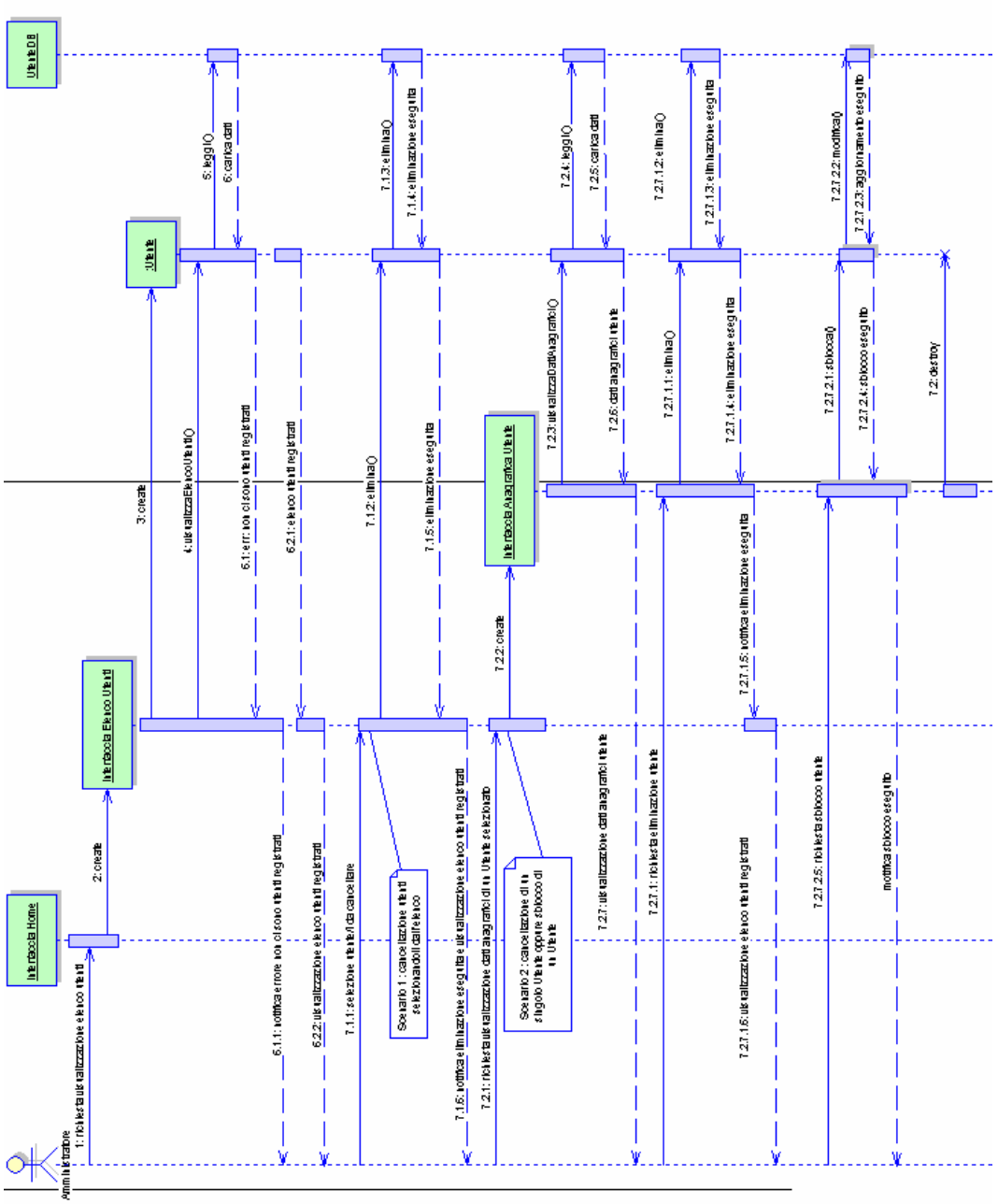


Figura 357 - Sequence Diagram Gestione Utenti

3.4 Statechart Diagrams

Gli Statechart Diagrams costituiscono uno strumento mediante il quale è possibile descrivere una state machine, enfatizzando il flusso di controllo tra gli state. Una state machine definisce un comportamento che specifica i vari state che un oggetto può assumere durante la sua vita in risposta a certi eventi.

Ogni state rappresenta una condizione di un oggetto, in cui sono soddisfatti certi requisiti , mentre un evento è la specificazione di un accadimento significativo, posizionabile nel tempo e nello spazio, che determina una transizione di stato da parte dell'oggetto. Quest'ultima esprime il fatto che, un oggetto a partire da un certo stato ed al verificarsi di un determinato evento passa in uno stato differente, poiché sono cambiate le condizioni in cui si trova. Sulla base di quanto detto, un diagramma di questo tipo può rappresentare un'estensione della descrizione di un automa a stati finiti.

Considerando il case study sviluppato, sono stati definiti gli Statechart Diagrams per quegli oggetti che possono compiere delle transizioni significative durante la vita della Web application. In particolare sono :

- Amministratore;
- Utente;
- Alimento;

Per quanto riguarda l'*Amministratore*, questo può trovarsi fondamentalmente nello stato Loggato. Ovviamente, bisogna tener conto che la condizione di *Amministratore* non loggato rappresenta il punto di inizio del diagramma. Lo stato "loggato" si raggiunge dopo aver eseguito correttamente il login.

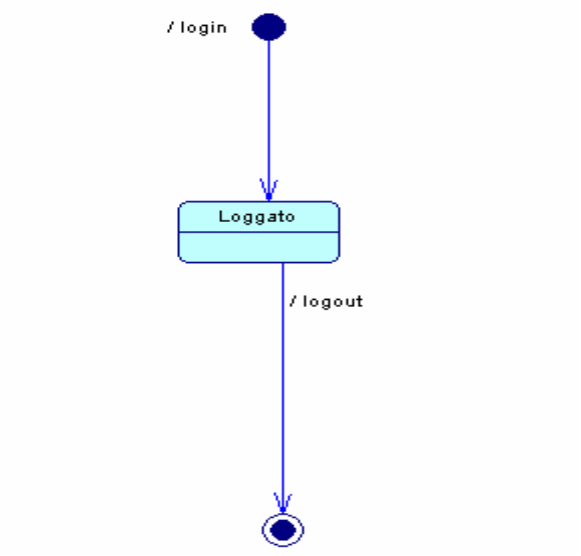


Figura 36 - Statechart Amministratore

L'*Utente* prevede una situazione leggermente complicata, in quanto si parte da uno stato iniziale in cui l'utente non è registrato. Una volta eseguita la registrazione, l'utente passa nello stato "convalida" successivamente passerà nello stato "registrato" che poi evolverà tra gli stati, "loggiato" e "bloccato".

Lo Statechart Diagram prevede una struttura innestata, nell'ambito della quale all'interno dello stato "registrato" è previsto un ulteriore diagramma che descrive le possibili transizioni di stato dell'utente che ha eseguito la registrazione.

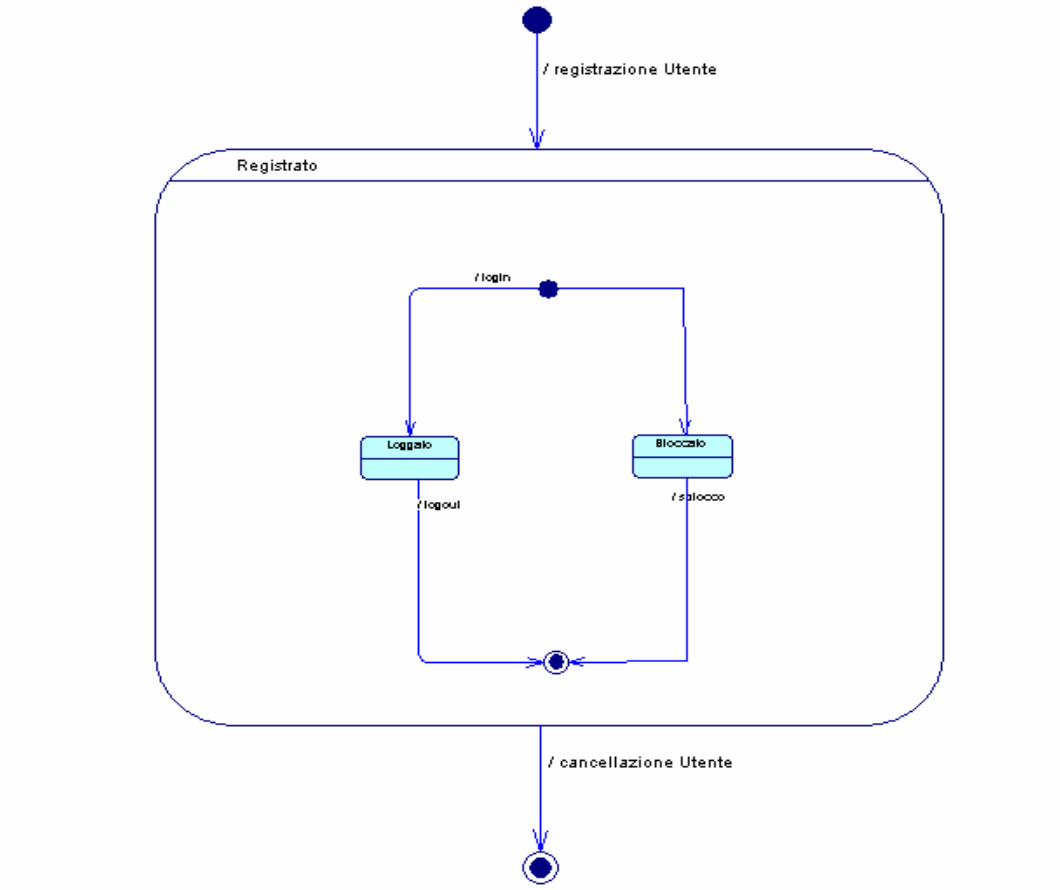


Figura 37 - Statechart Utente

L'Alimento parte da una condizione in cui non è stato inserito nell'archivio ed eseguendo tale operazione, si passa allo stato "archiviato"

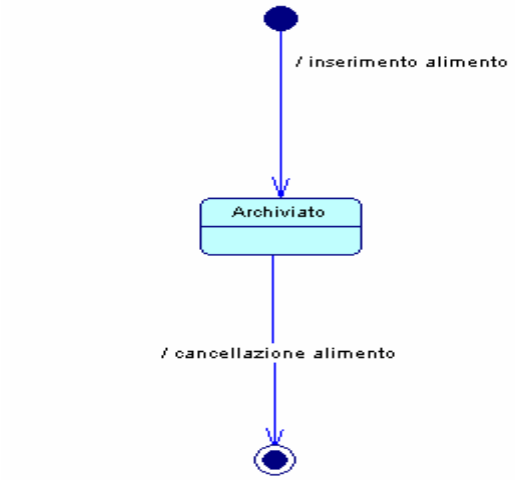


Figura 38 - Statechart Alimento

Capitolo V – Case Study : Alimenti

1. Introduzione

La Web application precedentemente progettata è stata implementata con il framework JavaServer Faces, gli strumenti software utilizzati per lo sviluppo e gli ambienti di esecuzione sono i seguenti :

- Piattaforma *J2SE* (Java 2 Second Edition);
- Ambiente IDE *Eclipse 3.1* con i plug-in di *Exadel Studio*, per lo sviluppo grafico del file di configurazione di JavaServer Faces, e *UML Omondo*, per la realizzazione dei Class Diagrams per le classi implementate;
- Web Container/Server *Jakarta Apache Tomcat*;

La piattaforma J2SE (Java2 Standard Edition) rappresenta la base per lo sviluppo e l'esecuzione delle applicazioni Java, in quanto mette a disposizione la JVM (Java Virtual Machine), oltre a tutte le classi principali e le librerie del linguaggio. Un'estensione di questa piattaforma è rappresentata dalla J2EE (Java2 Enterprise Edition) che fornisce il supporto per strumenti avanzati come i Web Services, gli EJB (Enterprise Java Beans) e JavaServer Faces. In questo caso, non si è reso necessario l'utilizzo della piattaforma J2EE, in quanto la distribuzione del framework JSF può essere gestita anche semplicemente mediante le librerie JAR (Java ARchive) che contengono le classi dell'architettura. Nelle successive versioni della piattaforma Java, in particolare a partire dalla Java5, JSF ne diventerà parte integrante.
Per l'ambiente di sviluppo IDE (Integrated Development Environment), la scelta è caduta sul progetto OpenSource Eclipse 3.1, dopo un rapido confronto con l'antagonista NetBeans. Ha nettamente prevalso il primo, considerando il supporto fornito per lo sviluppo di applicazioni Web basate appunto su JSF, grazie al plug-in free di Exadel. Il secondo, invece, fino alla versione 4.1 non ha ancora fornito tale supporto, introdotto a partire dalla versione 5 in fase di Beta Testing. Prendendo in esame tale versione, la semplicità ed il supporto offerti da Eclipse 3.1 sono comunque nettamente superiori.

L'ambiente è stato ulteriormente ampliato utilizzando il plug-in free di Omondo per la realizzazione dei diagrammi UML, a partire dalle classi implementate. Tale strumento è stato utilizzato solo ed esclusivamente per generare i Class Diagrams a livello di implementazione.
Per quanto riguarda il Web Container, la scelta è ovviamente caduta sul progetto OpenSource Jakarta Apache Tomcat, sicuramente tra i migliori per l'esecuzione di applicazioni Web realizzate in ambiente Java.
E' altresì necessario specificare che, per JavaServer Faces è stata scelta l'implementazione JSF-RI (Refernce Implementation) 1.0 e non MyFaces.

2. Struttura dell'applicazione

Il primo aspetto preso in considerazione riguarda la struttura dell'applicazione, in termini di eventuali moduli che la compongono, oltre che delle pagine, le immagini, gli eventuali fogli di stile (CSS) e di tutti gli altri elementi che ve ne fanno parte.
L'architettura del sistema prevede un unico modulo al quale è ovviamente associato un file di configurazione (*faces-config.xml*). I file componenti il progetto riguardano le pagine ed i relativi contenuti, le immagini, il foglio di stile (*styles.css*) per la formattazione della grafica ed un Tag Library Descriptor (*components.tld*) che contiene i tag associati ai componenti della UI implementati. Infine, oltre alla librerie che contengono le classi costituenti l'architettura del framework, sono utilizzati i seguenti gruppi di ulteriori librerie :

- *JavaMail* : framework per l'invio delle mail basato su *JAF* (JavaBeans Activation Framework);
- *JDBC MySQL Connector* : contiene le classi relative ai driver JDBC (Java DataBase Connectivity) per l'accesso alla base di dati realizzata con il DBMS MySQL;

Si osserva inoltre che dalla homepage sia possibile accedere alla funzionalità di login, di registrazione ed è possibile richiedere l'invio di una mail contenente i dati di accesso in caso di dimenticanza. Infine, è definita un'eccezione globale nel caso si verifichi un errore di accesso alla base di dati.

L'Utente invece ha accesso alle seguenti funzionalità :

- modifica dei dati dell'utente;
- visualizzazione e ricerca degli alimenti;
- operazione di logout;

Anche in questo caso è definita un'eccezione globale nel caso si verifichi un errore di accesso alla base di dati.
Infine, l'Amministratore ha a disposizione i seguenti servizi :

- modifica dei dati dell'amministratore;
- gestione degli utenti;
- visualizzazione e ricerca degli alimenti;
- inserimento, modifica e cancellazione degli alimenti e di tutte le sue componenti;
- operazione di logout;

Anche qui è stata definita un'eccezione globale che viene sollevata qualora si verifichi un errore di accesso alla base di dati.
Per completare l'analisi della struttura complessiva della Web application, bisogna descrivere quelli che sono i package contenenti le classi ed il loro ruolo.
Nel dettaglio, i package comuni sono i seguenti :

- *accesso* : classi relative alle informazioni di accesso dell'utente e dell'amministratore;
- *alimenti* : classi relative agli alimenti ed ai suoi componenti;
- *database* : classi relative all'accesso alla base di dati;
- *exception* : classi che implementano delle eccezioni personalizzate, estendendo la classe di base *Exception*;
- *listeners* : classi che intervengono in particolari situazioni di controllo;
- *mail* : classe per l'invio delle mail;
- *ruolo* : classi che implementano i possibili utilizzatori del sistema;
- *components* : classi che implementano dei Custom Components, realizzati per l'interfaccia utente (UI);
- *validators* : classi che implementano alcuni Custom Validators, utilizzati per la validazione dei dati;

Un' ulteriore considerazione riguarda l'internazionalizzazione, supportata dalla Web application attraverso la definizione di due Resource Bundle. In particolare, ne è previsto uno per la localizzazione di default, ossia l'Italia, e l'altro per un'ulteriore localizzazione supportata, in lingua Inglese.
Infine, ci sono da motivare le scelte relative alla realizzazione del Model; il sottolivello Data Access è completamente indipendente dagli altri. Questo ne favorisce la portabilità, qualora si prenda in considerazione l'ipotesi di realizzare la Web application con un ulteriore framework differente. Per quanto riguarda i sottolivelli di Business Logic ed External Interface, è stata fatta la seguente scelta: si è utilizzato l'approccio di "fusione", realizzando i backing beans in modo tale da contenere la business-logic e l'interfacciamento con le classi del framework. Tale scelta ha il vantaggio di non dover realizzare un ulteriore strato di classi che implementi l'External Interface sublayer, il cui scopo sia quello di invocare i metodi sulle classi della business-logic ed utilizzare gli oggetti interni al framework. Lo svantaggio è ovviamente la perdita di portabilità, in quanto i backing beans utilizzano delle classi che non sarebbero presenti in framework diversi.

3. Controller

Il framework JSF fornisce le proprie classi di base che implementano la struttura del Controller di default, è possibile intervenire per poter personalizzare tali classi, in modo da aggiungere particolari funzionalità che vanno applicate durante la fase di comunicazione tra Model e View. Nel caso di studio in esame, si è reso necessario l'intervento sul Controller, per poter gestire due situazioni particolarmente delicate :

- la gestione delle sessioni non concluse correttamente dell'utilizzatore;
- la possibilità di accesso a pagine protette senza la necessaria operazione di login;

La prima problematica è maggiormente legata alla gestione delle Servlet che non alle classi del framework

3.1 Gestione delle sessioni

Uno dei principali problemi da cui sono afflitte tutte le Web application, è la gestione delle sessioni che non vengono concluse correttamente da un utente. Nel caso specifico, quando l'utilizzatore, utente o amministratore che sia, effettua il login al sistema, vengono memorizzate all'interno della base di dati le informazioni relative all'accesso, ossia la data ed l'ora di ingresso, l'indirizzo IP del client ed il numero di tentativi effettuati. Inoltre, viene settato un flag nel database che indica che l'utente è online, in modo che un tentativo fraudolento di accesso da parte di un secondo utente che disponga dei dati del primo, venga ovviamente bloccato. Nell'effettuare correttamente la fase di logout, le informazioni di accesso vengono aggiornate settando opportunamente la data e l'ora di uscita dal sistema, nonché viene azzerato il flag nel database per segnalare che l'utente è offline. Ci sono moltissime situazioni in cui un utente chiude direttamente il browser, senza completare correttamente il logout. In questo caso è necessario intervenire per ristabilire la coerenza delle informazioni contenute nel database rispetto alla realtà. L'evento di chiusura del browser non può essere intercettato dal server, sul quale è in esecuzione la Web application, ma ovviamente solo dal client. Il linguaggio di scripting lato client, che permette di eseguire elaborazioni su quest'ultimo è ovviamente il JavaScript, il quale però non mette a disposizione la possibilità di intercettare un semplice evento come la chiusura di una finestra del browser, a meno che questa azione non venga eseguita sempre mediante codice. Per questo ed altri motivi, è stato introdotto il concetto di sessione, mediante la quale si stabilisce un legame tra l'utente che utilizza l'applicazione ed il server, su cui quest'ultima è in esecuzione. Alla sessione è associato un concetto di timeout, per cui se non si rileva attività da parte dell'utente fino allo scattare di quest'ultimo, ne viene eseguita l'invalidazione. E' in corrispondenza di questo evento, che si può intervenire effettuando le operazioni che garantiscano la coerenza delle informazioni nel sistema. Da qui, la necessità di realizzare un listener, che allo scadere del timeout di sessione, intervenga per poter permettere allo sviluppatore di effettuare delle elaborazioni necessarie.

La realizzazione di un listener di questo tipo è favorita dall'utilizzo dell'interfaccia *HttpSessionListener* e della classe *HttpSessionEvent*.

La prima mette a disposizione i metodi da eseguire al momento della creazione e distruzione di una sessione, mentre la seconda permette di ricavare un riferimento alla sessione stessa.
Quindi è stata realizzata la classe *InactivityListener*, nell'ambito della quale il metodo *sessionCreated()* ha il compito di settare il timeout di sessione e *sessionDestroyed()* ha l'onere di cambiare lo stato del flag nel database, per segnalare che l'utente è offline. Quest'ultimo metodo viene appunto invocato nel momento in cui scatta il timeout, in quanto l'utente non sta più generando attività sul server e presumibilmente ha chiuso il browser senza eseguire correttamente il logout.

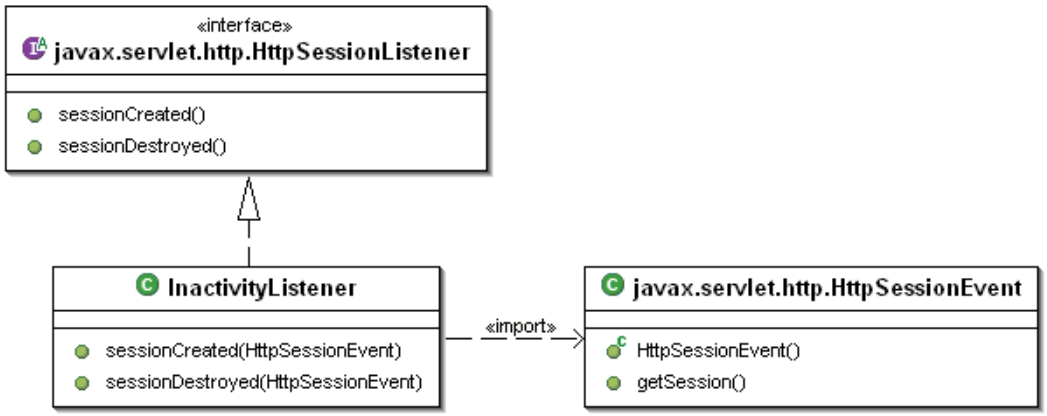


Figura 1 InactivityListener

La configurazione che specifica l'utilizzo del listener prevede l'intervento sul Deployment Descriptor *web.xml* mediante la seguente dichiarazione :

```
<listener>
  <listener-class>listeners.InactivityListener</listener-class>
</listener>
```

3.2 Protezione pagine

La seconda problematica in cui è necessario l'intervento sul Controller, riguarda la possibilità che un utilizzatore voglia accedere alle pagine protette dell'applicazione, senza avere eseguito correttamente l'operazione di login. Ovviamente, il sistema deve intercettare un'azione di questo tipo, redirezionando l'utente alla homepage, nella quale è costretto ad eseguire l'accesso.

Dall'analisi del problema, si evince che l'intervento deve essere eseguito per ciascuna richiesta che arrivi dal client, in modo da sincerarsi che l'utilizzatore sia correttamente loggato, qualunque sia la pagina a cui vuole accedere, escluse le pagine relative alla registrazione ed alla richiesta dei dati di accesso via mail.
Per la risoluzione di tale problema si fa riferimento alla gestione di una richiesta del client, la quale viene eseguita attraverso sei fasi distinte. In corrispondenza di ciascuna di esse, può essere utilizzato un listener che permetta di eseguire delle operazioni prima e dopo l'esecuzione della fase stessa.
La soluzione adottata prevede la realizzazione della classe *LoginCheck* che implementa l'interfaccia *PhaseListener* ed esegue i controlli necessari prima della fase di Render Response. Essa prevede che il metodo *afterPhase()* sia vuoto, mentre all'interno del metodo *beforePhase()* ci sia il controllo che l'utilizzatore, utente o amministratore che sia, risulti correttamente loggato oppure la richiesta si riferisca a pagine non protette da login. Nel caso di esito positivo, la navigazione procede normalmente, altrimenti viene ricavato un riferimento al *NavigationHandler*, il quale permette di spostare il flusso dell'applicazione verso la homepage. Le informazioni relative all'evento verificatesi, ossia l'esecuzione della fase di Render Response, sono contenute nella classe *PhaseEvent*.

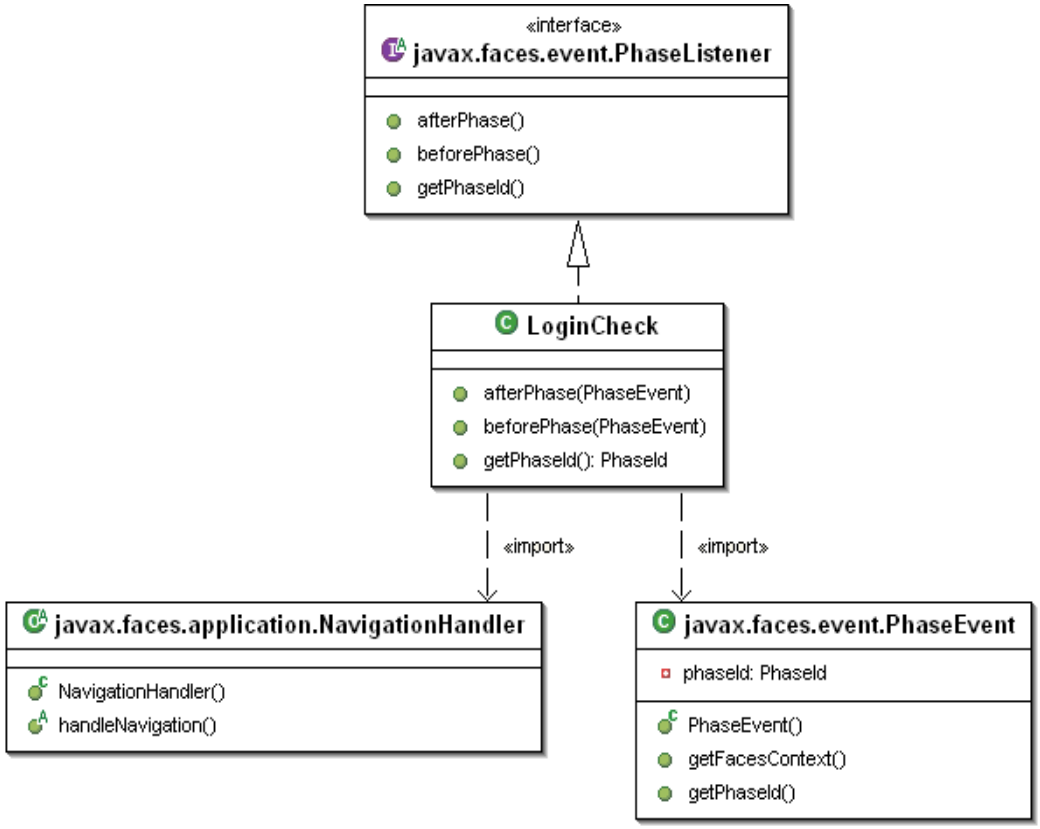


Figura 2 - LoginCheck

La configurazione del listener prevede di associare quest'ultimo all'istanza della classe *Lifecycle* che si occupa di gestire il ciclo di vita di ogni singola richiesta.

```
<lifecycle>
  <phase-listener>listeners.LoginCheck</phase-listener>
</lifecycle>
```

4. View

Per la realizzazione dell'interfaccia utente (UI) sono state utilizzate due librerie proprie del framework, ossia Core ed HTML, la prima per quanto riguarda aspetti avanzati come l'uso dei validatori mentre la seconda per la costruzione dei form, la visualizzazione dei messaggi e la formattazione del contenuto delle pagine.
Inoltre, sfruttando l'enorme potenzialità di JSF, sono stati realizzati dei Custom Components ed il relativo Tag Library Descriptor, in modo da utilizzare questi componenti all'interno delle pagine con un corrispondente tag.

4.1 Le librerie Standard

Per quanto riguarda le librerie di tag standard, il framework JSF utilizza solo ed esclusivamente le proprie librerie, Core ed HTML, in quanto non prevede alcun tipo di modifica nel Deployment Descriptor *web.xml*

4.1.1 I form : contenuto e layout

Per la definizione della struttura dei form ed il posizionamento degli elementi all'interno della pagina, viene introdotto il concetto di Panel per la formattazione dei contenuti. Tale concetto è ben noto nell'ambito dello sviluppo delle applicazione desktop tramite le librerie AWT e Swing ed è stato introdotto nelle Web application, come passo verso un'unificazione tra le due tipologie di applicazioni.

La libreria JSF HTML fornisce i tag <*h:panelGrid*> e <*h:panelGroup*> di cui il primo definisce appunto un Panel, costituito da un fissato numero di colonne, all'interno del quale vengono disposti gli elementi mentre il secondo permette di raggruppare più campi, facendo in modo che nel Panel vengano trattati come un unico elemento. Inoltre, all'interno del Panel è possibile specificare le intestazioni delle colonne, mediante il tag <*f:facet*>, così come gli stili da applicare, definiti all'interno di un file CSS esterno.

```
<h:form id="regForm">
<h:panelGrid columns="3" headerClass="formsHeaderText"
        columnClasses="columsLabelForm,
        columsInputForm, columsErrorForm">
    <f:facet name="header">
        <h:outputText value="#{bundle.regHeader}" />
    </f:facet>
    <h:outputText value="#{bundle.regUsernameLabel} * "/>
    <h:inputText id="userName"
        value="#{utente.userName}"
        required="true"  maxlength="20"/>
    <h:panelGroup>
    <h:message for="userName" styleClass="errorMessage" />
    <h:message for="regForm" id="regError" styleClass="errorMessage"/>
</h:panelGroup>
  ...
```

4.1.2 Costruzione condizionale dei contenuti

Un' ulteriore considerazione riguarda la costruzione condizionale di una pagina, che consiste nel fare in modo che un certo contenuto sia visualizzato o meno a seconda del verificarsi di opportune condizioni. Tale operazione viene eseguita mediante l'utilizzo dell'attributo *rendered* di cui è dotato ciascun tag della libreria JSF HTML.

Tale attributo può assumere un valore booleano che indica appunto se il tag ed il relativo contenuto debba essere renderizzato o meno. Facendo uso dell'Expression Language e del value-binding, è possibile definire un'espressione nella quale entrino in gioco le proprietà dei backing beans, in modo da esprimere condizioni complesse sulla base delle quali visualizzare o meno il componente.

Nell'esempio che segue, il contenuto del Panel viene visualizzato soltanto nel caso in cui l'utente abbia effettuato il login.

```
<h:panelGrid rendered="#{utente.loggedIn}">
...
</h:panelGrid>
```

4.1.3 DataTable JSF

Nell'ambito della Web application realizzata, si è più volte posto il problema di visualizzare un elenco di elementi in forma tabellare, come nel caso degli alimenti o degli utenti registrati.

L'implementazione JSF ha previsto l'utilizzo del potentissimo tag <*h:dataTable*>, il quale permette di specificare la lista degli elementi da visualizzare ed eventualmente le informazioni di formattazione. Inoltre, ciascuna colonna viene definita con il tag <*h:column*>, all'interno del quale ne viene specificato un'eventuale intestazione ed il relativo contenuto.

```
<h:dataTable value="#{sessionScope.alimenti}" var="alimento"
        headerClass="headerDataTable"
        columnClasses="linkDataTable, infoDataTable,
                infoDataTable, checkBoxDataTable" rowClasses="rowDataTable">
        <h:column>
                <f:facet name="header">
                <h:outputText value="#{bundle.DescrizioneAlimentoLabel}" />
                </f:facet>
                <h:commandLink action="#{alimento.visualizzaInfoAlimento}">
                        <h:outputText value="#{alimento.descrizioneAlimento}"/>
                </h:commandLink>
        </h:column>
        <h:column>
                <f:facet name="header">
                        <h:outputText value="#{bundle.fonteLabel}" />
                </f:facet>
                <h:outputText value="#{alimento.codiceFonte}" />
        </h:column>
    ...
```

Dall'esempio si osserva che l'attributo *value* specifica una variabile di sessione che contiene l'elenco degli alimenti, mentre l'attributo *var* definisce il nome della variabile che identifica ciascun elemento all'interno della lista. Nel momento in cui si clicca sulla descrizione dell'alimento, per poterne visualizzare le informazioni, l'oggetto corrente specificato nell'attributo *var* viene trasferito tra i parametri della request, per cui sarà disponibile alla pagina di visualizzazione.

4.2 Custom Components

Il framework JSF mette a disposizione una potenzialità notevole che consiste nel poter estendere le classi dei componenti della UI, per implementare componenti personalizzati, molto più potenti, da utilizzare nella propria interfaccia utente. Per ciascuno di essi, è necessario eseguire le seguenti fasi :

- realizzare la classe che definisce il comportamento del componente, oltre alle fasi di decoding e di encoding se vanno gestite in maniera autonoma;
- viceversa, realizzare la classe renderer che si occupi delle operazioni di decode ed encode;
- realizzare la classe che costituisce il Tag Handler, per poter gestire il tag associato al componente;
- definire le proprietà del tag all'interno di un file TLD (Tag Library Descriptor);

Nella Web application in questione, sono stati realizzati i seguenti componenti :

- *DateSelectComponent* : componente che visualizza tre drop-down list contenente giorni, mesi ed anni, per la selezione di una data. Utilizzato, ad esempio, per l'immissione della data di. Inoltre permette di visualizzare un'etichetta associata ed un eventuale messaggio di errore;
- *DDListComponent* : componente che visualizza semplicemente una drop-down list ma con la possibilità di specificarne un'etichetta e visualizzare un eventuale messaggio di errore;

Le classi che implementano i suddetti componenti estendono la classe *UIInput* e non direttamente la classe di base *UIComponent*, in modo tale da poter sfruttare il comportamento di default di un componente di input, potenziandone le caratteristiche.

Inoltre, il componente *DateSelectComponent* svolge in maniera autonoma le operazioni di deconding e di encoding, per cui non ha bisogno di un renderer. Viceversa, il componente *DDListComponent* delega tali operazioni ad una classe renderer corrispondente, implementata estendendo la classe base *Renderer*. Ovviamente, ciascun componente prevede un proprio Tag Handler realizzato mediante una classe che estende *UIComponentTag*.

4.2.1 DateSelectComponent

L'implementazione di questo componente è stata presa in considerazione, per poter permettere all'utente di specificare in maniera piuttosto banale la propria data di nascita, nelle fasi di registrazione.

Figura 3 - Esempio d'uso del DateSelectComponent

Si può osservare che il componente prevede la visualizzazione di un' etichetta, una serie di drop-down list ed un eventuale messaggio di errore. Inoltre, la sua introduzione all'interno di una pagina JSP, prevede un unico tag ed attraverso i relativi attributi è possibile specificare il testo dell'etichetta e quali drop-down list visualizzare, tra quelle del giorno, del mese e dell'anno. Nel caso della data di nascita sono necessarie tutte mentre nel caso della scadenza della carta di credito bastano le ultime due.

```
<components:dateSelect id="dataNascita"
                       value="#{utente.dataNascita}"
                       showDay="true"
                                showMonth="true"
                                showYear="true"
                                type="birth"/>
```

L'ulteriore vantaggio è dato dal fatto che una volta selezionata ed inviata la data, tramite un form, il componente non prevede due o tre valori separati che rappresentano giorno, mese ed anno ma bensì un unico valore di tipo *Date*.

Se si fosse pensato di utilizzare i tag delle librerie di base, sarebbero stati necessari due o tre tag <*h:selectOneListbox*> oppure <*h:selectOneMenu*> per le drop-down list, più un tag <*h:outputText*> per l'etichetta. Inoltre, i valori ottenuti sul server sarebbero stati completamente indipendenti e si sarebbero rese necessarie ulteriori elaborazioni per comporli e ricavare una data valida. Per quanto riguarda lo sviluppo, la prima classe realizzata è quella relativa al componente vero e proprio, ossia *DateSelectComponent*. Ques'ultima estende la classe *UIInput*, in modo da ereditarne il comportamento di base, al quale aggiunge le proprie caratteristiche specifiche.

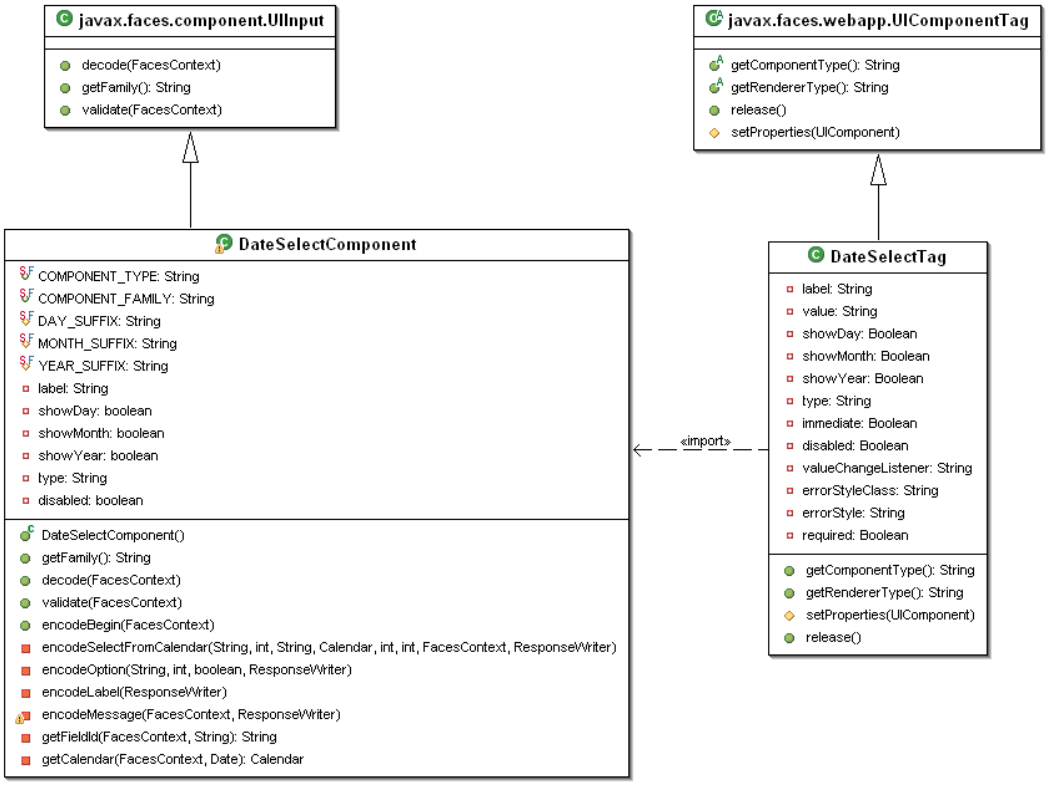


Figura 4 - DateSelectComponent, DateSelectTag

Gli attributi principali della classe servono per specificare l'etichetta da visualizzare al fianco del componente, quali campi visualizzare tra giorno, mese ed anno ed infine che tipo di data dovrà contenere, nascita o scadenza, in modo da generare l'elenco degli anni in maniera differente. L'operazione di deconding viene eseguita all'interno del componente stesso attraverso il metodo *decode()*, all'interno del quale vengono ricavati dalla richiesta i parametri che rappresentano i valori del giorno, del mese e dell'anno, i quali vengono utilizzati per creare un oggetto *Calendar* che rappresenti la data selezionata.

Si evince quindi che il componente restituisce un valore di tipo *Date*, che può essere destinato direttamente ad una proprietà di un backing bean dello stesso tipo. L'operazione di validazione è pressoché superflua, considerando che il componente "guida" l'utente nella scelta della data ed inoltre va sottolineato che, nel caso di data incoerente (es. 31 febbrario 2006), lo stesso oggetto *Calendar*, in maniera del tutto automatica, ripristina la data valida più vicina. La fase di encoding viene eseguita attraverso dei metodi privati, invocati nel metodo principale *encodeBegin()*, generando i tag HTML per la visualizzazione dell'etichetta e le drop-down list. Al componente è associato il Tag Handler realizzato mediante la classe *DateSelectTag* che estende *UIComponentTag*. Essa dispone di tutti gli attributi di cui sarà dotato il tag nella pagina JSP, ma in particolar modo del metodo *setProperties()*, che ha il compito di caricare i valori di tali attributi che il Page author specifica nel tag, in corrispondenza dei campi della classe *DateSelectComponent*. Inoltre, è previsto il metodo *release()*, che permette di rilasciare le risorse allocate per gli attributi, una volta che il componente sia stato renderizzato. Infine, per la descrizione del tag relativo a questo componente così come agli altri, è stato creato un Tag Library Descriptor (*components.tld*) in cui c'è la descrizione degli attributi associati al tag stesso.

```
<tag>
 <name>dateSelect</name>
 <tag-class>components.DateSelectTag</tag-class>
 <attribute>
  <name>id</name>
  <description>Identificativo del component</description>
 </attribute>
 <attribute>
  <name>value</name>
  <description>Valore del component</description>
 </attribute>
```

E' ovviamente necessaria la registrazione del componente all'interno del file di configurazione, specificandone la classe che lo implementa.

```
<component>
 <component-type>DateSelect</component-type>
  <component-class>components.DateSelectComponent </component-class>
</component>
```

4.2.2 DDListComponent

La realizzazione di questo componente potrebbe sembrare assolutamente superflua, considerando che le librerie di base mettono a disposizione i tag *<h:selectOneListbox>* e *<h:selectOneMenu>*, per la visualizzazione della drop-down list, insieme ai tag *<f:selectItem>* e *<f:selectItems>*, per specificare gli elementi contenuti nella lista.


```
<components:dropDownList id="stato"
                         value="#{utente.stato}"
                         items="#{itemsBean.stati}"
                         required="true"/>
```

Figura 5 – Esempio d'uso del DDListComponent

I vantaggi principali ottenuti dall'utilizzo del componente sono relativi alla possibilità di visualizzare un'etichetta ed un eventuale messaggio di errore, ma soprattutto di poter specificare la lista degli elementi da visualizzare, non mediante ulteriori tag, ma bensì con un proprio attributo (*items*).
Per quanto riguarda l'implementazione, si è pensato di delegare la fase di deconding e di encoding ad un renderer, in modo da evindeziare il differente approccio rispetto al *DateSelectComponent*. Tale modalità ha il notevole vantaggio di poter associare ad un medesimo componente più renderers differenti, magari relativi a linguaggi di markup diversi. In questo modo, il comportamento viene definito una sola volta ma le visualizzazioni sono molteplici, in virtù della possibilità di accedere alla Web application con dispositivi client diversi.
L'implementazione necessita quindi di tre o più classi : una relativa al componente, una relativa al corrispondente Tag Handler ed una o più classi relative ai renderer. Nel caso di studio in esame, è stato realizzato un unico rendeder per il linguaggio di markup HTML, assumendo che l'applicazione sia accessibile solo ed esclusivamente mediante un browser Web.

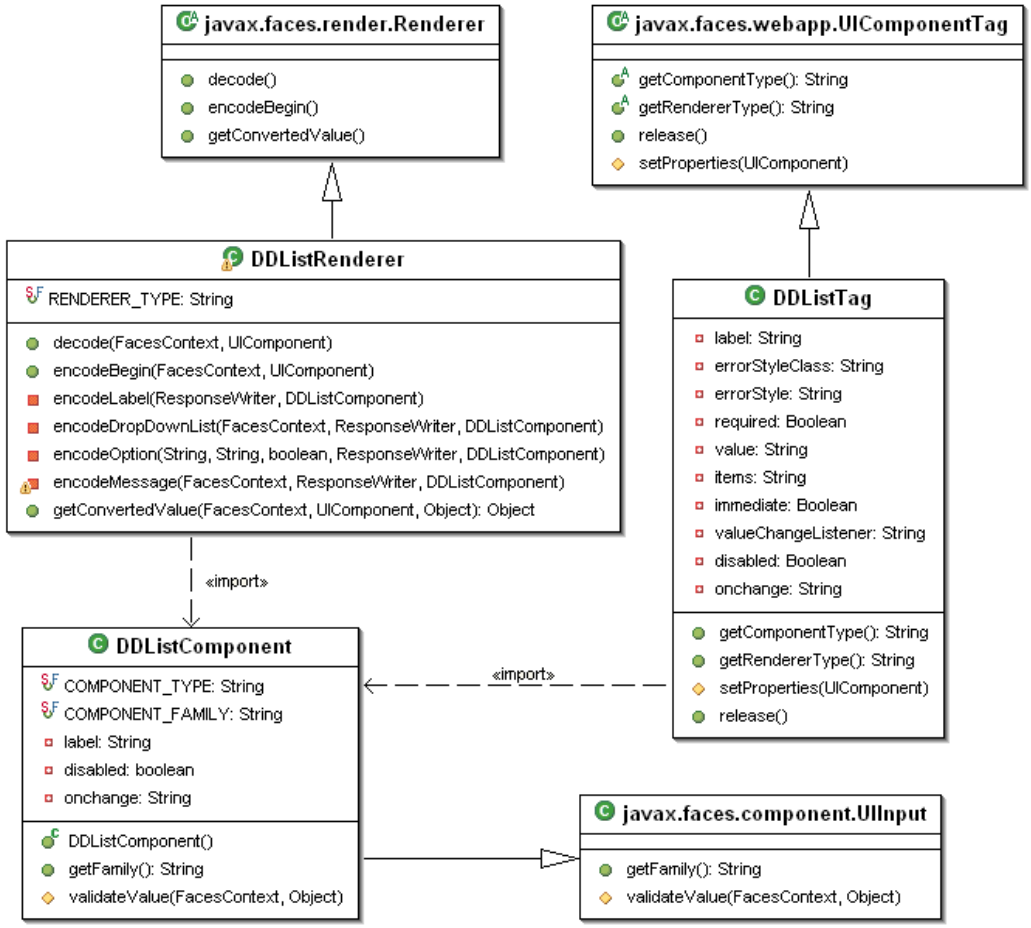


Figura 6 – DDListComponent, DDListTag, DDListRenderer

La classe *DDListComponent*, che estende *UIInput*, non prevede tutti gli attributi del componente, in quanto alcuni di essi possono essere direttamente gestiti dal Tag Handler, per cui in molti casi si preferisce ometterli nella definizione del componente. Essa prevede il metodo *validateValue()*, il quale esegue una validazione del valore immesso dall'utente, invocando semplicemente la versione della superclasse. Si osservi che in questo caso, mancano i metodi relativi al deconding ed all'encoding, poiché tali fasi vengono eseguite dal renderer.

La classe *DDListTag*, che deriva da *UIComponentTag*, specifica tutti gli attributi del componente che saranno presenti nel tag corrispondente ed implementa i tipici metodi *setProperties()* e *release()*, per caricare i valori degli attributi nel componente e rilasciare le risorse allocate.

Infine, la classe *DDListRenderer* estende la classe base *Renderer* per poter implementare il renderer HTML associato al componente.

Essa prevede al suo interno il tipico metodo *encodeBegin()*, il quale utilizza dei metodi privati per la visualizzazione del componente. Inoltre, è stato eseguito l'overidding del metodo *getConvertedValue()*, per poter gestire all'interno del renderer stesso la conversione del valore selezionato. In molte situazioni, tale operazione non è strettamente necessaria, in quanto è il framework che esegue le conversioni in maniera automatica. In questo caso, si rende necessaria l'operazione di conversione in quanto il componente viene utilizzato per visualizzare sia liste con valori numerici (es. importo scheda prepagata) che stringhe (es. stato).
La descrizione del tag è contenuta all'interno del file *components.tld* e per quanto riguarda la configurazione, è necessario registrare sia il componente che il renderer. Quest'ultimo va tipicamente inserito all'interno di un Render Kit, in base al dispositivo client di accesso, ma in questo caso è stato registrato in quello di "default" relativo al linguaggio HTML.

```
<renderer>
  <component-family>DropDownList</component-family>
  <renderer-type>DropDownListRenderer</renderer-type>
  <renderer-class>components.DDListRenderer</renderer-class>
</renderer>
```

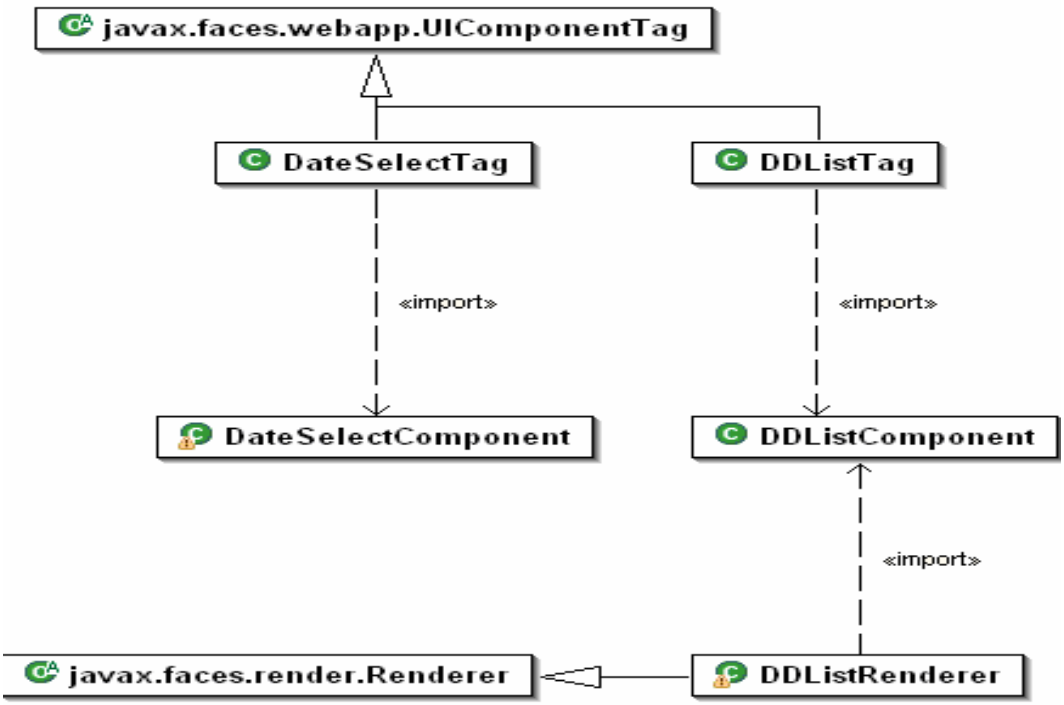


Figura 7 - Custom Components

4.3 Funzionalità del Sistema

4.3.1 Login, Registrazione e Richiesta password

La funzionalità di Login prevede un semplice form con due campi, all'interno dei quali specificare lo Username e la Password.

Figura 8 - Form di Login

L'implementazione è notevolmente semplice, in quanto le proprietà della classe che la implementa non sono altro che i campi previsti nel form : *userName* e P*assword*. Non è stato effettuato l'overriding del metodo *validate()*, in quanto la validazione dei dati immessi non è necessaria per due motivi :

- Lo username e la password possono contenere caratteri alfanumerici qualsiasi;
- Se non viene immesso alcun valore, si impedisce semplicemente l'accesso;

La funzionalità di Registrazione prevede un form per l'introduzione dei dati di accesso e dei dati anagrafici dell'utente.

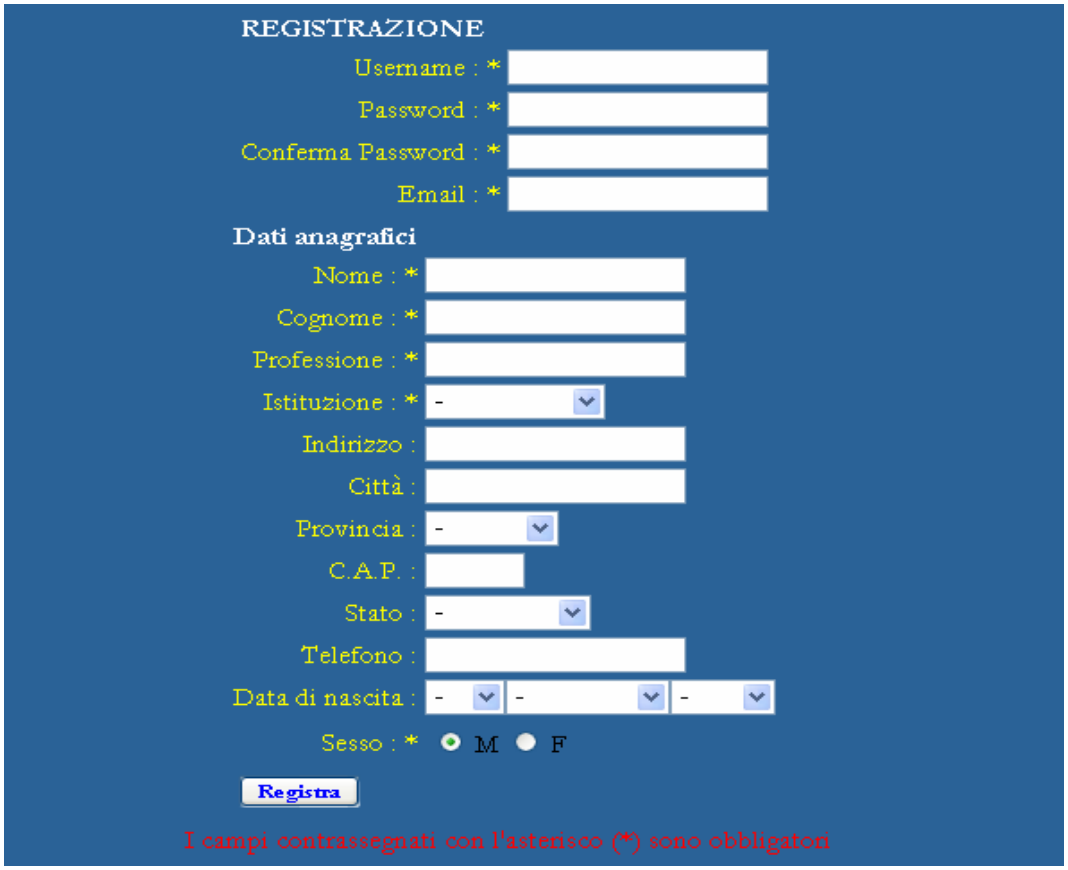


Figura 9 - Form di Registrazione

Infine, la funzionalità di richiesta della password prevede che l'utente inserisca molto semplicemente l'indirizzo email con cui si è registrato per poter ricevere una mail con i dati di accesso.

Figura 10 - Form di Richiesta Username/Password

4.3.2 Modifica dati utente ed amministratore

Attraverso la corrispondente funzionalità, l'utente ha la possibilità di modificare i propri dati di accesso ed i dati anagrafici.

Figura 11 - Form di Modifica dati Utente

Un'analoga funzionalità è prevista per l'amministratore, il quale però ha la possibilità di modificare la sua password, l'email ed il relativo smtp.

Figura 12 - Form di Modifica dati Utente

4.3.3 Visualizzazione e ricerca degli alimenti

La visualizzazione dell'elenco degli alimenti presenti in archivio è prevista sulla base dell'impostazione dei criteri di ricerca prefissati.

Tale modalità prevede in prima istanza la scelta delle fonti da consultare, la lingua degli alimenti su cui effettuare la ricerca ed infine il tipo di ricerca : ricerca per Alimento, ricerca per Componente e ricerca per Categoria.

Si avvisa inoltre che adoperare più fonti comporta, per alcune ricerche, anche l'utilizzo di certe caratteristiche sperimentali : vengono impiegati infatti i componenti e le categorie "standard", di cui si è già abbondantemente discusso precedentemente; se si seleziona invece una sola fonte, si rendono automaticamente disponibili le categorie ed i componenti originari di quella particolare fonte

Figura 13 – Impostazione Ricerca

Successivamente verranno visualizzati dei form per l'impostazione dei criteri di ricerca in base alla scelta effettuata.

4.3.3.1 Ricerca per Alimento

Questa ricerca dà la possibilià di inserire una o più parole descriventi l'alimento (le parole vengono poi utilizzate come se fossero legate dal connettivo logico "and"). E' poi possibile restringere la ricerca ad una particolare categoria, includendo o meno anche gli alimenti che non appartengono ad alcuna categoria. Viene poi visualizzata una lista degli alimenti che rispettano i criteri di ricerca impostati. Un elenco di esempio (dove è stata cercata la parola "pizza") è mostrato nelle figure seguenti.
Cliccando poi sul nome di un singolo alimento se ne ottiene la relativa scheda, che mostra delle informazioni generiche e la composizione del cibo stesso.

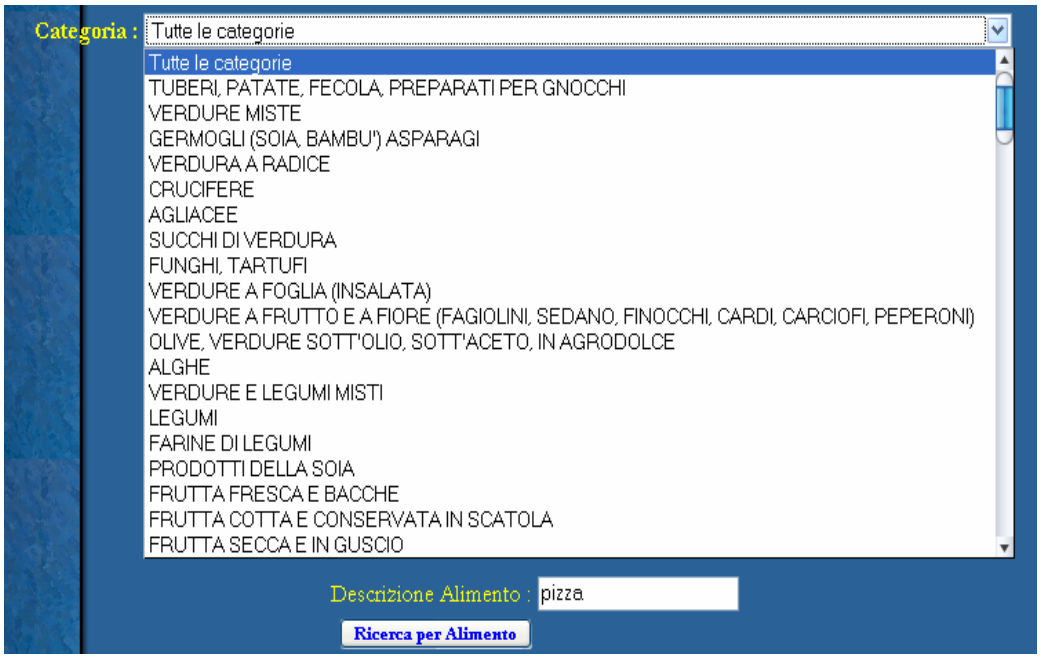


Figura 14 –Ricerca per Alimento

Codice Alimento	Descrizione Alimento	Nome Scientifico	Codice Nella Fonte
6704	PIZZA BIANCA		000700
6705	PIZZA CON POMODORO		000710
6706	PIZZA CON POMODORO E MOZZARELLA		000720

Figura 15 – Risultati della Ricerca per Alimento

4.3.3.2 Ricerca per Categoria

Con questo tipo di ricerca viene scelto il nome della categoria da un elenco a scomparsa. Il risultato di tale ricerca sarà un elenco molto simile a quello risultante dalla ricerca per alimento.

Anche da questa tabella è possibile accedere, come di consueto, alla scheda del singolo alimento: basta ciccare sul nome del cibo di cui si desidera ricevere maggiori informazioni.

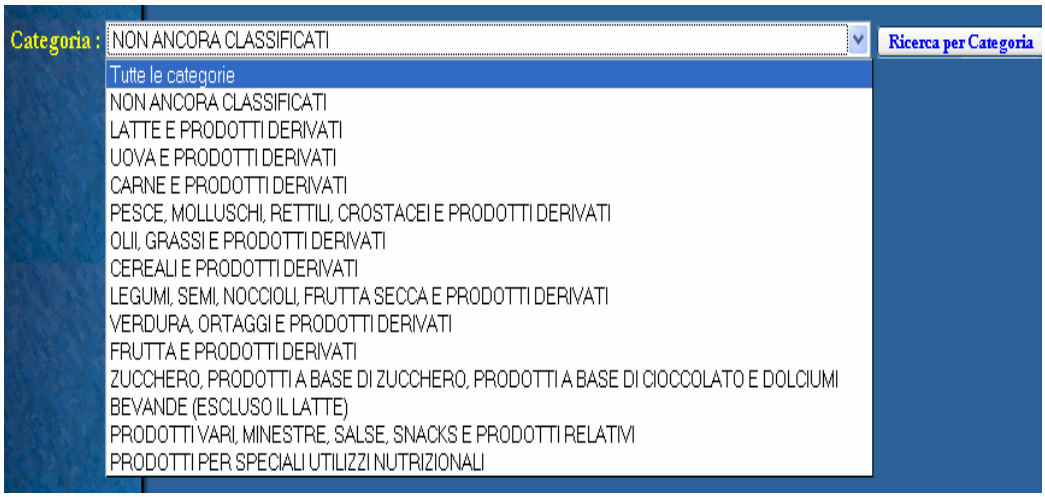


Figura 16 – Ricerca per Categoria in italiano

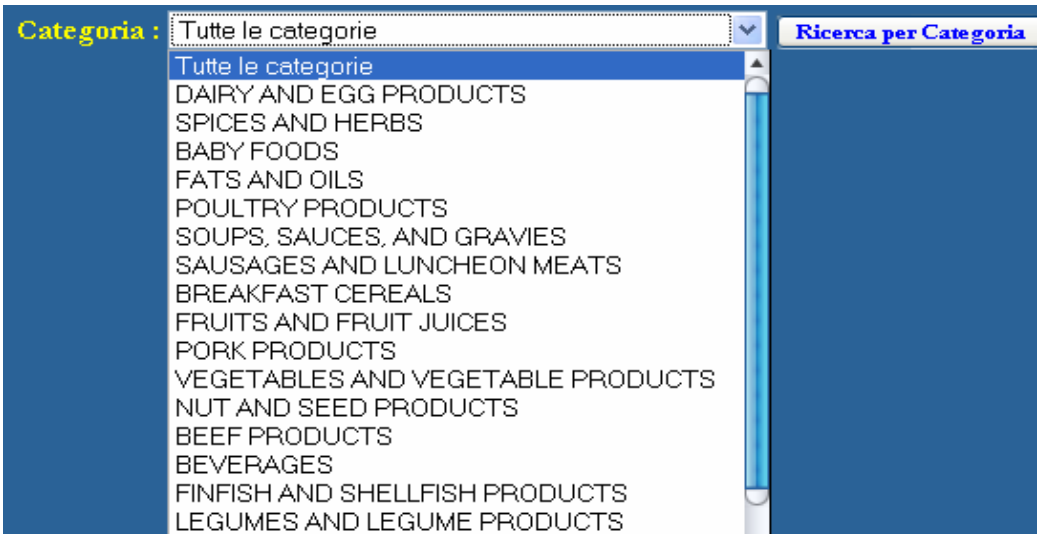


Figura 17 –Ricerca per Categoria in inglese

4.3.3.3 Ricerca per Componente

Un ulteriore utilissimo strumento di ricerca e confronto è quello della ricerca per componente. Tramite esso è possibile ottenere una lista di tutti gli alimenti delle fonti selezionate che contengono quel dato componente.
Tale lista può essere ordinata in vari modi e può includere o meno gli alimenti per cui il valore del componente è zero o non quantificabile (il nutriente si dice in tal caso presente "in tracce").
Si consiglia comunque, per motivi di leggibilità della lista e per velocizzare la query, di evitare di inserire gli alimenti con contenuto nullo o quasi nullo del componente. Questa ricerca infatti, essendo presenti più di ottomila e ottocento alimenti nel database, è intrinsecamente lenta e dà performance abbastanza buone solo quando il numero di fonti selezionate è esiguo.
Nella figura immediatamente successiva viene mostrato il form da riempire per effettuare questa ricerca.Si noti come, a partire da quest'ultimo elenco, si abbia immediato accesso ad altre importanti informazioni: cliccando sul nome dell'alimento si ottiene la scheda ad esso relativa.

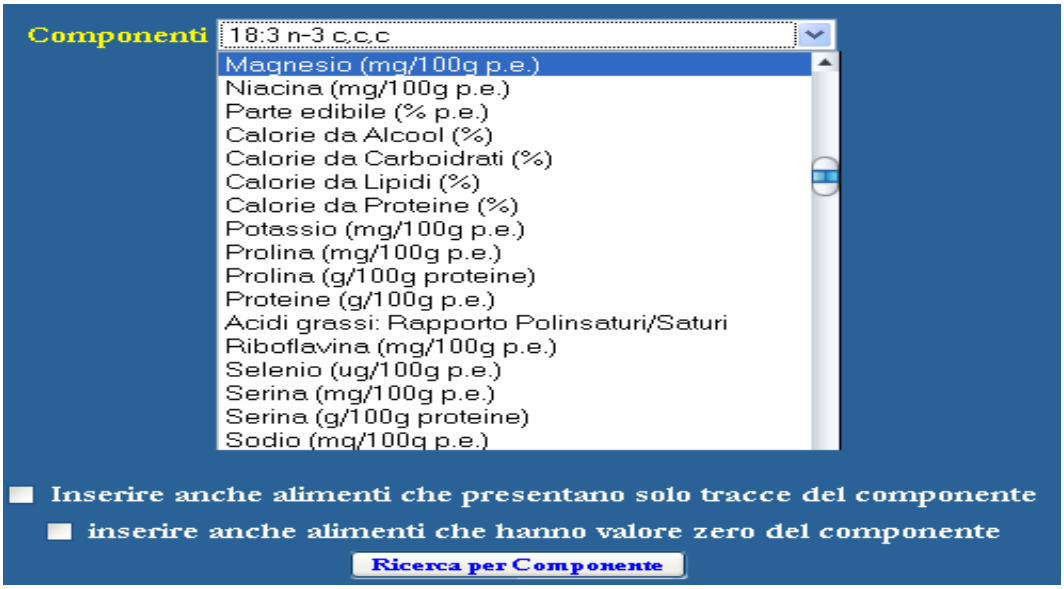


Figura 18 –Ricerca per Componente

Descrizione Alimento	Valore	Tracce	Fonte
FARRO	797.0	No	1
FRUMENTO TENERO	576.0	No	1
GRANO SARACENO	665.0	No	1
MAIS	482.0	No	1
MIGLIO	637.0	No	1

Figura 19 –Risultati della Ricerca per Componente

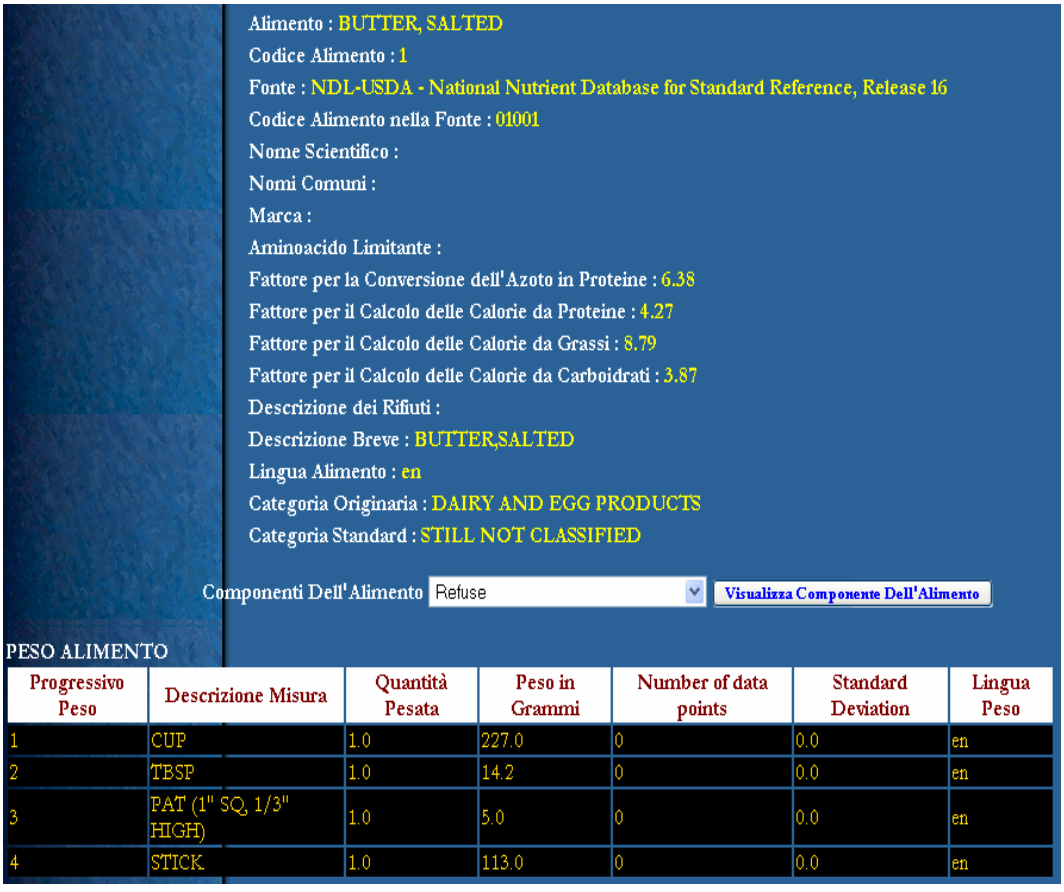
4.3.3.4 Visualizzazione informazioni Alimento

Una volta eseguita la richiesta di visualizzazione oppure terminata la ricerca, se ci sono degli alimenti che soddisfano i criteri impostati, viene visualizzato un elenco degli stessi. Cliccando poi sul nome di un singolo alimento se ne ottiene la relativa scheda, che mostra delle informazioni generiche e la composizione del cibo stesso.

Nell'esempio vediamo la scheda dell'alimento "butter, salted", preso dalla fonte USDA (che è quella in cui sono presenti più informazioni aggiuntive). La prima immagine mostra le caratteristiche generiche dell'alimento, la seconda parte della scheda invece illustra la composizione vera e propria, arricchita dalle più importanti informazioni statistiche.

Da notare che in queste schede compositive vengono sempre utilizzate le denominazioni dei componenti della fonte (possibilmente tradotte nella lingua in uso) e non i nomi del vocabolario "standard".

L'ordine con cui vengono mostrati i componenti inoltre è lo stesso dei report creati dalle fonti primarie nei rispettivi applicativi e/o libri.



Progressivo Peso	Descrizione Misura	Quantità Pesata	Peso in Grammi	Number of data points	Standard Deviation	Lingua Peso
1	CUP	1.0	227.0	0	0.0	en
2	TBSP	1.0	14.2	0	0.0	en
3	PAT (1" SQ, 1/3" HIGH)	1.0	5.0	0	0.0	en
4	STICK	1.0	113.0	0	0.0	en

Figura 20 – Scheda Alimento

INFORMAZIONI SUI VALORI DEL CIBO

Descrizione Componente : Calcium, Ca
Valore : 24.0
Tracce : No
Unità di Misura : mg
Espressione Misura : per 100g di sostanza grassa
Componente Standard [SIGLA] : caffeine [CAFFN]
Valore minimo : 19.0
Valore massimo : 30.0
Standard Error : 0.789
Number of data points : 17
Età dei Data Points (in ore) :
Lower 95% Error Bound : 22.021
Upper 95% Error Bound : 26.496
Number of Studies : 7
Degrees of Freedom : 3.8
Nutriente Arricchito Artificialmente : No
Codice dell'alimento utilizzato per attribuire il valore :
Determinazione Valore : ANALYTICAL DATA
Tipo di Valore : ANALYTICAL OR DERIVED FROM ANALYTICAL

Torna In Visualizza Alimento

COMMENTO STATISTICO

Descrizione Commento Statistico	Lingua
THE DISPLAYED DEGREES OF FREEDOM WERE COMPUTED USING SATTERTHWAITES APPROXIMATION (KORZ AND JOHNSON, 1988).	en
THE PROCEDURE USED TO ESTIMATE THE RELIABILITY OF THE GENERIC MEAN REQUIRES THAT THE DATA ASSOCIATED WITH EACH STUDY BE A SIMPLE RANDOM SAMPLE FROM ALL THE PRODUCTS ASSOCIATED WITH THE GIVEN DATA SOURCE(FOR EXAMPLE,MANUFACTURER,VARIETY,CULTIVAR).	en

FONTE SECONDARIA

Titolo	Autori	Editore	Anno	Nome Rivista	Numero Rivista	Volume	Citta	Stato	Pagina Iniziale	Pagina Finale
FDA TOTAL DIET STUDY	FOOD AND DRUG ADMINISTRATION (FDA), DHHS		1995							

Figura 21 – Scheda Componente dell'Alimento

4.3.4 Inserimento/Modifica e Cancellazione alimenti

Per effettuare l'inserimento di un nuovo alimento all'interno dell'archivio, l'amministratore ha a disposizione un form nel quale poter inserire le informazioni dell'alimento e successivamente dei suoi componenti. Per quanto riguarda la modifica e la cancellazione, tale funzionalità è disponibile nell'interfaccia che visualizza le informazioni di un certo alimento. Il form è praticamente uguale al precedente.

Figura 22 - Form di Inserimento di un Alimento

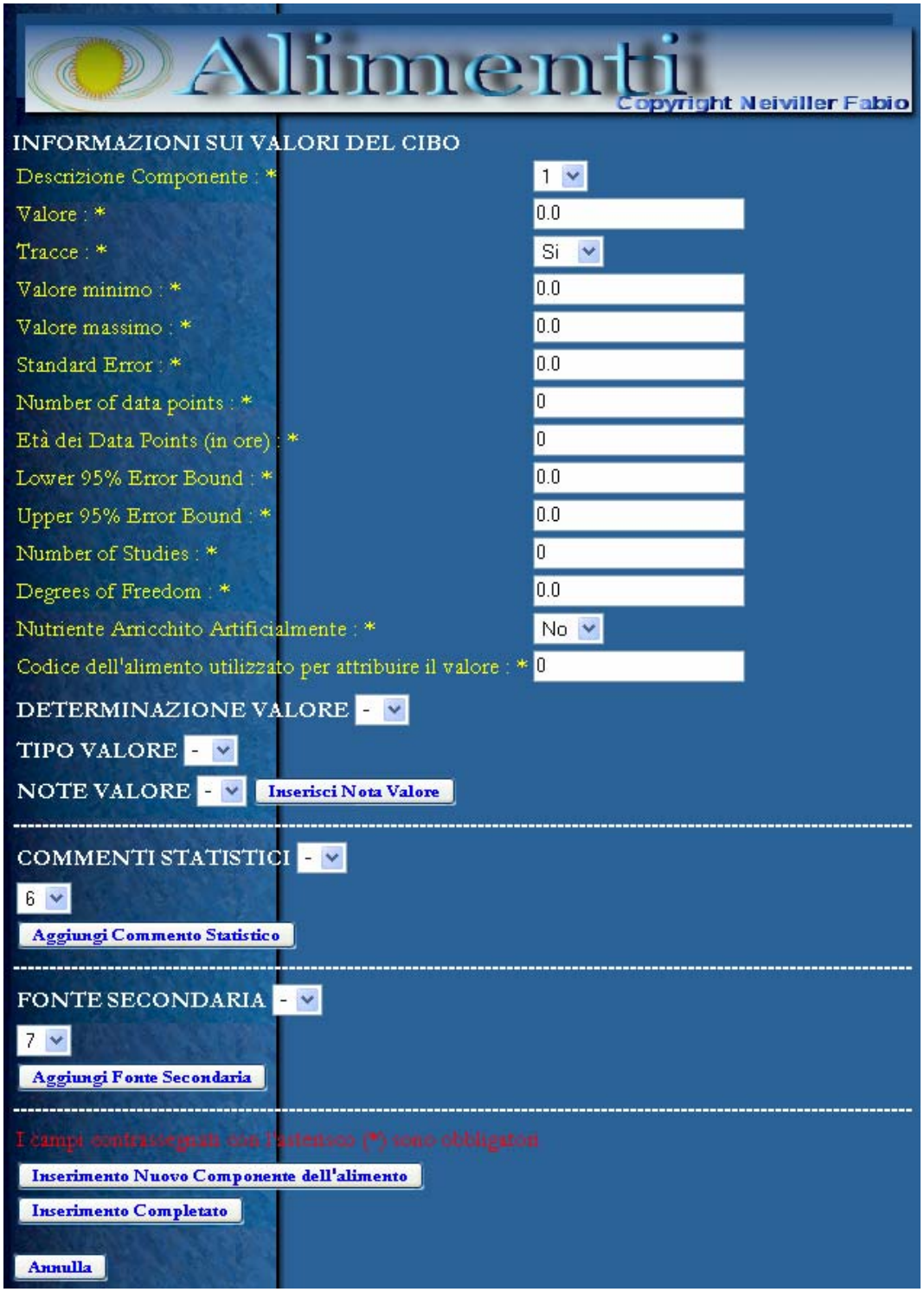


Figura 23 - Form di Inserimento del Componente di un Alimento

Figura 1 - Form di Modifica di un Alimento

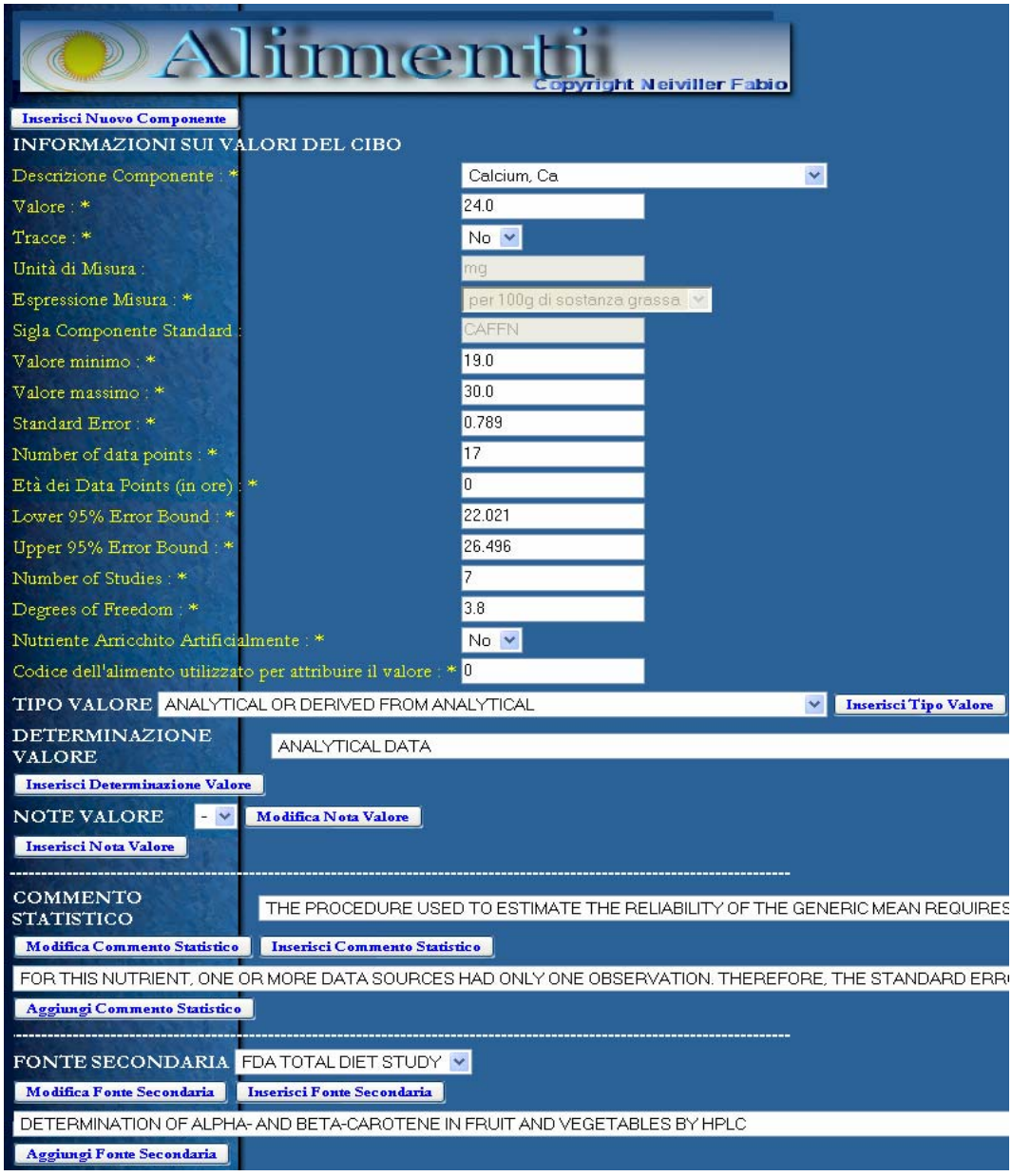


Figura 25 - Form di Modifica del Componente di un Alimento

4.3.5 Gestione Utenti

L'amministratore dispone di semplici funzionalità per poter gestire gli utenti registrati. In particolare, è possibile visualizzare l'elenco degli utenti e selezionare quelli da cancellare attraverso delle checkbox.

Figura 26 - Form di Visualizzazione ed eliminazione degli utenti

4.4 Internazionalizzazione (I18N)

Attraverso il meccanismo dell'Internazionalizzazione (I18N), è possibile visualizzare il testo presente nella Web application in lingue diverse, in relazione alla localizzazione geografica da cui si collega l'utente oppure sulla base delle impostazioni della lingua del browser. Nel caso dell'applicazione realizzata, si è previsto il supporto per la lingua italiana, considerata di default, e per la lingua inglese. In questo modo, senza la necessità di dover sviluppare due volte la medesima Web application, è possibile comunque garantire l'utilizzo di quest'ultima a persone di lingua differente.

Per quanto riguarda l'implementazione sono stati realizzati due Resource Bundle contenenti i messaggi da visualizzare nelle due lingue. Ciascuno di essi è costituito da una serie di coppie "key-message", attraverso le quali è possibile visualizzare un certo messaggio semplicemente specificando la chiave ad esso associata.

```
...
regStatoLabel=Stato :
regTelefonoLabel=Telefono :
regDataNascitaLabel=Data di nascita :
...
```

ResourceBundle IT

```
...
regStatoLabel=State :
regTelefonoLabel=Phone Number :
regDataNascitaLabel=Date of birth :
...
```

ResourceBundle EN

Come si può osservare, in entrambi i file (con estensione *properties*) le chiavi sono perfettamente identiche mentre i messaggi sono in lingua diversa. Inoltre, i file stessi hanno praticamente il medesimo nome, ossia *ApplicationResources* ed *ApplicationResources_en*, a meno del file in lingua inglese che ha il suffisso "*en*". Tipicamente, il file che non ha alcun suffisso è quello relativo alla lingua di default mentre i file associati alle altre lingue devono avere lo stesso nome del primo ma seguito da un suffisso che identifica la lingua.

Per la registrazione dei Resource Bundle è stato necessario specificare la localizzazione (*Locale*) di default e quella supportata, oltre al nome del file contenente i messaggi.

```
<application>
 <locale-config>
  <default-locale>it</default-locale>
  <supported-locale>en</supported-locale>
 </locale-config>
 <message-bundle>ApplicationResources</message-bundle>
</application>
```

Per quanto riguarda l'utilizzo dell'internazionalizzazione all'interno delle pagine dell'applicazione, prevede l'utilizzo del tag <*f:loadBundle*> in cui l'attributo *basename* specifica il nome del Resource Bundle mentre *var* permette di assegnargli un nome mediante il quale farvi riferimento all'interno della pagina JSP.

```
<f:loadBundle basename="ApplicationResources" var="bundle" />
```

Il cui risultato è il seguente :

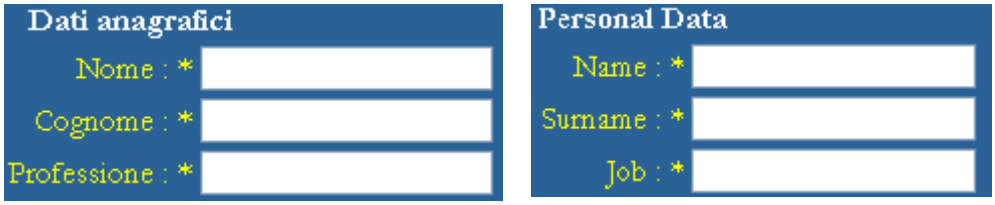


Figura 27 - Esempio di Form con localizzazioni differenti

Un limite legato all'Internazionalizzazione riguarda il fatto che, nel caso in cui la navigazione tra le pagine non sia definita attraverso collegamenti ipertestuali ma attraverso elementi grafici, non è possibile visualizzare un elemento al posto di un altro in base alla differente lingua di accesso. I Resource Bundle supportano il contenuto testuale e non eventuali immagini che costituisco i bottoni per la navigazione.

5. Validazione

Per la validazione dei dati immessi nei form da parte dell'utente JSF mette a disposizione un proprio meccanismo per la validazione e per la sua estensione, la scelta fatta nell'implementazione è fondata sulla riusabilità e sulla rapidità di sviluppo, sfruttando lì dove possibile tutti gli strumenti a disposizione.

5.1 Custom Validators

JavaServer Faces mette a disposizione soltanto tre validatori standard che permettono di valutare se un certo valore ha una certa lunghezza entro un minimo ed un massimo prefissati oppure se il medesimo valore rientra in un certo range. Per quanto riguarda l'obbligatorietà di un campo, quest'ultima viene specificata mediante l'attributo *required* del tag associato al componente in questione.
Un esempio di obbligatorietà di un campo può essere la richiesta dello Username in fase di registrazione, come mostrato in figura.

Figura 38 - Esempio di campo obbligatorio

Per quanto riguarda il controllo che un valore di un campo debba avere una lunghezza minima prefissata, si può fare riferimento al tag <*f:validateLength*> utilizzato in fase di registrazione per la richiesta della password, per la quale è stata fissata una lunghezza minima di 5 caratteri alfanumerici.

```
<h:inputSecret id="password"
                    value="#{utente.password}"
                    required="true"
                    maxlength="15">
  <f:validateLength minimum="5"/>
</h:inputSecret>
```

Figura 4 - Esempio di ValidateLength

Considerando il numero limitato di controlli standard che possono essere eseguiti, il framework mette a disposizione due meccanismi per la realizzazione di ulteriori validatori. Il primo consiste nel realizzare un metodo che esegua la validazione all'interno di un backing bean, il secondo prevede lo sviluppo di una classe che implementi l'interfaccia *Validator*, alla quale può poi essere associato o meno un tag. Tipicamente la prima soluzione è strettamente limitata all'uso dello specifico backing bean e non permette di utilizzare il validatore su più componenti che prevedono lo stesso tipo di controllo, a meno di non effettuare un copia-incolla del metodo di validazione nei vari backing beans. Per questo motivo, si è preferita la seconda soluzione grazie alla quale un unico validatore può essere facilmente associato a più componenti.

L'implementazione in JSF ha previsto lo sviluppo dei tre seguenti validatori :

- *EmailValidator* : permette di valutare se il formato di un indirizzo email sia o meno corretto;
- *SeqCifreValidator* : per valutare se un certo valore è rappresentato da una sequenza di cifre;
- *TelefonoValidator* : per verificare se un numero di telefono abbia un formato valido, del tipo prefisso-numero;

5.1.1 EmailValidator

Questo validatore viene utilizzato nella fase di registrazione di un utente oppure nel momento in cui quest'ultimo richiede l'invio di una mail con i propri dati di accesso, specificando il proprio indirizzo. La classe *EmailValidator* implementa l'interfaccia *Validator* della quale esegue l'overriding del metodo *validate()* per valutare se l'indirizzo email immesso dall'utente abbia un formato corretto.

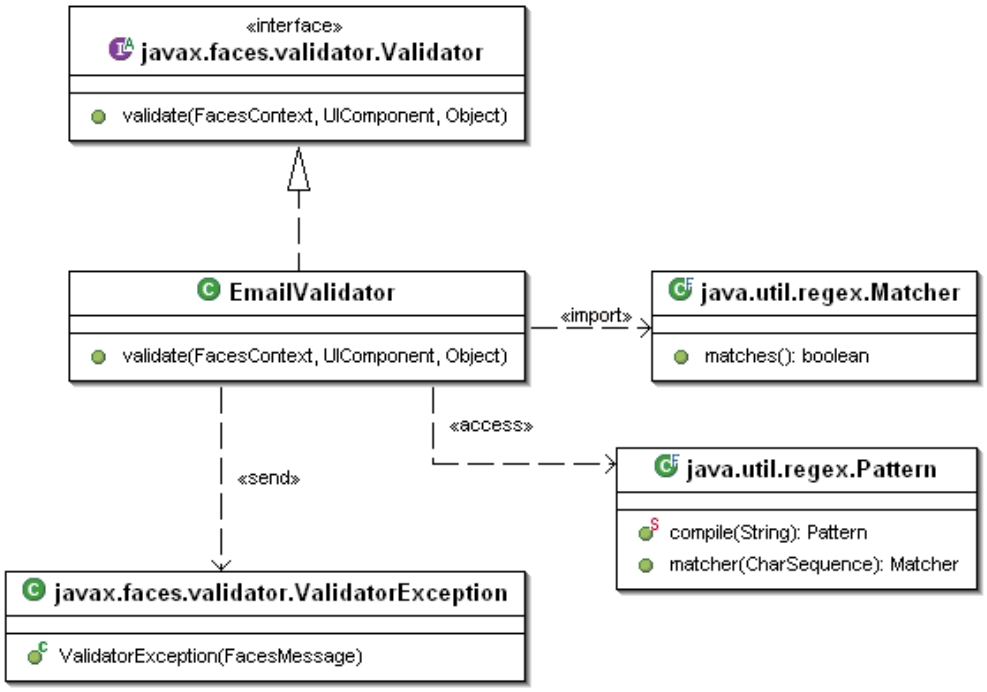


Figura 5 - EmailValidator

Per poter effettuare il controllo, all'interno del metodo *validate()* viene utilizzato lo strumento delle *Regular Expression* (Espressione Regolare) fornito dal linguaggio Java. In particolare, si specifica quale sia il pattern a cui deve essere conforme un indirizzo email per essere considerato corretto e si valuta se il valore immesso dall'utente corrisponda ad esso.

```
...
Pattern mask = Pattern.compile("\\w+@{1}\\w+\\.{1}\\w+");
Matcher matcher = mask.matcher(value.toString());

// se il valore non corrisponde al pattern
if (!matcher.matches()){
  ...
  ...
  throw new ValidatorException(msg);
}
...
```

Si osserva che se il valore immesso dall'utente non è conforme al pattern specificato, viene sollevata un' eccezione la quale contiene il messaggio da visualizzare all'utente. Tale messaggio viene prelevato dal Resource Bundle.
Il validatore ha ovviamente previsto una registrazione all'interno del file di configurazione, così come tutti gli elementi personalizzati che si creano con JSF.

```
<validator>
  <validator-id>emailValidator</validator-id>
  <validator-class>validators.EmailValidator</validator-class>
</validator>
```

Per quanto riguarda il suo utilizzo all'interno di una pagina JSP, si è preferito non realizzare un tag appropriato ma di utilizzare il tag standard messo a disposizione da JSF, ossia <*f:validator*>, il quale prevede l'attributo *validatorId* che permette di specificare l'identificativo del validatore da utilizzare, così come è stato registrato.

```
<h:inputText id="email"
                value="#{utente.email}"
                required="true"
                maxlength="50">
   <f:validator validatorId="emailValidator"/>
</h:inputText>
```

L'immissione da parte dell'utente di un indirizzo con un formato errato, prevede una segnalazione del tipo mostrato in figura.

Figura 61 - Esempio d'uso dell'EmailValidator

5.1.2 SeqCifreValidator

Nell'ambito dell'applicazione realizzata, ci sono una serie di campi dei form che prevedono l'immissione di un valore caratterizzato solo ed esclusivamente da una sequenza di cifre, come ad esempio il C.A.P.

Per verificare che l'utente immetta un valore corretto, è stato realizzata la classe *SeqCifreValidator* che implementa l'interfaccia *Validator*.

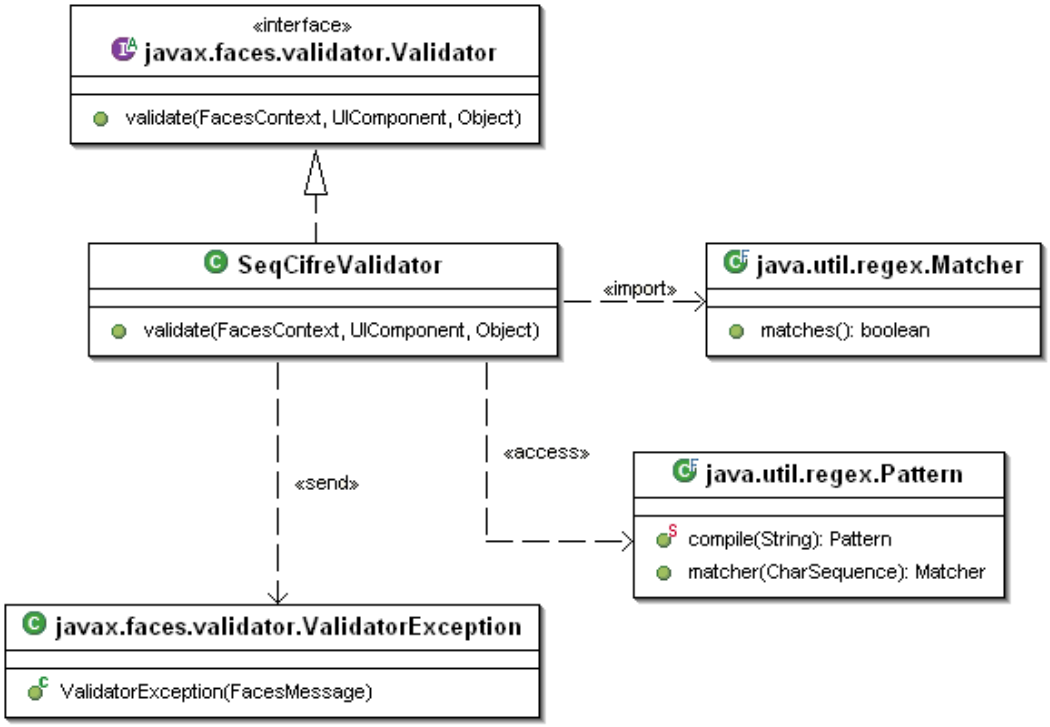


Figura 32 - SeqCifreValidator

All'interno del metodo *validate()* viene definito il pattern a cui deve essere conforme un valore caratterizzato esclusivamente da cifre e viene sollevata un'eccezione qualora il controllo abbia esito negativo, in modo da visualizzare un messaggio di errore all'utente.

```
...
Pattern mask = Pattern.compile("\\d+");
Matcher matcher = mask.matcher(value.toString());
// se il valore non corrisponde al pattern
if (!matcher.matches()){
  ...
  throw new ValidatorException(msg);
}
...
```

Infine, eseguita la registrazione nel file di configurazione, il validatore viene utilizzato mediante il tag <*f:validator*>.

```
<h:inputText id="cap"
                    value="#{utente.cap}"
                    maxlength="5"
          size="5">
  <f:validator validatorId="seqCifreValidator" />
</h:inputText>
```

Nel caso in cui il controllo abbia un esito negativo, viene visualizzato un opportuno messaggio di errore prelevato dal Resource Bundle.

Figura 73 - Esempio d'uso dell'EmailValidator

5.1.3 TelefonoValidator

Generalmente, il valore di un campo che deve rappresentare un numero di telefono è caratterizzato da un formato particolare del tipo “prefisso-numero”. Nel caso dell’applicazione realizzata, una situazione di questo tipo si verifica nella fase di registrazione di un utente o comunque nell’acquisto di una scheda prepagata, in cui tra i dati facoltativi c’è anche la richiesta del numero di telefono. Per poter garantire che l’utente immetta un valore valido, si è resa necessaria la realizzazione di un validatore opportuno mediante la classe *TelefonoValidator* che implementa l’interfaccia *Validator*.

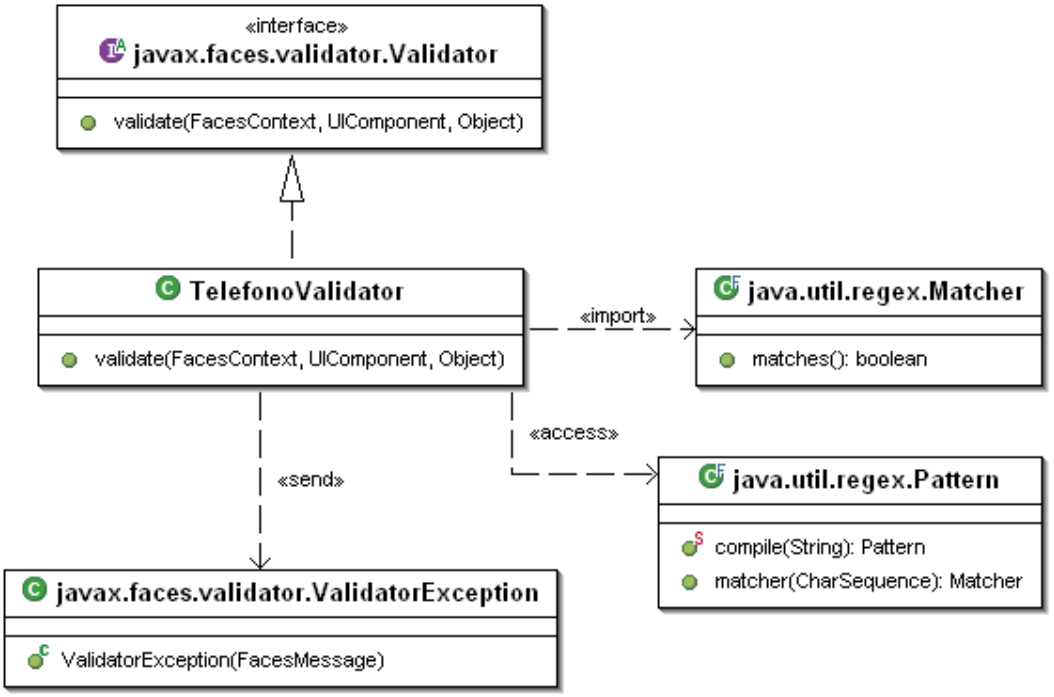


Figura 38 - TelefonoValidator

Il criterio adottato per la validazione è il medesimo degli altri validatori ed è basato sulla definizione di un pattern a cui deve essere conforme un numero di telefono per essere considerato valido.

```
...
Pattern mask = Pattern.compile("\\d+-{1}\\d+");
Matcher matcher = mask.matcher(value.toString());

// se il valore non corrisponde al pattern
if (!matcher.matches()){
  ...
  ...
  throw new ValidatorException(msg);
}
...
```

Allo stesso modo, il validatore è stato registrato nel file di configurazione ed utilizzato nelle pagine JSP mediante il tag <*f:validator*>.

```
<h:inputText id="telefono"
                    value="#{utente.telefono}"
                    maxlength="15">
  <f:validator validatorId="telefonoValidator" />
</h:inputText>
```

Nel caso di esito negativo della validazione, il messaggio di errore prelevato dal Resource Bundle è del tipo in figura.

Figura 9 - Esempio d'uso del TelefonoValidator

6. Model

Il Model può essere considerato il cuore dell'applicazione realizzata, in quanto è costituito da tutte quelle classi che definiscono la business-logic. Così come anticipato in precedenza, l'implementazione in JSF ha previsto lo sviluppo dei backing beans e dei relativi metodi che realizzano le funzionalità del sistema.
Inoltre, sono state implementate alcune classi comuni che hanno i seguenti ruoli :

- una serie di eccezioni che possono essere sollevate da particolari elaborazioni;
- una serie di classi che gestiscono le operazioni di accesso alla base di dati;

6.1 Classi Exception

L'applicazione prevede alcune funzionalità nell'ambito delle quali si possono verificare delle condizioni di errore che vanno opportunamente gestite. Uno dei metodi più "eleganti" è l'utilizzo del meccanismo delle eccezioni messo a disposizione dal linguaggio Java. Per questo motivo, sono state realizzate le seguenti classi che estendono la classe base *Exception* :

- *PasswordErrata* : eccezione che si verifica nel caso in cui un utilizzatore abbia tentato un accesso al sistema sbagliando la password;
- *UsernameErrata* : eccezione sollevata nel momento in cui un utilizzatore tenta di effettuare il login con una username non registrata;
- *UsernameEsistente* : eccezione che si verifica in fase di registrazione, quando l'utente ha specificato una username già presente nella base di dati;
- *RuoloBloccato* : eccezione sollevata quando un utilizzatore tenta di eseguire il login con username e password corretti, ma il suo account risulta bloccato in quanto è in attesa di convalida da parte dell'amministratore;
- *RuoloLoggato* : eccezione che viene sollevata nel momento in cui un utilizzatore tenta di accedere al sistema ma risulta già loggato;

Queste classi sono un'estensione della classe *Exception* alla quale non aggiungono alcuna funzionalità. Lo scopo della loro realizzazione è soprattutto concettuale, per poter distinguere all'interno del codice le eccezioni di tipo diverso. Soltanto la classe *PasswordErrata* è leggermente diversa dalla altre, in quanto ha la proprietà *idRuolo* che permette di individuare l'utilizzatore che ha immesso lo username corretto ma la password sbagliata.

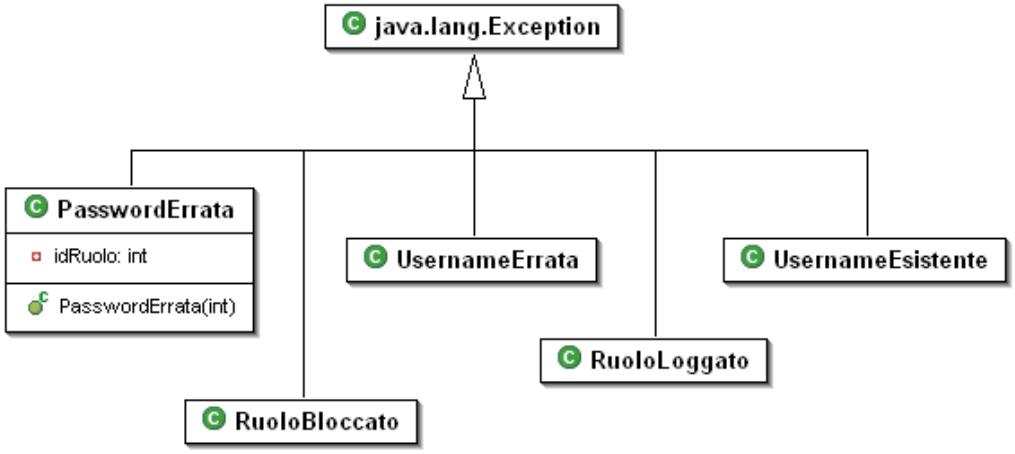


Figura 10 - Classi Exception

6.2 Classi di interfacciamento con il DataBase

Nella realizzazione di un sistema software che debba funzionare in locale oppure in rete, ci sono tipicamente degli oggetti per i quali va garantita la persistenza attraverso l'utilizzo delle basi di dati. Per poter gestire le operazioni di lettura e scrittura nel database, sono previste due possibili soluzioni :

- ogni oggetto gestisce la propria persistenza in maniera autonoma, prevedendo dei metodi che accedono alla base di dati;
- ogni oggetto delegata la gestione delle propria persistenza ad un'altra classe che mette a disposizione una serie di metodi di accesso al database;

Con lo scopo di disaccoppiare le funzionalità della business-logic dalle operazioni di accesso alla base di dati, generalmente si preferisce adottare la seconda soluzione. Ciò è stato fatto anche nella Web application realizzata, implementando il sottolivello Data Access del Model attraverso un'ulteriore insieme di classi, una per ciascuna corrispondente classe della business-logic.

Per l'accesso vero e proprio alla base di dati, sono state utilizzate le seguenti classi JDBC (Java DataBase Connectivity) :

- *Connection* : rappresenta una connessione alla base di dati;
- *Statement* : definisce una generica query da eseguire sul database;
- *ResultSet* : rappresenta un gruppo di record, risultato di una query eseguita sulla base di dati;
- *DriverMananger* : classe che stabilisce la connessione con la base di dati utilizzando i driver JDBC;
- *SQLException* : definisce un'eccezione generica che è sollevata nel caso in cui si verifichi un errore di accesso al database;

Ogni classe realizzata prevede una serie di proprietà private che rappresentano la connessione (*conn*), una generica query (*stmt*), un resultSet (*rs*) ed infine la stringa di interrogazione (*sQuery*).
Per poter garantire l'utilizzo dell'applicazione con un qualsiasi tipo di DBMS (es. MySQL, MS SQL Server, MS Access, Oracle, ...) e facilitarne il passaggio dall'uno all'altro, è stata realizzata la classe astratta *InfoDBMS* che non ha metodi ma eslusivamente le seguenti proprietà :

- *driver* : il driver da utilizzare per la connessione al database;
- *connString* : la stringa di connessione che specifica l'URL del database;
- *user* : il nome dell'utente che accede al database con opportuni privilegi;
- *userPassword* : la password dell'utente suddetto;

Le classi che si interfacciano con il database utilizzano le proprietà della classe suddetta, per impostare i parametri di accesso con i quali stabilire la connessione alla base di dati. In questo modo, è possibile utilizzare di volta in volta un DBMS differente, semplicemente modificando i campi statici all'interno della classe *InfoDBMS*, senza alcun intervento all'interno delle altre classi. Un ulteriore vantaggio è dato dal fatto che la configurazione è completamente centralizzata in un'unica classe.

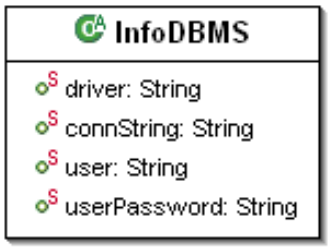


Figura 37 - InfoDBMS

Per quanto riguarda l'implementazione dei metodi di accesso, la struttura tipica di ciascuno di essi è la seguente.

```
...
try {

 Class.forName(InfoDBMS.driver);
 conn=DriverManager.getConnection(InfoDBMS.connString,
                                  InfoDBMS.user,
                                  InfoDBMS.userPassword);

 stmt=conn.createStatement();

... creazione della query string ...

 rs=stmt.executeQuery(sQuery);

... gestione del resultset ...

} catch (SQLException e) {

... gestione eccezioni ...

} finally {

 try {if (rs !=null) rs.close();} catch (SQLException se) {}
 try {if (stmt !=null) stmt.close();} catch (SQLException se) {}
 try {if (conn !=null) conn.close();} catch (SQLException se) {}
}
...
```

Viene eseguita un'operazione di caricamento dei Driver JDBC e di apertura della connessione alla base di dati specificata, dopodichè si crea la query string contenente l'interrogazione, la si esegue e si manipola il gruppo di record risultanti. E' prevista inoltre la gestione di un'eventuale eccezione sollevata e la chiusura definitiva degli oggetti utilizzati.

Per quanto riguarda le informazioni di accesso alla Web application è stata realizzata la classe *AccessoDB*, la quale dispone dei metodi per inserire, aggiornare e leggere un record nella tabelle degli accessi. Il metodo *inserisci()* ed *aggiorna()* prevedono in ingresso un oggetto della classe *Accesso*, oltre all'identificativo del ruolo a cui le informazioni si riferiscono e la tipologia del ruolo stesso, utente oppure amministratore. Essi non restituiscono alcun valore, in quanto il loro scopo è eseguire un'operazione di scrittura nella base di dati. Il metodo *leggi()* prevede soltanto gli ultimi due parametri suddetti, accede al database e restituisce l'oggetto *Accesso* corrispondente. Infine, ci sono due metodi privati che permettono la conversione e la formattazione degli oggetti *Date*, sottolineando però che le date memorizzate all'interno del database non sono di tipo stringa.

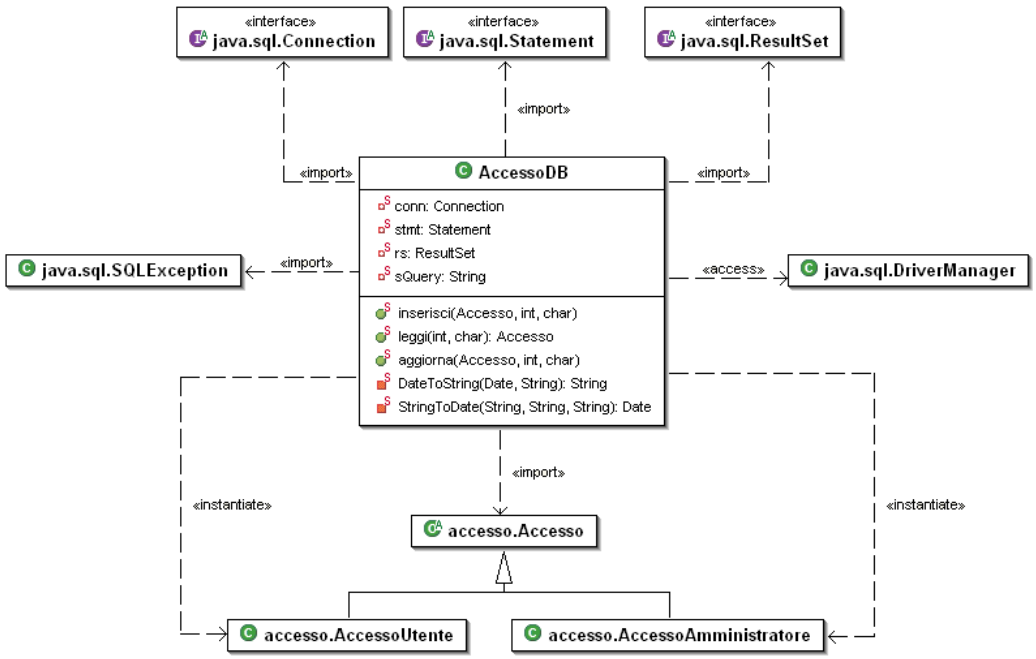


Figura 38 - AccessoDB

Le informazioni che riguardano l'amministratore sono gestite attraverso la classe *AmministratoreDB* che mette a disposizione i metodi per eseguire l'operazione di login, per leggere ed aggiornare un record ed infine bloccare l'amministratore.

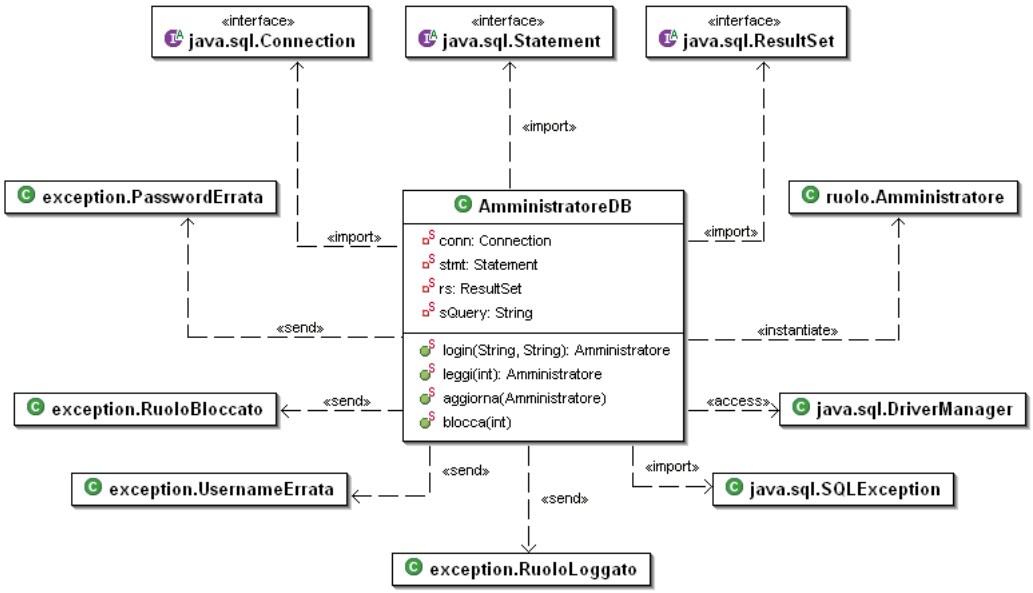


Figura 39 - AmministratoreDB

In particolare, il metodo *login()* prevede in ingresso la username e la password dell'amministratore e ne verifica la validità, con la possibilità di sollevare una delle eccezioni introdotte in precedenza, se si verifica una particolare condizione di errore. Ovviamente, questo metodo è invocato dalla funzionalità della business-logic che permette di effettuare il login ed il suo scopo è solo ed esclusivamente di accesso alla base di dati. I metodi *leggi()* e *blocca()* prevedono in ingresso l'identificativo dell'amministratore di cui leggere le informazioni oppure da bloccare, mentre *aggiorna()* riceve un oggetto *Amministratore*.

In maniera del tutto analoga è definita la classe *UtenteDB* che permette di garantire la persistenza delle informazioni dell'utente. Esso mette a disposizione i metodi tipici per la lettura, l'aggiornamento, l'inserimento e l'eliminazione di un record. I metodi *leggi()* ed *elimina()* prevedono in ingresso l'identificativo dell'utente di cui leggere le informazioni oppure da eliminare, mentre i metodi *aggiorna()* ed *inserisci()* prevedono un oggetto *Utente*. E' da osservare che l'operazione di inserimento è piuttosto generica, in quanto essa viene utilizzata nella funzionalità di registrazione della business-logic, certamente più complessa.

Il metodo *login()* opera allo stesso modo come per la classe *AmministratoreDB*. Inoltre sono previsti i seguenti ulteriori metodi :

- *blocca()* : permette di bloccare o di sbloccare un utente;
- *elencoUtenti()* : produce la lista degli utenti registrati;
- *esisteUsername()* : verifica l'esistenza di uno username in fase di registrazione;
- *esisteEmail()* : verificano l'esistenza di un indirizzo email. Viene utilizzato nella funzionalità di richiesta via mail delle informazioni di accesso da parte dell'utente e restituisce l'identificativo di quest'ultimo in caso di esito positivo;

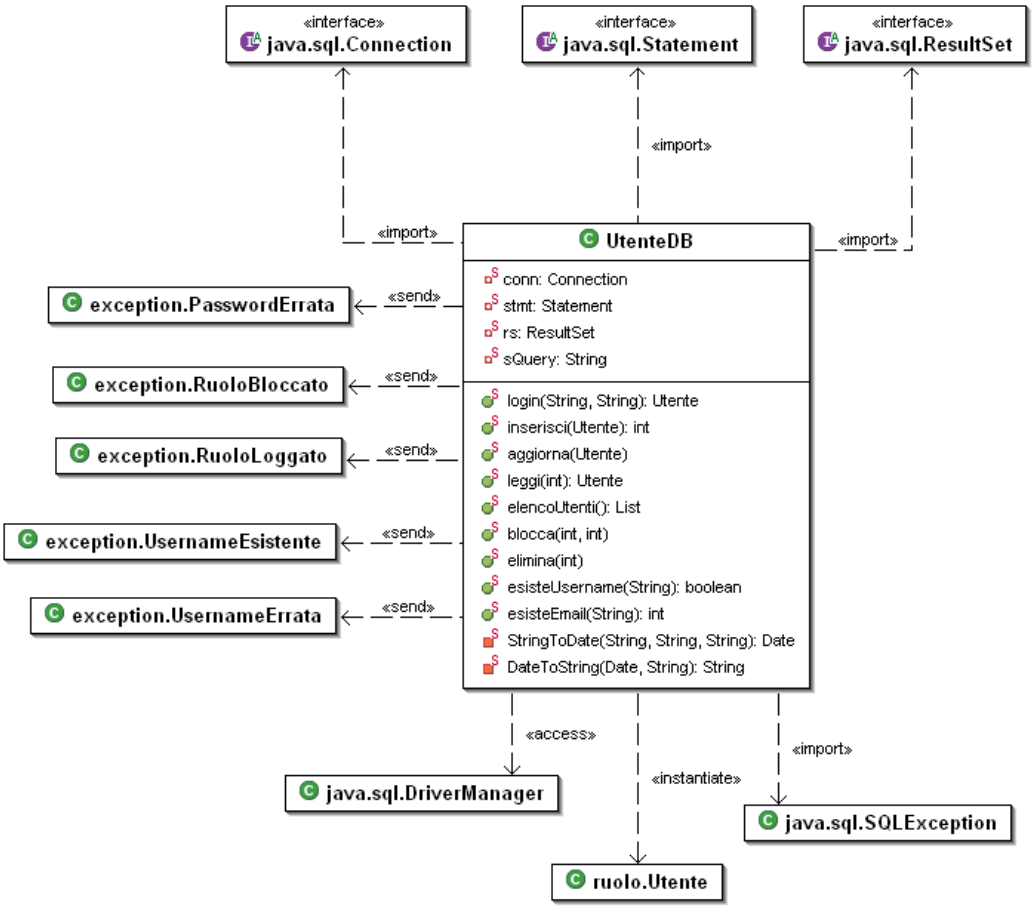


Figura 40 - UtenteDB

La gestione della permanenza di tutto ciò che è strettamente legato agli alimenti viene eseguita mediante le seguenti classi :

- *AlimentoDB*;
- *CategoriaDB*;
- *CategoriaOriginariaDB*;
- *CategoriaStandardDB*,
- *CommentoStatisticoDB*,
- *ComponenteDB*,
- *ComponenteStandardDB*,
- *DeterminazioneValoreDB*,
- *FonteDB*,
- *FonteSecondariaDB*,
- *TipoValoreDB*

Attraverso la classe *AlimentoDB* è possibile gestire le informazioni di ciascun alimento all'interno dell'archivio. Oltre ai tipici metodi di inserimento, eliminazione, lettura ed aggiornamento è previsto il metodo *ricerca()* che permette di caricare dal database una lista di alimenti. Esso viene utilizzato nel caso in cui viene effettuata un'operazione di ricerca in base a dei criteri specificati.

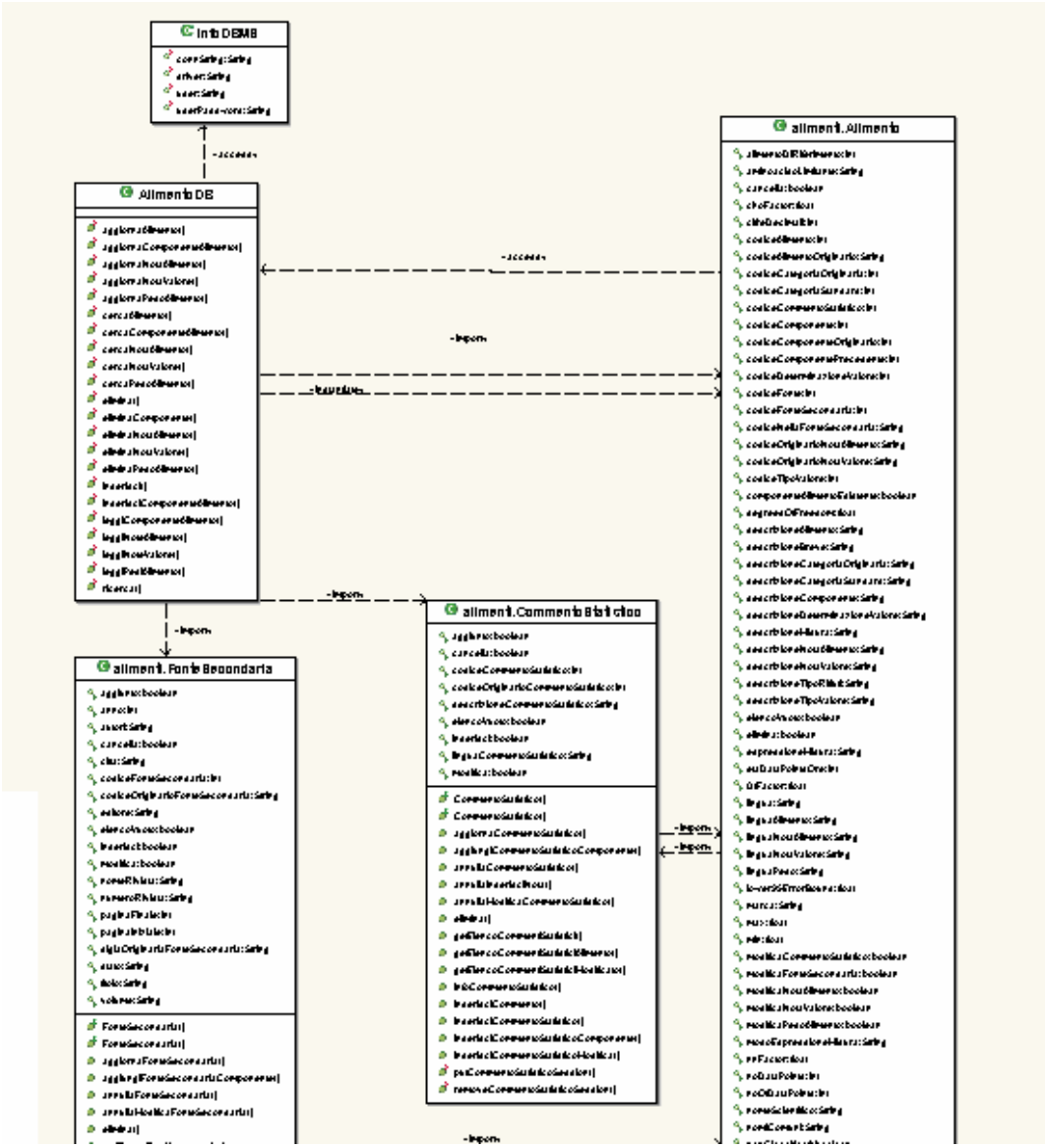


Figura 41 - AlimentoDB

Le altre classi implementate, hanno una struttura simile e svolgono pressocchè le stesse operazioni, ovviamente su dati diversi, quindi risulta inutile rappresentare

6.4 Business-Logic e funzionalità del sistema

Le classi che costituiscono la business-logic sono caratterizzate dalle seguenti caratteristiche :

- costituiscono a tutti gli effetti i backing beans, per cui la maggior parte dei metodi restituiscono gli *outcome* per la navigazione essendo invocati direttamente dalle pagine JSP;
- i metodi accedono agli oggetti del framework, così come alle variabili di sessione e di ciascuna richiesta;

Facendo riferimento alle elaborazioni eseguite per ciascuna funzionalità, basta osservare che sono concentrate nei metodi delle classi stesse.
Una precisazione è doverosa per quanto riguarda l'inserimento e la lettura degli oggetti del sistema all'interno delle variabili di sessione. Queste ultime sono ampiamente utilizzate per memorizzare alcuni oggetti della business-logic che contengono le informazioni associate all'utilizzatore corrente, di cui bisogna mantenerne lo stato durante la navigazione. Per poter gestire le operazioni di lettura e scrittura in sessione, le classi sono state dotate di alcuni metodi statici che ovviamente necessitano dell'accesso all'istanza della classe *HttpSession*.
Tali metodi statici, previsti esclusivamente per alcune classi, hanno una nomenclatura di questo tipo :

- *get*Classe*Session* : permette di recuperare dalla sessione un oggetto appartenente a questa classe;
- *remove*Classe*Session* : elimina dalla sessione l'oggetto della classe;
- *put*Classe*Session* : inserisce all'interno della sessione un oggetto di questa classe;
- *is*Classe*Session* : verifica se un oggetto della classe è presente all'interno della sessione corrente;

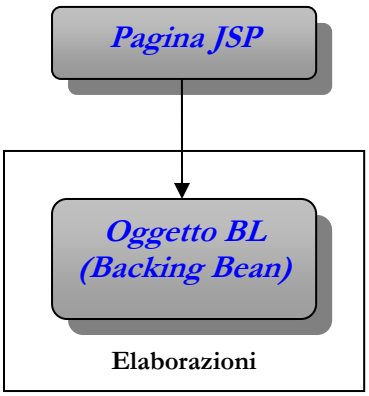


Figura 42 - Accesso agli oggetti della Business-Logic

Di seguito sono descritte le classi che compongono il Model, evidenziando l'interazione tra di esse.

6.4.1 Package accesso

Questo package contiene le classi che permettono di gestire le informazioni che riguardano tutte le operazioni di login e di logout eseguite dagli utenti e dall'amministratore. Esso è caratterizzato dalla classe astratta *Accesso* dalla quale derivano le classi *AccessoUtente* ed *AccessoAmministratore* per distingure, unicamente da un punto di vista concettuale, le informazioni di accesso dell'utente e dell'amministratore.
In entrambe le implementazioni, le proprietà delle classi sono le seguenti :

- *id* : identificativo dell'accesso;
- *dataOraLogin*, *dataOraLogout* : data ed ora in cui è stato eseguito il login ed il logout;
- *loginValido* : valore booleano che indica se l'accesso è andato a buon fine o meno;
- *indirizzoIP* : indirizzo IP del client;

Ovviamente, nella classe *Accesso* i metodi sono tutti astratti ma sono implementati all'interno delle classi derivate e sono i seguenti :

- *inserisci()* : permette di inserire le informazioni relative ad un'azione di login. Prevede in ingresso l'identificativo dell'utilizzatore che ha effettuato l'accesso;
- *leggi()* : permette di leggere le informazioni dell'accesso di un utilizzatore, specificando in ingresso il suo identificativo;
- *aggiorna()* : permette di aggiornare le informazioni dell'accesso in fase di logout, ricevendo in ingresso l'identificativo dell'utilizzatore che sta abbandonando il sistema;

Per quanto riguarda i metodi di accesso alla sessione, è stato utilizzata la classe del framework *FacesContext*.

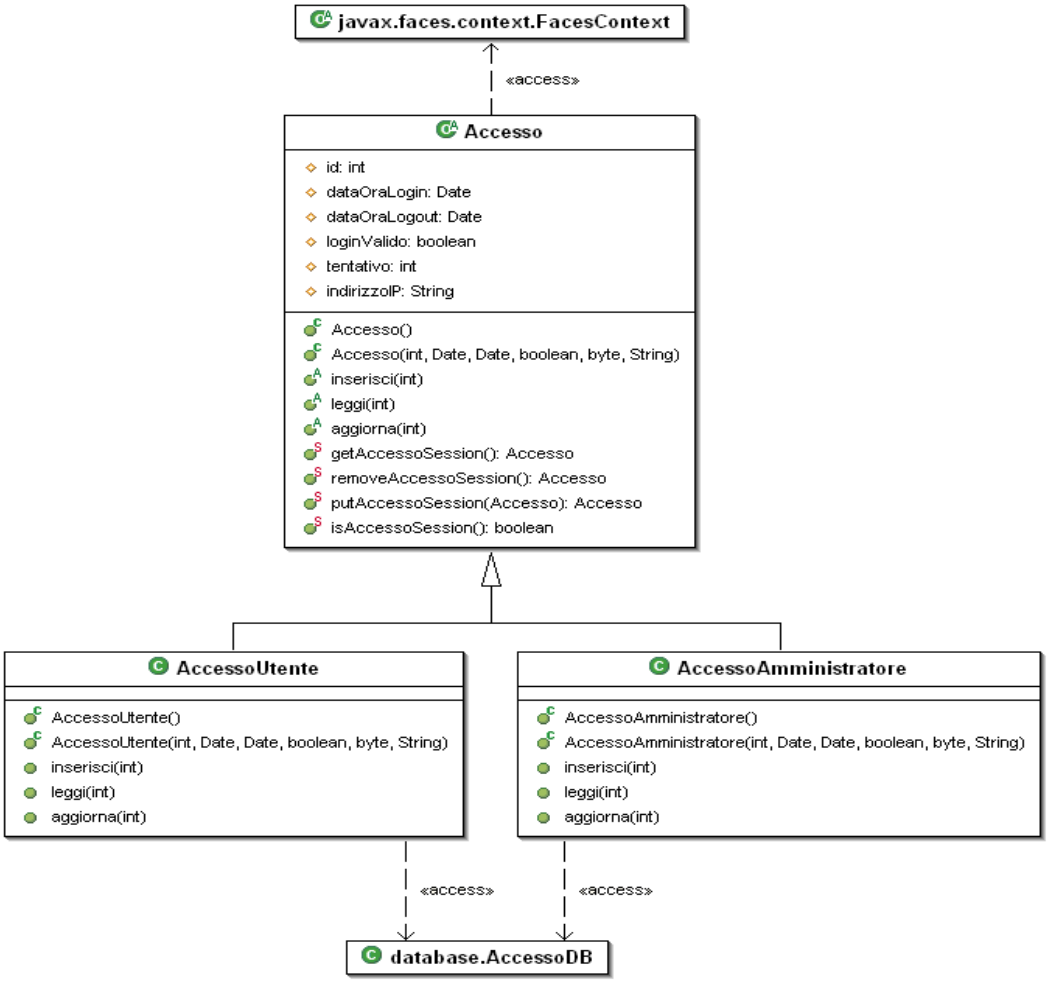


Figura 43 – package *accesso*

6.4.2 Package ruolo

Il package *ruolo* contiene le classi relative agli utilizzatori della Web application. In particolare, la classe *Ruolo* generalizza le due sottoclassi *Utente* ed *Amministratore*, tenendo conto del fatto che prima di essere registrato o comunque di eseguire il login, un utilizzatore dell'applicazione non può essere classificato. In entrambe le implementazioni, essa prevede le seguenti proprietà :

- *id* : identificativo univoco del ruolo;
- *userName* : username del ruolo;
- *password* : password di accesso del ruolo;
- *bloccato* : indica se il ruolo risulta bloccato o meno;
- *loggedIn* : indica se il ruolo è loggato o meno al sistema;

Ovviamente, esse sono ereditate dalle classi *Utente* ed *Amministratore* essendo informazioni comuni ad entrambi, con la differenza che mentre la seconda non aggiunge ulteriori proprietà, la prima ha delle proprietà in più che rappresentano tutti i dati anagrafici dell'utente. Inoltre, è prevista la proprietà *dataOraRegistrazione*. Nell'implementazione la classe *Utente* necessita di una proprietà in più, *elimina*, che tiene conto del fatto che l'utente sia stato selezionato o meno all'interno dell'elenco degli utenti registrati per poter essere cancellato.
Prima di descrivere in linea generale il comportamento dei metodi previsti nelle classi, è opportuno osservare le dipendenze che ci sono tra di esse e le altre classi del Model oltre ad eventuali classi del framework adottato, in modo da poter comprendere alcune differenze tra le due implementazioni.
Nell'implementazione con JSF, le classi utilizzano fortemente *FacesContext* per poter accedere alle variabili di sessione, mentre con Struts soltanto le classi *Utente* ed *Amministratore* accedono alle istanze di *HttpServletRequest* ed *HttpSession*. Queste ultime sono state utilizzate per far gestire a tali classi la propria permanenza in sessione ma, come anticipato, non sono strettamente necessarie e potrebbero essere usate dalle action esterne. La classe *Ruolo* non è assolutamente legata a queste classi, in quanto il suo uso principale avviene nella action di login , *LoginAction*, che gestisce la persistenza in sessione delle informazioni di un utente oppure dell'amministratore, utilizzando i metodi delle classi corrispondenti.

La classe *Ruolo* utilizza le classi *UtenteDB* ed *AmministratoreDB* per poter effettuare l'operazione di login, così come è legata alle eccezioni che possono essere sollevate in fase di accesso al sistema. Inoltre, *Ruolo* utilizza la classe *Accesso* e le sue derivate per poter gestire il salvataggio delle informazioni di accesso. Infine, la classe *Utente* accede a tutte quelle classi legate a funzionalità accessibili dall'utente, tra cui *Mail*.

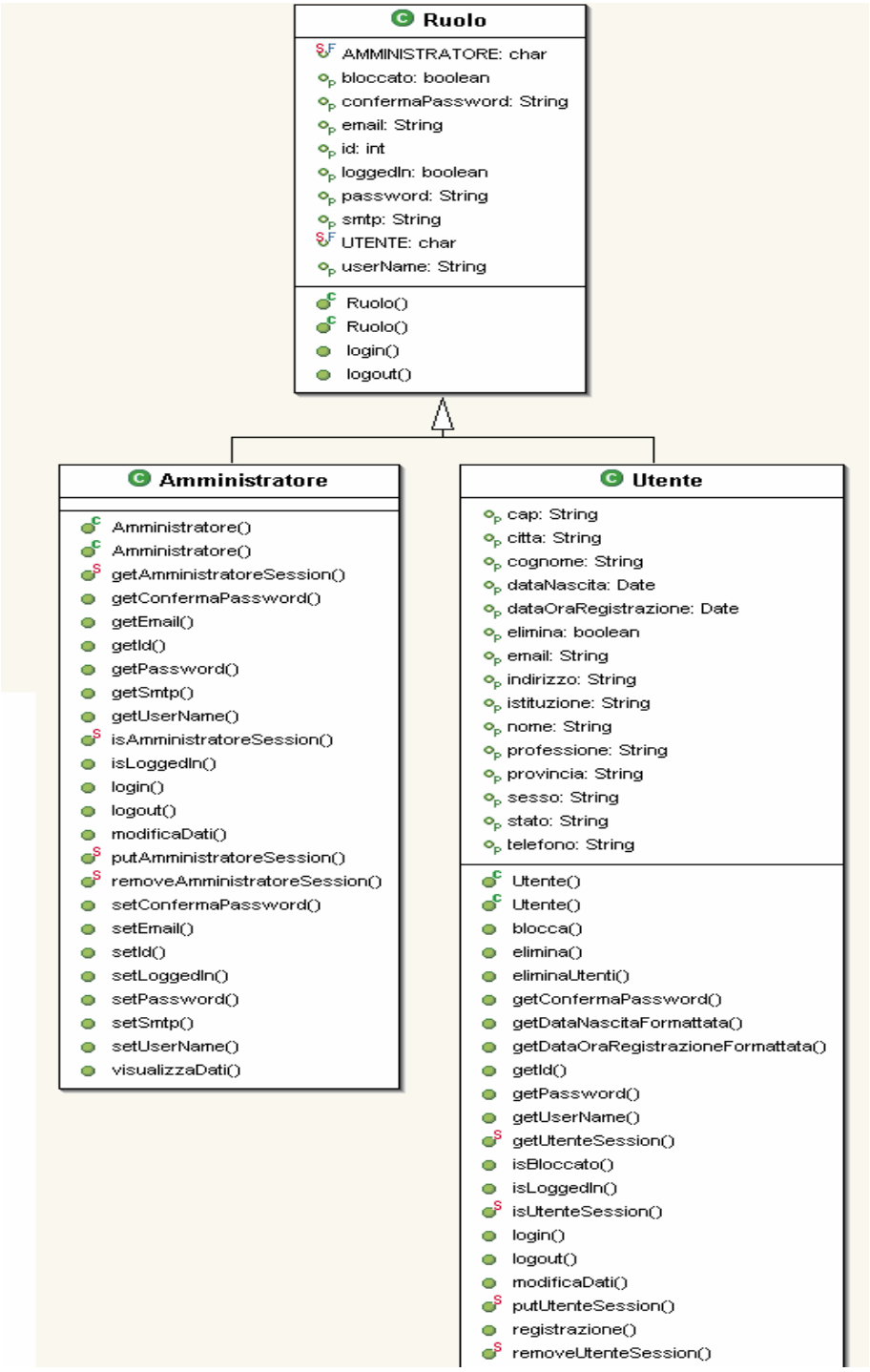


Figura 44 - package *ruolo*

Per quanto riguarda i metodi, la classe *Ruolo* prevede :

- *login()*, *logout()* : per effettuare le operazioni di login e di logout al sistema;
- *blocca()*, *sblocca()* : per bloccare o sbloccare un ruolo. Essi non hanno corpo in entrambe le implementazioni ma vengono utilizzate le versioni soggette ad overriding delle classi *Utente* ed *Amministratore*;

Il metodo *login()* opera secondo una modalità di questo tipo :

- tenta di valutare se al sistema sta accedendo un utente invocando *UtenteDB.login()*;
- se lo username non è registrato tra gli utenti, tenta di valutare se sta accedendo l'amministratore invocando *AmministratoreDB.login()*;
- nel caso di username inesistente in entrambi i casi viene segnalato un errore, mentre in caso di esito positivo e di password corretta viene consentito l'accesso. Se si tratta di un utente, vengono caricate le informazioni di un'eventuale scheda prepagata e ne viene valutata la scadenza;
- in tutti gli altri casi, le classi di accesso alla base di dati possono sollevare le eccezioni relative a password errata, ruolo loggato e ruolo bloccato;

Inoltre, il parametro di ritorno del metodo sarà una stringa che rappresenta l'*outcome* con il quale proseguire la navigazione, mentre per le eccezioni sollevate nelle possibili condizioni di errore vengono catturate e poi ne viene gestita la segnalazione. Per quanto riguarda la memorizzazione delle informazioni di accesso, questa viene eseguita all'interno della classe *Ruolo* attraverso dei metodi privati.

Il metodo *logout()* esegue semplicemente l'invalidazione della sessione mentre altre operazioni specifiche vengono eseguite dalla versione soggetta ad overriding delle classi *Utente* ed *Amministratore.*

La classe *Amministratore* prevede i seguenti metodi :

- *login(), logout()* : versioni sovrascritte delle medesime funzioni di *Ruolo*. La *login()* non fa altro che invocare la versione della superclasse mentre la *logout()*, aggiorna le informazioni di uscita dal sistema;
- *visualizzaDati()* : ha il compito di permettere la visualizzazione dei dati dell'amministratore per un'eventuale modifica, ossia ricava le informazioni dall'oggetto *Amministratore* in sessione e restituisce l'*outcome* per la navigazione alla pagina di visualizzazione;
- *modificaDati()* : esegue la modifica dei dati dell'amministratore, ossia esegue l'aggiornamento invocando *AmministratoreDB.aggiorna()* oppure segnala eventuali errori e fa proseguire la navigazione restituendo l'*outcome*;

La classe *Utente* è sicuramente una delle più complesse, in quanto prevede i seguenti metodi :

- *login(), logout()* : versioni sovrascritte delle medesime funzioni di *Ruolo*. La *login()* non fa altro che invocare la versione della superclasse mentre la *logout()*, in entrambe le implementazioni, aggiorna le informazioni di uscita dal sistema;
- *registrazione()* : permette di effettuare la registrazione dell'utente;
- *visualizzaDati()* : ha il compito di permettere la visualizzazione dei dati dell'utente, ossia ricava le informazioni dall'oggetto *Utente* in sessione e restituisce un *outcome* diverso per la navigazione, in base al fatto che la richiesta sia stata fatta dall'utente, che vorrà modificare i propri dati, oppure dall'amministratore che potrà soltanto visionarli;
- *modificaDati()* : esegue la modifica dei dati dell'utente, ossia esegue l'aggiornamento invocando *UtenteDB.aggiorna()* oppure segnala eventuali errori e fa proseguire la navigazione restituendo l'*outcome*.;
- *blocca(), sblocca()* : eseguono le operazioni di blocco e sblocco di un utente, mediante i corrispondenti metodi della classi che accedono alla base di dati;
- *sbloccaUtente()* : è utile per non alterare la struttura comune dei metodi precedenti e può essere invocato da una pagina JSP per sbloccare l'utente e restituire l'*outcome* per la navigazione;

- *visualizzaElencoUtenti()* : permette la visualizzazione dell'elenco degli utenti registrati, invoca *UtenteDB.elencoUtenti()* e salva la lista in una variabile di sessione restituendo l'*outcome* per la navigazione;
- *elimina(), eliminaUtenti()* : permettono di eliminare un singolo utente oppure più utenti selezionati dall'elenco, viene utilizzato uno o più volte il metodo *UtenteDB.elimina() e viene* gestito l'aggiornamento dell'elenco in sessione e restituito l'*outcome* per la navigazione;
- *verificaUsername()* : permette di valutare se un certo username risulta essere già associato ad un utente registrato. Esso viene utilizzato in fase di registrazione ed una volta terminata la verifica, restituisce un *outcome* per la navigazione;
- *richiediUsernamePassword()* : permette all'utente di ricevere una mail contenente i dati di accesso al sistema, prima verifica l'esistenza della mail utilizzando *UtenteDB.esisteEmail()*, e poi invia i dati mediante la classe *Mail* oppure segnala eventuali errori;

Il metodo *registrazione()* merita un'analisi a parte ha una struttura notevolmente complessa, poichè esegue tutte le operazioni legate alla registrazione che non prevede soltanto il salvataggio dell'utente in archivio. La funzionalità di registrazione è caratterizzata dalle seguenti fasi :

- si valuta se l'utente abbia inserito i dati necessari per la registrazione;
- viene bloccato il suo account ed inviata una mail all'amministratore indicando che un nuovo utente si è iscritto;
- una volta che l'amministratore avrà sbloccato l'account, viene inviata una email all'utente, avvisandolo che ora può effettuare l'accesso al sito;

6.4.3 Package alimenti

Il package alimenti contiene fondamentalmente le classi relative alla gestione degli alimenti presenti in archivio : Alimento, Categoria, CategoriaOriginaria, CategoriaStandard, CommentoStatistico, Componente, ComponenteStandard, DeterminazioneValore, Fonte, FonteSecondaria e TipoValore.

Per quanto riguarda le proprietà che le caratterizzano:

- *Alimento* : contiene le informazioni relative ad un generico alimento;
- *Categoria : contiene le informazioni per la scelta della categoria (tra categoriaStandard e categoriaOriginale) e della ricerca per categoria;*
- *CategoriaOriginaria : contiene le informazioni relative alla generica categoriaOriginaria;*
- *CategoriaStandard : contiene le informazioni relative alla generica categoriaStandard;*
- *DeterminazioneValore : contiene le informazioni relative alle determinazioni valore degli alimenti;*
- *Fonte : contiene le informazioni relative alle Fonti Primarie;*
- *FonteSecondaria : contiene le informazioni relative alle fonti Secondarie;*
- *TipoValore : contiene le informazioni relative ai tipi di valore degli alimenti;*

Dai diagrammi UML è possibile evidenziare le dipendenze che ci sono tra queste classi e le restanti classi del Model, con le quali interagiscono.

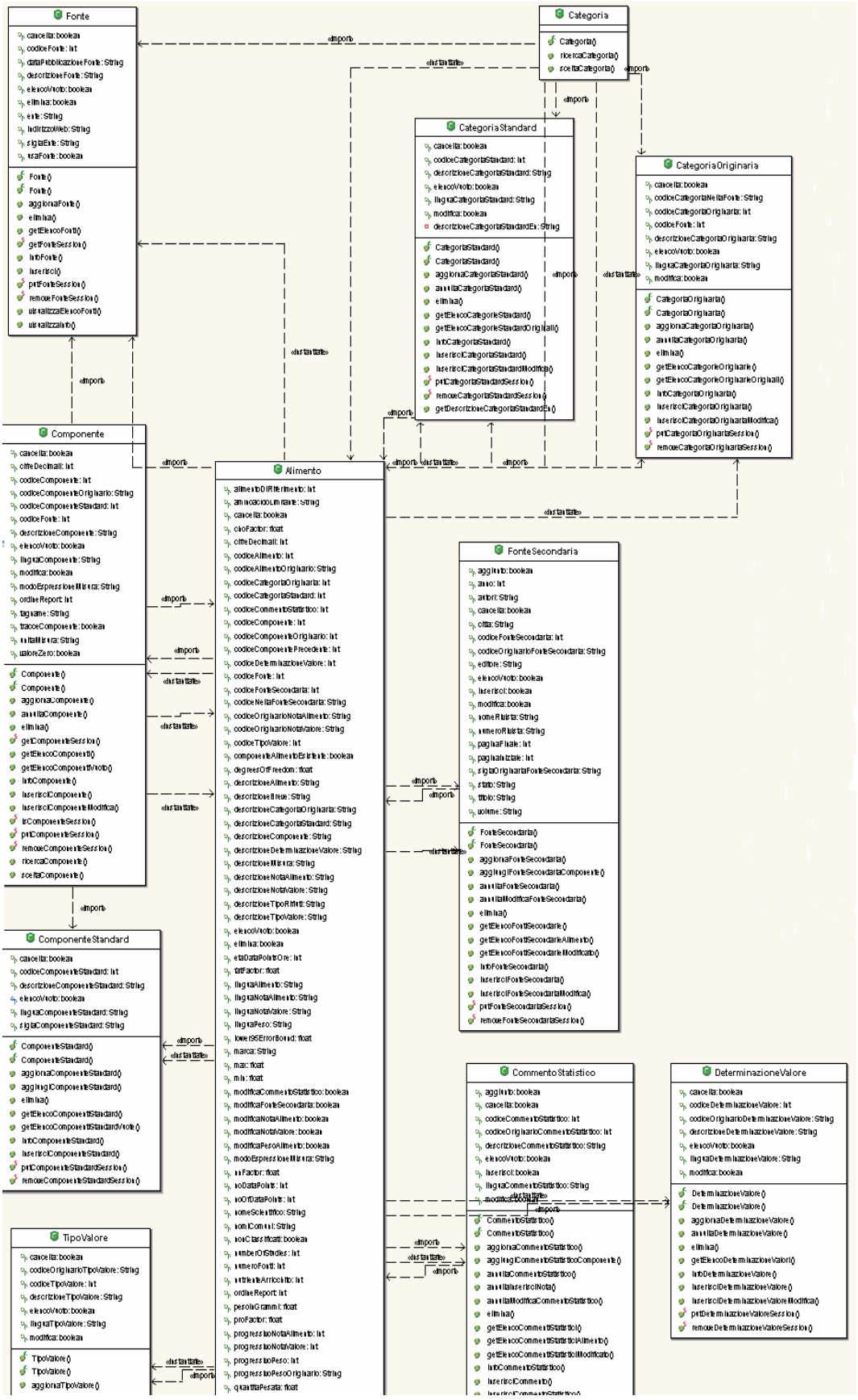


Figura 45 - package alimenti

Analizzando in linea generale i metodi della classe *Alimento*, essi svolgono le seguenti funzionalità :

- *visualizzaInfo()* : permette la visualizzazione delle informazioni di un alimento selezionato. L'alimento è automaticamente presente nella request e viene spostato in sessione, restituendo un *outcome* diverso per la navigazione in base al fatto che la richiesta sia stata fatta dall'amministratore, che potrà modificare le informazioni del brano, oppure dall'utente che potrà esclusivamente visionarle*;*
- *modificaInfo()* : permette la modifica delle informazioni di un alimento;
- *visualizzaElencoAlimenti(), ricerca()* : questi due metodi operano allo stesso modo, nel senso che permettono la visualizzazione di un elenco di alimenti sulla base di un criterio di ricerca, utilizzano il metodo *AlimentoDB.ricerca()* e caricano la lista ottenuta all'interno della sessione, restituendo poi l'*outcome* per la navigazione;
- *inserisci()* : permette l'inserimento di un nuovo alimento in archivio. Opera allo stesso modo del metodo *modifica()*, ossia viene invocato il metodo Alimento*DB.inserisci()* mentre in caso di esito negativo vengono segnalati gli errori. In entrambi i casi viene restituito l'*outcome* per proseguire la navigazione;
- *elimina()* : permette di eseguire l'eliminazione di un singolo alimento, utilizza il metodo Alimento*DB.elimina()*, per cancellare le informazioni del brano dall'archivio.

Tali metodi sono comuni a tutte le classi prima citate, quindi risulta inutile ripetere una descrizione per ogni classe.

6.4.8 Package mail

Il package *mail* contiene solo la classe *Mail*, la quale non ha alcuna proprietà e prevede soltanto il metodo *sendMessage()* mediante il quale è possibile inviare una mail. Esso utilizza la classe *Properties* per fissare nelle proprietà del sistema l'host che dovrà occuparsi dell'invio e la classe *Session* per stabilire una sessione con il mail server.

Genera un messaggio del tipo *MimeMessage* definendone il mittente, il destinatario, l'oggetto ed il corpo che riceve come parametri di ingresso. Infine, invia il messaggio invocando il metodo *Transport.send()*.

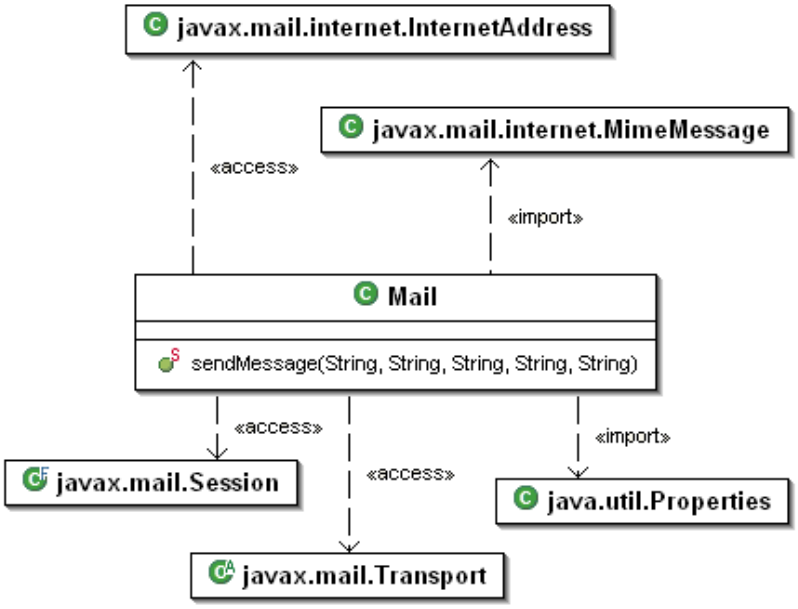


Figura 46 - package *mail*

6.5 I Backing Beans

Nell'implementazione realizzata tutte le classi che costituiscono il Model sono dichiarate all'interno del file di configurazione come backing beans (managed beans).
Questo tipo di approccio garantisce due vantaggi fondamentali:

- ciascun componente dell'interfaccia utente può essere associato ad una proprietà di un backing bean attraverso il value-binding, in modo tale che quando il form a cui appartiene il componente viene trasmesso, il valore che assume il componente viene trasferito nella corrispondente proprietà a cui è legato;
- utilizzando gli eventi , action e value-change, è possibile invocare direttamente da un componente un metodo di un backing bean, attraverso il method-binding, in modo da poter effettuare delle elaborazioni sui dati contenuti nel backing bean stesso;

JSF fa in modo che i valori dei campi dei form vengano copiati direttamente nelle corrispondenti proprietà di un backing bean.
Per quanto riguarda l'avvio delle elaborazioni da una pagina JSP, JSF permette di avviare direttamente da una pagina JSP un metodo di un backing bean, ossia di un oggetto del Model, nel quale ci saranno automaticamente i dati da elaborare. All'interno del file di configurazione, *faces-config.xml*, è possibile dichiarare opportunamente ciascun backing bean specificandone il tipo, ossia la classe che lo implementa, e l'ambito di visibilità (scope) per indicare se sia accessibile soltanto nell'ambito di una richiesta (*request*), di una sessione utente (*session*) oppure nel contesto della servlet (*application*). In maniera del tutto automatica è compito del framework gestire l'allocazione e la deallocazine dei backing bean quando necessario, grazie al meccanismo dell' *IoC (Inversion of Control)* noto anche come *DI (Dependency Injection)*.
Un semplice esempio è la seguente dichiarazione del backing bean relativo all'utente, al quale viene associato un nome, ne viene specificata la classe di appartenenza, *ruolo.Utente*, ed infine indicato l'ambito di visibilità, *session*, in modo che le informazioni dell'utente siano sempre accessibili durante la sua sessione di utilizzo dell'applicazione.

```
<managed-bean>
  <managed-bean-name>utente</managed-bean-name>
  <managed-bean-class>ruolo.Utente</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

8. Navigazione

Diamo uno sguardo ora alla navigazione all'interno dell'applicazione, questa si basa sul concetto delle *navigation-rules*, all'interno di ciascuna delle quali va definita la pagina di partenza della specifica "regola" e vanno definiti uno o più *navigation-case* che specificano, sulla base dell'*outcome* rinvenuto dal *NavigationHandler*, quale sia la pagina verso la quale proseguire la navigazione.

In base a quanto detto, ogni regola prevede :

- la pagina che rappresenta il punto di partenza di riferimento;
- uno o più pagine destinazione che è possibile raggiungere dalla suddetta pagina, ciascuna delle quali è distinta da un differente *outcome*;

Ciò vuol dire che, nel momento in cui viene invocato un metodo di un backing bean da una certa pagina JSP, quest'ultimo dovrà restituire come parametro di uscita una stringa che rappresenti l'*outcome* di navigazione. Sulla base di quest'ultimo, il *NavigationHandler* esegue una ricerca nel file di configurazione e determina verso quale pagina deve proseguire la navigazione.

9. Eccezioni

La gestione delle eccezioni dichiarative è una potenzialità che non è sfruttata in maniera ottimale da JSF, per ovviare a questo problema si è fatto ricorso alla semplice funzionalità di navigazione. Più precisamente, nel caso in cui si verifichi un errore di accesso alla base di dati, viene sollevata un'eccezione che, intercettata all'interno del codice stesso, fa in modo che il metodo del backing bean in questione restituisca l'*outcome* verso la pagina di errore.

10. Sicurezza

La Web application realizzata prevede una funzionalità particolare, nell'ambito della quale sono trasmessi dei dati fortemente sensibili, quali ad esempio i dati anagrafici di un utente registrato.. In una situazione di questo tipo, si rende necessario l'utilizzo di un meccanismo di sicurezza mediante il quale sia possibile crittografare i dati inviati attraverso la rete, in modo da renderli incomprensibili a chiunque riesca ad intercettarli in maniera fraudolenta.

Gli elementi principali che permettono un approccio di questo tipo sono fondamentalmente due :

- l'acquisizione di un certificato digitale;
- uso del protocollo HTTPS, ossia HTTP basato su SSL (Secure Socket Layer);

Attraverso un certificato digitale, rilasciato dalla CA (Certification Authority), è sempre possibile essere a conoscenza dell'identità dell'interlocutore e decidere se accettare o meno di stabilire una comunicazione con qeust'ultimo. Uno strumento di questo tipo risulta notevolmente utile, in virtù del fatto che ci si potrebbe ritrovare a scambiare informazioni con un'entità sulla rete che non è quella da noi attesa, ma che ne abbia preso il posto in maniera fraudolenta. Chiunque sia in possesso di un certificato digitale è stato rinosciuto da terze parti e quindi si ha la certezza che i dati trasmessi verranno acquisiti dal destinatario corretto. Una delle funzionalità che è possibile sfruttare mediante i certificati digitali è la crittografia, mediante la quale è possibile cifrare i dati trasmessi e renderli illeggibili durante il loro invio. Soltanto il destinatario, riconosciuto sulla base di un certificato digitale, avrà la possibilità di decrittografarli attraverso l'utilizzo di una chiave. La comunicazione avviene attraverso il protocollo HTTPS, ossia il tipico HTTP basato su SSL (Secure Socket Layer), che garantisce una cifratura con chiave a 128 bit.
Nel caso di studio in esame, per garantire la sicurezza, si sono rese necessarie le seguenti operazioni :

- creazione di un certificato digitale di esempio;
- abilitazione del *Connector SSL* nel Web Container Apache Tomcat;
- configurazione della pagine protette nel Deployment Descriptor dell'applicazione;

La creazione del certificato digitale è stata effettuta facendo uso di uno degli strumenti messi a disposizione dall'SDK di Java2, ossia *keytool*, che produce un cosiddetto *keystore*. Quest'ultimo permette inoltre di specificare il tipo di algoritmo da utilizzare per la crittografia, che nel caso specifico è l'RSA.

```
keytool -genkey -alias tomcat -keyalg RSA
```

Una volta creato il certificato, bisogna abilitare il Connector SSL del Web Container Tomcat, mediante il quale è possible redirezionare tutte le comunicazioni protette, verso una porta specificata e tramite protocollo HTTP. Per fare questo è necessario apportare una modifica al file di configurazione *server.xml*, all'interno del quale va specificato il *keystore* da utilizzare e la relativa password.

```
<Connector
    port="8443" maxThreads="150" minSpareThreads="25"
    maxSpareThreads="75" enableLookups="false"
    disableUploadTimeout="true" acceptCount="100" debug="0"
    scheme="https" secure="true" clientAuth="false"
    sslProtocol="TLS" keystorePass="changeit"
    keystoreFile="C:\.keystore"/>
```

Per quanto riguarda la modifica da applicare alla Web application, tenendo conto del fatto che JSF demanda la gestione della sicurezza al Web Container sottostante e non fornisce un proprio particolare strumento, all'interno del Deployment Descriptor *web.xml* è possibile specificare le pagine, la cui trasmissione deve essere effettuata su protocollo HTTPS. In questo caso si è deciso di rendere tutta l'applicazione sicura, garantendo la trasmissione tramite SSL per tutte le pagine a partire dalla pagina di ben venuto *index.jsp*.

```
<security-constraint>
 <display-name>SSL Constraint</display-name>
 <web-resource-collection>
  <web-resource-name>Automatic SLL Forwarding</web-resource-name>
  <url-pattern>/index.jsp</url-pattern>
  <http-method>GET</http-method>
  <http-method>PUT</http-method>
  <http-method>POST</http-method>
  <http-method>DELETE</http-method>
 </web-resource-collection>
 <user-data-constraint>
  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
 </user-data-constraint>
</security-constraint>
```

In questo modo, alla richiesta dell'URL iniziale viene segnalata la redirezione verso una connessione sicura garantita da un certificato digitale. La trasmissione dei dati tra client e server avverrà tramite il protocollo SSL in modalità crittografata.

Nella figura seguente, è visibile il messaggio del browser Web che segnala il passaggio ad una modalità di comunicazione protetta mediante protocollo SSL. E' possibile visionare le informazioni relative al certificato digitale, in modo da essere a conoscenza dell'interlocutore e decidere se accettare o meno la trasmissione dei dati.

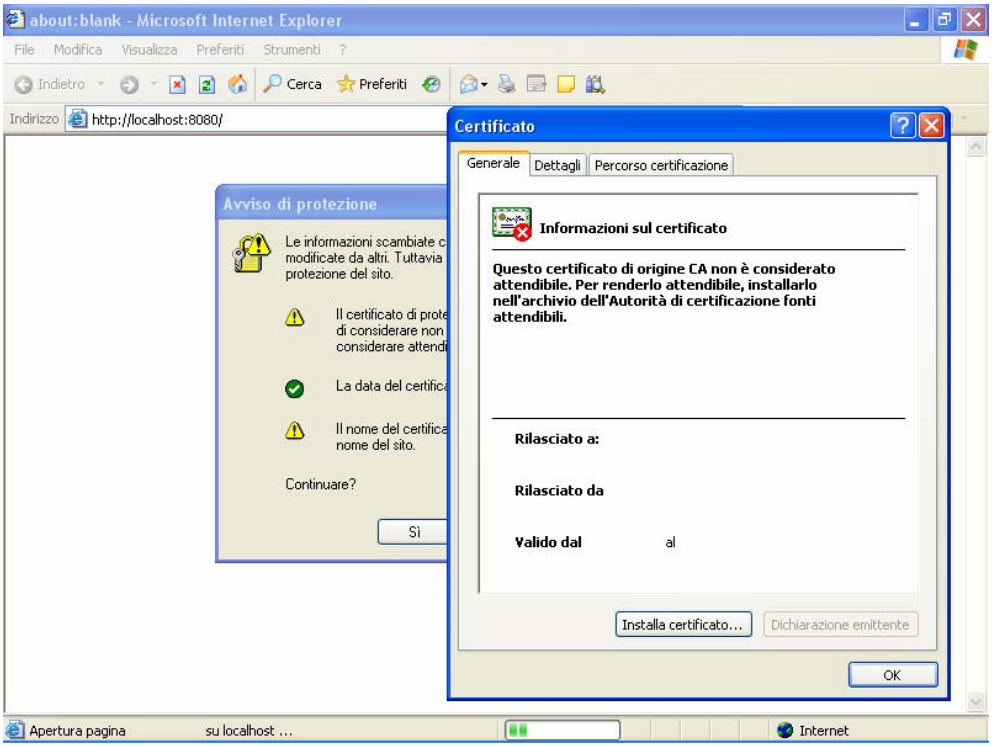


Figura 47 - Segnalazione del certificato digitale

Una volta accettato il passaggio al protocollo HTTPS, la navigazione all'interno della Web application prosegue normalmente, ma è visibile nella parte bassa del browser (es. Internet Explorer 6) un lucchetto che indica la garanzia di una comunicazione protetta e cifrata.

Appendice A – SRS Web Application Alimenti

1. Introduzione

1.1 Propositi

Questo documento ha lo scopo di definire i requisiti e le specifiche del prodotto software "*Alimenti*", al fine di facilitarne la realizzazione e la validazione. E' destinato sia al committente del prodotto software che allo sviluppatore, al fine di definire una base di riferimento per la validazione del prodotto e creare le premesse per la produzione di un software di alta qualità.

1.2 Obiettivi

Il prodotto software "*Alimenti*" ha come obiettivo la realizzazione di una Web application che permetta agli utilizzatori di usufruire del contenuto della Data WareHouse riguardante la composizione Chimico-Fisica degli alimenti. Il sistema permette l'automatizzazione delle seguenti operazioni, per quanto riguarda l'interazione con l'Utente :

- Registrazione al sito;
- Identificazione dell'Utente tramite lo Username e la Password;
- Accesso all'archivio degli alimenti;
- Ricerche avanzate nell'archivio degli alimenti;
- Visualizzazione delle informazioni di un singolo aliment;
- Gestione dei propri dati personali;
- Richiesta dei propri dati di accesso via mail;
- Uscita dal sistema;

L'Amministratore disporrà di funzionalità avanzate, mediante le quali saranno consentite le seguenti operazioni :

- Identificazione dell'Amministratore tramite lo Username e la Password;
- Gestione dell'archivio degli alimenti per l'accesso, la ricerca, l'inserimento, la modifica e la visualizzazione;
- Gestione degli utenti registrati;
- Gestione dei propri dati di accesso;
- Uscita dal sistema;

1.3 Definizioni, Acronimi ed Abbreviazioni

Utente : persona che ha effettuato la registrazione e che risulta abilitata all'utilizzo dei servizi resi disponibili dalla scelta effettuata;

Amministratore : persona addetta alla gestione degli archivi degli utenti e alla gestione dei servizi offerti dal sistema;

Utilizzatore – Ruolo : persona non ancora identificata tra Utente ed Amministratore, che non ha effettuato l'accesso al sistema;

Servizio : ciascuna delle attività accessibili e consentite ad un qualunque tipo di utente o all'amministratore;

Registrazione : contratto stipulato dall'Utente, necessario per poter usufruire dei servizi offerti dal sistema. Che permette di accedere a tutti i servizi

Username : identificativo univoco dell'Utente oppure dell'Amministratore;

Password : password di accesso al sistema dell'Utente oppure dell'Amministratore;

1.4 Riferimenti

Standard IEEE 830-1993.

1.5 Generalità

L'intento di questo documento è quello di descrivere le funzionalità che il software deve soddisfare, le quali saranno specificate nel capitolo successivo in modo chiaro e conciso.

Il resto del documento contiene l'elenco delle funzioni che il prodotto software deve avere insieme ai vincoli che deve soddisfare. Più avanti nel testo sono presentate l'interfaccia (sia verso l'Utente sia verso l'Amministratore) dei servizi messi a disposizione dal prodotto e l'interfaccia verso l'hardware. Al presente documento ne sono allegati altri contenenti i diagrammi realizzati secondo lo standard UML, necessari per la miglior comprensione del dominio applicativo.

2. Descrizione Generale

2.1 Prospettive del prodotto

Questo prodotto non si integra con nessun altro sistema software ed è indipendente ed autonomo per quanto riguarda l'elaborazione(stand-alone).

2.2 Funzionalità

Il prodotto software "*Alimenti*" dovrà :

nei confronti dell'Amministratore fornire le seguenti funzionalità :

- accesso al sistema;
- registrazione dell'accesso;
- modifica della propria Password di accesso;
- gestione dell'archivio degli alimenti, che prevede :
  - visualizzazione degli alimenti in base a criteri di ricerca specifici;
  - inserimento di un nuovo alimento in archivio;
  - modifica delle informazioni di un alimento;
  - eliminazione di un alimento dall'archivio;
  - visualizzazione delle informazioni di un alimento;
- gestione degli utenti registrati, che prevede :
  - eliminazione di un utente;
  - sblocco di un utente bloccato;
- uscita dal sistema;
- registrazione dell'uscita;

nei confronti dell'Utente fornire le seguenti funzionalità :

- registrazione;
- richiesta via mail dei dati di accesso;
- accesso al sistema;
- registrazione dell'accesso;
- modifica dei propri dati personali;
- accesso all'archivio degli alimenti, che prevede :
  - visualizzazione degli alimenti in base a criteri di ricerca specifici;
  - visualizzazione delle informazioni di un alimento;
- uscita dal sistema;
- registrazione dell'uscita;

2.3 Caratteristiche Utente

Il prodotto software è destinato ad utenti che abbiano una conoscenza di base nell'utilizzo di Personal Computer e relativi software per la produttività personale; mentre gli amministratori, che usano il software di gestione, hanno bisogno di una conoscenza informatica di base.

2.4 Vincoli Generali

I vincoli generali sono i seguenti :

- L'Utente che richiede l'accesso ai servizi del sistema deve digitare esclusivamente il proprio Username e la relativa Password;
- L'Utente appena registrato, non può ancora accedere al sistema, finchè il suo accesso non verrà sbloccato dall'amministratore;
- L'Amministratore che richiede l'accesso ai servizi del sistema deve digitare esclusivamente il proprio Username e la relativa Password;
- La Password di accesso dell'Utente o dell'Amministratore può essere composta da almeno 5 caratteri alfanumerici;

2.5 Assunzioni e Dipendenze

Il Sistema software dovrà essere utilizzato su una macchina dotata di sistema operativo Windows 95/98/Me/2000/Xp oppure di una distribuzione Linux, con 128 Mb di memoria RAM, Hard Disk da 10 GB ed una connessione Internet consigliata ADSL.

3. Requisiti Specifici

3.1 Requisiti di interfaccia esterna

3.1.1 Interfaccia Utente

E' richiesta un'interfaccia utente visuale, con finestre, bottoni e form di immissione dati.

3.1.2 Interfaccia Hardware

Il sistema software non si interfaccia con alcun particolare sistema hardware.

3.1.3 Interfaccia Software

Il sistema software non si integra e non comunica con nessun altro sistema software.

3.1.4 Interfaccia di Comunicazione

La comunicazione fra il sistema software "lato client" e quello "lato server", si svolge utilizzando la rete Internet e si basa fondamentalmente sull'accesso di un database condiviso localizzato sul server stesso.

3.2 Requisiti Funzionali

Classe Amministratore

3.2.1 Login

3.2.1.1 Introduzione

Questa funzionalità permette all'Amministratore di accedere al sistema.

3.2.1.2 Input

- Username (obbligatorio)
- Password (obbligatorio)

3.2.1.3 Elaborazione

Visualizzazione di un form per l'input dei dati, sui quali il sistema esegue le seguenti operazioni :

- Controlla che la Username e la Password siano corrette e cioè corrispondano a quelle memorizzate nell'archivio. Se il controllo va a buon fine, viene permesso l'accesso, altrimenti viene concesso un altro;
- Controlla che l'Amministratore non sia già loggato;
- Registrazione dei dati relativi all'accesso;
- Verifica dei diritti di accesso;
- Visualizzazione del menù per accedere alle aree di gestione;

3.2.1.4 Output

Menu di accesso alle aree di gestione.

Dati relativi all'accesso registrati in archivio :

- Data ed Ora di login
- Indirizzo IP
- Validità login
- Tentativo

Messaggi :

"Username e/o Password errati"

"Già loggato"

3.2.2 Modifica Password Amministratore

3.2.2.1 Introduzione

Questa funzionalità permette all'Amministratore di modificare i propri dati di accesso al sistema, nella fattispecie soltanto la password. E' ovviamente previsto che l'Amministratore abbia già eseguito il login, altrimenti la funzionalità non è disponibile.

3.2.2.2 Input

- Identificativo Amministratore (obbligatorio)
- Password nuova (obbligatorio)
- Conferma Password nuova (obbligatorio)

3.2.2.3 Elaborazione

Visualizzazione di un form per l'input dei dati, sui quali il sistema esegue le seguenti operazioni :

- Controlla che la Password nuova e Conferma Password siano almeno di 5 caratteri e visualizza eventualmente un messaggio di errore;
- Controlla che la Password nuova e la Conferma Password coincidano, altrimenti visualizza un messaggio di errore;
- Registrazione dei nuovi dati inseriti;

3.2.2.4 Output

Parte dei dati letti in input che vengono registrati in archivio :

- Password nuova;

Messaggi :

"La Password nuova e la sua Conferma non coincidono"

"La Password deve avere una lunghezza di almeno 5 caratteri"

"La Conferma Password deve avere una lunghezza di almeno 5 caratteri"

"Aggiornamento effettuato con successo"

3.2.3 Ricerca di un alimento secondo uno o più criteri

3.2.3.1 Introduzione

Questa funzionalità permette all'Amministratore di cercare uno o più alimenti all'interno dell'archivio, sulla base di propri criteri, per poter effettuare su di essi delle eventuali operazioni.

3.2.3.2 Input

- Fonti Selezionate
- Lingua
- Tipo di Ricerca
- A seconda del tipo di ricerca dovrà essere indicata : la Descrizione dell'Alimento nel caso di ricerca per Alimento, il nome del Componente nel caso di ricerca per Componente, il nome della categoria nel caso di ricerca per Categoria;

3.2.3.3 Elaborazione

Visualizzazione di un form che permette all'Amministratore di stabilire i propri criteri di ricerca degli alimenti. Inoltre, il sistema effettua le seguenti operazioni :

- Controlla che i campi su cui effettuare la ricerca non siano vuoti;
- Caricamento dei dati presenti in archivio;
- Visualizzazione dell'elenco degli alimenti;

3.2.3.4 Output

Elenco degli alimenti che corrispondono ai criteri impostati.

Messaggi :

"Nessun Alimento Trovato"

"Nessuna Fonte Selezionata"

3.2.4 Inserimento di un alimento

3.2.4.1 Introduzione

Questa funzionalità permette all'Amministratore di inserire un nuovo alimento in archivio. Le informazioni relative all'alimento devono essere specificate dall'Amministratore.

3.2.4.2 Input

- Descrizione Alimento (obbligatorio)
- Fonte (obbligatorio)
- Codice Alimento nella Fonte (obbligatorio)
- Nome Scientifico
- Nomi Comuni
- Marca
- Aminoacido Limitante
- Fattore per la Conversione dell'Azoto in Proteine
- Fattore per il Calcolo delle Calorie da Proteine
- Fattore per il Calcolo delle Calorie da Grassi
- Fattore per il Calcolo delle Calorie da Carboidrati
- Descrizione dei rifiuti
- Descrizione Breve
- Lingua Alimento (obbligatorio)
- Note Alimento
- Peso Alimento
- Categoria Originaria (obbligatorio)
- Categoria Standard (obbligatorio)
- Componenti Alimento

3.2.4.3 Elaborazione

Visualizzazione di un form per l'input dei dati sui quali il sistema esegue le seguenti operazioni :

- Verifica delle informazioni introdotte dall'Amministratore;
- Registrazione dei dati di input in archivio;

3.2.4.4 Output

Tutti i dati letti in input registrati in archivio.

Messaggi :

"Inserimento alimento effettuato con successo"

"Il tag relativo alla descrizione dell'alimento è vuoto"

"Il tag relativo alla fonte è vuota"

"Il tag relativo alla Lingua dell'Alimento è vuoto"

"Il tag relativo alla categoria Originaria è vuoto"

"Il tag relativo alla categoria Standard è vuoto"

3.2.5 Visualizzazione informazioni di un alimento

3.2.5.1 Introduzione

Questa funzionalità permette all'Amministratore di visualizzare le informazioni dettagliate di un alimento presente in archivio.

3.2.5.2 Input

- Identificativo alimento (obbligatorio)

Dati caricati dal sistema provenienti dall'archivio.

3.2.5.3 Elaborazione

- Caricamento dei dati presenti in archivio;
- Visualizzazione delle informazioni dettagliate dell'alimento selezionato;

3.2.5.4 Output

Informazioni dettagliate dell'alimento selezionato.

3.2.6 Modifica informazioni di un alimento

3.2.6.1 Introduzione

Questa funzionalità permette all'Amministratore di modificare le informazioni di un alimento presente in archivio.

3.2.6.2 Input

- Descrizione Alimento (obbligatorio)
- Fonte (obbligatorio)
- Codice Alimento nella Fonte (obbligatorio)
- Nome Scientifico
- Nomi Comuni
- Marca
- Aminoacido Limitante
- Fattore per la Conversione dell'Azoto in Proteine
- Fattore per il Calcolo delle Calorie da Proteine
- Fattore per il Calcolo delle Calorie da Grassi
- Fattore per il Calcolo delle Calorie da Carboidrati
- Descrizione dei rifiuti
- Descrizione Breve
- Lingua Alimento (obbligatorio)
- Note Alimento
- Peso Alimento
- Categoria Originaria (obbligatorio)
- Categoria Standard (obbligatorio)
- Componenti Alimento

3.2.6.3 Elaborazione

Visualizzazione di un form, contenente i dati attuali relativi all'alimento, caricati dall'archivio, per l'input dei nuovi dati sui quali il sistema esegue le seguenti operazioni:

- Controlla che i campi relativi alla Descrizione dell'Alimento, della Fonte, del codice originario nella Fonte, della Lingua dell'Alimento, della Categoria Standard e della Categoria Originaria non siano vuoti;
- Registrazione dei dati di input in archivio;

3.2.6.4 Output

Tutti i dati letti in input registrati in archivio.

Messaggi :

"Aggiornamento effettuato con successo"

3.2.7 Cancellazione di un alimento

3.2.7.1 Introduzione

Questa funzionalità permette all'Amministratore di cancellare uno o più alimenti presenti in archivio.

3.2.7.2 Input

- Identificativo alimento (obbligatorio)

3.2.7.3 Elaborazione

- Visualizzazione di un form contenente le informazioni di un alimento.

3.2.7.4 Output

Elenco degli alimenti.

Messaggi :

"Eliminazione effettuata con successo"

3.2.8 Visualizzazione di un alimento

3.2.8.1 Introduzione

Questa funzionalità permette all'Amministratore di visionare le informazioni di un singolo alimento.

3.2.8.2 Input

- Identificativo alimento (obbligatorio)

3.2.8.3 Elaborazione

- Visualizzazione di un form per la visione di tutte le informazioni di un singolo alimento

3.2.8.4 Output

Informazioni alimento.

Messaggi :

"Nessun alimento è stato selezionato"

3.2.9 Visualizzazione elenco utenti

3.2.9.1 Introduzione

Questa funzionalità permette all'Amministratore di visualizzare l'elenco degli utenti iscritti.

3.2.9.2 Input

Dati caricati dal sistema provenienti dall'archivio.

3.2.9.3 Elaborazione

- Caricamento dei dati presenti in archivio;
- Visualizzazione dell'elenco degli utenti;

3.2.9.4 Output

Elenco degli utenti.

Messaggi :

"Nessun utente registrato"

3.2.10 Cancellazione di un utente

3.2.10.1 Introduzione

Questa funzionalità permette all'Amministratore di cancellare uno o più utenti presenti in archivio.

3.2.10.2 Input

- Identificativo Utente (obbligatorio)

3.2.10.3 Elaborazione

Visualizzazione di un form, contenente l'elenco degli utenti, con possibilità di selezionare gli utenti da cancellare. E' possibile cancellare un utente anche dalla funzionalità di visualizzazione dei suoi dati anagrafici. Inoltre, il sistema effettua le seguenti operazioni :

- Controlla che sia stato selezionato almeno un utente;
- Eliminazione dei dati presenti in archivio relativi all'/gli utente/i selezionato/i;
- Visualizzazione dell'elenco degli utenti;

3.2.10.4 Output

Elenco degli utenti.

Messaggi :

"Eliminazione effettuata con successo"

"Nessun utente è stato selezionato"

3.2.11 Visualizzazione dati anagrafici utente

3.2.11.1 Introduzione

Questa funzionalità permette all'Amministratore di visualizzare i dati anagrafici di un utente presente in archivio.

3.2.11.2 Input

- Identificativo Utente (obbligatorio)

Dati caricati dal sistema provenienti dall'archivio.

3.2.11.3 Elaborazione

- Caricamento dei dati presenti in archivio;
- Visualizzazione dei dati anagrafici dell'utente selezionato;

3.2.11.4 Output

Dati anagrafici dell'utente selezionato.

3.2.12 Sblocco di un utente

3.2.12.1 Introduzione

Questa funzionalità permette all'Amministratore di sbloccare un Utente, il cui account è in attesa di approvazione.

3.2.12.2 Input

- Identificativo Utente (obbligatorio)

3.2.12.3 Elaborazione

- Registrazione dei dati in archivio, ossia del sblocco dell'Utente.

3.2.12.4 Output

Sblocco Utente.

3.2.13 Logout

3.2.13.1 Introduzione

Questa funzionalità permette all'Amministratore di eseguire il logout dal sistema.

3.2.13.2 Input

- Identificativo Amministratore (obbligatorio)

3.2.13.3 Elaborazione

- Registrazione dei dati in archivio che tengono traccia del logout;

3.2.13.4 Output

Dati relativi all'uscita dal sistema registrati in archivio :

- Data ed Ora di logout

Classe Utente

3.2.14 Login

3.2.14.1 Introduzione

Questa funzionalità permette all'Utente di accedere al sistema.

3.2.14.2 Input

- Username (obbligatorio)
- Password (obbligatorio)

3.2.14.3 Elaborazione

Visualizzazione di un form per l'input dei dati, sui quali il sistema esegue le seguenti operazioni :

- Controlla che la Username e la Password siano corrette e cioè corrispondano a quelle memorizzate nell'archivio. Se il controllo va a buon fine, viene permesso l'accesso;
- Controlla che l'Utente non sia già loggato;
- Registrazione dei dati relativi all'accesso;
- Verifica dei diritti di accesso;
- Visualizzazione del menù per l'accesso alle aree di utente;

3.2.14.4 Output

Menù di accesso alle aree di utente.

Dati relativi all'accesso registrati in archivio :

- Data ed Ora di login
- Indirizzo IP
- Validità login
- Tentativo

Messaggi :

"Username e/o Password errati"

"Già loggato"

3.2.15 Registrazione

3.2.15.1 Introduzione

Questa funzionalità permette all'Utente di effettuare la registrazione, per usufruire dei servizi offerti dalla web application.

3.2.15.2 Input

Dati dell'Utente :

- Email (obbligatorio)
- Username (obbligatorio)
- Password (obbligatorio)
- Conferma Password (obbligatorio)
- Nome (obbligatorio)
- Cognome (obbligatorio)
- Indirizzo (obbligatorio)
- Professione (obbligatorio)
- Istituzione (obbligatorio)
- CAP
- Stato (obbligatorio)
- Provincia
- Città (obbligatorio)
- Telefono
- Data di Nascita
- Sesso

3.2.15.3 Elaborazione

Visualizzazione di un form per l'input dei dati, sui quali il sistema esegue le seguenti operazioni :

- Controlla che i campi relativi all'Email, Username, Password, Conferma Password, Nome, Cognome, Indirizzo, Professione, Istituzione, Stato e Città non siano vuoti o non validi, in particolare l'Email abbia una struttura corretta e la Password abbia una lunghezza di almeno 5 caratteri;
- Controlla che lo Username non sia già presente in archivio;
- Controlla che la Password e la Conferma Password siano uguali;
- Registrazione dei dati di input in archivio;

3.2.15.4 Output

Tutti i dati letti in input registrati in archivio.

Menù di accesso alle aree di utente.

Altri dati registrati :

- Data/Ora registrazione

Messaggi :

"Registrazione effettuata con successo"

"Email non valida"

"Il campo relativo all'email non può essere vuoto"

"Il campo relativo alla Username non può essere vuoto"

"Il campo relativo alla Password non può essere vuoto"

"La Password deve essere almeno di 5 caratteri"

"Password e Conferma Password non coincidono"

3.2.16 Modifica dati personali

3.2.16.1 Introduzione

Questa funzionalità permette all'Utente di modificare i propri dati personali.

3.2.16.2 Input

Dati dell'Utente :

- Nome (obbligatorio)
- Cognome (obbligatorio)
- Indirizzo (obbligatorio)
- CAP
- Stato (obbligatorio)
- Provincia
- Città (obbligatorio)
- Istituzione (obbligatorio)
- Professione (obbligatorio)
- Telefono
- Data di Nascita
- Sesso
- Email (obbligatorio)
- Username (obbligatorio)
- Password nuova (obbligatorio)

- Conferma Password nuova (obbligatorio)

3.2.16.3 Elaborazione

Visualizzazione di un form, contenente i dati attuali relativi all'Utente caricati dall'archivio, per l'input dei nuovi dati sui quali il sistema esegue le seguenti operazioni

- Controlla che i campi relativi all'Email, Username, Password vecchia e nuova e Conferma Password non siano vuoti o non validi, in particolare l'Email abbia una struttura corretta e la Password abbia una lunghezza di almeno 5 caratteri;
- Controlla che i campi relativi al Nome, Cognome, Indirizzo, Città, Professione, Istituzione e Stato non siano vuoti;
- Controlla che la Password nuova e la Conferma Password coincidano, altrimenti visualizza un messaggio di errore;
- Registrazione dei dati di input in archivio;

3.2.16.4 Output

Tutti i dati letti in input registrati in archivio.

Menù di accesso alle aree di utente.

Messaggi :

"Aggiornamento effettuato con successo"

"Email non valido"

"Il campo relativo all'email non può essere vuoto"

"Il campo relativo alla Username non può essere vuoto"

"Il campo relativo alla Password nuova non può essere vuoto"

"Il campo relativo alla Conferma Password non può essere vuoto"

"La Password deve essere di almeno 5 caratteri"

"la Password ed il relativo campo di Conferma non coincidono"

"Il campo relativo al nome non può essere vuoto"

"Il campo relativo al cognome non può essere vuoto"

"Il campo relativo all'indirizzo non può essere vuoto"

"Il campo relativo alla città non può essere vuoto"

"Il campo relativo allo stato non può essere vuoto"

3.2.17 Ricerca di un alimento secondo uno o più criteri

3.2.17.1 Introduzione

Questa funzionalità permette all'Utente di cercare uno o più alimenti all'interno dell'archivio, sulla base di propri criteri, per accedere alle informazioni di un singolo alimento.

3.2.17.2 Input

- Fonti Selezionate
- Lingua
- Tipo di Ricerca
- A seconda del tipo di ricerca dovrà essere indicata : la Descrizione dell'Alimento nel caso di ricerca per Alimento, il nome del Componente nel caso di ricerca per Componente, il nome della categoria nel caso di ricerca per Categoria;

3.2.17.3 Elaborazione

Visualizzazione di un form che permette all'Utente di stabilire i propri criteri di ricerca degli alimenti. Inoltre, il sistema effettua le seguenti operazioni :

- Controlla che i campi su cui effettuare la ricerca non siano vuoti;
- Caricamento dei dati presenti in archivio;
- Visualizzazione dell'elenco degli alimenti;

3.2.17.4 Output

Elenco degli alimenti che corrispondono ai criteri impostati.

Messaggi :

"Nessun Alimento Trovato"

"Nessuna Fonte Selezionata"

3.2.18 Visualizzazione informazioni di un alimento

3.2.18.1 Introduzione

Questa funzionalità permette all'Utente di visualizzare le informazioni dettagliate di un alimento presente in archivio.

3.2.18.2 Input

- Identificativo alimento (obbligatorio)

Dati caricati dal sistema provenienti dall'archivio.

3.2.18.3 Elaborazione

- Caricamento dei dati presenti in archivio;
- Visualizzazione delle informazioni dettagliate dell'alimento selezionato;

3.2.18.4 Output

Informazioni dettagliate dell'alimento selezionato.

3.2.19 Richiesta username/password

3.2.19.1 Introduzione

Questa funzionalità permette all'Utente di chiedere l'invio di una mail contenente la username e la password, qualora se ne fosse dimenticato.

3.2.19.2 Input

- Email (obbligatorio)

3.2.19.3 Elaborazione

Visualizzazione di un form per l'inserimento dei dati, sui quali il sistema effettua le seguenti operazioni :

- Controlla che il campo relativo alla Email non sia vuoto e sia valido;
- Caricamento dei dati presenti in archivio;
- Invio della mail contenente username e password;

3.2.19.4 Output

Messaggi :

"Mail inviata con successo"

"Il campo relativo alla Email non può essere vuoto"

"L'Email ha un formato non valido"

"Email non registrata in archivio"

3.2.20 Logout

3.2.20.1 Introduzione

Questa funzionalità permette all'Utente di eseguire il logout dal sistema.

3.2.20.2 Input

- Identificativo utente (obbligatorio)

3.2.20.3 Elaborazione

- Registrazione dei dati in archivio che tengono traccia del logout;

3.2.20.4 Output

Dati relativi all'uscita dal sistema registrati in archivio :

- Data ed Ora di uscita

3.3 Requisiti di prestazioni

Nessuno.

3.4 Vincoli di progetto

Nessuno.

3.5 Attributi

3.5.1 Sicurezza

Come strumento di protezione è previsto l'utilizzo di una Password al fine di evitare accessi al sistema da parte di persone non autorizzate.

3.6 Altri requisiti

3.6.1 Database

Il sistema software prevede l'utilizzo di un database relazionale per l'archiviazione dei dati.

Riferimenti Bibliografici

Capitolo I

N. Ford - "*Art of Java Web Development*" - Ed. Manning

K. D. Mann - "*JavaServer Faces in Action*" - Ed. Manning

M. Pianciamore - "*Principi di Ingegneria del Software – Design Pattern*" - CEFRIEL

G. Mecca - "*Applicazioni Web J2EE : Java Servlet*", corso di Tecnologie di sviluppo per il Web - Università della Basilicata

J. Falkner, **K. Jones** - "*Servlets and JavaServer Pages : The J2EE technology Web tier*" - Ed. Addison Wesley

Capitolo II

E. Armstrong, **J. Ball**, **S. Bodoff**, **D.B. Carson**, **I. Evans**, **D. Green**, **K. Haase**, **E. Jendrock** - "*The J2EE 1.4 Tutorial*" (cap. 17,18,19,20,21) - Sun Microsystems

K. D. Mann - "*JavaServer Faces in Action*" - Ed. Manning

Doris Chen Ph.D. - "*JavaServer Faces and Sun Java Studio Creator : Experience the Next Generation Web-Application Framework*", Sun Tech Days 2005 - Sun Microsystems

D. Goyal, **V. Varma** - "*Introduction to JavaServer Faces (JSF)*" - Sun Microsystems

E. Burns - *"About Faces : The JavaServer Faces API and how it relates to Struts"* - Sun Microsystems

S. Shin - *"J2EE Advanced"* (www.javapassion.com) - Sun Microsystems

M. Hall - Tutorial *"JavaServer Faces 1.1"* (www.coreservlets.com)

A. Winder, **C. Straub** - *"Extreme Reuse in JavaServer Faces Technology"* , JavaOne - Sun's 2005 Worldwide Java Developer Conference - Oracle

J. Pardi – *"JavaServer Faces"*, 25 Maggio 2005

Capitolo IV

R.S. Pressman - *"Principi di Ingegneria del Software"* - Ed. McGraw/Hill

P. Atzeni, **S. Ceri**, **S. Paraboschi**, **R. Torlone** - *"Basi di dati"*, seconda edizione - Ed. McGraw/Hill

G. A. Di Lucca - *"Corso di Ingegneria del Software"*, slide a.a. 2003-2004 - Università degli Studi del Sannio

Sybase - *"Sybase Power Designer : Conceptual Data Model , User's Guide"*

Sybase - *"Sybase Power Designer : Object Oriented Model , User's Guide"*

Sybase - *"Sybase Power Designer : Physical Data Model , User's Guide"*

Capitolo V

B. Dudney - *"Creating JSF Custom Components"* (www.java.net)

R. Hightower - *"JSF for nonbelievers : JSF component development"* (www.ibm.com)

Apache Software Foundation - *"Apache Jakarta Tomcat 5 official documentation : SSL Configuration How-to"*

Appendice A – SRS Web Application Alimenti

Software Engineering Standards Committee of the IEEE Computer Society – *"IEEE Recommended Practice for Software Requirements Specifications"*, IEEE Std 830-1993