POLITECNICO DI MILANO

Corso di Laurea Specialistica in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

PROGETTAZIONE E SVILUPPO DI UNA SOLUZIONE WEB PER GESTIRE LA MODULISTICA AZIENDALE

Relatore: Prof.ssa Chiara FRANCALANCI
Correlatore: Dott.re Maurizio BARTOLI

Tesi di Laurea di:
Matteo STEFANI, matricola 682220

ANNO ACCADEMICO 2011 – 2012

Alla mia famiglia

Indice

Elenco delle figure

Capitolo 1

Introduzione

Nell'ultimo decennio la diffusione del Web ha registrato una crescita esponenziale in termini di utenza e di risorse disponibili. Tutto ciò ha cambiato radicalmente e semplificato le regole d'accesso a qualsiasi tipo d'informazione.
In questo scenario le aziende italiane hanno colto sempre più l'importanza del Web, considerandolo lo strumento di comunicazione principale e ponendolo al centro dei rapporti con i propri dipendenti.
E' quindi fondamentale gestire in maniera completa e ottimizzata e con opportuni strumenti informatici, il ciclo di acquisizione, elaborazione e ottimizzazione del flusso dati relativo alla singola persona, oltre a rendere ancora più efficace e produttiva l'amministrazione del personale.

Questa tesi descrive la realizzazione di un'applicazione Web per lo sviluppo, compilazione e gestione di interfacce per la gestione di dati ed informazioni

legate a processi gestionali complessi. Nel dettaglio si illustra la sua declinazione operativa in ambito aziendale, permettendo così di migliorare il processo di comunicazione tra dipendente e azienda, rendendo veloci e automatiche le operazioni di raccolta dati, controllo e aggiornamento sistemi aziendali, che ancora oggi sono svolte manualmente o con l'ausilio di sitemi di office automation.

La fase iniziale del lavoro ha comportato l'analisi delle esigenze, identificando il target d'utenza e una macro definizione funzionale. Nella fase successiva è stata fatta un'analisi atta ad individuare l'esistenza di prodotti esistenti o framework che potessero garantire una copertura funzionale sufficente.

Si è quindi deciso di procedere con lo sviluppo di un prodotto custom sviluppato sfruttando le potenzialità e le caratteristiche del frame work **JSF** (Java Server Faces [1]) basato sul design pattern architetturale Model-View-Controller cui è stato aggiunto l'utilizzo di **Hibernate** [2] che è uno strato di middleware che consente di automatizzare le procedure per le operazioni cosiddette CRUD (Create, Read, Update, Delete) dei database.

Il successivo passo del progetto è stato lo sviluppo della struttura delle pagine Web, che doveva integrarsi ai prodotti già esistenti e doveva massimizzare quanto più possibile l'accessibilità e l'usabilità, senza stravolgere le abitudini e le procedure aziendali già presenti e avviate.

A questo punto si è passati alla fase di realizzazione, mettendo in pratica le idee, gli schemi e le procedure ideate in fase di progettazione.

Per quanto concerne gli strumenti utilizzati si è fatto ampio uso dell'ambiente IDE Eclipse, integrato con alcuni plug-in per migliorare lo sviluppo delle pagine JSF. L'applicazione è distribuita per essere installata sul Web Container Apache Tomcat 6.0.x [3] e sfruttare le potenzialità di Oracle 10g.

Capitolo 2

Il Progetto “Modulistica WEB”

Prima di mostrare l’applicazione Modulistica Web, va chiarito il contesto in cui è nata l'idea e si è sviluppato il progetto. In questo capitolo verrà dunque mostrato brevemente lo stato dell'arte in termini di fruibilità e gestione della modulistica nella maggior parte delle piccole medie imprese.

2.1 Scenario “as-is”

La gestione della modulistica all'interno dell'azienda è per lo più un'azione manuale. Per tutti i moduli interni, si utilizzano spesso strumenti non adatti che non permettono di controllare l'accesso e la gestione delle informazioni, come documenti condivisi tramite la rete aziendale, mail per comunicare eventuali modifiche o addirittura moduli in versione cartacea scambiati manualmente tra i diversi dipendenti.

Il processo di gestione della modulistica è schematizzabile in cinque fasi ben distinte:

- Stesura del modulo
- Diffusione ai propri dipendenti
- Compilazione
- Controllo
- Inserimento a sistema

Tutte queste azioni, svolte spesso dall'ufficio del personale, richiedono molto tempo e un notevole dispendio di risorse a causa della poca organizzazione e differenza tra i diversi moduli e regole di compilazione.

Le prime due fasi del processo sono già in parte automatizzate. Tramite i sistemi di office automation, infatti, è possibile generare documenti con alcune informazioni già precompilate, chiedendo ai dipendenti di fornire solo le informazioni mancanti o le eventuali variazioni.

Le fasi più esose in termini di risorse e tempo, sono quelle successive alla compilazione, cruciali per l'azienda. Queste fasi permettono di controllare le informazioni inserite dai dipendenti e di trasferirle all'interno dei sistemi aziendali per poi essere utilizzate per eseguire dell'analisi, statistiche e report dell'andamento dei propri dipendenti.

La compilazione dei moduli, oltre ad essere formalmente corretta, spesso deve rispettare delle regole stringenti indicate dalla normativa vigente. L'azienda deve quindi possedere delle competenze interne per analizzare le diverse normative e applicarle ai moduli compilati.

Il trasferimento delle informazioni all'interno dei sistemi aziendali, avvenendo tramite una digitazione manuale, oltre ad essere una procedura lunga e dispendiosa può generare anomalie ed errori che inficiano così le informazioni fornite. Alcuni sistemi aziendali offrono degli strumenti di controllo che validano le informazioni inserite, evitando così gli errori. Tale controllo però è eseguito solo alla fine del processo e non durante, introducendo così un eventuale ritardo per la correzione. Chi gestisce il modulo, spesso l'ufficio del personale, deve rintracciare il compilatore del modulo e richiedere una correzione delle informazioni in modo da rispettare i vincoli imposti dal modulo stesso.

2.2 Specifiche e requisiti

L'applicazione nasce dall'esigenza di automatizzare tutte e cinque le fasi del processo di gestione della Modulistica aziendale. Per raggiungere tutti i dipendenti dell'azienda si è scelto di creare un'applicazione web che fornisse tutti gli strumenti necessari al raggiungimento di tale obiettivo.

Un'applicazione web permette di utilizzare come terminali, normali web browser che accedono alle funzioni applicative tramite l'utilizzo di un network, come ad esempio un'intranet o attraverso la rete Internet.

L'applicazione, tramite un editor visuale, aiuterà a velocizzare la definizione dei moduli permettendo di caricare sul sistema PDF o moduli già esistenti e configurando le caratteristiche di pubblicazione sarà possibile limitare la visibilità del modulo ad un insieme ristretto dell'azienda, sfruttando un unico applicativo.

La sicurezza delle informazioni inserite, essendo dati sensibili, è un nodo fondamentale per lo sviluppo dell'applicazione. L'accesso ai moduli deve essere garantita da una procedura di autorizzazione molto rigida, che possibilmente si deve integrare con i diversi sistemi aziendali già presenti (Active Directory, LDAP, Token, ecc...). Data la possibilità di gestire moduli di natura completamente diversa, l'autorizzazione alla visualizzazione e gestione dei moduli deve dipendere dal ruolo dell'utente oltre che dall'organizzazione gerarchia dell'azienda.

Le informazioni inserite dai dipendenti devono essere validate tramite procedure automatiche che garantiscano la corretta compilazione del modulo,

sia dal punto di vista formale che dal punto di vista legislativo. Inoltre, devono essere presenti dei controlli che aiutino la compilazione da parte dell'utente, offrendo funzionalità di auto completamento, dinamicità dei campi, calcolo automatico per tutti i campi numerici.

La comunicazione con i sistemi aziendali già esistenti non deve essere unidirezionale ma bidirezionale. L'applicazione deve offrire strumenti configurabili che permettono di recuperare qualsiasi tipo d'informazione da qualsiasi tipo di sistema aziendale, estendendo così le funzionalità di precompilazione sui diversi moduli. A conclusione del flusso approvativo, le informazioni contenute all'interno dei moduli devono poter essere trasferite, tramite l'utilizzo di procedure automatiche, nei diversi sistemi aziendali, offrendo così la possibilità di tenere aggiornati tutti i diversi sistemi informativi.

Queste procedure devono poter essere personalizzate con semplicità, in modo da gestire al meglio l'eterogeneità delle informazioni e le loro differenti destinazioni.

Il processo approvativo non deve essere rigido, ma anch'esso configurabile nei minimi dettagli per rispettare tutte le procedure aziendali già esistenti.

L'applicazione dovrà tener conto della multi-nazionalità dei dipendenti dell'azienda, perciò si dovrà adottare una modalità efficiente di gestione dell'internazionalizzazione.

A livello tecnico l'applicazione deve poter essere installata su qualsiasi application server che rispetti le specifiche Servlet 2.5 e JSP 2.1, su cui è installata una versione Java 1.6 o superiore.

Lato Database non è richiesto alcun vincolo, l'applicazione deve essere in grado di poter funzionare su un qualsiasi DBMS opportunamente configurato.

2.3 Architettura del sistema

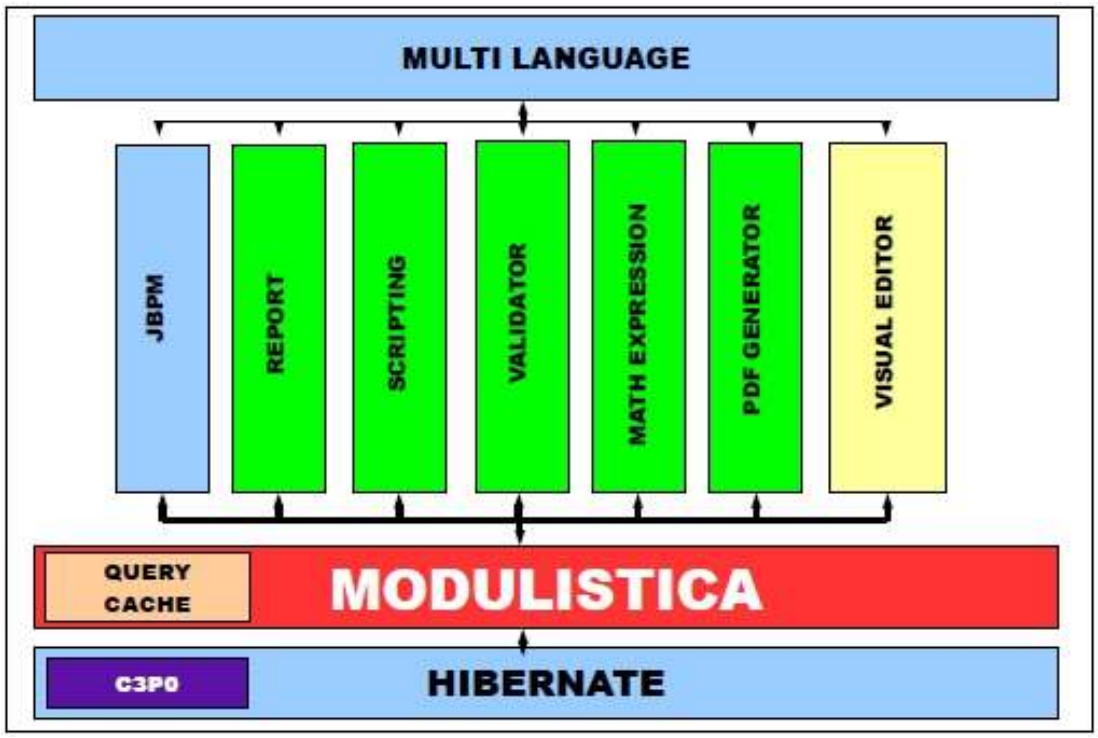


Figura 1 Architettura Applicativo Modulistica WEB.

In Figura 1 è rappresentata in modo astratto la struttura dell'applicazione, evidenziando i moduli principali e i legami presenti tra essi.

In azzurro sono riportate tutte quelle librerie sviluppate da terze parti, open source, che sono state utilizzate per lo sviluppo dell'applicazione e che quindi sono alla base della stessa.

Il cuore dell'applicazione, in rosso, racchiude tutte le logiche necessarie a gestire i moduli, gli utenti e le diverse funzionalità applicative. Al suo interno trovano spazio anche tutte le logiche di presentazione, cioè tutti quei componenti che regolano l'aspetto dell'applicazione verso l'utente finale.

In verde sono indicati tutti i moduli aggiuntivi che svolgono una funzione ben precisa. I principali sono sicuramente:

- Il modulo dei Report che permette di eseguire e di configurare dei report in modo da poter svolgere delle funzioni di analisi sui dati presenti a sistema.
- Il modulo di Scripting e Validator che offrono funzionalità di validazione della compilazione, rendendo lo strumento molto flessibile.
- MultiLanguage che permette di rendere lo strumento internazionale non legandolo a una singola lingua, in base all'utente che si collega.

A supporto di tutta l'applicazione sono presenti due moduli di non poca importanza. Il primo, la QueryCache, permette di introdurre un'ottimizzazione per il funzionamento dell'applicativo. Esso, infatti, offre una cache per tutte le query necessarie ai moduli che saranno impostati, aumentando enormemente le prestazioni delle stesse. Il Visual Editor è praticamente il modulo che permette di configurare l'intero applicativo. Tramite quest'ultimo, infatti, è possibile disegnare un modulo configurando i singoli campi sfruttando un ambiente di lavoro completamente visuale.

2.3.1 Livelli di autorizzazione

Il modello di gestione proposto all'interno di Modulistica Web prende in considerazione **gli attori** di seguito indicati:

- Il **redattore del modulo**, che predispone la modulistica da automatizzare e la rende disponibile per la compilazione impostando tutti i filtri necessari alla sua pubblicazione.
- **L'utente**, che si autentica sul sistema e, tramite una procedura guidata, accede al modulo, provvede alla sua compilazione e alla successiva trasmissione agli uffici o figure aziendali competenti.
- **Utente Responsabile** autorizza eventuali moduli dei propri collaboratori, analizzandone il contenuto e trasmettendoli agli uffici o figure aziendali competenti.
- **Ufficio del personale** gestisce i moduli compilati dall'utente e effettua gli eventuali trasferimenti all'interno dei diversi sistemi aziendali.

2.3.2 Layout applicativo

L'applicazione deve integrarsi ai prodotti già esistenti rendendo l'esperienza d'uso la meno complicata possibile per gli utenti.
Si è scelta una struttura molto semplice e lineare che ricalca la classica struttura delle applicazioni a finestra, abitualmente utilizzate nel mondo office. È possibile dividere l'area di lavoro in tre distinte zone che racchiudono informazioni di diverso tipo.

La parte in alto dell'applicazione è riservata ai menu, sempre visibili, in modo da rendere accessibili tutte le informazioni e azioni dell'utente. Da questa sezione sarà possibile richiamare le varie funzionalità e accedere alle diverse sezioni dell'applicativo. Inoltre sarà possibile utilizzare alcune funzionalità particolari, come l'opera Come o il cambio lingua.

La parte centrale visualizza tutte le informazioni richiamate dall'utente. Dall'eventuale modulo all'elenco dei dipendenti che hanno compilato un modulo. In questa sezione l'applicazione visualizzerà tutti i controlli necessari a interagire con l'utente.

Infine la parte bassa, visualizza tutte le informazioni aggiuntive che possono informare l'utente su eventuali errori o anomalie.

2.3.3 Architettura Database

L'applicazione deve poter gestire qualsiasi tipo di modulo, indipendentemente dalla sua struttura e della sua lunghezza. Non deve essere posto quindi, alcun limite per quanto riguarda il numero dei campi e del loro formato. La gestione dei moduli e delle informazioni in essi contenuti è quindi la parte fondamentale dell'architettura del database. Per gestire queste informazioni sono state create delle tabelle differenti in funzione del formato del dato da gestire. Sarà presente una tabella dove verranno memorizzati i campi di tipo testo, un'altra di tipo numerico, un'altra di tipo data e così via. Queste tabelle sono indicizzate in base alla posizione del campo all'interno del modulo e al nome del campo stesso. In questo modo è possibile gestire qualsiasi modulo e qualsiasi tipo di campo.

Per quanto riguarda invece la struttura del modulo, verrà sfruttata la Serializzazione degli oggetti java, memorizzando nel database, un'oggetto che contiene tutte le informazioni necessarie a disegnare ed impostare il modulo. All'interno dell'oggetto saranno memorizzati il numero di pagine del modulo, il numero, tipo e disposizione dei campi, così come le caratteristiche di ogni singolo campo, l'aspetto grafico, le procedure di controllo, ecc...

Capitolo 3

Tecnologie Utilizzate

In questo capitolo saranno riportate le informazioni necessarie per introdurre e descrivere le tecnologie scelte per lo sviluppo della Modulistica Web. L'utilizzo di una tecnologia rispetto ad altre, influenza notevolmente lo sviluppo di un'applicazione web, per questo saranno elencati i motivi principali che hanno portato alla scelta di una particolare tecnologia rispetto alle possibili concorrenti.

3.1 Java Server Faces (JSF)

JSF, Java Server Faces [1], è il frame work della Java EE per lo sviluppo dell'interfaccia web e per la gestione del flusso di navigazione dell'applicazione.
JSF pur essendo simile ad altri frame work, per alcuni servizi che fornisce introduce numerosi aspetti di novità che lo rendono senza dubbio il modello di sviluppo dominante nel breve futuro.
JSF è strutturato secondo il pattern MVC, ma propone un modello di sviluppo per componenti a eventi che lo rende molto simile a quello che è Swing per lo sviluppo di applicazioni Client-Server.
JSF si propone come un frame work che rende possibile sviluppare l'interfaccia web di una applicazione server-side secondo la logica RAD, Rapid Application Development, tipica degli ambienti di sviluppo di Swing, in cui si ha a disposizione una palette di componenti predefiniti, realizzati secondo lo standard dei JavaBeans, da selezionare e trascinare nell'area in cui si costruisce l'interfaccia.

JSF si occupa di indirizzare lo sviluppo dell'interfaccia dell'applicazione e non del livello di business-logic, per il quale restano validi tutti i concetti conosciuti.

Uno degli elementi fondamentali di JSF è quindi un insieme di componenti che rappresentano gli elementi più comuni di un'interfaccia web e alcune librerie di custom-tag per l'utilizzo di questi componenti nelle pagine JSP.

In JSF gli elementi di una pagina JSP sono rappresentati lato server da un albero di oggetti che rappresenta lo stato dell'interfaccia in ogni momento del suo ciclo di vita.

Gli elementi principali che costituiscono JSF possono essere quindi così riassunti:

- Un insieme di componenti di interfaccia predefiniti. JSF comprende un insieme di componenti pronti per l'uso che rispondono alla gran parte delle comuni necessità di sviluppo di una interfaccia web.
- Un modello di sviluppo "event-driven". Il modello di sviluppo di JSF è a eventi. Gli eventi sono costituiti dalle azioni che l'utente effettua sui componenti di interfaccia. A questi eventi vengono registrati dei listener per la gestione server-side degli stessi.
- Un modello a componenti. Ogni elemento dell'interfaccia è realizzato da un componente, una classe che implementa una interfaccia stabilita. Questo modello fornisce al framework modularità, riusabilità ed estensibilità in quanto è possibile realizzare i propri componenti custom qualora le esigenze lo richiedessero.

3.1.1 I servizi JSF

Gli strumenti messi a disposizione da JSF forniscono i servizi tipici di cui ha bisogno uno sviluppatore di applicazioni web:

- **Conversioni di tipo**. Com'è noto, un client web può inviare via HTTP al server solo delle stringhe. È quindi poi necessario convertire queste stringhe in valori corrispondenti ai tipi necessari per il livello di Model. JSF fornisce gli strumenti per automatizzare e rendere molto semplice questa conversione.
- **Validazione**. La validazione dei campi di un form è uno dei compiti più comuni e ripetitivi nello sviluppo dell'interfaccia web. JSF fornisce una serie di validatori standard per le esigenze tipiche, che sollevano lo sviluppatore dal realizzare quest'attività di scarso valore aggiunto, e mette a disposizione un modello di sviluppo che consente di realizzare molto semplicemente validatori custom.
- **Binding** tra campi d'interfaccia e attributi di JavaBean. I dati immessi lato client nell'interfaccia web di un'applicazione devono poi essere contenuti in oggetti lato server che modellano le entità sulle quali devono andare ad agire le elaborazioni del livello di Model. I dati d'interfaccia, dopo essere stati convertiti in valori di tipo opportuno ed essere stati sottoposti a validazione, devono andare a popolare opportuni attributi di JavaBean. JSF fornisce gli strumenti per eseguire quest'operazione di "binding" in modo automatico e dichiarativo.

- **Gestione del Flusso di Navigazione**. Una parte molto importante di un'applicazione web è la gestione del flusso di navigazione tra le pagine web che la costituiscono. JSF, analogamente a quanto fatto ad esempio in Struts, consente di definire in maniera semplice ed esternamente al codice la logica di navigazione della propria applicazione.
- **Rendering dei componenti**. Un aspetto molto importante in JSF è la separazione tra il comportamento di un elemento d'interfaccia e la sua rappresentazione grafica. L'elemento d'interfaccia è modellato da una classe che ne definisce il comportamento. L'aspetto grafico, in altre parole la sua "renderizzazione", è a carico di una classe completamente distinta, un Renderer appunto. Ciò significa che una stessa interfaccia può essere "renderizzata" in formati diversi senza modificarne il comportamento funzionale. Ovviamente il Renderer Kit standard fornito con la reference implemetation di JSF è quello per l'HTML, ma ne esistono altri e nulla vieta di personalizzarli o di svilupparne di nuovi.
- **Gestione degli errori**. JSF ha i meccanismi necessari per la gestione degli errori e per la segnalazione degli stessi al livello d'interfaccia.
- **Internazionalizzazione**. JSF supporta la gestione dell'internazionalizzazione di un'applicazione web con strumenti molto simili a quelli già in uso in altri framework.

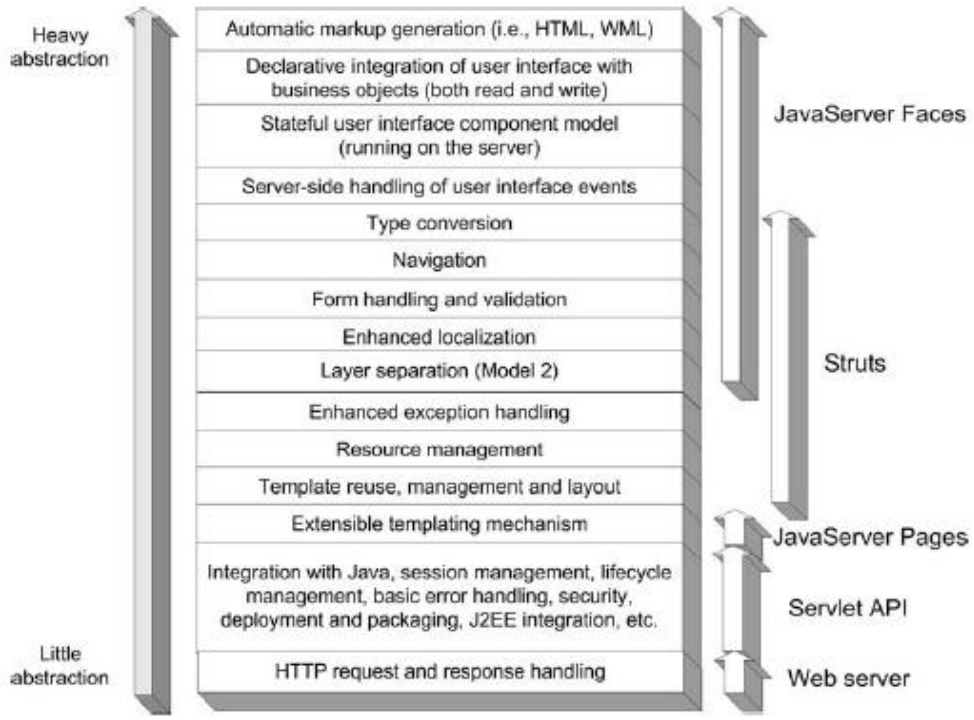


Figura 2 Architettura JSF.

Com'è facile osservare, JSF oltre a fornire servizi già presenti in altri framework introduce un livello di astrazione più elevato, analogamente a quanto fece Struts rispetto a Servlet e JSP. Come raffigurato molto efficacemente nella Figura 2, tratta da [1], se pensiamo l'infrastruttura di un'applicazione web come uno stack di servizi che partono da un livello di astrazione minimo fino a un livello più alto, vediamo come i servizi di JSF forniscono uno strato di astrazione più elevato rispetto a quelli forniti da Struts, che già a loro volta erano più in alto nella pila rispetto a quelli dell'implementazione standard di Servlet e JSP.

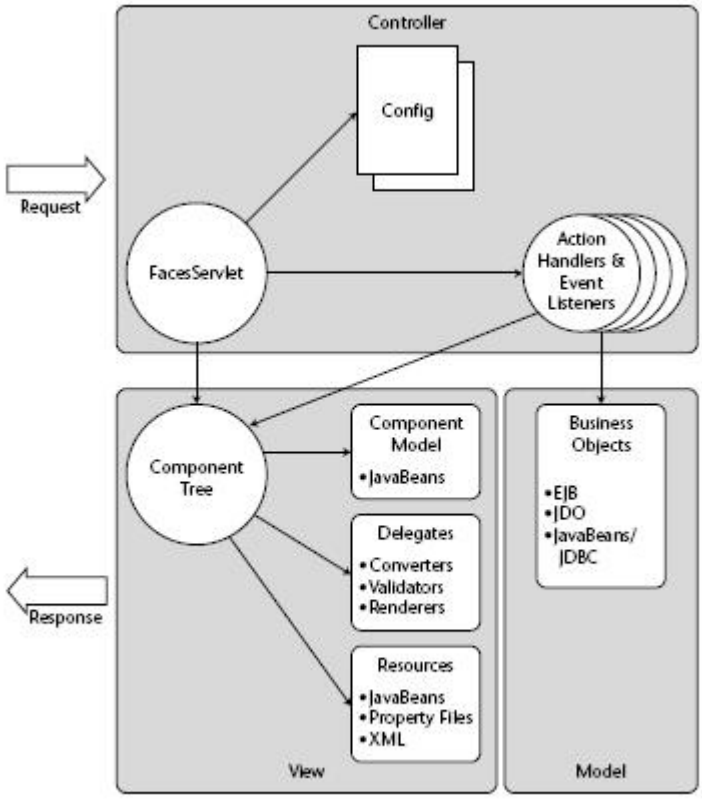


Figura 3 I servizi di un'infrastruttura web

3.1.2 Perché JSF?

È lecito chiedersi la ragione nell'adottare un nuovo frame work quando ne esistono già numerosi altri, uno su tutti Struts, già sperimentati, conosciuti e ormai maturi e affidabili.

La discussione potrebbe essere lunga ed eventuali scelte andrebbero valutate caso per caso. Va anche detto che è possibile ad esempio integrare JSF con Struts se si volesse reingegnerizzare l'interfaccia web di un'applicazione lasciando però inalterato tutto il codice già realizzato per la parte di business-logic.

Alcuni aspetti, però, fanno di JSF senza dubbio l'opzione privilegiata nella scelta di un frame work di sviluppo in un progetto che parte da zero:

- **Standardizzazione**. A differenza di tutti gli altri frame work, JSF fa parte della specifica Java EE, ed è quindi oggi il frame work standard per lo sviluppo dello strato web. Questo fa sì che tutti gli sviluppatori di applicazioni web possano riferirsi a un unico modello riducendo così la frammentazione oggi esistente e accrescendo la condivisione delle conoscenze e le possibilità di estensione delle funzionalità del frame work stesso.
- **Supporto negli ambienti di sviluppo**. JSF è standard ed è supportato da tutti gli ambienti di sviluppo Java EE. Ciò prima non avveniva, fatta eccezione probabilmente solo per Struts che costituiva uno standard de facto. La logica a componenti di JSF consente a questi tool di fornire allo sviluppatore un ambiente RAD per lo sviluppo dell'interfaccia web aumentando notevolmente la produttività rispetto alle metodologie di sviluppo precedentemente in uso nel mondo Java EE.
- **Struttura a componenti**. Questa è la vera novità di JSF. Ogni elemento dell'interfaccia è un componente riusabile e ciò rende l'architettura molto ben strutturata, modulare ed estensibile. In JSF una pagina web corrisponde lato server a un albero di oggetti che ne rappresentano gli elementi. Esistono implementazioni di componenti JSF, quali ad esempio i componenti Richfaces di JBoss, che vengono distribuiti in modalità open source, che estendono le potenzialità di JSF e costituiscono un tipico esempio di come sia possibile arricchirne le funzionalità.

3.2 RichFaces

RichFaces [4] è un framework open source che aggiunge funzionalità Ajax ad applicazioni JSF esistenti senza dover ricorrere a JavaScript.

RichFaces sfrutta tutte le funzionalità messe a disposizione da Java Server Faces, la convalida, i meccanismi di conversione e gestione delle risorse statiche e dinamiche. Per questo i componenti RichFaces, con supporto integrato Ajax, possono essere facilmente incorporati all'interno delle applicazioni JSF.

RichFaces permette di:

- Intensificare l'intero insieme di benefici JSF aggiungendo il pieno supporto Ajax alle applicazioni. RichFaces è completamente integrato nel ciclo di vita JSF. Permette di abilitare il supporto Ajax a livello di pagina e non solo a livello di componente. In questo modo è possibile definire regioni più o meno estese della pagina che devono essere aggiornate in seguito ad una richiesta Ajax invocata dal client.
- Velocizza la creazione delle pagine sfruttando l'insieme dei componenti contenuti nel framework, permettendo così di creare un'interfaccia utente ricca di funzionalità. Inoltre, i componenti RichFaces sono progettati per essere utilizzati senza problemi con altri librerie di componenti, in modo da avere più opzioni per sviluppare le proprie applicazioni.
- Permette di creare dei propri componenti che integrino il supporto Ajax. Sfruttando il Component Development Kit (CDK) che è stato

utilizzato per la creazione di RichFaces. Il CDK include un impianto di generazione del codice e una struttura di template che utilizza una sintassi simile a JSP. Queste funzionalità aiutano a evitare un processo di routine durante la creazione di un componente.

- RichFaces fornisce un supporto avanzato per la gestione delle varie risorse: immagini, codice JavaScript e fogli di stile CSS.
- Creare una moderna e ricca interfaccia utente personalizzabile nei colori e nella Skin. RichFaces ha una funzione che permette facilmente di definire e gestire le diverse combinazioni di colori ed altri parametri dell'interfaccia utente. Questi parametri sono accessibili direttamente da codice Java, offrendo così la possibilità di modificarli a runtime. RichFaces viene fornito con una serie di skin predefinite, ma è anche possibile creare facilmente le proprie skin personalizzate.

Il frame work è implementato come una libreria di componenti che aggiunge funzionalità Ajax in pagine già esistenti, quindi non c'è bisogno di scrivere codice JavaScript o di sostituire i componenti esistenti con nuovi widget Ajax. RichFaces consente di abilitare il supporto Ajax a livello di pagina al posto del tradizionale supporto a livello di componente. Quindi, è possibile definire l'evento sulla pagina che invoca una richiesta Ajax e le aree della pagina che devono essere sincronizzate con l'albero dei componenti JSF, dopo che la richiesta Ajax cambia i dati sul server in base alla eventi generati sul client.

La figura seguente mostra come funziona:

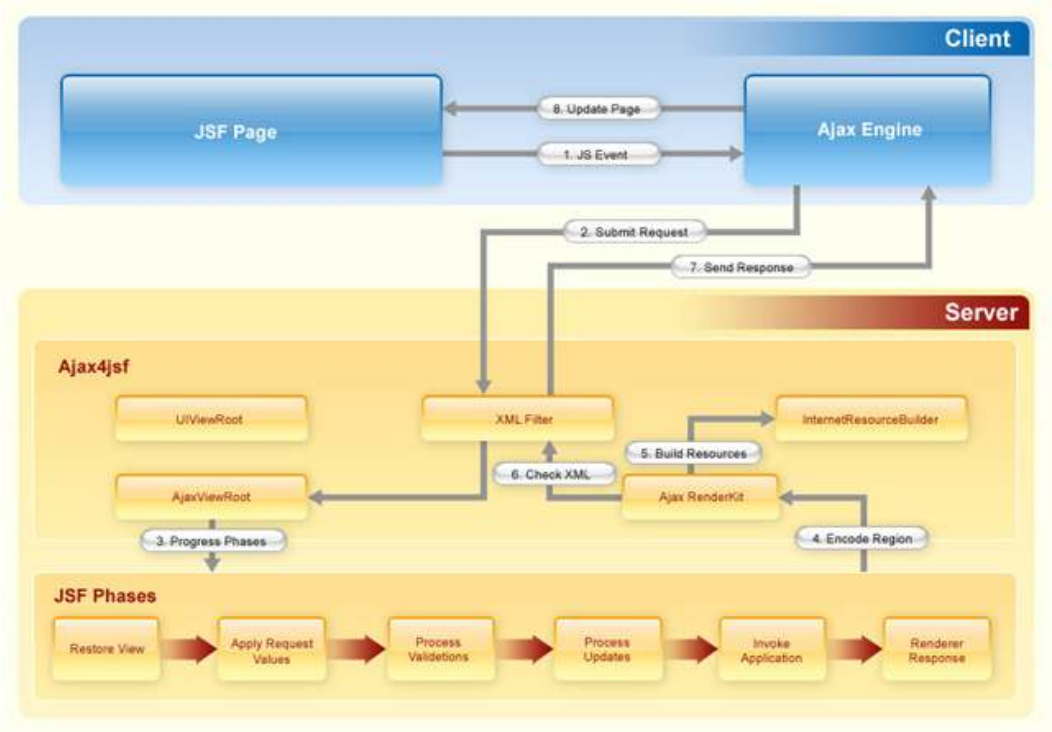


Figura 4 Richiesta di elaborazione del flusso

RichFaces permette di definire (per mezzo di tag JSF) diverse parti di una pagina JSF che si desidera aggiornare con una richiesta Ajax e fornisce alcune opzioni per inviare richieste Ajax al server senza dover scrivere alcun JavaScript, tutto viene fatto automaticamente.

3.3 RichFaces Panoramica dell'architettura

La Figura successiva mostra alcuni importanti elementi di RichFaces

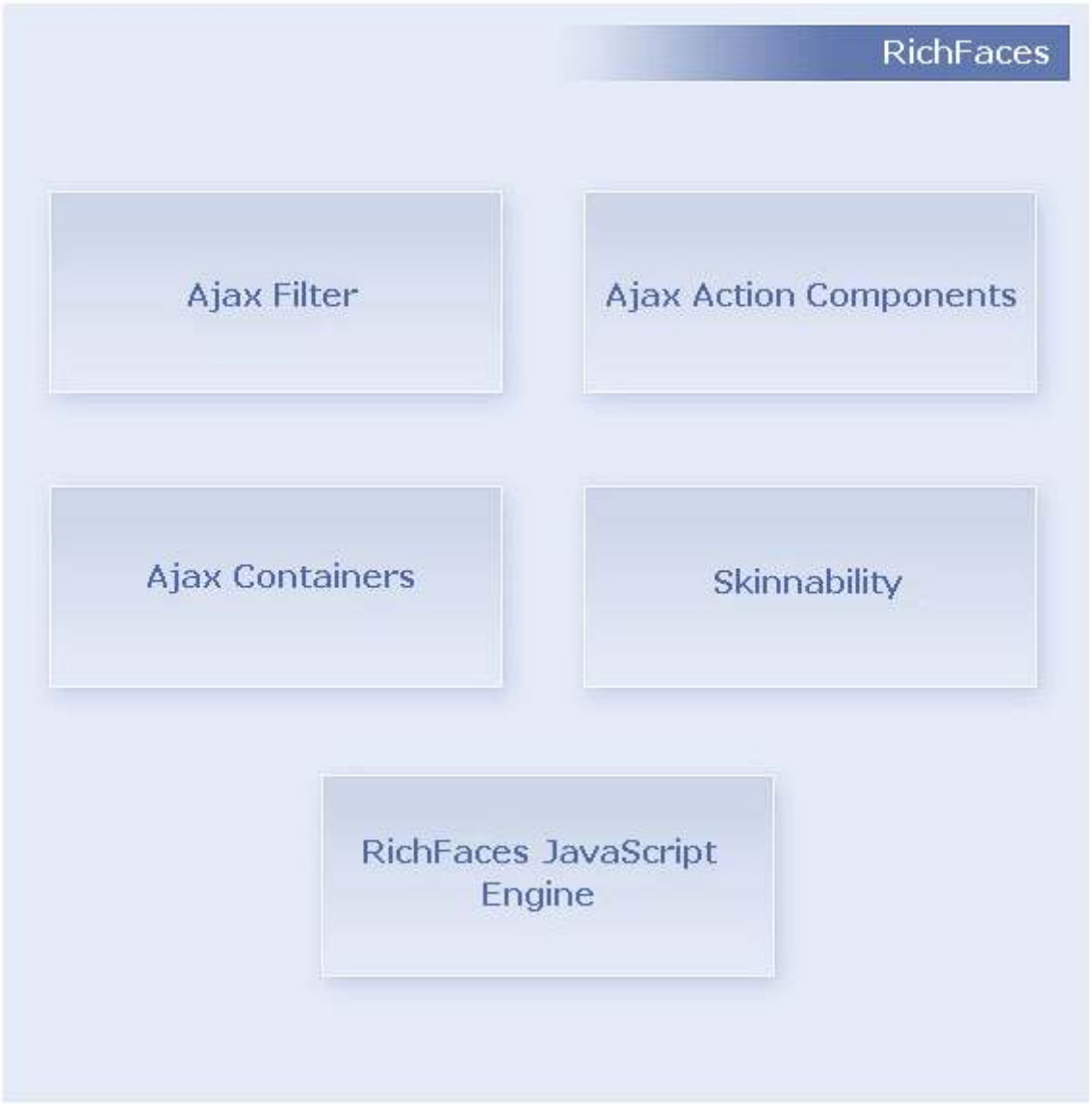


Figura 5 Nucleo Ajax di RichFaces.

Per ottenere tutti i benefici di RichFaces, si deve registrare un filtro nel file web.xml dell'applicazione. Il filtro riconosce più tipi di richiesta. Le informazioni necessarie sulla configurazione del filtro possono essere trovate nella

documentazione del framework alla sezione Configurazione Filtro[1]. Il diagramma di sequenza nella Figura 6 mostra la differenza di elaborazione di una richiesta "normale" JSF e una richiesta Ajax.

Nel primo caso l'intero albero JSF viene codificato, nel secondo caso invece, vengono codificati solo i componenti contenuti nella regione Ajax. Come si può vedere, nel secondo caso il filtro analizza il contenuto di una risposta Ajax prima di inviarlo al lato client.

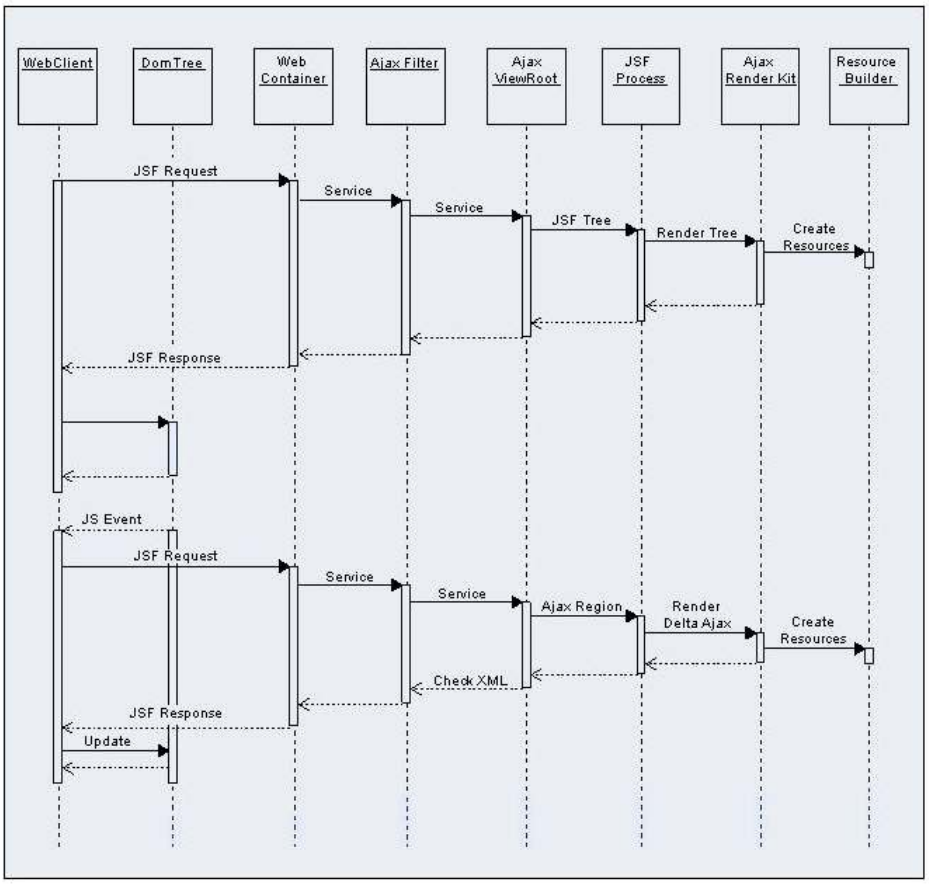


Figura 6 Diagramma di sequenza per la richiesta di elaborazione

In entrambi i casi, le informazioni sulle risorse statiche o dinamiche richieste dall'applicazione vengono registrate nella classe ResourseBuilder.

[1] http://docs.jboss.org/richfaces/latest_3_3_X/en/devguide/html/ArchitectureOverview.html

Quando viene richiesta una risorsa (Figura 7 Diagramma di sequenza per la richiesta risorsa), il filtro RichFaces controlla la cache di risorse, e se viene individuata, la risorsa viene inviata al client. In caso contrario, il filtro cerca la risorsa tra quelle registrate dal ResourceBuilder. Se la risorsa è registrata, il filtro RichFaces invierà una richiesta al ResourceBuilder di creare e consegnare la risorsa.

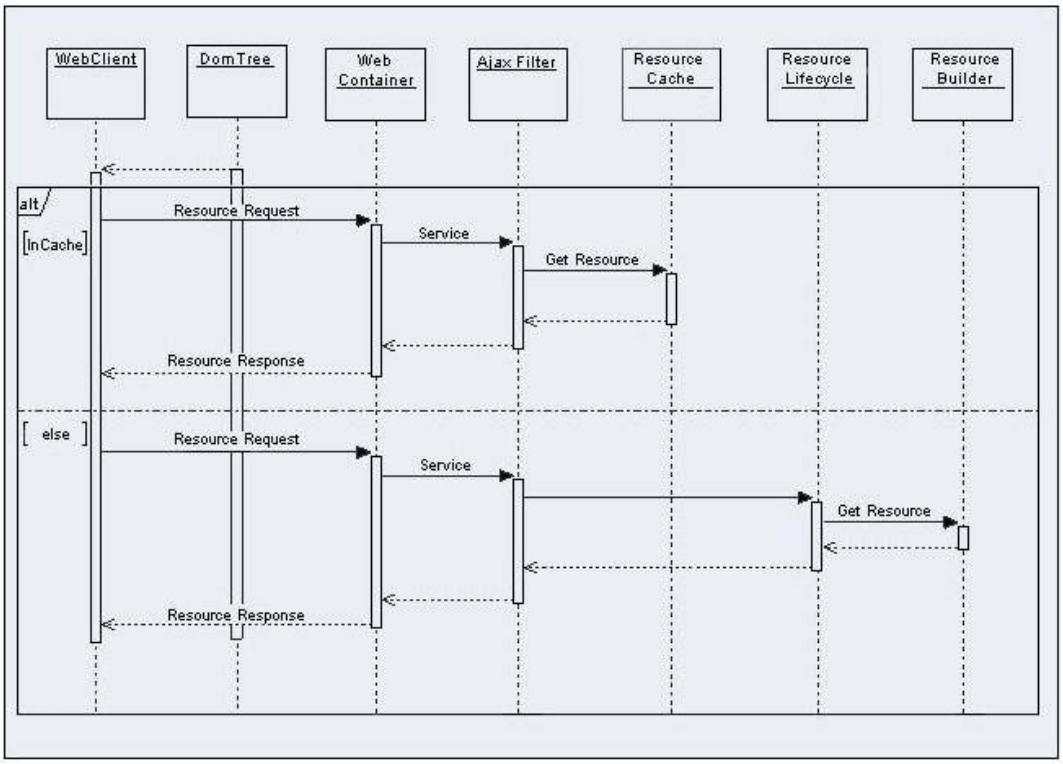


Figura 7 Diagramma di sequenza per la richiesta risorsa

Lato client l'aggiornamento avviene tutto tramite un motore JavaScript che elabora la risposta Ajax e con le informazioni in essa contenute aggiorna le diverse aree della pagina JSF.

3.4 Hibernate

Lavorare con linguaggi orientati agli oggetti e un database relazionale può essere difficoltoso e impegnativo. Hibernate è uno strumento di mappaggio tra oggetti e relazioni (OR) per gli ambienti basati su java. Il termine "mappaggio oggetto-relazionale" ("object relational mapping" o ORM) si riferisce alla tecnica di creare una corrispondenza (mappare) tra una rappresentazione di dati secondo il modello a oggetti e quella secondo il modello relazionale, con uno schema basato su SQL.

Hibernate [5] si occupa non solo del mappaggio dalle classi Java alle tabelle della base di dati (e dai tipi di dato Java a quelli SQL), ma fornisce anche funzionalità di interrogazione e recupero dei dati (query), e può ridurre significativamente i tempi di sviluppo altrimenti impiegati in attività manuali di gestione dei dati in SQL e JDBC.

Lo scopo di Hibernate è di alleviare lo sviluppatore dal 95% dei più comuni compiti di programmazione legati alla persistenza dei dati. Hibernate può non essere la soluzione migliore per le applicazioni data-centriche che usano solo stored-procedures per implementare la logica di business nel database; è principalmente utile con modelli di dominio orientati agli oggetti in cui la logica di business sia collocata nello strato intermedio di oggetti Java (middle-tier). In ogni caso, Hibernate aiuta senza dubbio a rimuovere o incapsulare codice SQL che sia dipendente dal particolare fornitore, e aiutare con i compiti più comuni di traduzione dei set di risultati (result set) da una rappresentazione tabellare a un grafo di oggetti.

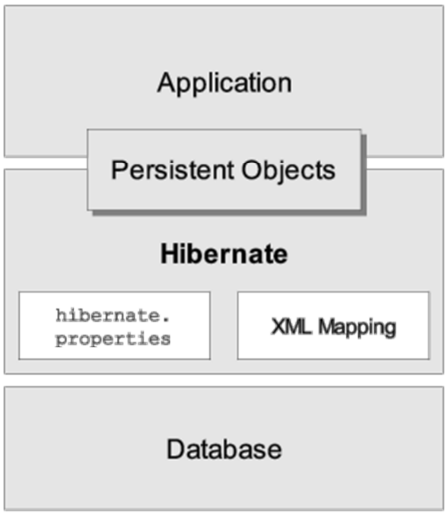


Figura 8 Una visione di alto livello sull'architettura di Hibernate

La Figura 8 mostra come Hibernate utilizzi il database e i dati di configurazione per fornire servizi di persistenza (e oggetti persistenti) all'applicazione.

Mostrare una vista più dettagliata dell'architettura di runtime è molto complesso perché essendo Hibernate molto flessibile, rende possibili diversi approcci. Di seguito saranno mostrati i due casi estremi.

L'architettura "leggera" è quella in cui l'applicazione fornisce le sue connessioni JDBC e gestisce le transazioni. Quest'approccio usa un sottoinsieme minimale delle API di Hibernate.

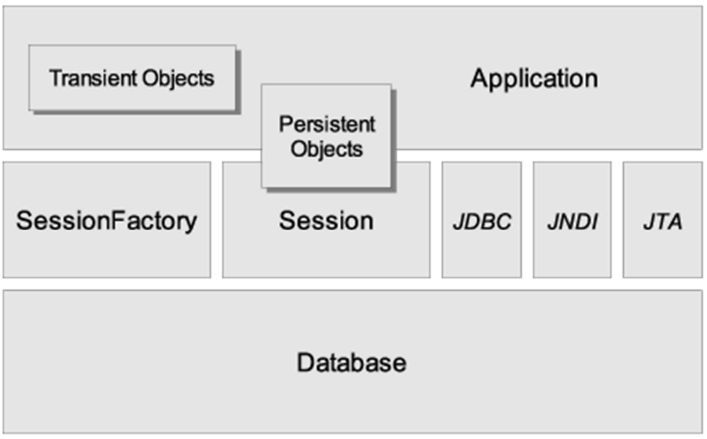


Figura 9 Architettura "leggera"

L'architettura "completa" di Hibernate, permette all'applicazione di astrarre dai dettagli delle API JDBC/JTA e lascia che se ne occupi Hibernate.

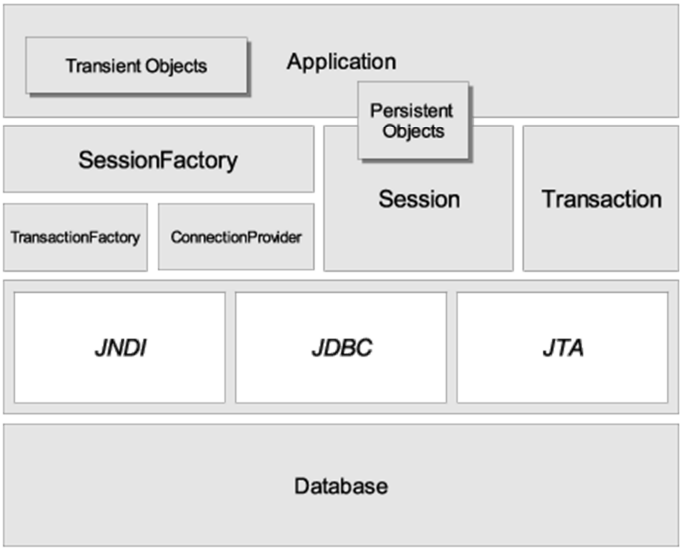


Figura 10 Architettura "completa"

Ecco alcune definizioni degli oggetti presenti nei diagrammi:

- **SessionFactory** È una cache immutabile e "thread-safe" di mappaggi compilati per un database singolo. Allo stesso tempo è un factory per oggetti Session e un client di ConnectionProvider. Potrebbe contenere

una cache di secondo livello opzionale riutilizzabile tra le transazioni, sia a livello di processo, sia a livello di cluster.

- **Session** È un oggetto "mono-thread", di corta durata, che rappresenta una conversazione tra l'applicazione e il contenitore persistente. Incapsula una connessione JDBC. È un factory per oggetti Transaction. Mantiene una cache obbligatoria (di primo livello) per gli oggetti persistenti, usata quando si naviga il grafo degli oggetti o si ricercano oggetti per identificatore.
- **Oggetti persistenti e collezioni** Sono oggetti di breve durata, a thread singolo, che contengono stato persistente e funzioni applicative. Potrebbero essere normali oggetti POJO/Javabeans, con l'unica particolarità che in un dato momento sono associati con (esattamente) una Session. Nel momento in cui la Session viene chiusa, verranno staccati e saranno liberi di essere usati in qualsiasi strato applicativo (ad esempio direttamente come oggetti di trasferimento dei dati da e allo strato di presentazione).
- **Oggetti transienti e collezioni** Sono le istanze delle classi persistenti che in un dato momento non sono associate con una Session. Possono essere state istanziate dall'applicazione e non (ancora) rese persistenti, o possono essere state istanziate da una Session poi chiusa.
- **Transaction** È un oggetto a thread singolo, di corta durata, usato dall'applicazione per specificare unità di lavoro atomiche. Separa le

applicazioni dalla transazione JTA, CORBA o JDBC sottostante. Una Session potrebbe estendersi lungo varie Transaction in certi casi.

- **ConnectionProvider** Un *factory* (e *pool*) di connessioni JDBC. Astrae le applicazioni dai dettagli dei sottostanti Datasource o DriverManager. Non viene esposta all'applicazione, ma può essere estesa/implementata dagli sviluppatori.
- **TransactionFactory** Un *factory* per istanze di Transaction. Non viene esposta all'applicazione, ma può essere estesa/implementata dagli sviluppatori.

3.4.1 Configurazione

Un modo di configurare Hibernate è di specificare una configurazione completa in un file chiamato *hibernate.cfg.xml*. Questo file può essere usato come un'alternativa al file *hibernate.properties* o, se sono presenti entrambi, per ridefinirne le proprietà.

Il file di configurazione XML viene caricato da Hibernate dalla radice del *CLASSPATH*.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>

        <property name="hibernate.connection.url">
                jdbc:oracle:thin:@[HOST][:PORT]:SID
        </property>
        <property name="hibernate.cglib.use_reflection_optimizer">
                true
        </property>
        <property name="hibernate.connection.password">password</property>
        <property name="hibernate.connection.username">username</property>
        <property name="hibernate.connection.driver_class">
                oracle.jdbc.Driver
        </property>
        <property name="hibernate.dialect">
                org.hibernate.dialect.Oracle10gDialect
        </property>
        <property name="c3p0.acquire_increment">1</property>
        <property name="c3p0.idle_test_period">30</property>
        <property name="c3p0.max_size">5</property>
        <property name="c3p0.max_statements">0</property>
        <property name="c3p0.min_size">1</property>
        <property name="c3p0.timeout">60</property>

        <mapping resource="mapping/…" />
</session-factory>
</hibernate-configuration>
```

Figura 11 : File di configurazione di Hibernate

Tutti le etichette (*tag*) *property* vengono utilizzati per impostare la connessione al DBMS o altre opzioni di Hibernate secondarie. Le impostazioni sono relative esclusivamente all'accesso al database e spaziano dall'url di connessione ai parametri utilizzati per l'accesso fino ad arrivare alla definizione di un dialetto (*dialect*) sql. Questo non è strettamente necessario, a meno che si desidera usare

la generazione di chiavi primaria native o *sequence* o il *locking* pessimistico. Comunque, se si specifica un *dialect*, Hibernate imposterà dei valori predefiniti adatti per alcune proprietà qui non elencate, risparmiando lo sforzo di specificarle manualmente.

La parte successiva, riguarda la configurazione del pool di connessione. Questo è un insieme di connessioni prestabilite che vengono lasciate in attesa, pronte per essere utilizzate. Quando il client ha bisogno di eseguire un'operazione, richiede una connessione al pool, esegue l'operazione e la rilascia immediatamente, occupando la risorsa solo per il tempo strettamente necessario. È statisticamente improbabile che tutti gli utenti dell'applicazione eseguano un'operazione su quella risorsa contemporaneamente, quindi il sistema risulta efficiente. L'algoritmo di "pooling" (mantenimento nel lotto) di Hibernate è abbastanza rudimentale. Ha lo scopo di aiutare a cominciare a lavorare, ma non è fatto per l'uso in un sistema in produzione, o anche solo per dei test di performance. La documentazione di Hibernate suggerisce di sfruttare un'altra libreria di pooling per contare su performance migliori e maggior stabilità. Per far questo, bisogna sostituire la proprietà *hibernate.connection.pool_size* con le proprietà specifiche per il settaggio del pool scelto. Hibernate fornisce la possibilità di utilizzare diverse librerie, come C3P0. C3P0 è una libreria open source di pooling per connessioni JDBC che viene distribuita insieme ad Hibernate, e i cui file *jar* devono essere inseriti nella directory *lib* dell'applicazione per poterne sfruttare le potenzialità. Impostando le proprietà *hibernate.c3p0.**, Hibernate userà il C3P0ConnectionProvider integrato per il pooling delle connessioni. Tali proprietà possono essere definite in vario modo:

attraverso il file di configurazione di Hibernate come mostrato in Figura 11, oppure aggiungendo un file di configurazione, denominato c3p0.proprierties, all'interno del quale vengono definite tutte le proprietà relative a questa libreria.

Finita la configurazione di Hibernate per quel che riguarda la connessione alla base dati, si elencano tutti i file di "mappaggio" che legano ogni singola tabella con oggetti persistenti e permettono a Hibernate di svolgere il suo lavoro. Questi file sono fondamentali per l'uso di Hibernate. Un esempio viene riportato in Figura 12 File di configurazione per la tabella Docente.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="mapping.Docente" table="docente" schema="public">
        <id name="matricola" type="integer">
            <column name="matricola" />
            <generator class="assigned" />
        </id>
        <property name="nome" type="string">
            <column name="nome" length="50" not-null="true" />
        </property>
        <property name="cognome" type="string">
            <column name="cognome" length="50" not-null="true" />
        </property>
        <property name="email" type="string">
            <column name="email" length="50" />
        </property>
        <property name="password" type="string">
            <column name="password" length="20" />
        </property>
               …
        <set name="gestisces" inverse="true">
            <key>
                <column name="matricoladocente" not-null="true" />
            </key>
            <one-to-many class="mapping.Gestisce" />
        </set>
        <set name="avvisis" inverse="true">
            <key>
                <column name="dest" />
            </key>
            <one-to-many class="mapping.Avvisi" />
        </set>
               …
        <set name="insegnas" inverse="true">
            <key>
                <column name="matricoladocente" not-null="true" />
            </key>
            <one-to-many class="mapping.Insegna" />
        </set>
    </class>
</hibernate-mapping>
```

Figura 12 File di configurazione per la tabella Docente

La prima cosa da notare è che tale file è classe-centrico: descrive cioè il modo in cui far corrispondere una singola classe Java a una o più tabelle dello schema relazionale e non viceversa. Nel *tag* radice <hibernate-mapping> vi troviamo il *package* di default da utilizzare, se non diversamente specificato per i singoli *tag* <class>. In quest'ultimo troviamo invece tutti i dettagli per il mapping della classe *Docente*: il nome della tabella che ne conterrà i dati e, di seguito elencate, tutte le proprietà che corrispondono ai vari campi di ciascun record.

Tralasciando la descrizione delle proprietà più ovvie soffermiamoci sui *tag* <id> e <set>. Il primo permette di specificare la configurazione della proprietà che

farà da chiave primaria per le istanze della classe e, rispettivamente, per le righe della tabella utilizzata. Con Hibernate le chiavi primarie delle tabelle assumono un'importanza fondamentale, per esempio per poter capire se un'istanza è già presente nel database o se l'oggetto in questione non è mai stato salvato prima. Il consiglio è di utilizzare sempre una chiave primaria "tecnica" per ogni tabella, priva di alcun particolare significato applicativo. Hibernate consente in proposito l'utilizzo di diversi algoritmi per la generazione di chiavi univoche e supporta inoltre le opzioni di ciascun DB server per questa funzionalità (campi ad incremento automatico, sequenze…). La proprietà *generator* serve per configurare questo comportamento.

Il *tag* <set> consente invece la configurazione in modo facile e intuitivo di una tipica relazione uno a molti. Con le relative proprietà (e *tag* xml innestati) è possibile specificare quanto segue:

- Il nome dell'istanza Java Collection che contiene le istanze degli oggetti.
- La proprietà *inverse*, impostata a true, indica che la relazione tra le istanze delle due classi di oggetti è biunivoca. Anche gli oggetti script cioè manterranno riferimento all'applicazione cui appartengono.
- Il *sotto-tag* <key> permette di specificare la colonna contenente la chiave esterna della relazione, in questo caso "matricoladocente" nella tabella *Gestisce*.
- Infine, il *tag* <one-to-many> lega la classe *Docente* con *Gestisce* in una relazione uno a molti. Nel file di configurazione *Gestisce.hbm.xml* potrete ritrovare il corrispondente *tag* <many-to-one>. In tal modo Hibernate è a

conoscenza delle informazioni necessarie per gestire la relazione (navigabile in entrambe le direzioni) tra *Docenti* e *Gestisce*.

3.5 Oracle

Oracle è uno tra i più famosi database management system (DBMS), cioè sistema di gestione di basi di dati, ed è stato scritto in linguaggio C. Fa parte dei cosiddetti RDBMS (Relational DataBase Management System) ovvero di sistemi di database basati sul Modello relazionale che si è affermato come lo standard dei database dell'ultimo decennio.

Una base di dati Oracle comprende istanze e dati memorizzati. Un'istanza è costituita da un insieme di processi di sistema e strutture di memoria che interagiscono con i dati memorizzati. Tra questi processi i seguenti sono necessari per il funzionamento dell'istanza:

- PMON (monitor dei processi)
- SMON (monitor di sistema)
- DBWR (scrive nei datafile)
- LGWR (scrive nei logfile)
- CKPT (scrive i checkpoint controllandone la consistenza)
- ARCH (archiviatore dei log delle transazioni per il DB in modalità archive log mode)

Un compito importante è svolto dalla System Global Area (SGA), una regione di memoria condivisa che contiene dati ed informazioni per il controllo di

un'istanza Oracle. La SGA si occupa della cache, i dati bufferizzati, i comandi SQL e informazioni sull'utente.

Le strutture fisiche fondamentali per un'istanza sono:

- Control files: qui sono memorizzate informazioni essenziali al corretto funzionamento del database. Tra queste il DBID identificativo dell'istanza, il valore di CKPT per la sincronizzazione dei datafile e dati relativi ad alcune viste V$ da interrogare quando il DB stesso non è in stato di Open. È necessario averne almeno uno associato all'istanza; per maggior sicurezza possono esserne creati più di uno, il database stesso si occuperà della loro sincronizzazione, in modo da poter avviare il DB anche in stato di mount ed avviare un recovery.
- L'archivio delle transazioni (online redo logs): i redologs sono necessari per il funzionamento del Db stesso, il numero minimo di redo logs e è 2.
- I rollback/undo segments
- Il tablespace system
- Un tablespace di tipo "temporaneo"

Oracle memorizza i dati sia logicamente, sotto forma di tablespace, sia fisicamente, sotto forma di file (datafile). Un tablespace, formato da uno o più datafile, contiene vari tipi di segment; ogni segment a sua volta si suddivide in uno o più extent. Ogni extent comprende gruppi contigui di blocchi di dati (data block), quest'ultimi sono la più piccola informazione memorizzabile da Oracle. A livello fisico, i file comprendono almeno due o più extent. Fino alla versione 8i la

dimensione del blocco di dati era stabilita alla creazione del database e non poteva più essere modificata; dalla versione 9i in poi i blocchi di dati possono essere di dimensione variabile, sebbene ogni tablespace debba necessariamente essere costituita da datafile con la stessa dimensione di blocco dati.

Oracle tiene traccia dei dati memorizzati tramite l'aiuto di informazioni presenti nelle tabelle di sistema. Esse contengono il dizionario dei dati e se presenti indici e cluster. Un dizionario dati consiste di una collezione di tabelle che contengono informazioni riguardo tutti gli oggetti del database.

Se un amministratore della base di dati ha attivato la funzione RAC (Real Application Clusters), allora istanze multiple, solitamente su server differenti, si collegano ad una Storage Area Network o sistema similare i cui dischi sono visibili e utilizzabili da tutti i nodi del cluster. Questo scenario può offrire numerosi vantaggi, tra cui maggiori performance, scalabilità e ridondanza. Comunque, il supporto e la gestione diviene più complesso e molti siti evitano di usare RAC.

Tra le varie potenzialità possiamo memorizzare ed eseguire stored procedure e funzioni. Grazie al PL/SQL, un'estensione procedurale del linguaggio SQL, sviluppato da Oracle, e a Java possiamo scrivere funzioni, procedure, trigger e package.

Oracle è un RDBMS che se configurato e gestito in maniera appropriata, garantisce una sicurezza dei dati molto elevata. È possibile attivare a questo proposito la modalità detta ARCHIVING (o ARCHIVELOG MODE). Essa consiste nel registrare tutte le transazioni che avvengono nel DB anche in file di sistema operativo che dovranno essere utilizzati in caso di DB RECOVERY dovuta a crash totale o parziale del sistema. In questa modalità è possibile sfruttare l'HOT BACKUP ossia il salvataggio dei dati a sistema acceso senza effettuare fermi. Le modalità per il backup a caldo (hot backup) sono diverse. Quella standard oracle è denominata RMAN ossia Recovery Manager. Nulla vieta comunque all'amministratore del DB di gestire il backup/restore delle istanze oracle in maniera manuale o automatica tramite scripting.

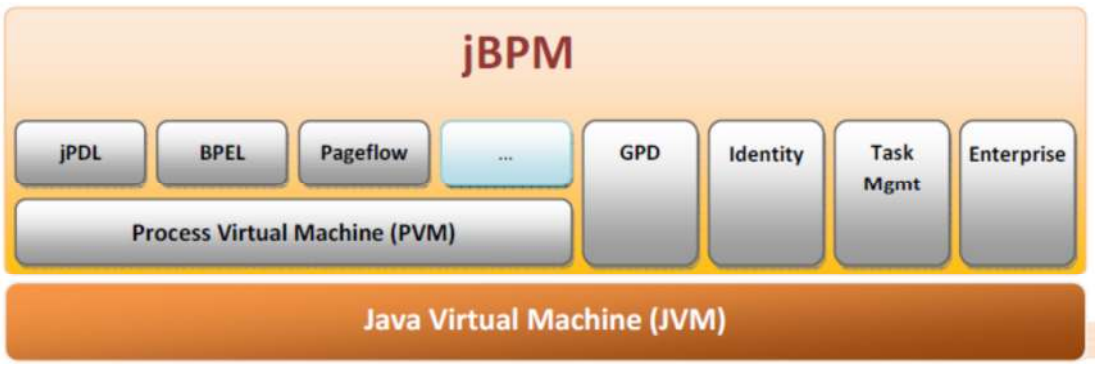
3.6 Business Process Management – jBPM

Il Business Process Management (BPM), cioè la gestione automatica dei processi di business, è la naturale evoluzione delle metodologie di progettazione del software. L'approccio BPM permette uno sviluppo agile dei processi di business:

- Riduce il numero di attori che devono interpretare i requisiti (lo sviluppatore entra in un secondo tempo) e di conseguenza aumenta la produttività.
- Aumenta la velocità di costruzione dei prototipi (integrazioni simulate con sistemi esterni).

jBPM [6], realizzato dalla Jboss, è una piattaforma open source, basata sulla tecnologia Java, per sviluppare e gestire processi applicati ai dati di business.

L'architettura di jBPM è strutturata su numerosi componenti appoggiati sulla Java Virtual Machine (JVM).



- La Process Virtual Machine (PVM) è il motore che interpreta ed esegue i processi, scritti in diversi linguaggi (jPDL, BPEL, ..).
- Il GPD è il designer. Un plugin Eclipse che permette di disegnare e deployare i processi.

- Identity è il componente in grado di gestire utenti e permessi, e si può associare ad una directory aziendale.
- Il task Management si occupa della gestione dei task, ad esempio per l'interazione temporizzata con l'utente. Il componente Enterprise permette l'accesso ai componenti dell'infrastruttura Java Enterprise (EJB, Data Source, Code, etc..).
- L'architettura jBPM fornisce un sistema ampiamente collaudato e un'architettura flessibile.
- Il Process Engine gira sul sistema interno e colloquia con il database contenente i dati di business. Si noti che è possibile configurare il sistema con ogni database noto.
- Il Process Designer è un applicazione esterna tramite la quale si sviluppano i processi e colloquia con il Process Engine per il deployment (distribuzione o la messa in produzione) dei processi
- L'integrazione con sistemi esterni è gestita dall'engine, eseguendo il codice prodotto dagli sviluppatori del processo. Tale codice è archiviato sul database insieme alle definizioni di processo.

3.6.1 La definizione del processo

Lo sviluppo di un processo coincide con la definizione grafica dello stesso, attraverso il Designer oppure manualmente con la stesura di un file in formato XML che rappresenta il flusso del processo.

```
<process-definition name="sample">
        <start-state name="start">
                <transition to="fork"/>
        </start-state>
        ...
        <fork name="fork">
                <transition to="state-one"/>
                <transition to="state-two"/>
        </fork>
        <state name="state-one">
                <transition to="join"/>
        </state>
        ...
        <end-state name="end"></end-state>
</process-definition>
```

Figura 13 Definizione di un processo JBPM

Al termine della costruzione del flusso di processo, questo può essere deployato sul server. Quest'operazione consiste nel salvataggio della definizione di processo (ProcessDefinition) su DataBase. In seguito al deploy la definizione del processo è nota alla piattaforma, è può essere utilizzata per avviare istanze concrete del processo (ProcessInstance).

Quando una definizione di processo è presente nel database è possibile richiamarla e richiedere all'engine la costruzione di un'istanza concreta del process. In questo modo qualsiasi applicazione client (J2SE, JEE, EJB) può operare su diverse istanze di processo, mentre all'engine è delegata la gestione della persistenza nelle diverse fasi.

3.7 JasperReports

JasperReports [7] è una valida alternativa, non solo all'ipotesi di realizzare ex novo il motore per la creazione di report professionali ma anche rispetto all'adozione di software commerciali (quali Crystal Reports), per integrare delle funzionalità di reportistica all'interno di un'applicazione Java.

JasperReports è una libreria Open Source scritta interamente in linguaggio Java che consente la generazione dinamica di report a partire da una fonte dati e la successiva renderizzazione in diversi formati, tra i quali PDF, HTML e XML.
Il processo di generazione di un report mediante Jasper Report consiste in tre step:

1. definizione della struttura e del layout del report sotto forma di file jrxml;
2. compilazione del file jrxml e generazione di un file jasper;
3. rendering del file jasper mediante la libreria Jasper Reports.

JasperReports prevede già delle API per la realizzazione di tutte le funzionalità tipiche di un software di reportistica:

- Formattazione di ogni elemento di un testo (font, allineamento, spaziatura, colore, etc.)
- Definizione di parti standard predefinite (header, footer, sommario, etc.)
- Gestione di raggruppamenti
- Valutazione di espressioni

- Campi calcolati
- Gestione di subreport
- Inserimento di immagini
- Stampa
- Salvataggio dell'output su file di diverso formato

Primo passo necessario è la creazione del report è la generazione del file **jrxml** (Jasper Report XML). Nel file jrxml vengono definite diverse informazioni, tra le quali la dimensione delle sezioni, gli elementi e la loro posizione specifica, i colori e la dimensione del carattere, le query SQL da eseguire per reperire le informazioni dalla base dati ed altro ancora. Scrivere manualmente tale file non è un'operazione semplice ed immediata. Per tale motivo si può utilizzare iReport, un tool gratuito che permette di disegnare la struttura ed il layout del report sfruttando le classiche tecniche del drag & drop.

A partire da tale layout, tramite JasperReports, andrà prodotto un file con estensione .jasper. Quest'ultimo è il semi compilato che verrà utilizzato dall'applicazione a runtime per la generazione del report vero e proprio, a partire da dati recuperati da un database.

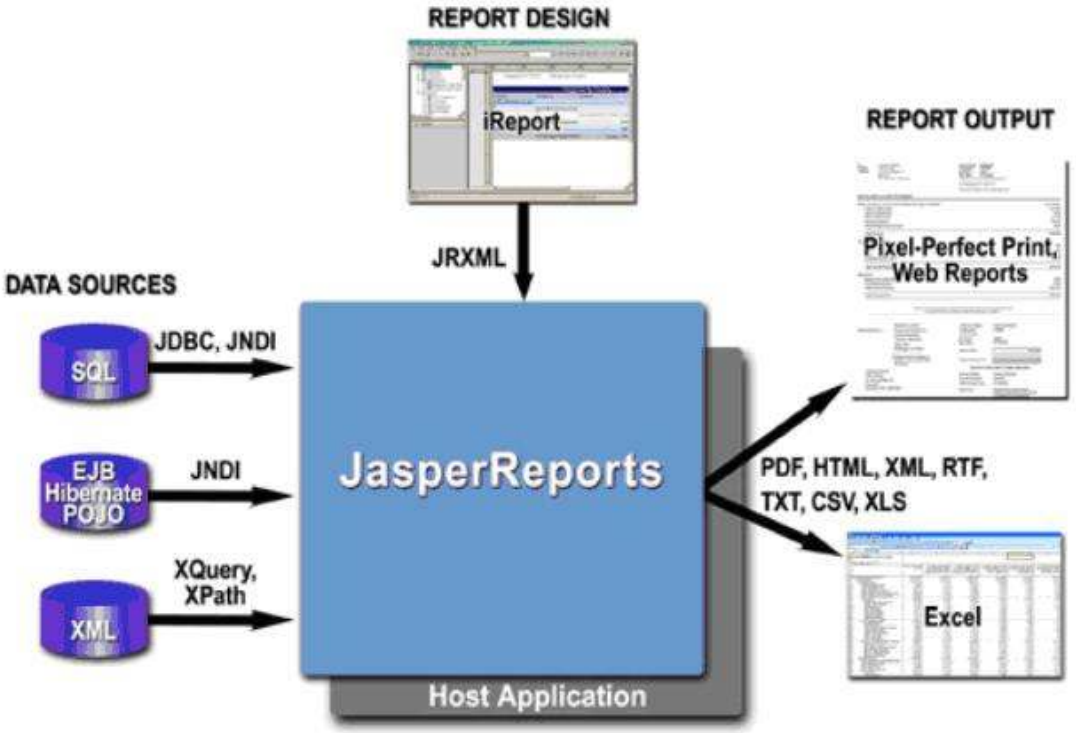


Figura 14 Schema funzionamento JasperReport

Ciascun report è costituito dalle seguenti sezioni:

- **Title**: contiene il titolo del report che viene stampato solo nella prima pagina;
- **Page Header**: contiene il titolo di ciascuna pagina del report;
- **Column Header**: contiene le intestazioni delle colonne riguardanti i dati che vogliamo visualizzare;
- **Detail**: contiene i dati estratti dal nostro database;
- **Column Footer**: contiene la parte che viene visualizzata al di sotto di ciascuna colonna;
- **Page Footer**: contiene tutto ciò che viene visualizzato alla fine di ogni pagina come ad esempio il numero della pagina;

- **Last Page Footer**: contiene ciò che viene visualizzato alla fine del report, solo nell'ultima pagina;
- **Summary**: contiene le informazioni che saranno visualizzate subito dopo l'ultima riga contenente i dati.

Nessuna di queste sezioni è obbligatoria. Possono liberamente essere utilizzate solo quelle adatte alle esigenze applicative.

Capitolo 4

Realizzazione

Dopo aver analizzato le tecnologie e gli strumenti utilizzati, saranno ora illustrate le scelte progettuali e i problemi incontrati durante lo sviluppo. In particolare sarà trattato, nel dettaglio, come la Modulistica Web gestisce le informazioni inserite dall'utente descrivendo le relazioni e i legami tra i diversi oggetti che si occupano di gestire la struttura di un modulo.

4.1 Compilazione Modulo

La compilazione di un modulo deve essere il più possibile simile alla compilazione cartacea, così che gli utenti non devono essere formati sull'utilizzo dell'applicazione e la stessa risulti molto facile e abbia un impatto minimo sulle abitudini delle persone.

Per questo si è pensato di comporre la Modulistica Web in modo che a video l'utente veda lo stesso identico modulo che prima compilava sulla carta o che riceveva precompilato via mail. In modalità di compilazione o semplice visualizzazione di un modulo, l'applicazione si presenta con una schermata molto semplice, dove le informazioni hanno il risalto maggiore occupando grande parte dello schermo, come evidenziato in Figura 15.

Figura 15 Compilazione di un modulo.

In alto, tramite una serie d'icone, sono presentate tutte le azioni che possono essere utilizzate dall'utente, come il salvataggio, la firma, la stampa in PDF e la navigazione tra le diverse pagine, con la possibilità di navigare pagina per pagina, oppure visualizzare un menù a comparsa che offre l'elenco di tutte le

pagine contenute nel modulo, permettendo all'utente di accedere direttamente a una determinata pagina, saltando le pagine non utili per l'utente stesso.

La parte centrale è interamente utilizzata per visualizzare il modulo e le informazioni in esso contenute. Per garantire al massimo la coerenza tra il modulo cartaceo e quello presentato a video, le immagini ottenute dal PDF sono utilizzate come sfondo della finestra e sopra sono posizionati i campi che verranno utilizzati dall'utente per compilare il modulo. In questo modo, all'utente è presentato lo stesso identico modulo che compilava sulla carta e in più, sfruttando tutte le funzionalità offerte dalla Modulistica Web, la compilazione è guidata passo passo riducendo al minimo i possibili errori.

4.2 Struttura Modulo

Il nucleo fondamentale dell'applicativo è sicuramente il modulo. Per poter gestire qualsiasi tipo di modulo, indipendentemente dal tipo e dalla quantità delle informazioni in esso contenuto, è di notevole importanza il modo in cui l'applicazione memorizza e gestisce tali informazioni. Deve essere sempre possibile ricostruire l'intera struttura del modulo nel minor tempo possibile.

Il problema della memorizzazione del modulo è stato separato e affrontato in due passi:

1. Definizione e memorizzazione della struttura del modulo
2. Compilazione e memorizzazione delle informazioni inserite nel modulo

La prima fase ha permesso di individuare la logica con cui il sistema memorizza tutte quelle informazioni che permettono di disegnare e controllare il modulo,

cioè tutte quelle informazioni necessarie all'applicativo per consentire all'utente una veloce e corretta compilazione del modulo.

La seconda fase invece ha individuato la logica con cui i dati inseriti dall'utente sono memorizzati e organizzati sulla base dati, per poi poter essere gestiti dalle diverse funzionalità dell'applicativo. In Figura 16 sono riportate tutte le tabelle e tutti i legami logici presenti tra di esse, che vengono sfruttate dal sistema per poter memorizzare i moduli compilati dagli utenti. La tabella MODULO è ovviamente la tabella principale cui sono collegate tutte le tabelle relative ai diversi campi. La frammentazione di queste tabelle si è resa necessaria per ottimizzare al meglio la memorizzazione della singola informazione, senza dover utilizzare più colonne diverse per tipo oppure procedure di conversione che avrebbero inserito dei ritardi nel gestire e recuperare le informazioni. La tabella VERSIONE è quella che si occupa di memorizzare tutti gli schemi dei moduli, mentre la tabella DEF_MODULO permette di creare e organizzare le diverse tipologie di modulo.

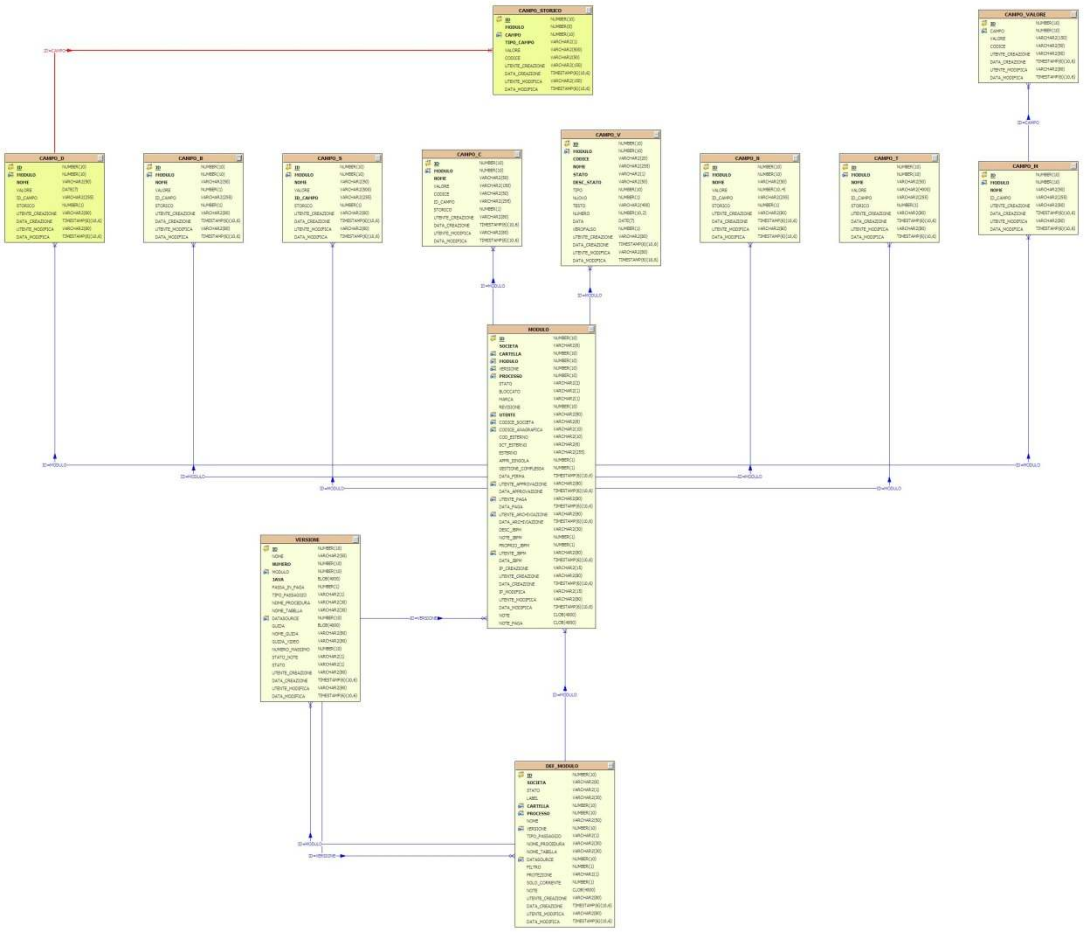


Figura 16 Schema E-R per la memorizzazione del modulo.

Conclusa la parte riguardante il database, la Figura 17 descrive tutti i legami logici tra le diverse classi utilizzate nella gestione dei moduli.

La parte alta dell'immagine descrive tutte le classi utilizzate per la memorizzazione della definizione. La classe **ModuloSchemaJava** è il punto iniziale dove sono memorizzate tutte quelle informazioni che caratterizzano il modulo a livello generale, come per esempio le dimensioni delle pagine, le azioni javascript da invocare in base alle azioni dell'utente, il template PDF e alcune opzioni aggiuntive da utilizzare durante la fase di stampa più molte altre caratteristiche che influenzano l'interazione dell'utente con il modulo.

Da questa classe è possibile accedere alla lista delle pagine dalle quali è possibile accedere alla lista degli oggetti in esse contenute. Gli oggetti contenuti all'interno delle pagine devono essere due classi **OggettoModulo** o **CampoDefinizione**. Questa differenza si è resa necessaria per separare tutti quelli oggetti puramente grafici, come le immagini, da oggetti che invece richiedono l'inserimento d'informazioni da parte dell'utente e che quindi richiedono controlli e metodi completamente diversi dai primi. La gerarchia completa dei componenti è visibile nella Figura 20 ed è descritta in modo esteso nel Capitolo 4.3.

Ogni singolo oggetto è identificato da un nome, che ovviamente è univoco a livello di modulo, tramite il quale è possibile ricercare e identificare il singolo campo all'interno del modulo. Inoltre, nel momento in cui un campo è creato e inserito nel modulo, è associato a esso un progressivo numerico che sarà utilizzato in fase di memorizzazione per distinguere l'informazione inserita dall'utente.

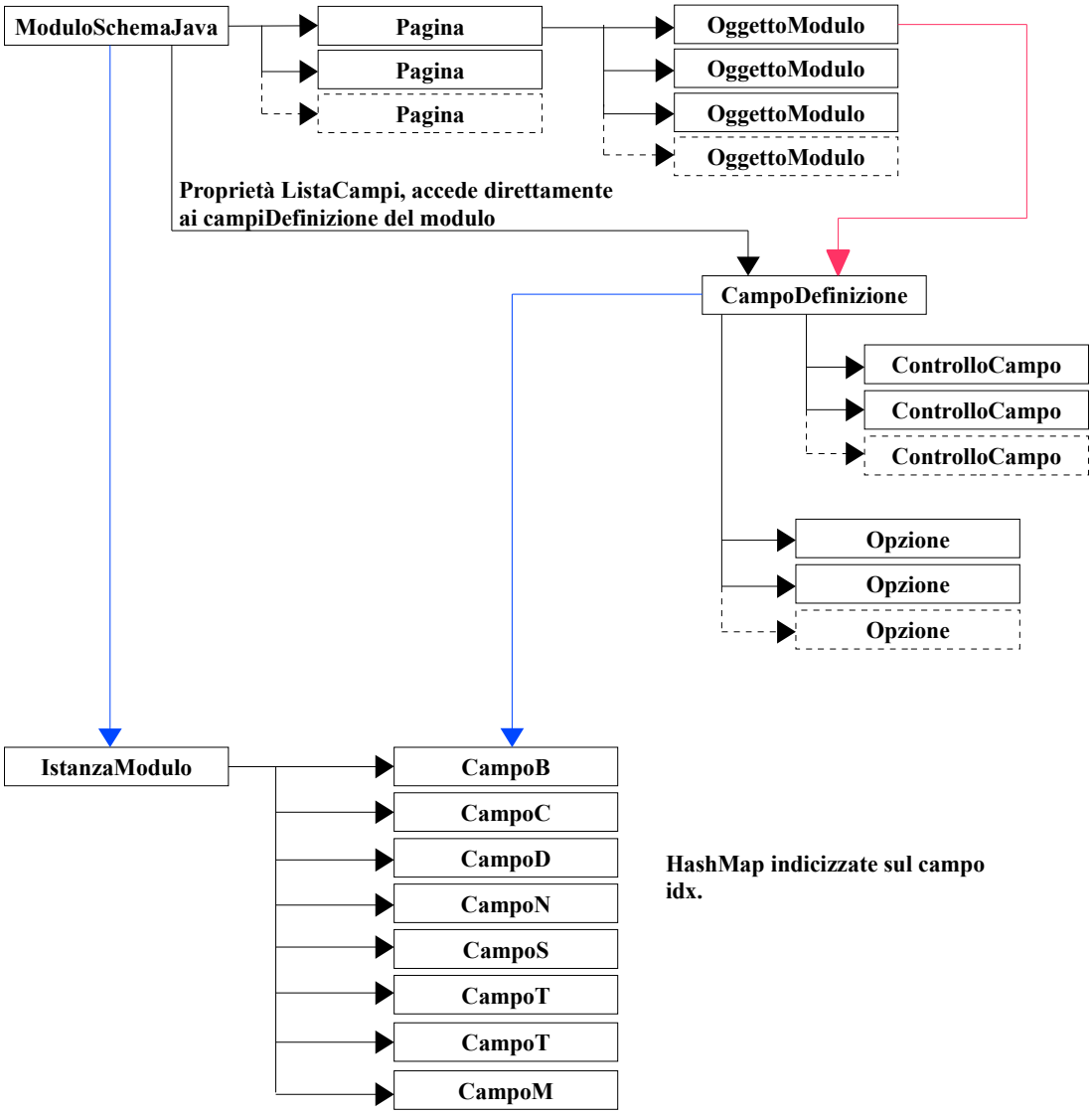


Figura 17 Schema Logico di un modulo.

Legenda:

_____ Legame logico (tramite lista o HashMap)

_____ Estende la classe

_____ Genera la classe Hibernate che memorizza l'oggetto sul DB

La sezione inferiore dell'immagine, descrive tutte le logiche e i legami tra le classi predisposte alla memorizzazione del modulo sulla base dati. Sono tutte classi mappate da Hibernate e che saranno trasferite una ad una sul DB. La

classe **IstanzaModulo** rappresenta la testata del modulo e si occupa di memorizzare le informazioni generiche di un modulo come il tipo, l'utente che lo compila, la data di compilazione e di firma, più altre informazioni aggiuntive. Da essa tramite l'accesso a delle HashMap indicizzate su un progressivo numerico è possibile accedere all'informazione di ogni singolo campo. Con questa logica l'informazione di un modulo è spacchettata su più tabelle distinte, esattamente otto, dove il fattore discriminante è il tipo d'informazione trattenuta dal campo.

La scelta di spacchettare l'informazione e ricostruirla in fase di lettura, permette di avere la massima libertà a livello di struttura del modulo, senza alcun limite a livello di formato e numero di campi.

L'utilizzo delle HashMap invece che ArrayList dipende dalla diversa efficienza nel ricercare un determinato elemento non conoscendo a priori la posizione dello stesso all'interno della collezione. Negli ArrayList, infatti, tale procedura raggiunge una complessità O(N) che può essere ridotta con ordinamenti e ricerche binarie a $O(\log_2(N))$, mentre con le HashMap la ricerca di un determinato elemento costa O(1).

4.3 Editor Visuale

La funzionalità di pubblicazione deve integrare nel miglior modo due requisiti fondamentali:

- **Facilità d'uso:** lo strumento deve essere il più semplice possibile, garantendo una curva di apprendimento molto bassa.
- **Velocità e flessibilità:** passare alla Modulistica Web deve essere quasi trasparente all'azienda e il sistema deve poter gestire qualsiasi tipo di modulo.

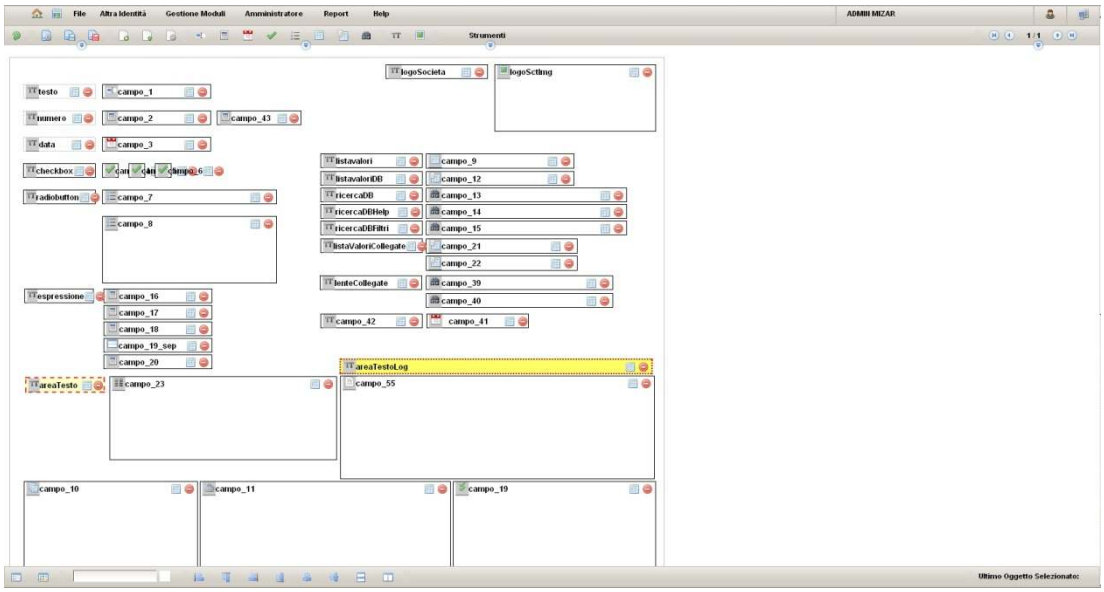


Figura 18 Editor Visuale utilizzato per creare i moduli.

Per garantire la facilità d'uso si è pensato di creare un ambiente web completamente configurabile tramite l'utilizzo di tecniche ormai comuni, come il drag&drop, copia e incolla, ecc... La funzionalità di editing di un modulo, offre la piena libertà all'utente, sulla posizione di un campo, sulle sue caratteristiche e sulle sue proprietà permettendo così di configurare qualsiasi modulo già

esistente, partendo da un formato digitale già esistente. La prima fase della configurazione definisce l'aspetto di un modulo, dichiarandone le dimensioni, il numero di pagine e lo sfondo delle stesse. Applicando un'immagine di sfondo alle pagine, si riduce notevolmente il lavoro d'impostazione di un modulo potendo così sfruttare i moduli già esistenti sia cartacei sia digitali (PDF e JPEG).

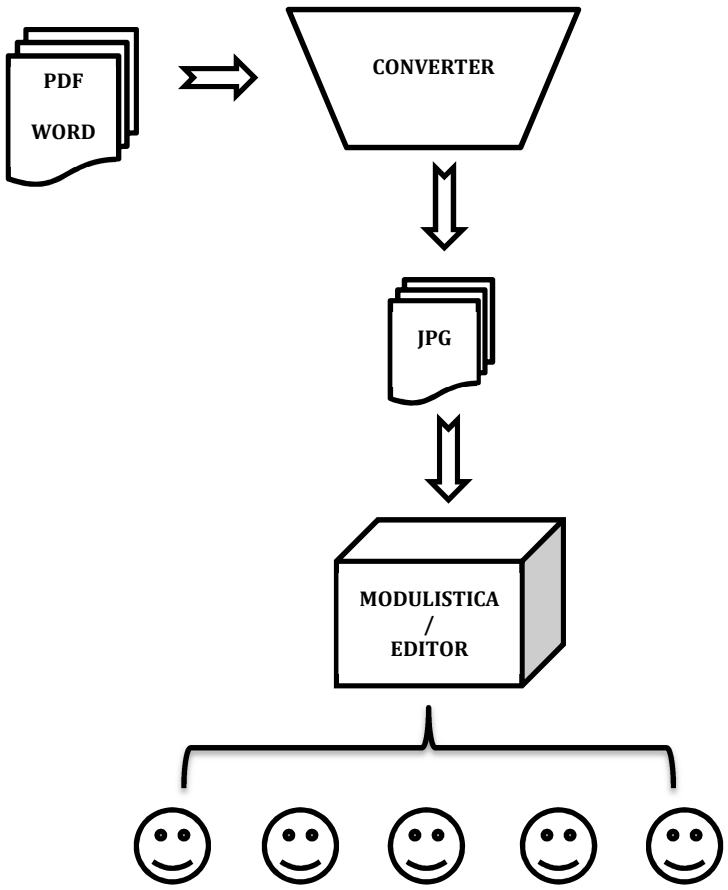


Figura 19 Processo di generazione di un modulo partendo dal PDF.

L'applicazione integra alcune funzionalità di elaborazione PDF che permettono di generare tutte le risorse necessarie da utilizzare poi durante la configurazione e compilazione, così come illustrato dalla Figura 19. Analizzato il PDF all'utente è richiesto solo di posizionare e dimensionare i diversi campi sopra di esso così che poi gli utenti possano compilare il modulo inserendo le proprie informazioni. La maggior parte delle azioni principali, necessarie per

configurare un modulo, sono completamente guidate, in modo da non richiedere particolari conoscenze informatiche o la conoscenza di un particolare linguaggio di programmazione. Visto però la necessità di poter gestire qualsiasi modulo, l'editor della modulistica integra alcune utility che permettono di configurare l'ambiente in ogni singolo dettaglio, offrendo così la massima libertà all'utente. Ovviamente per poter controllare al meglio queste utility è necessaria una buona conoscenza informatica ed eventualmente la conoscenza di alcuni linguaggi di programmazione. La validazione del modulo, per esempio, può essere effettuata tramite un motore interno, interamente configurabile dall'interfaccia web dell'editor, ma può essere tranquillamente estesa integrando nel sistema alcuni script con cui la modulistica interagirà per poter fornire all'utente le informazioni necessarie. La scrittura di questi script richiede ovviamente conoscenze informatiche approfondite e la conoscenza di alcuni linguaggi di programmazione come JavaScript e Python. In questo modo un modulo è l'insieme di più risorse:

- **PDF:** definisce il template del modulo
- **Schema:** indica la posizione, la dimensione e le caratteristiche dei campi
- **Script:** permettono di eseguire operazioni complesse che esulano dal comportamento standard dell'applicativo

4.4 Gerarchia dei Componenti

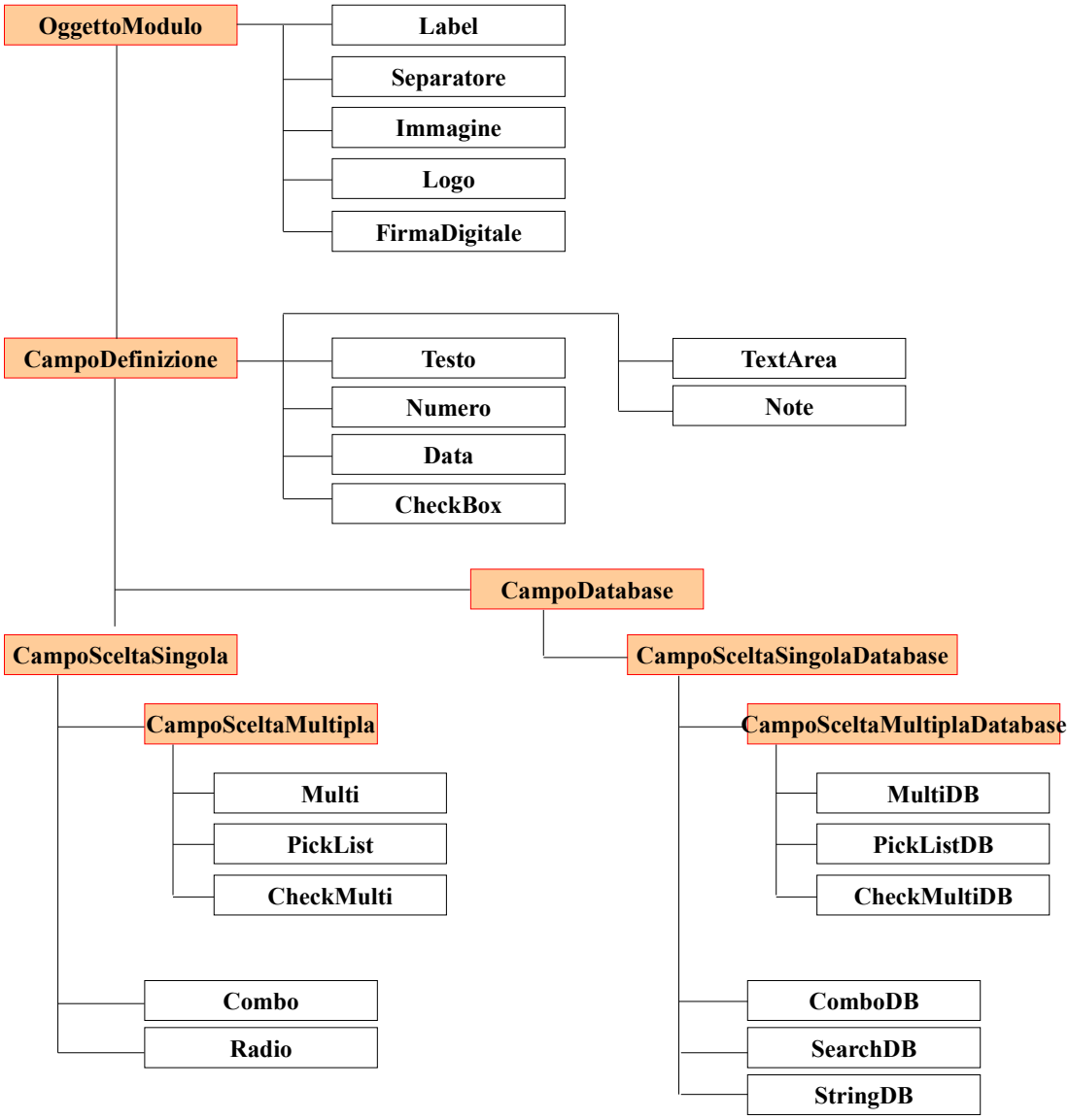


Figura 20 Gerarchia dei componenti.

In Figura 20 è illustrato l'intero albero gerarchico dei componenti, evidenziando così le dipendenze tra di essi. La scelta di mantenere un legame gerarchico tra i componenti, deriva da uno dei paradigmi della programmazione ad oggetti: l'ereditarietà. Il meccanismo dell'ereditarietà permette di derivare nuove classi a partire da quelle già definite. Una classe derivata, o *sottoclasse*, mantiene i

metodi e gli attributi delle classi da cui deriva (*classi base*, o *superclassi*); inoltre, può definire i propri metodi o attributi, e può ridefinire il codice eseguibile di alcuni dei metodi ereditati tramite un meccanismo chiamato overriding.

L'ereditarietà può essere usata come meccanismo per ottenere l'estensibilità e il riuso del codice, e risulta particolarmente vantaggiosa quando viene usata per definire sottotipi, sfruttando le relazioni *is-a* esistenti nella realtà di cui la struttura delle classi è una modellizzazione. Oltre all'evidente riuso del codice della superclasse, l'ereditarietà permette la definizione di codice generico attraverso il meccanismo del polimorfismo.

Tramite l'ereditarietà è quindi possibile aggiungere nuovi componenti, sfruttando quelli già esistenti, senza dover intervenire nel codice dell'applicazione.

4.4.1 Testo

Figura 21 Casella di Testo.

Permette all'utente di inserire un testo libero. Ha una dimensione massima che non può superare i 150 caratteri. È possibile impostare il campo in modo che sia automaticamente compilato dal sistema con valori appartenenti all'utente collegato, come il cognome, il nome, residenza, ecc...

Proprietà	Descrizione
Lunghezza	Numero massimo di caratteri consentito nel campo.
Espressione	Espressione che permette di validare il campo.
Controlla se compilato	Indica al sistema se l'espressione inserita vale solo se il campo è compilato.
Testo Predefinito	Indica il testo con cui inizializzare il campo.

Figura 22 Caratteristiche campo testo.

4.4.2 Area di testo

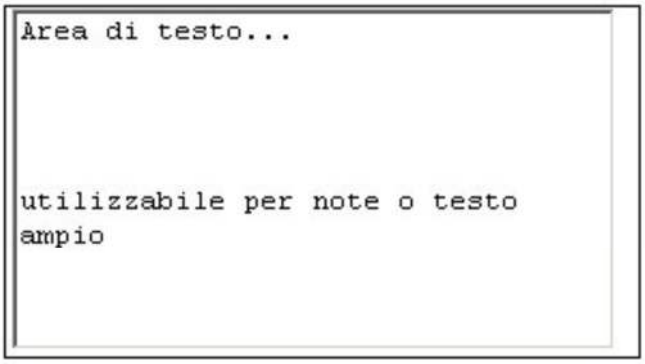


Figura 23 Area di testo.

Permette all'utente di inserire un testo libero, più ampio rispetto a quello inseribile nella singola casella di testo. Ha una dimensione massima di 4000 caratteri.

4.4.3 Note

Permette all'utente di inserire un testo libero, sotto forma di log. Il sistema annoterà l'ora e l'utente che scriverà la nota. Selezionando la corretta icona all'interno del componente, verrà mostrando l'intera cronologia del campo.

Figura 24 Campo Note.

4.4.4 Casella Numerica

Figura 25 Casella di tipo numerico.

Permette di definire un campo numerico. Può essere specificato un valore massimo e \ o minimo che deve essere rispettato in fase di compilazione.

È possibile impostare alcune regole di auto compilazione, sfruttando le informazioni dell'utente collegato.

Proprietà	Descrizione
Massimo	Valore massimo consentito dal campo
Minimo	Valore minimo consentito dal campo
Valore predefinito	Indica il numero con cui inizializzare il campo.
Separa decimali	Indica se utilizzare il separatore dei decimale nel formattare il campo.
Decimali	Indica le cifre minime obbligatorie per formattare la parte decimale del numero.
Virgola	Indica il numero delle cifre da utilizzare per rappresentare la parte dopo la virgola.

Figura 26 Caratteristiche campo numerico.

4.4.5 Casella Data

Figura 27 Campo Data.

Permette di definire un campo Data. Può essere specificata una data massima e una minima che deve essere rispettata in fase di compilazione, così come una data predefinita. Il controllo prevede anche la possibilità di specificare la data tramite un calendario grafico, che permette di navigare tra i mesi e gli anni in modo che possa essere scelta la data voluta in modo semplice e veloce.

È possibile impostare alcune regole di auto compilazione, sfruttando le informazioni dell'utente collegato.

Proprietà	Descrizione
Massimo	Indica la data massima consentita nel campo.
Minimo	Indica la data minima consentita nel campo.
Predefinita	Indica la data con cui inizializzare il campo. Può essere specificata manualmente con una data fissa, oppure utilizzare dei valori previsti dal sistema, come il giorno corrente, mese corrente, ecc.
Formato	Indica il formato in cui la data sarà stampata a video e nei PDF.

Figura 28 Caratteristiche Campo Data.

4.4.6 Casella si\no

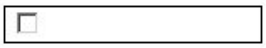


Figura 29 Casella si\no.

Permette di definire un campo di scelta booleana, cioè un campo in cui l'utente può eseguire una scelta di tipo vero o falso. A video è visualizzata come una checkbox che potrà essere selezionata o meno dall'utente. È possibile impostare una *label* che sarà visualizzata al fianco della casella come descrizione e definire anche la scelta di default del campo. È possibile cioè impostare se la casella deve essere inizialmente selezionata oppure no.

Proprietà	Descrizione
selezionato	Indica al sistema se il campo deve essere selezionato di default o meno.
testo	Indica il testo che sarà visualizzato a fianco della casella.

Figura 30 Caratteristiche campo si \ no.

4.4.7 Casella si\no multipla

Figura 31 Casella si \ no multipla.

Permette di definire una serie di opzioni, che verranno visualizzate sotto forma di checkbox. È possibile impostare il componente per accettare una singola opzione, oppure accettare più di una opzione.

Proprietà	Descrizione
Direzione	Indica la direzione in cui saranno visualizzate le singole opzioni. A scelta tra orizzontale e verticale
Opzione default	Indica l'opzione che verrà selezionata in modo automatico dal sistema. Solo per opzioni definite dall'utente
Numero massimo di Opzioni	Indica il numero massimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno
Numero Minimo di Opzioni	Indica il numero minimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno
Vincola Singola Opzione	Indica che l'utente può selezionare una singola opzione. Il vincolo è bloccante.

Figura 32 Caratteristiche casella si \ no multipla.

4.4.8 Casella si\no multipla (DataBase)

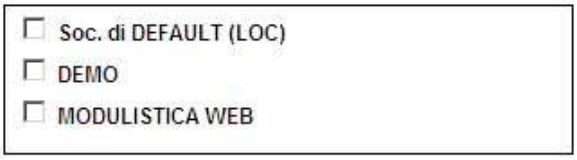


Figura 33 Casella si \ no multipla (DataBase).

Identico come visualizzazione al controllo precedente, con la differenza che l'elenco delle opzioni viene definito tramite una query SQL. Il sistema permette di inserire una query SQL qualsiasi e di legare il campo codice e il campo descrizione delle varie opzioni, alle colonne restituite dalla query. **Per utilizzare questo tipo di componente è necessario definire almeno un datasource.**

Proprietà	Descrizione
direzione	Indica la direzione in cui verranno visualizzate le singole opzioni. A scelta tra orizzontale e verticale
Opzione default	Indica l'opzione che verrà selezionata in modo automatico dal sistema. Solo per opzioni definite dall'utente
Numero massimo di Opzioni	Indica il numero massimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno.
Numero Minimo di Opzioni	Indica il numero minimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno.
Vincola Singola Opzione	Indica che l'utente può selezionare una singola opzione. Il vincolo è bloccante.

Figura 34 Opzioni casella si \ no multipla (DataBase)

4.4.9 RadioButton

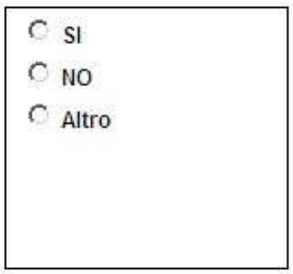


Figura 35 RadioButton

Permette di definire un RadioButton. Il numero di opzioni non è fisso, ed è definibile tramite l'editor. Un'opzione è caratterizzata da un codice univoco e da una label descrittiva.

Proprietà	Descrizione
direzione	Indica la direzione in cui saranno visualizzate le singole opzioni. A scelta tra orizzontale e verticale
Opzione default	Indica l'opzione che verrà selezionata in modo automatico dal sistema. Solo per opzioni definite dall'utente

Figura 36 Caratteristiche RadioButton.

4.4.10 ComboBox

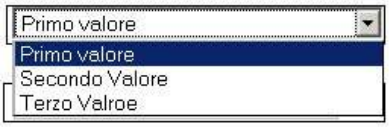


Figura 37 ComboBox

Tramite questa casella è possibile scegliere un'opzione tra una lista precaricata dal sistema. Il numero di opzioni non è fisso, ed è definibile tramite l'editor. Un'opzione è caratterizzata da un codice univoco e da una descrizione.

Proprietà	Descrizione
Inizializza a vuoto	Indica se il primo valore della combo deve essere un valore vuoto
Opzione default	Indica l'opzione che verrà selezionata in modo automatico dal sistema. Solo per opzioni definite dall'utente

Figura 38 Caratteristiche Campo di Scelta.

4.4.11 ComboBox (DataBase)

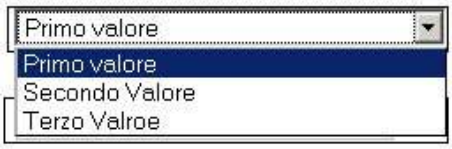


Figura 39 ComboBox (DataBase)

Identico come visualizzazione al controllo precedente, con la differenza che l'elenco delle opzioni viene definito tramite una query SQL. Il sistema permette di inserire una query SQL qualsiasi e di legare il campo codice e il campo descrizione delle varie opzioni, alle colonne restituite dalla query. **Per utilizzare questo tipo di componente è necessario definire almeno un datasource.**

Proprietà	Descrizione
query	Indica la query che verrà eseguita per generare tutte le opzioni
datasource	Indica il datasource su cui verrà eseguita la query
Colonna riferimento	Indica la colonna che verrà utilizzata per la descrizione delle opzioni
Colonna codice	Indica la colonna che verrà utilizzata per il codice delle opzioni
Aggiorna i campi	Permette di indicare la lista di campi che dovranno essere aggiornati ogni volta che viene selezionato un valore

Figura 40 Caratteristiche Campo di Scelta da DB.

4.4.12 Casella di scelta multipla

Figura 41 Casella di scelta Multipla.

Tramite questa casella è possibile scegliere una serie di opzioni tra una lista precaricata dal sistema. Il numero di opzioni non è fisso, ed è definibile tramite il propertyEditor. Un'opzione è caratterizzata da un codice univoco e da una descrizione.

Il codice è il valore utilizzato dal sistema per il passaggio in paga del campo, cioè è il valore che viene trasferito dal sistema verso il sistema di paghe.

Proprietà	Descrizione
Opzione default	Indica l'opzione che verrà selezionata in modo automatico dal sistema. Solo per opzioni definite dall'utente
Numero massimo di Opzioni	Indica il numero massimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno
Numero Minimo di Opzioni	Indica il numero minimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno
Vincola Singola Opzione	Indica che l'utente può selezionare una singola opzione. Il vincolo è bloccante

Figura 42 Opzioni Campo di Scelta Multipla.

4.4.13 Casella di scelta multipla (DataBase)

Figura 43 Casella scelta multipla (DataBase)

Identico come visualizzazione al controllo precedente, con la differenza che l'elenco delle opzioni viene definito tramite una query SQL. **Per utilizzare questo tipo di componente è necessario definire almeno un datasource.**

Proprietà	Descrizione
query	Indica la query che verrà eseguita per generare tutte le opzioni
datasource	Indica il datasource su cui verrà eseguita la query
Colonna riferimento	Indica la colonna che verrà utilizzata per la descrizione delle opzioni
Colonna codice	Indica la colonna che verrà utilizzata per il codice delle opzioni
Aggiorna i campi	Permette di indicare la lista di campi che dovranno essere aggiornati ogni volta che viene selezionato un valore
Numero massimo di Opzioni	Indica il numero massimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno
Numero Minimo di Opzioni	Indica il numero minimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno
Vincola Singola Opzione	Indica il numero massimo di opzioni che l'utente può selezionare. Il vincolo è bloccante

Figura 44 Caratteristiche Campo di Scelta Multipla da DB

4.4.14 Casella di scelta con filtri (DataBase)

Figura 45 Casella di scelta con filtri (DataBase)

È un campo complesso. Presenta una casella di testo e una icona che permette di accedere ad un'area di scelta delle informazioni. La casella di testo può essere libera e utilizzata per inserire un testo qualsiasi, o il campo può essere impostato solo per accettare valori contenuti nel database. **Per utilizzare questo tipo di componente è necessario definire almeno un datasource.**

La fase di filtro e scelta visuale del valore, permette di visionare sotto forma di tabella tutte le informazioni estratte dalla query SQL. Una volta individuato il valore, basta semplicemente eseguire la selezione tramite l'apposita icona.

Proprietà	Descrizione
query	Indica la query che verrà eseguita per generare tutte le opzioni
datasource	Indica il datasource su cui verrà eseguita la query
Colonna riferimento	Indica la colonna che verrà utilizzata per la descrizione delle opzioni
Colonna codice	Indica la colonna che verrà utilizzata per il codice delle opzioni
Accetta solo valori DB	Indica se il campo può accettare qualsiasi valore o solo valori contenuti nel database. Se selezionato, in fase di validazione verrà eseguita una query per controllare il valore inserito dall'utente
Abilita Suggeritore	Permette di abilitare un suggeritore per il campo di testo. L'utente inserendo solo una parte della parole verrà aiutato dal sistema, con alcune opzioni possibili
Abilita Ricerca	Permette all'utente di eseguire una ricerca \ selezione visuale del valore scelto
Abilita Filtri	Permette all'utente di filtrare i dati, specificando i filtri voluti
Aggiorna i campi	Permette di indicare la lista di campi che dovranno essere aggiornati ogni volta che viene selezionato un valore

Figura 46 Caratteristiche Campo Ricerca da DB.

4.4.15 PickList

Figura 47 PickList

Tramite questa casella è possibile scegliere una serie di opzioni tra una lista precaricata dal sistema. Il numero di opzioni non è fisso, ed è definibile tramite l'editor. Un'opzione è caratterizzata da un codice univoco e da una descrizione.

Proprietà	Descrizione
Opzione default	Indica l'opzione che verrà selezionata in modo automatico dal sistema. Solo per opzioni definite dall'utente
Numero massimo di Opzioni	Indica il numero massimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno.
Numero Minimo di Opzioni	Indica il numero minimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno.
Vincola Singola Opzione	Indica che l'utente può selezionare una singola opzione. Il vincolo è bloccante.
Visualizza Controlli Semplici	Visualizza i controlli per la copia e la rimozione della singola opzione.
Visualizza Controlli Multipli	Visualizza i controlli per la copia e la rimozione di tutte le opzione.
Visualizza Label Controlli	Utilizzare la label sui controlli
Copia	Label utilizzata sul pulsante di copia per la singola opzione.
Copia Tutti	Label utilizzata sul pulsante di copia per tutte le opzioni.
Rimuovi	Label utilizzata sul pulsante di rimozione per la singola opzione
Rimuovi Tutti	Label utilizzata sul pulsante di rimozione per tutte le opzioni..

Figura 48 Caratteristiche PickList

4.4.16 PickList (DataBase)

Figura 49 PickList (DataBase)

Identico come visualizzazione al controllo precedente, con la differenza che l'elenco delle opzioni viene definito tramite una query SQL. **Per utilizzare questo tipo di componente è necessario definire almeno un datasource.**

Proprietà	Descrizione
query	Indica la query che verrà eseguita per generare tutte le opzioni
datasource	Indica il datasource su cui verrà eseguita la query
Colonna riferimento	Indica la colonna che verrà utilizzata per la descrizione delle opzioni
Colonna codice	Indica la colonna che verrà utilizzata per il codice delle opzioni
Aggiorna i campi	Permette di indicare la lista di campi che dovranno essere aggiornati ogni volta che viene selezionato un valore
Numero massimo di Opzioni	Indica il numero massimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno
Numero Minimo di Opzioni	Indica il numero minimo di opzioni che l'utente può selezionare. Il vincolo può essere bloccante o meno
Vincola Singola Opzione	Indica che l'utente può selezionare una singola opzione. Il vincolo è bloccante
Visualizza Controlli Semplici	Visualizza i controlli per la copia e la rimozione della singola opzione
Visualizza Controlli Multipli	Visualizza i controlli per la copia e la rimozione di tutte le opzione
Visualizza Label Controlli	Utilizzare la label sui controlli
Copia	Label utilizzata sul pulsante di copia per la singola opzione
Copia Tutti	Label utilizzata sul pulsante di copia per tutte le opzioni
Rimuovi	Label utilizzata sul pulsante di rimozione per la singola opzione
Rimuovi Tutti	Label utilizzata sul pulsante di rimozione per tutte le opzioni

Figura 50 Caratteristiche Campo PickList da DB.

4.4.17 Testo Fisso (DataBase)

Permette di visualizzare a video un testo fisso costruito tramite la concatenazione delle colonne di una query SQL. **Per utilizzare questo tipo di componente è necessario definire almeno un datasource.**

Proprietà	Descrizione
query	Indica la query che verrà eseguita per generare tutte le opzioni
datasource	Indica il datasource su cui verrà eseguita la query
espressione	Indica la sequenza dei campi che dovranno essere utilizzati per costruire l'espressione da visualizzare a video. Le colonne della query devono essere racchiuse tra **#**, es. **#NOME#** riporterà il valore della colonna NOME.
Colonna codice	Indica la colonna della query che verrà utilizzata per il passaggio in paga
descrizione	Indica il testo che verrà visualizzato come popup, nel momento in cui l'utente passerà il cursore del mouse sull'oggetto
testo predefinito	Indica il testo che verrà visualizzato nel caso in cui la query non restituisce alcun risultato

Figura 51 Opzioni Testo Fisso (DataBase)

4.4.18 Testo fisso

Permette di inserire un testo fisso all'interno del documento. Il testo sarà visualizzato all'interno dell'area indicata nell'editor. Potrà quindi essere visualizzato su più righe, oppure su singola riga. Il testo inserito non sarà modificabile in visualizzazione.

4.4.19 Logo Società

Permette di riportare il logo della società cui appartiene il dipendente che dovrà compilare il modulo stesso. Ovviamente non sarà modificabile in alcun modo.

4.4.20 Immagine

Permette di inserire un'immagine fissa all'interno del modulo.

4.4.21 Separatore

Disegnerà un separatore grafico nel modulo. Utile se bisogna separare graficamente due distinte aree di una pagina.

4.4.22 Firma Digitale

Permette di definire l'area in cui sarà inserito nel PDF il campo per la firma digitale. Questo è un campo invisibile per l'utente fino alla creazione del PDF e vale appunto solo per il PDF generato dal documento.

Proprietà	Descrizione
Invia il pdf	Permette di definire il link a cui il pdf dovrà essere inviato. Se selezionato, sarà inserito nel pdf un pulsante che permetterà l'inoltro del **PDF** all'indirizzo specificato. Se l'indirizzo è vuoto, verrà utilizzato l'indirizzo di default.
Rispetta le condizioni seguenti	Permette di indicare le condizioni che regolano l'inserimento della firma digitale nel **PDF.**
Firma Privata	La firma digitale sarà inserita nel **PDF** solo se è il proprietario a richiedere il PDF al sistema.
Solo dopo Firma Modulo	La firma digitale sarà inserita nel **PDF** solo se il modulo è stato firmato dall'utente. (Non sarà inserita quindi nello stato di Bozza)
Campo Nascosto	Permette di indicare dei valori che saranno aggiunti al **PDF** sotto forma di campi nascosti. Questi campi sono caratterizzati da un nome e da un valore. È possibile far in modo che il valore dei campi venga letto dall'anagrafica dell'utente
Condizioni Firma Digitale	Permette di indicare, in base all'anagrafica dell'utente del modulo, quando la firma digitale deve essere inserita o meno.

Figura 52 Opzione Firma Digitale

4.5 Profilazione degli utenti

Essendo l'applicazione completamente web e visto la natura delle informazioni trattate, l'accesso alla stessa è subordinato a una procedura di autenticazione, in modo da garantire la massima sicurezza dei dati inseriti dall'utente. La procedura di autenticazione può essere configurata per interagire con i diversi sistemi aziendali come Active Directory, LDAP. È possibile anche attivare una procedura di accesso automatico tramite Single Sign-On per permettere l'accesso a utenti già riconosciuti all'interno della rete aziendale.

A ogni utente è assegnato un profilo, che descrive nel dettaglio tutte le funzionalità e le procedure richiamabili dall'utente stesso. In questo modo è possibile personalizzare il comportamento dell'applicazione sulla base della tipologia d'utente. Un profilo utente caratterizza:

- **Menu:** definisce tutte le voci di menù disponibili. Le diverse voci di menu permettono all'utente di richiamare le diverse funzionalità e di poter così accedere alle diverse sezioni dell'applicativo. Se una determinata funzionalità non è presente nel proprio menù, l'utente non potrà in alcun modo richiamare tale funzione.
- **Compilazione:** i moduli disponibili all'utente dipendono strettamente dal proprio profilo. In questo modo è possibile distribuire i moduli basandosi sull'organizzazione interna dell'azienda. Definiti i profili, è possibile scendere ancor più nel dettaglio, distribuendo i moduli sulla base delle informazioni anagrafiche di ogni singolo utente. Questo

permette di combinare sia l'organizzazione gerarchica (profili utente) sia l'organizzazione geografica dell'azienda (stabilimento, sede, reparto)

- **Gestione:** i moduli che possono essere gestiti dall'utente, così come quelli compilabili, sono dipendenti dal profilo. Così come la compilazione, anche la gestione può essere distribuita basandosi sull'organizzazione interna dell'azienda.
- **Report:** tutta la reportistica e quindi i report disponibili all'utente sono collegati al profilo. Anche i profili, così come i moduli saranno quindi soggetti alla distribuzione condizionata sui profili utente.

4.6 Processo di Approvazione

Il processo di approvazione standard, prevede che il modulo possa essere firmato dall'utente e in seguito approvato da un ufficio o da un altro utente che eventualmente può anche passare il modulo in paga.

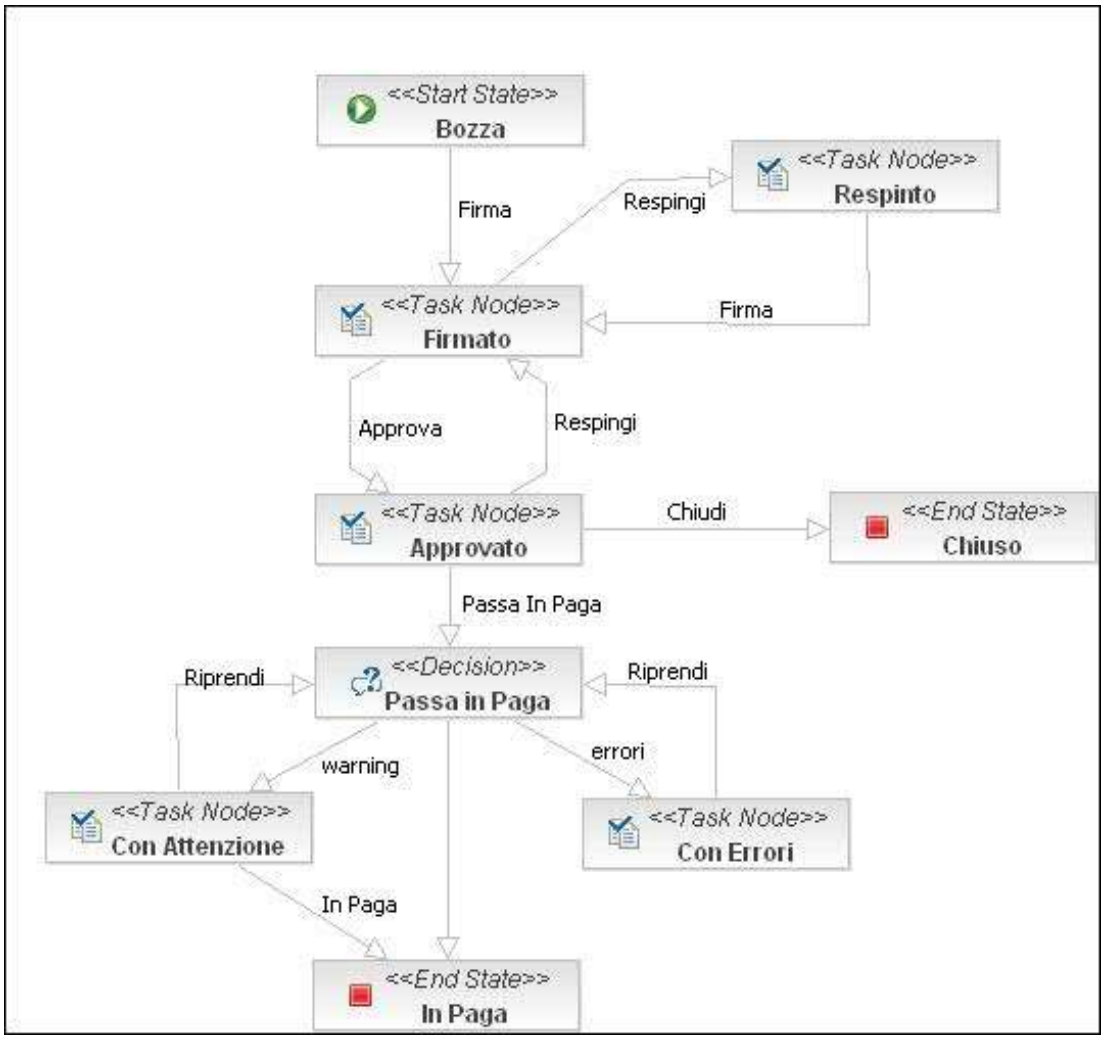


Figura 53 Processo Standard.

Se il modulo necessita un passaggio in paga sarà abilitato lo stato di approvazione e successivamente di passaggio in paga, altrimenti il modulo passerà automaticamente nello stato di **Chiuso**.

È possibile personalizzare i singoli stati, in modo da eseguire delle azioni di validazione prima d'entrare in uno stato, e delle azioni successive allo stato stesso. Le azioni **PRE-stato** sono azioni bloccanti, cioè se è sollevato un errore o qualche anomalia il sistema non permette al modulo di passare nello stato voluto, fino a quando le anomalie non saranno corrette. Queste azioni \ controlli, sono svolte tramite degli script che ricevono in ingresso tutti i campi del modulo e possono quindi validare il contenuto del modulo stesso.

Con passaggio in paga di un modulo s'intende la generazione di un flusso d'informazioni dalla Modulistica Web verso qualsiasi altro sistema aziendale. Questo passaggio d'informazioni, può essere automatizzato tramite il richiamo di una procedura personalizzata che si occuperà di prelevare i dati dalla Modulistica e di travasarli nel sistema aziendale di destinazione. Il comportamento standard dell'applicazione è di generare un tracciato ben definito contenente tutte le informazioni del modulo. Questo tracciato sarà poi analizzato dalle procedure o da qualsiasi sistema esterno.

Il motore di workflow jBPM, completamente integrato all'interno della Modulistica Web permette di definire nel dettaglio un qualsiasi processo aziendale, rendendo così l'applicazione completamente configurabile in base alle necessità dell'azienda.

4.7 Compilazione condivisa

L'applicazione oltre a permettere la compilazione dei moduli dai singoli utenti, e quindi automatizzare e regolare la comunicazione utente -> ufficio personale, è stata strutturata e pensata per gestire la compilazione condivisa, da parte di più soggetti, dello stesso modulo. La Modulistica Web, infatti, permette di configurare nel dettaglio i permessi di visibilità e scrittura sul singolo campo, in funzione del ruolo dell'utente che accede all'applicativo e allo stato corrente del modulo. Configurando opportunamente tale funzionalità è quindi possibile guidare passo passo l'informazione contenuta nel modulo, garantendo la sicurezza e la privacy dell'informazione stessa. Spesso per ottenere lo stesso risultato si utilizzano strumenti non adatti, come documenti condivisi dove l'informazione non è assolutamente controllata e completamente libera: tutti vedono tutto e possono modificare tutto, indipendentemente dal loro ruolo gerarchico o funzionale. Inoltre si utilizzano spesso canali non efficienti e ridondanti, come l'email, che non fanno altro che generare confusione nelle figure coinvolte e aumentare la disorganizzazione del flusso informativo.

Integrando tale procedura all'interno della Modulistica Web è possibile ridurre notevolmente la ridondanza delle comunicazioni e aumentare esponenzialmente l'organizzazione delle stesse perché è utilizzato un singolo strumento e un unico canale di comunicazione per gestire l'intero flusso informativo.

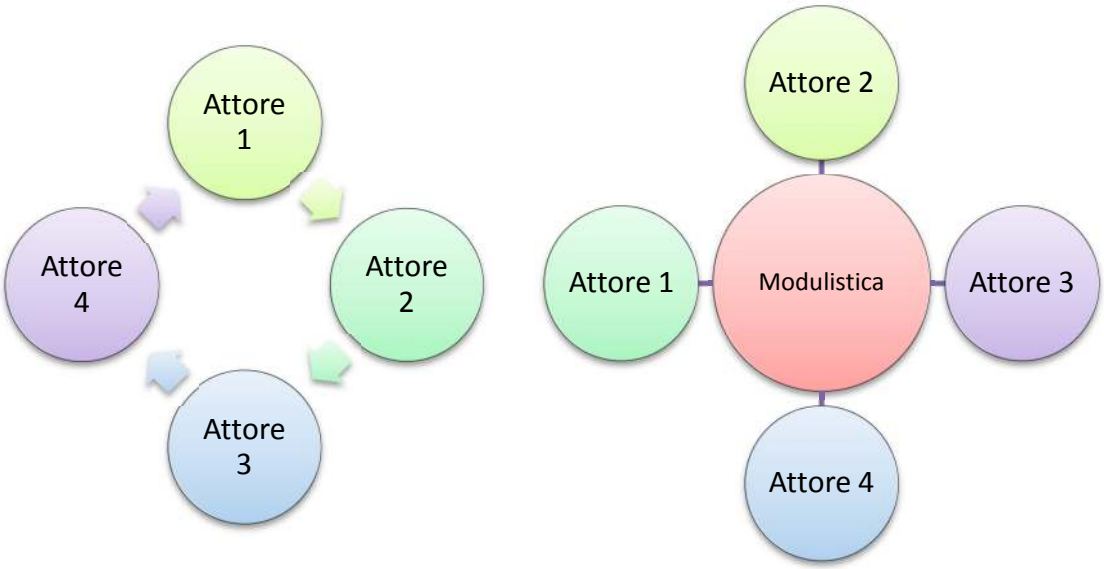


Figura 54 Come cambia la compilazione condivisa.

Per rendere questa funzione utilizzabile, il modulo viene trasformato ed elaborato dal sistema come se fosse una matrice tridimensionale (Figura 55), le cui dimensioni sono i profili utenti coinvolti nel processo, gli stati del processo seguito dal modulo e i campi del modulo stesso.

L'accesso a questa matrice avviene attraverso il seguente percorso: Profilo Utente -> Stato del Modulo -> Campo del modulo. Ogni singola cella della matrice contiene lo stato del campo, cioè indica se questo dev'essere visibile, nascosto o compilabile da parte dell'utente che accede al sistema.

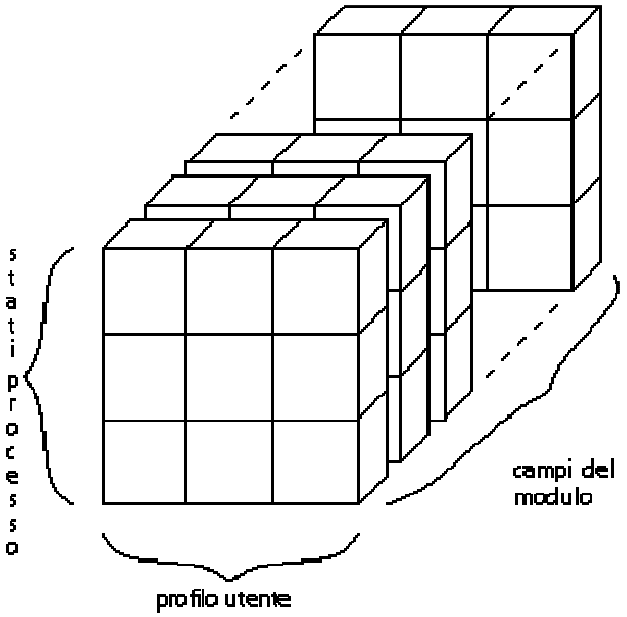


Figura 55 Struttura logica del modulo durante la compilazione condivisa.

Attraverso quest'estrazione il sistema è in grado di individuare per il singolo campo, lo stato da presentare all'utente. Se un campo sarà solamente visibile, l'utente potrà soltanto consultarne il contenuto senza modificarlo. Se sarà invece modificale, l'utente potrà inserire la propria informazione rendendola così disponibile a tutti gli utenti coinvolti nel processo.

4.8 Reportistica

La Modulistica Web oltre a permettere di raccogliere le informazioni, tramite la compilazione dei diversi moduli pubblicati, deve poter gestire anche tutte le azioni necessarie all'analisi dei dati.

L'utente di alto livello, con funzionalità di approvazione e gestione, deve poter sfruttare il sistema per analizzare lo stato dei singoli moduli, eseguire statistiche sulla compilazione ed eventualmente estrarre le informazioni contenute all'interno dei singoli moduli in modo semplice e veloce.

Per raggiungere questi obiettivi, si è pensato di includere nella Modulistica Web, Jasper Report, con il quale è possibile creare e definire report in modo rapido e guidato e che permette di esportare i dati per qualsiasi tipo di file (PDF, Excel, CSV, ecc.).

In questo modo la Modulistica non genera alcun report, ma si occupa solamente di raccogliere i parametri necessari e invocare il motore di Jasper Report ottenendo come risultato finale, un report contenente tutte le informazioni.

Così come per i moduli, anche i report possono essere distribuiti sulla base del profilo utente, offrendo la possibilità a chi gestisce il sistema di differenziare l'offerta della reportistica in base al ruolo e alle caratteristiche dell'utente.

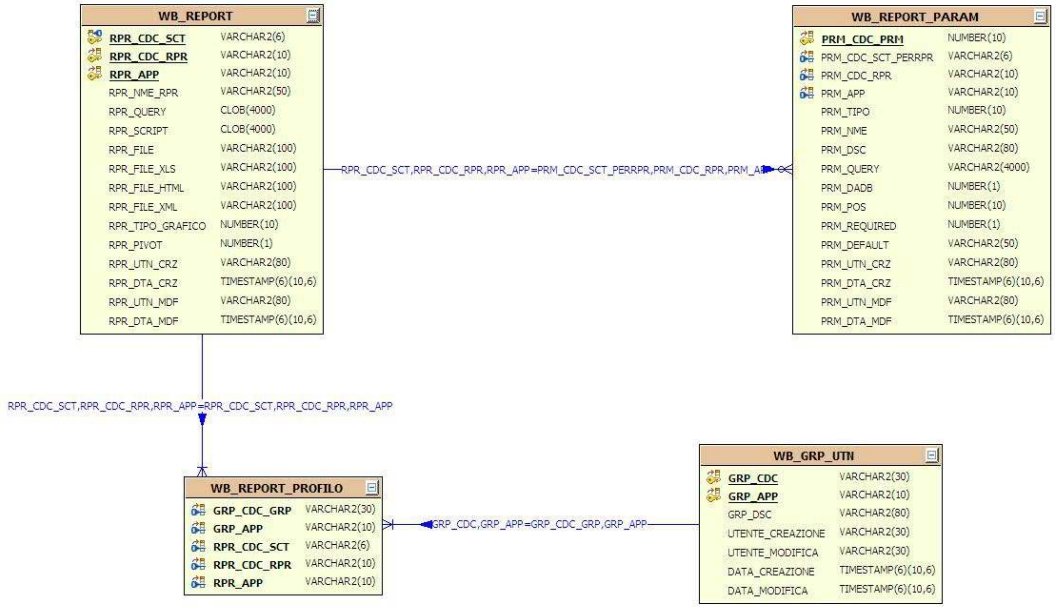


Figura 56 Schema E-R riguardante la gestione dei report.

In Figura 56 sono riportate tutte le tabelle e tutte le relazioni usate dal sistema per gestire i report. Le strutture sono molto semplici, ovviamente la tabella principale è WB_REPORT, che identifica il report vero e proprio. Da cui, attraverso la tabella WB_REPORT_PARAM e WB_REPORT_PROFILO vengono definiti rispettivamente i parametri e i profili utente che possono utilizzare il report.

Oltre ad integrare un motore di reportistica statico, poiché i report prodotti tramite Jasper Report sono file finiti ed esterni alla Modulistica Web, si è pensato di inserire alcune funzionalità di reportistica dinamica con le quali è possibile costruire dei semplici cruscotti, che permettono di offrire e di consultare una panoramica sui dati richiesti dall'utente. È, infatti, possibile raccogliere le informazioni e rappresentarle sotto forma di grafici interattivi o semplici tabelle.

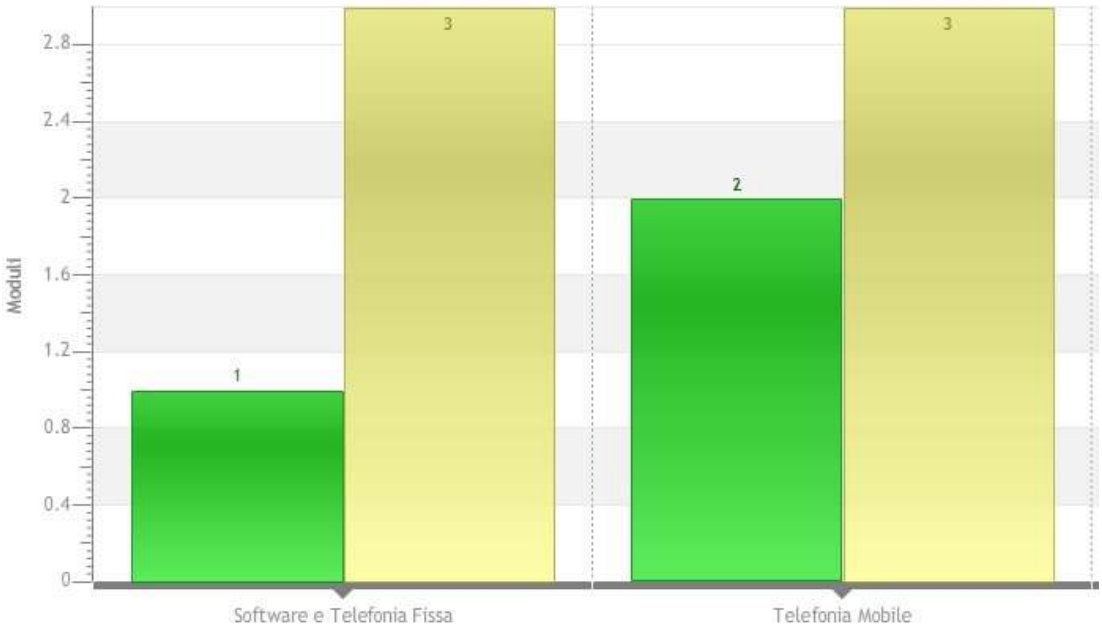


Figura 57 Esempio di cruscotto.

Così come per i moduli attraverso l'editor, anche per i cruscotti e i report si è cercato di sviluppare un'interfaccia che permettesse all'utente di definire i propri report in modo indipendente a seconda delle necessità e dei moduli creati, anche se per poter configurare questi cruscotti è necessaria una buona conoscenza di SQL e della base dati sottostante.

Per definire i cruscotti, infatti, è necessario impostare una serie di query SQL che saranno analizzate dal sistema per poi essere rappresentate sotto forma di grafico o di semplice tabella, a seconda delle richieste dell'utente. In questo modo, essendo le informazioni ottenute tramite interrogazione SQL, l'utente avrà sempre a disposizione dati in tempo reale, che potrà ulteriormente affinare attraverso i filtri e gli strumenti di ricerca messi a disposizione dall'applicativo.

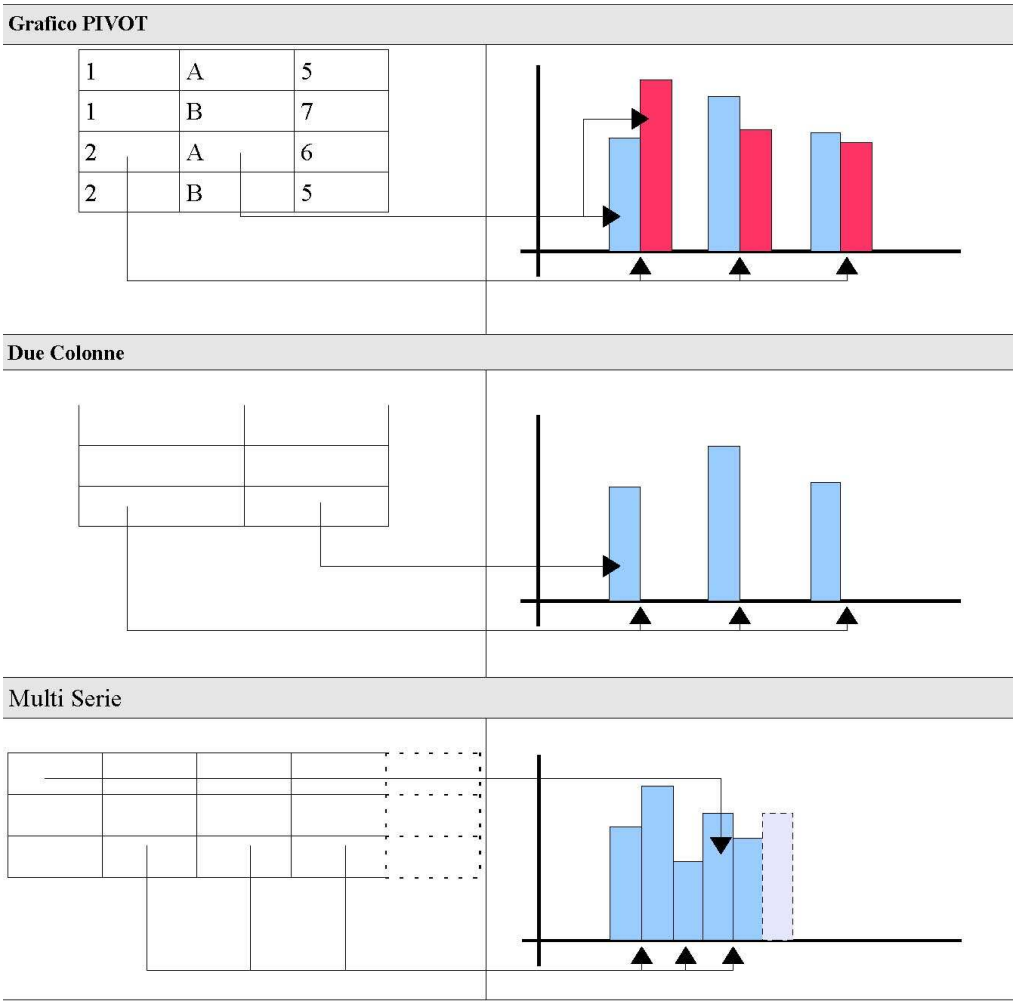


Figura 58 Logiche di generazione dei report grafici.

In Figura 58 sono riportate le tre diverse logiche con cui il sistema elabora la query e genera il grafico.

1. Se il grafico è **PIVOT** vengono considerate solo le prime tre colonne della query. La prima colonna indicherà il valore delle ordinate. La seconda colonna indica la serie del valore. Per ogni valore di X bisogna avere gli stessi identici valori in numero o ordine per la seconda colonna. La terza colonna è il valore delle ascisse che verrà utilizzato nel report.

2. Se la query restituisce due sole colonne, la prima sarà il valore delle ordinate e la seconda il valore delle ascisse. In questo caso la serie è unica e avrà il nome della seconda colonna.
3. Se la query restituisce più colonne, la prima sarà sempre il valore delle ordinate e le colonne successive saranno le diverse serie con i rispettivi valori delle ascisse.

Selezionando la serie e un valore delle ordinate è possibile espandere le informazioni e ottenere il dettaglio di un determinato valore. In questo modo l'utente può sfruttare i report sia per avere informazioni di alto livello sia per avere informazioni dettagliate.

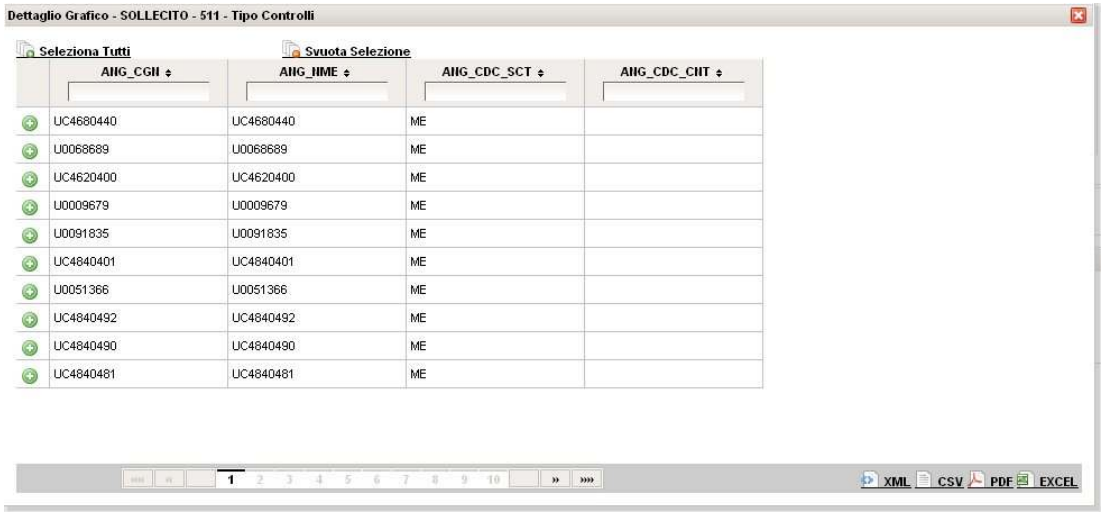


Figura 59 Report di dettaglio ottenuto da un report grafico.

4.8.1 Generatore di report

Così come per la definizione dei moduli, anche per i report si è scelto di sviluppare una funzionalità integrata nell'applicazione che permetta a un utente con poche conoscenze informatiche di creare i propri cruscotti e report.

Definita la struttura del modulo, l'utente può scegliere quali informazioni verranno estratte dal report, con quale ordine e in quale formato. In questo modo viene nascosta all'utente tutta la complessità relativa alla scrittura delle query SQL, che vengono impostate automaticamente dal sistema e non viene richiesta la conoscenza di programmi particolari per generare i file di Jasper Report.

Ovviamente l'utilizzo di questa funzionalità impone alcuni limiti, soprattutto sul formato dei cruscotti e sulla struttura dei report ottenuti. Infatti per offrire all'utente la possibilità di generare report in modo quasi automatico, sono state inserite nel sistema alcune logiche, di carattere generico, che possono adattarsi a qualsiasi tipologia di modulo.

I cruscotti che si ottengono tramite il generatore automatico, infatti, permettono di analizzare i dati di un solo modulo, raggruppandoli per stato. Mentre i report sono semplici rappresentazioni tabellari dei moduli compilati dagli utenti, le cui colonne sono i campi definiti dall'utente durante la creazione del report. Questi report possono anche essere dei punti di partenza che l'utente può elaborare e migliorare sia nell'aspetto che nelle informazioni estratte.

Capitolo 5

Robustezza e conformità

Dopo aver analizzato nel dettaglio l'applicazione Modulistica Web, in questo capitolo verrà effettuata un'analisi della robustezza di tutto il sistema e la sua conformità agli obblighi di legge.

In particolare verrà effettuata un'analisi sugli strumenti utilizzati per effettuare i test del software, sulla sicurezza del sistema (e sulle sue potenziali falle) e su come la Modulistica Web rispetti gli obblighi di legge circa la privacy.

5.1 Test

Periodicamente l'applicazione è sottoposta a sessioni di test programmati. I test vengono effettuati con più modalità e a più livelli.

I componenti più piccoli e controllabili vengono testati attraverso JUnit [8], un sistema di test completamente automatizzato. I test tramite JUnit consistono nel creare tramite il linguaggio Java gli scenari d'uso dei singoli componenti, provando a chiamarli con tutti gli input possibili e indicando a JUnit quali sono gli output che ci si aspetta.

Chiaramente le parti più complesse del sistema non sono gestibili con sistemi di test come JUnit. I test di più ampio respiro sono pianificati attraverso il software open source JMeter [9] e vengono eseguiti in parte manualmente dai beta tester e in parte in automatico. Il software registra le azioni effettuate all'interno dell'applicazione (tasti premuti, movimenti e click del mouse ecc.) e li ripete in automatico, analizzando che il risultato delle azioni eseguite sia quello atteso.

Infine, prima del rilascio di ogni nuova versione dell'applicazione, viene effettuata dai beta tester anche una sessione di test libero: utilizzo dell'applicazione in modalità casuale per cercare di rilevare eventuali bug non riscontrabili dai test programmati.

5.2 Sicurezza

Come già trattato nel capitolo 4.5, l'applicazione permette un alto grado di personalizzazione dei permessi d'accesso di ogni utente. La sicurezza delle informazioni è garantita dal fatto che non c'è modo di accedere direttamente al database se non passando dall'applicazione stessa.

Oltre a robustezza e sicurezza, vengono fatte ulteriori analisi per riscontrare la compatibilità con i dettami relativi all'accessibilità e al rispetto della privacy, come mostrato nelle sezioni seguenti.

5.3 Privacy

Poiché la Modulistica Web ha come scopo quello di raccogliere qualsiasi informazione da parte degli utenti, spesso anche informazioni sensibili, è stata pensata e progettata per rispettare le direttive legislative nazionali sul rispetto della privacy. Il riferimento principale è il "Disciplinare tecnico in materia di misure minime di sicurezza", allegato B della Legge 196.

In particolare la conformità alla legge è garantita da:

- La possibilità, attraverso opportuna configurazione del servlet container, di effettuare le comunicazioni in modo criptato utilizzando il protocollo HTTPS.
- Il componente che si occupa dell'autenticazione, espone diversi parametri di configurazione che permettono di imporre ad ogni utente di avere una password di almeno otto caratteri e di cambiarla ogni tre mesi.
- La password di accesso di ogni utente è una "stringa criptata".

- È possibile integrare la procedura di autenticazione con i diversi sistemi aziendali (Active Directory, LDAP) offrendo così un livello di sicurezza ancora più elevato.
- E possibile impostare i permessi di accesso ai dati in modo molto granulare su moduli e istanze in base a gruppi di utenti.

Si conclude così l'analisi della robustezza della Modulistica Web, cioè quali meccanismi sono stati utilizzati dal team di sviluppo per testare il sistema e per garantire il rispetto dei dettami su accessibilità e privacy.

Capitolo 6

Casi d'uso

Dopo aver descritto nei minimi particolari tutte le scelte progettuali e tecniche della Modulistica Web, verranno ora analizzati alcuni esempi di utilizzo da parte dei diversi clienti. Come campione d'analisi, sono stati scelti tre diversi clienti leader nel loro settore, e verranno descritte le diverse scelte nell'utilizzo dell'applicativo, evidenziando così la flessibilità e le potenzialità della Modulistica Web.

6.1 Settore Assicurativo

Un'azienda leader nel settore assicurativo ha installato e utilizza la Modulistica Web per ottimizzare molte funzionalità interne di gestione. Oltre alla compilazione della semplice modulistica da parte del dipendente, e quindi oltre ad automatizzare la comunicazione verso l'ufficio del personale, la scelta del cliente è stata quella di inserire all'interno della Modulistica la gestione delle Performance Management (PM), dei questionari valutativi interni all'azienda, pubblicazione di nuovi posti di lavoro e altri moduli interni non forniti con l'applicativo. Tutti questi moduli, particolari e specifici per il cliente in oggetto, sono stati creati dal cliente stesso tramite l'utilizzo dell'editor interno all'applicativo, senza dover ricorrere a un'assistenza particolare a testimonianza della facilità e dell'usabilità dello strumento stesso.

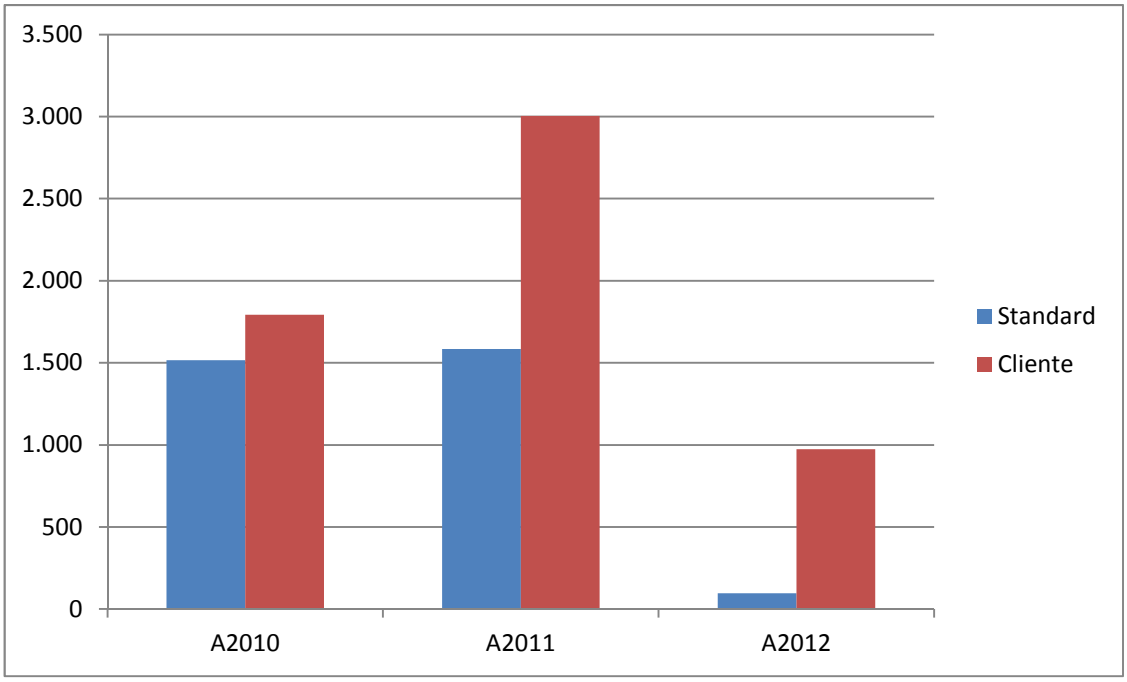


Figura 60 Moduli compilati su base annua.

Come si può notare dalla Figura 60, dopo un primo anno di utilizzo in cui i moduli standard e quelli interni del cliente sono stati utilizzati in egual misura, dal secondo anno in poi il cliente, capite le potenzialità dello strumento, ha spinto moltissimo sull'utilizzo dello stesso, pubblicando sempre più moduli e cercando di automatizzare e ottimizzare al massimo tutte le procedure interne.

Il modulo che ha sfruttato a pieno tutte le funzionalità della Modulistica Web è senz'ombra di dubbio la Performance Management. Con questo modulo il cliente ha cercato di ottimizzare e rendere meno confuso il processo di raccolta dati relativo alle valutazioni delle prestazioni dei dipendenti da parte dei vari responsabili. Definito e disegnato il modulo è stato impostato il processo così come riportato in Figura 61. Quest'ultimo si divide fondamentalmente in cinque fasi ben distinte, all'interno delle quali il responsabile e il dipendente a cui appartiene la scheda commentano gli obiettivi, le aspettative e i risultati raggiunti dalla persona durante l'anno.

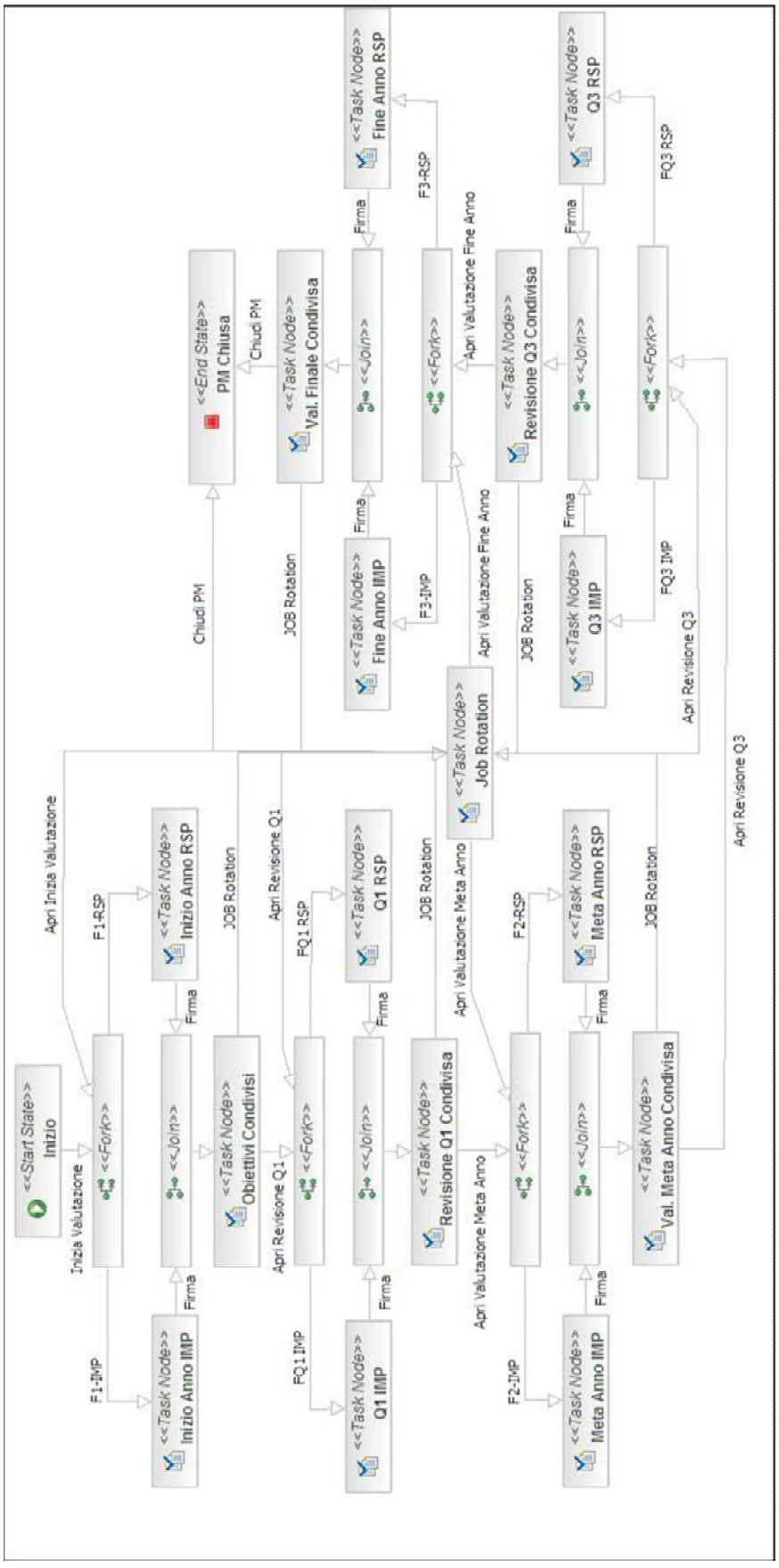

<<Start State>>
Inizio
Inizia Valutazione
<<Fork>>
F1-IMP
F1-RSP
Apri Inizia Valutazione
<<Task Node>>
Inizio Anno IMP
<<Task Node>>
Inizio Anno RSP
<<Join>>
Firma
<<Task Node>>
Obiettivi Condivisi
JOB Rotation
Apri Revisione Q1
<<Fork>>
FQ1 IMP
FQ1 RSP
<<Task Node>>
Q1 IMP
<<Task Node>>
Q1 RSP
<<Join>>
<<Task Node>>
Revisione Q1 Condivisa
<<Task Node>>
Job Rotation
Chiudi PM
<<End State>>
PM Chiusa
Apri Valutazione Meta Anno
<<Fork>>
F2-IMP
F2-RSP
<<Task Node>>
Meta Anno IMP
<<Task Node>>
Meta Anno RSP
<<Task Node>>
Val. Meta Anno Condivisa
Apri Revisione Q3
<<Task Node>>
Val. Finale Condivisa
Apri Valutazione Fine Anno
<<Fork>>
F3-IMP
F3-RSP
<<Task Node>>
Fine Anno IMP
<<Task Node>>
Fine Anno RSP
<<Task Node>>
Revisione Q3 Condivisa
FQ3 IMP
FQ3 RSP
<<Task Node>>
Q3 IMP
<<Task Node>>
Q3 RSP

Figura 61 Processo Performance Management.

Conclusa la singola fase di valutazione, cioè sia quando il responsabile che il dipendente firmano la scheda e accettano quindi il contenuto inserito dagli stessi, il modulo passa in uno stato temporaneo, detto Job Rotation, in carico al Responsabile PM. Quest'utente, che ha il compito di raccogliere tutte le schede e di riassumere tutte le valutazioni, ha anche il compito di aprire le singole fasi di valutazione, portando le diverse schede dallo stato di Job Rotation allo stato scelto.

Prima dell'utilizzo della modulistica, questo processo veniva svolto manualmente tramite l'utilizzo di file Word. La compilazione contemporanea del modulo non esisteva e questo inseriva una perdita di tempo sia al responsabile sia al dipendente, poiché dovevano bloccare le loro attività e riunirsi per discutere i diversi obiettivi. Sfruttando la modulistica, la compilazione è contemporanea, avviene tramite il web, e il dipendente vede quello che scrive il responsabile e viceversa, riducendo così i tempi necessari alla valutazione. Un altro processo che ha avuto un notevole sviluppo tramite l'utilizzo dell'applicazione è sicuramente la raccolta e l'analisi dei dati. In precedenza questa fase era svolta manualmente, raccogliendo le schede di valutazioni tramite mail o addirittura su carta e trascrivendo le valutazioni manualmente all'interno dei diversi report. Sfruttando il motore di reportistica integrato nella modulistica, è ora possibile, per l'utente Responsabile PM, raccogliere le informazioni in pochissimo tempo e organizzarle in appositi report, in tempo reale.

6.2 Settore Farmaceutico

L'azienda di lunga tradizione farmaceutica e oggi leader nel settore della salute e del benessere: tra le prime quattro aziende farmaceutiche in Italia per volumi di vendita, ha utilizzato la Modulistica Web per automatizzare la comunicazione dei dipendenti verso l'ufficio del personale, utilizzando i moduli standard che vengono forniti con l'applicazione e che ricoprono gli obblighi fiscali e legislativi del dipendente. Così sono stati distribuiti i moduli per le detrazioni fiscali, gli assegni familiari e alcuni moduli riguardanti le variazioni anagrafiche (domicilio, coordinate bancarie, ecc.).

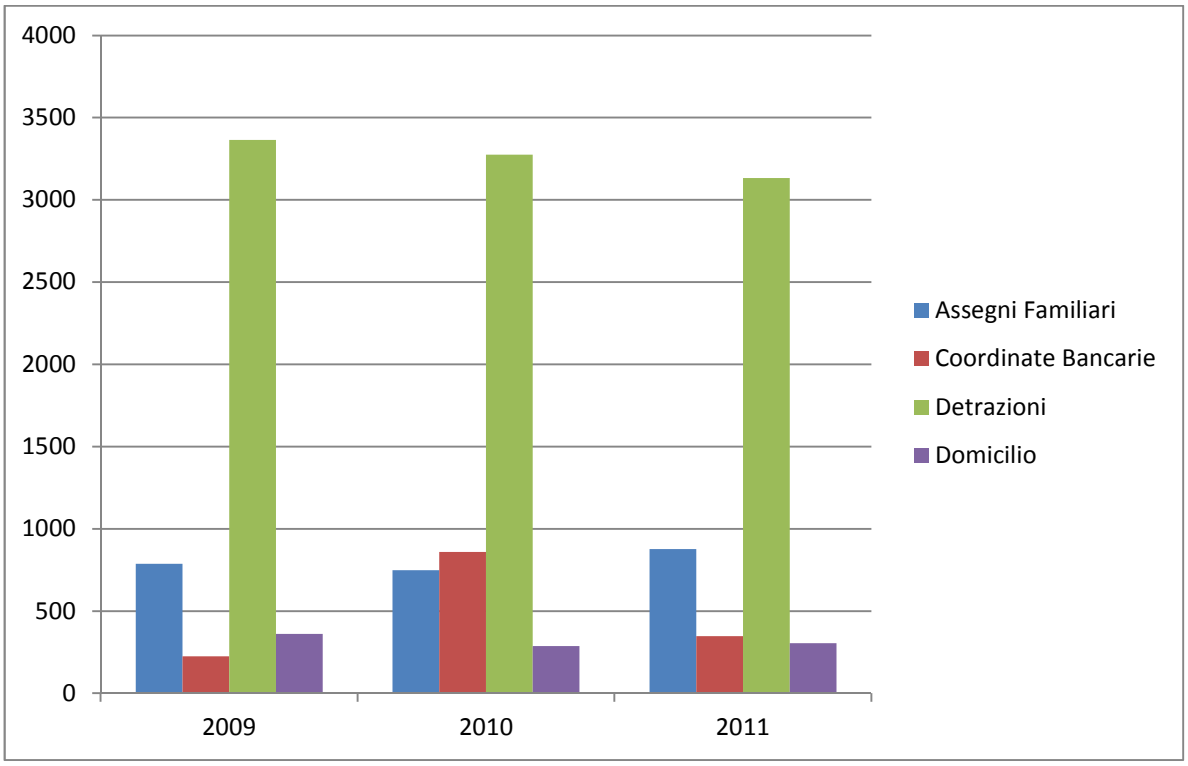


Figura 62 Moduli compilati su base annua.

Come si può notare dalla Figura 62 il modulo più utilizzato dai dipendenti dell'azienda è quello riguardante le Detrazioni, visto anche la normativa che obbligava il dipendente a compilarlo annualmente.

Prima dell'utilizzo della Modulistica Web la compilazione di questi moduli e la comunicazione dipendente -> ufficio personale era basata esclusivamente sui moduli cartacei. Ai dipendenti veniva offerto il modulo vuoto o in parte precompilato, da riconsegnare compilato in ogni singola parte all'ufficio competente. Il quale doveva controllare le informazioni inserite, validarle e in seguito inserirle manualmente nei sistemi aziendali.

L'introduzione della Modulistica Web ha permesso di automatizzare molte di queste azioni sgravando notevolmente il carico di lavoro dell'ufficio del personale che si dovrà occupare solo di eseguire le opportune procedure. Tramite il motore di controllo e di scripting, la compilazione dei moduli è ora controllata in modo automatico così che all'ufficio personale pervengano solo moduli formalmente corretti e gli eventuali errori vengono subito segnalati al dipendente, riducendo notevolmente i tempi di correzione.

Un altro vantaggio dovuto all'introduzione della Modulistica Web è la possibilità di distribuire le responsabilità di approvazione e controllo ai diversi uffici periferici, lasciando come unico compito dell'ufficio del personale l'acquisizione dell'informazioni contenute nei moduli tramite l'esecuzione delle diverse procedure automatiche.

6.3 Settore Chimico

Il maggior produttore mondiale di adesivi e prodotti chimici per l'edilizia ha deciso di sfruttare la Modulistica Web inizialmente per migliorare la comunicazione dei dipendenti verso l'ufficio del personale.

L'introduzione della Modulistica Web come sistema aziendale, la sua flessibilità e le sue possibilità hanno spinto il cliente a introdurre all'interno dello strumento la gestione del processo delle dotazioni aziendali.

Individuati in modo preciso i ruoli e le persone che ricoprono tali ruoli, è stato definito e configurato un processo descritto dalla Figura 63. Questo processo si può suddividere in tre distinte linee autorizzative, che vengono attivate a seconda delle dotazioni richieste dalla persona. Queste linee sono Risorse Umane, Servizi Generali e Sistemi Informativi, che a loro volta si suddividono in tre diverse sezioni che hanno compiti ben precisi: Telefonia, Hardware e Software.

L'integrazione del motore di workflow jBPM, permette alla Modulistica Web di gestire questo processo con estrema facilità e flessibilità, offrendo la visibilità di una scheda alle sole persone interessate, in base al loro ruolo e alle loro responsabilità.

L'introduzione di tale processo all'interno della modulistica permette di tenere traccia di tutte le azioni svolte, di analizzare la velocità con cui le diverse linee operano e quindi evidenziare le eventuali criticità, permettendo così al cliente di intervenire e risolvere gli eventuali problemi.

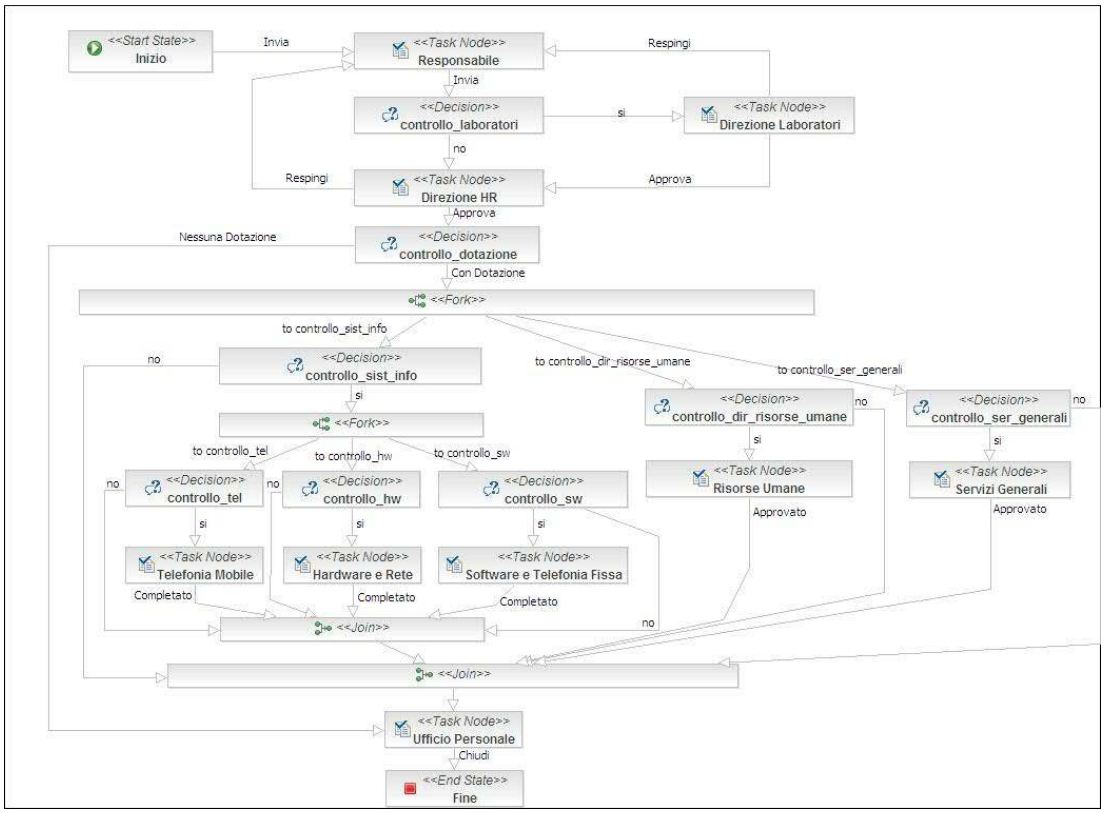


Figura 63 Processo Dotazioni Aziendali.

Capitolo 7

Conclusioni

Questa tesi descrive tutte le fasi che hanno caratterizzato un progetto effettuato all'interno di un'azienda informatica che si occupa di soluzioni per la gestione delle risorse umane. Il risultato finale è un'applicazione web per lo sviluppo e la gestione completa di tutta la modulistica aziendale. La creazione di questo servizio nasce dall'esigenza, delle piccole medie imprese, di regolare e informatizzare la comunicazione dal dipendente verso l'azienda attraverso uno strumento flessibile e di facile utilizzo.

La nuova applicazione descritta in questa tesi, sostituisce tutte quelle azioni svolte manualmente dai vari dipendenti dell'azienda, dalla compilazione alla correzione e all'inserimento delle informazioni nei diversi sistemi aziendali.

La parte più complessa dell'intero progetto è stato garantire due dei principali obiettivi dell'applicazione: facilità d'uso e flessibilità. Da una parte infatti,

l'applicazione doveva essere il più semplice possibile, immediata e di facile comprensione per evitare di creare un problema nell'aziende che sceglievano di utilizzarla. Dall'altra doveva essere estremamente flessibile, così da poter gestire qualsiasi tipologia di modulo senza dover richiedere alcuna modifica nel codice o nelle logiche applicative. Il simbolo di questa integrazione, tra flessibilità e facilità d'uso, è la funzionalità di editor visuale che permette a qualsiasi utente, con minime capacità informatiche, di disegnare ed impostare un qualsiasi modulo, indipendentemente dal numero di pagine, dalle informazioni trattate e dai destinatari di tale modulo.

L'applicazione è stata strutturata in moduli ben distinti così da poter ampliarne le funzionalità senza incontrare troppe difficoltà. In futuro, infatti, potranno essere apportati numerosi miglioramenti, quali:

- Aumento della velocità e della reattività dell'applicazione, nonostante le prestazioni già conseguite.
- Estensione della gerarchia dei componenti, aggiungendone di nuovi ad oggi non ancora sviluppati ed integrati.
- Aumento delle potenzialità degli strumenti di reportistica.
- Maggior integrazione con i sistemi aziendali già esistenti.

Questo progetto ha rappresentato il punto di partenza per l'approfondimento di molti aspetti, marginalmente affrontati in ambito universitario, sia a livello di codice di scripting, sia su argomenti come l'accessibilità, l'usabilità e la sicurezza.

Bibliografia

[1] K. D. Mann, JavaServer Faces in Action, Manning, 2004.

[2] J. Community, «Hibernate - Relational Persistence for Idiomatic Java,» [Online]. Available: http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/.

[3] Vivek Chopra, Sing Li, and Je, Professional Apache Tomcat 6, Wrox Press, 2007.

[4] J. Community, «RichFaces Developer Guide,» [Online]. Available: http://docs.jboss.org/richfaces/latest_3_3_X/en/devguide/html_single/.

[5] J. Community, «Hibernate,» [Online]. Available: http://www.hibernate.org/.

[6] J. Community, «JBoss jBPM - Workflow in Java,» [Online]. Available: http://docs.jboss.com/jbpm/v3.2/userguide/html_single/.

[7] Jaspersoft, JasperReports Library Ultimate Guide, Jaspersoft.

[8] K. Beck, JUnit Pocket Guide, O'Reilly Media, 2004.

[9] A. S. Foundation, «JMeter: User's Manual,» [Online]. Available: http://jmeter.apache.org/usermanual/index.html.

[10] J. Community, «RichFaces,» [Online]. Available: http://www.jboss.org/richfaces.

[11] Christian Bauer and Gavin King, Java Persistence with Hibernate, Manning Publications, 2006.

[12] E. Gianfelici, La legge sulla privacy, Fag, 1997.

[13] S. Holzner, Eclipse, O'Reilly Media, 2004.