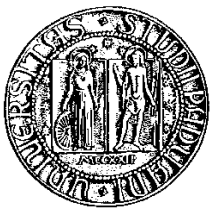

Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

Tesi di laurea

PROCEDURE PER IL WORKFLOW MANAGEMENT IN UN'AZIENDA CERTIFICATA ISO 9001

Relatore: Ch.mo Prof. Matteo Bertocco
Correlatore: Ing. Mauro Franchin

Laureando: Qifeng Zhu

05 Ottobre 2010

Alle mie famiglie

Prefazione

Il presente lavoro di tesi per il corso di Laurea Specialistica in Ingegneria Informatica è stato svolto con l'obiettivo di reingegnerizzare un sistema di *WorkFlow Management*, in seguito alla completa ridefinizione di alcuni requisiti fondamentali da parte del committente. Il punto di partenza del presente lavoro è stato fornito dalle seguenti tesi e può essere considerato il loro proseguo naturale:

> "*Progettazione e implementazione di procedure per il workflow management in un'azienda certificata ISO:9001*", di Molinaroli Andrea.
>
> "*Analisi e realizzazione di procedure per il workflow management in un'azienda certificata ISO:9001*", di Righetto Enrico.

Parti del presente elaborato hanno di fatto aggiornato le citate tesi, mentre le parti originali sono rappresentate dall'analisi del sistema esistente e implementazioni da zero del nuovo sistema in conformità ai nuovi requisiti, sotto la supervisione del Relatore e del Correlatore. Come i precedenti lavori, anche la presente tesi è frutto di una collaborazione tra un'azienda padovana che intende intraprendere il percorso di certificazione ISO 9001 e l'Università degli Studi di Padova .

Qifeng Zhu

Indice

Capitolo 1

Introduzione

Sempre più spesso le aziende, per poter far fronte alla concorrenza, devono offrire ai clienti certezze sulla qualità dei prodotti sviluppati e trasparenza nei processi aziendali.

La necessità di avere un riferimento internazionale, attraverso il quale stabilire la *qualità dei prodotti*, ha determinato, nel 1987, l'emanazione, da parte dell'ISO[1] (International Organization for Standardization), di una famiglia di *norme internazionali*. Tale famiglia è meglio conosciuta con il nome di ISO 9001 e contiene un insieme di regole il cui scopo è garantire che un'azienda implementi un sistema di gestione interna in grado di garantire la qualità dei prodotti.

Sebbene la volontà di intraprendere un percorso di certificazione, con lo scopo di beneficiarne, sia spesso dettata da logiche di mercato, va ricordato che rivedere l'*asset* aziendale per renderlo adatto allo standard ISO, e il lavoro per mantenerlo tale, può, spesso, richiedere un'overhead notevole ai dipendenti aziendali. Per fronteggiare tale problema vengono, generalmente, impiegati strumenti al fine di automatizzare il più possibile il lavoro di produzione della documentazione necessaria. Le aziende si trovano, quindi, costrette ad adottare strumenti, in particolare software, che spesso impongono la loro struttura, la quale obbliga chi li utilizza a modificare, anche pesantemente, il proprio modo di lavorare.

Il presente lavoro è il risultato di un lavoro di tesi svolto da uno studente del corso di Laurea Specialistica in Ingegneria Informatica che pone come obiettivo la reingegnerizzazione di un sistema di *WorkFlow Management* utilizzabile da aziende che lavorano in conformità alle norme ISO 9001 o che hanno come obbiettivo tale certificazione. In particolare, il sistema deve essere capace di adattarsi il più possibile a tutte le realtà aziendali, assimilare la struttura dell'azienda e, su di essa, applicare l'insieme delle regole che permettano di raggiungere la certificazione.

Portare un'azienda ad ottenere la certificazione ISO 9001 è un'operazione che richiede un avanzamento per passi successivi. Il lavoro descritto nella pre-

[1] `http://www.iso.org`

sente tesi è il frutto di una collaborazione tra un'azienda padovana che intende intraprendere il percorso di certificazione e l'Università degli Studi di Padova.

La tesi inizia con un'introduzione alle normative e descrive la struttura aziendale mettendola in relazione con i requisiti prescritti dalle norme stesse. In particolare, nel capitolo 2 viene presentata la normativa ISO 9001. Il capitolo 3 presenta il modello per la descrizione e la rappresentazione dei processi.

La progettazione e implementazione del un software per il sistema di gestione della qualità (SGQ) che rispetti i requisiti normativi precedentemente individuati è descritto nel capitolo 4, il quale presenta il progetto WQF (acronimo di *Work-Quality-Flow*), evidenziandone le idee di fondo e gli obbiettivi prefissati. Nel capitolo 5 sono descritte le fasi per la definizione del piano di lavoro.

L'analisi funzionale del software, che rappresenta il corpo centrale della tesi, in cui si sono analizzati i requisiti, in seguito alla ridefinizione di quest'ultimi, si trova nel capitolo 6. L'analisi tecnica, in cui viene presentata l'architettura dell'applicazione e le tecnologie utilizzate, è presentato nel capitolo 7. Nel capitolo 8 viene descritta, sotto forma di tutorial, la realizzazione di una semplice applicazione di prova per sperimentare i meccanismi di autenticazione e gestione dei permessi. La struttura della base di dati su cui lavora l'applicazione è riportata nel capitolo 9. I capitoli 10, 11 e 12, infine, contengono la manualistica per l'installatore, il disegnatore dei processi e l'utente abituale.

L'appendice A riporta i sorgenti e la struttura delle directory dell'applicazione di prova sulla sicurezza. I sorgenti dell'applicazione WQF sono stati volutamente omessi per ragioni di spazio: essi sono comunque disponibili nel disco allegato.

Capitolo 2

La Norma ISO 9001

2.1 La norma e i suoi obbiettivi

Nell'ultimo ventennio il mondo delle imprese, sia manifatturiere che di servizio, sia pubbliche che private, è stato protagonista di una vera e propria rivoluzione della qualità che ha profondamente influenzato le strategie d'impresa, il management, il ruolo delle persone e il quotidiano approccio alle attività aziendali. Questa rivoluzione globale ha, per le aziende, un passaggio cruciale, che in molti casi diviene obbligato per la sopravvivenza delle stesse nel mercato: instaurare un Sistema di Gestione per la Qualità (SGQ) in grado di essere certificato da un ente accreditato di parte terza.

L'ISO[1] (International Organization for Standardization) e il CEN[2] (European Committee for Standardization) hanno redatto le regole organizzative minimali cui le aziende devono conformarsi per garantire in modo continuo e costante la qualità di prodotti e/o servizi; queste regole sono rappresentate dalle norme UNI EN ISO 9000. Tali norme non fanno riferimento a particolari settori industriali e commerciali, ma sono applicabili a qualunque attività e a tutti i tipi di impresa e di organizzazione, qualsiasi sia la loro dimensione e il livello di tecnologia impiegata. Le norme UNI EN ISO 9000, recepimento italiano

[1] ISO è un'associazione mondiale di organismi nazionali di formazione, i cui comitati tecnici effettuano l'elaborazione delle norme internazionali. Ogni organismo nazionale di formazione interessato a un argomento per il quale è stato insediato un comitato tecnico ha il diritto di essere rappresentato in tale comitato. Partecipano ai lavori anche le organizzazioni internazionali di estrazione governativa e non, che intrattengono rapporti con l'ISO. L'ISO collabora strettamente con l'IEC (International Electrotechnical Commission) in tutti i campi di formazione del settore elettrotecnico.

[2] CEN (Comitato Europeo di Normazione) è un ente normativo che ha lo scopo di armonizzare e produrre norme tecniche in Europa in collaborazione con enti normativi nazionali e sovranazionali, quali per esempio l'ISO. Il CEN lavora in accordo con le politiche dell'Unione Europea e dell'EFTA (European Free Trade Association) per favorire il libero scambio e la tutela dei consumatori, dei lavoratori e dell'ambiente, producendo standard che sono poi normalmente recepiti dagli enti nazionali dei singoli paesi.

delle norme ISO 9000, forniscono alle aziende italiane un pacchetto di regole riguardanti la conduzione aziendale per la qualità e l'assicurazione della stessa.

L'UNI (Ente Nazionale Italiano di Unificazione) definisce la norma un *"documento prodotto mediante consenso di tutte le parti interessate e approvato da un organismo riconosciuto, che fornisce, per usi comuni e ripetuti, regole, linee guida o caratteristiche relative a determinate attività o ai loro risultati, al fine di ottenere il miglior ordine in un determinato contesto"*. Gli obiettivi fondamentali della normazione sono:

- la realizzazione di un mezzo chiaro e univoco di espressione di comunicazione fra tutte le parti interessate all'interno dell'organizzazione;
- il miglioramento dell'economia generale, attraverso la razionalizzazione della produzione di materiali grezzi, semilavorati e finiti;
- la salvaguardia della salute e della sicurezza degli individui e la tutela dell'ambiente;
- la protezione del consumatore mediante un livello di qualità dei prodotti e dei servizi debitamente controllato e adeguato alle sue necessità.

Alla base della definizione di Sistema di Gestione per la Qualità c'è il concetto di sistema: esso è un insieme di oggetti (parti, componenti, funzioni, ecc.) legati tra loro da relazioni di interdipendenza. Per interdipendenza s'intende che un intervento effettuato esclusivamente su una singola parte del sistema ha ripercussioni anche sulle altre; perciò quando si parla di sistema è necessario un approccio globale e mai parziale. Con l'approccio sistemico non si vuole certo sminuire l'importanza dei singoli componenti, ma si cerca piuttosto di considerarli e di studiarli in ragione del loro essere parti del tutto. Per ottenere questo importante risultato è necessario pianificare e organizzare le attività, i ruoli, le responsabilità e i supporti operativi in modo da coinvolgere tutti i funzionari aziendali. Un'azienda che operi in ottica di qualità e che abbia una gestione per processi vede all'interno di essi una sequenza di attività, ciascuna delle quali fornisce a quella successiva il proprio prodotto,ovvero l'attività a monte viene vista come un vero e proprio fornitore mentre l'attività a valle come un cliente, lungo una catena di processi interni che inizia con l'identificazione delle esigenze e delle aspettative del cliente (esterno) e si conclude con il suo soddisfacimento.

In seguito a questa interpretazione, la qualità ha assunto le prerogative di una filosofia che coinvolge tutti gli attori dell'organizzazione e tutti i processi che ne ottimizzino le risorse, diventando totale. Una qualità così intesa, e considerata dall'azienda come costante ricerca delle caratteristiche più importanti da individuare e fare proprie, si compone di tre caratteristiche principali: quella operativa (il prodotto/servizio è efficiente e affidabile); quella relativa alle caratteristiche tecniche (le prestazioni del prodotto/servizio sono frutto di indagini di marketing, analisi di mercato e valutazioni sulla soddisfazione del cliente);

quella dei servizi connessi al prodotto/servizio (sostituzione, tempi e modalità di consegna, forme di pagamento e assistenza post-vendita). E' proprio su aspetti quali la qualità dei servizi post-vendita che, in un mercato attestato su elevati standard produttivi, si va sempre più concentrando l'attenzione: trasporto, montaggio e manutenzione sono elementi fondamentali e possono determinare il consolidamento o la perdita di clientela.

Ai fini di una loro maggiore spendibilità operativa, si riassumono i concetti base per il raggiungimento del Total Quality Management (TQM) in dieci punti sequenziali:

1. la priorità assoluta dell'azienda, cioè la condizione essenziale per garantirne la sopravvivenza, è costituita dal cliente, senza il quale l'azienda non ha ragione di esistere;

2. in quest'ottica il tipo di cliente più importante è il cliente consolidato. Un fatturato realizzato con clienti consolidati è molto più sicuro di quello realizzato con clienti occasionali;

3. un cliente diventa consolidato se è soddisfatto del precedente acquisto. La soddisfazione del cliente diventa quindi la vera priorità operativa dell'azienda;

4. il profitto è il premio di questa soddisfazione, il fatturato ne è la sua misura;

5. la soddisfazione del cliente si ottiene essenzialmente fornendogli un prodotto o un servizio di alta qualità: è proprio la qualità di quanto ha già acquistato il fattore che più ne condizionerà il prossimo acquisto (il prezzo è, invece, un fattore prioritario per l'acquisizione di un nuovo cliente). La qualità assume un significato più ampio e contiene tutto ciò che garantisce la soddisfazione del cliente;

6. per garantirsi il consolidamento del cliente occorre assicurarsi la sua continua soddisfazione a ogni successivo acquisto. Questo risultato non è ottenibile semplicemente fornendo un elevato grado di qualità e mantenendolo costante nel tempo. La soddisfazione presume il miglioramento. Solo miglioramenti continui del prodotto e/o del servizio fornito possono garantire un elevato grado di soddisfazione del cliente, in modo tale da condizionarlo positivamente per il successivo acquisto;

7. la qualità del prodotto e/o del servizio fornito non è altro che il risultato dei processi aziendali attuati per realizzarlo. La qualità del prodotto è il risultato della qualità dei processi;

8. se occorre migliorare continuamente i prodotti, occorre dunque migliorare continuamente i processi aziendali;

9. per migliorare continuamente i processi aziendali occorre mobilitare la massima quantità di risorse aziendali. Presupposto del miglioramento è, dunque, il massimo coinvolgimento;

10. non è sufficiente mobilitare un elevato numero di persone per ottenere miglioramento, ma occorre organizzare questa attività e addestrare le persone a queste nuove abilità per il miglioramento richiesto.

2.1.1 La Qualità Totale

Quando si parla di Qualità Totale si fa sempre riferimento ai principi della competitività e del libero mercato; le norme, in questa dimensione verticale, rappresentano degli stati consolidati che permettono all'azienda di entrare nel mercato e con cui, se vorrà raggiungere l'eccellenza, dovrà continuamente confrontarsi per migliorare le proprie prestazioni.

Agire in quest'ottica dinamica e proattiva significa anzitutto:

- analizzare i processi lavorativi dell'azienda in funzione della soddisfazione del cliente al costo minimo;

- individuare e fissare ciò che viene fatto bene, affinché l'azienda possa concentrarsi sul miglioramento continuo dei processi che hanno impatto sulla qualità.

Emerge, quindi, la necessità di definire ruoli, interrelazioni e responsabilità di ogni partecipante, stabilire chi fa cosa e come lo fa, descrivere le attività svolte e documentarne i risultati, usufruendo di procedure e istruzioni scritte per ogni attività che abbia influenza sulla qualità.

Al punto 4.3 della ISO 9004:2000 sono elencati otto principi di gestione, pensati per fornire al management aziendale una guida al miglioramento delle prestazioni dell'organizzazione, apportando benefici in termini di valore, di ritorno monetario e di maggiore stabilità complessiva. I principi nascono dalla raccolta di *best practices* e dai pareri di esperti internazionali affinché i loro utilizzatori possano raggiungere il successo stabile delle organizzazioni per le quali lavorano. I concetti contenuti in questi otto principi costituiscono il fondamento su cui si basa l'intera famiglia delle norme ISO 9000 sui Sistemi di Gestione per la Qualità:

1. *Orientamento al Cliente*: le organizzazioni dipendono dai propri clienti e seguono le loro esigenze presenti e future, soddisfacendo i loro requisiti e mirando a superare le loro stesse aspettative. Benefici per l'organizzazione:

    - aumento del reddito e delle quote di mercato, grazie a una risposta più rapida e flessibile alle opportunità che il mercato stesso è in grado di offrire;

- maggiore efficacia nell'utilizzo delle risorse nel perseguire la soddisfazione dei clienti;
- maggiore fidelizzazione dei clienti, che porta non solo a una stabilità del business, ma a un suo potenziale incremento dovuto all'immagine aziendale positiva.

2. *Leadership*: la direzione generale stabilisce unità di intenti e di indirizzo dell'organizzazione. Essa crea e mantiene un ambiente interno che coinvolge pienamente il personale nel perseguimento degli obiettivi dell'organizzazione. Benefici per l'organizzazione:

   - maggiore compartecipazione e motivazione da parte di tutto il personale nel perseguire gli obiettivi e i traguardi dell'organizzazione;
   - maggiore trasparenza e coerenza nell'attuazione e successiva valutazione delle attività dell'organizzazione;
   - minor rischio che avvengano problemi di comunicazione tra i vari livelli dell'organizzazione.

3. *Coinvolgimento del personale*: le persone costituiscono l'essenza dell'organizzazione e il loro pieno coinvolgimento permette di porre le loro capacità al servizio dell'organizzazione. Benefici per l'organizzazione:

   - motivazione, rispondenza e coinvolgimento del personale nell'ambito dell'organizzazione;
   - stimolo all'innovazione e alla creatività nel raggiungimento degli obiettivi dell'organizzazione;
   - responsabilizzazione del personale per i compiti loro assegnati;
   - desiderio del personale di partecipare e contribuire al miglioramento continuo.

4. *Approccio per processi*: un risultato desiderato si ottiene con maggiore efficacia quando le relative attività e risorse sono gestite come un processo. Benefici per l'organizzazione:

   - minori costi e cicli più brevi, mediante un efficace uso delle risorse;
   - risultati migliori, coerenti e prevedibili;
   - occasioni per la messa a fuoco e la scelta delle priorità dei miglioramenti.

5. *Approccio sistemico alla gestione*: identificare, capire e gestire (come fossero un sistema) processi tra loro correlati contribuisce all'efficacia e alla efficienza dell'organizzazione. Benefici per l'organizzazione:

   - integrazione e allineamento dei processi per meglio favorire il raggiungimento dei risultati desiderati;

- capacità di mettere a fuoco i processi che più contano;
- dar fiducia alle parti interessate sulla solidità, efficacia ed efficienza dell'organizzazione.

6. *Miglioramento continuo*: il miglioramento continuo delle prestazioni complessive è un obiettivo permanente dell'organizzazione delle attività. Benefici per l'organizzazione:

   - vantaggi prestazionali attraverso migliorate potenzialità organizzative;
   - razionalizzazione delle attività di miglioramento a tutti i livelli, per perseguire gli obiettivi strategici dell'organizzazione;
   - flessibilità nel rispondere con prontezza alle opportunità che si presentano.

7. *Decisioni basate su dati di fatto*: le decisioni efficaci si basano sull'analisi di dati e di informazioni, gestite attraverso indicatori oggettivi e concordati con le persone coinvolte, sui principali parametri dei prodotti/processi. Benefici per l'organizzazione:

   - decisioni razionali, non basate esclusivamente su impressioni o intuizioni, ma su informazioni concrete;
   - maggior capacità nel dimostrare l'efficacia di precedenti decisioni, sulla base di situazioni di fatto;
   - miglior capacità di esaminare, confrontare e modificare opinioni e decisioni.

8. *Rapporti di reciproco beneficio cliente/fornitore*: il cliente e il suo fornitore sono interdipendenti e un rapporto di reciproco beneficio migliora, per entrambi, la capacità di creare valore. Benefici per l'organizzazione:

   - maggior capacità di creare valore, per entrambe le parti;
   - flessibilità e prontezza nel dare risposte congiunte al mutare del mercato o delle esigenze e aspettative dei clienti;
   - ottimizzazione di costi e risorse.

Capitolo 3

Modellizzazione di processi

In questo capitolo viene presentato un possibile formalismo adottato per la creazione del Workflow aziendale. Vengono descritti tutti gli elementi che compongono il formalismo e infine vengono illustrati esempi concreti tratti da procedure aziendali reali.

3.1 Modello di descrizione dei processi aziendali

Il primo passo per l'informatizzazione del sistema di gestione della qualità è la definizione di un modello descrittivo per la rappresentazione dei processi aziendali. In questa prima fase devono essere individuate tutte le esigenze aziendali con il fine di isolare le possibili soluzioni; esse devono essere caratterizzate da espressività descrittiva, presenza di tool per la gestione, semplicità nel disegno e nella lettura degli schemi. Per raggiungere un buon compromesso tra flessibilità ed espressività, analisti (coloro i quali studiano e propongono le possibili soluzioni) e utenti finali (chi realmente utilizzerà in futuro la soluzione adottata) devono collaborare tra di loro instaurando un canale di comunicazione; esso permette lo scambio di informazioni finalizzate alla ricerca e alla definizione del modello descrittivo.

Per modellare la rappresentazione dei processi aziendali in principio si era pensato di utilizzare lo standard UML[1].

Tuttavia questo impone i propri vincoli obbligando chi lo utilizza a rispettare regole a volte troppo restrittive. Per far fronte a queste limitazioni, si è pensato di impiegare quest'ultimo come base per un nuovo modello che si potesse adattare facilmente alle esigenze.

[1]UML (Unified Model Language) è un insieme di diagrammi formali e semiformali usato per aiutare il progettista informatico nella descrizione del problema e della soluzione. La prima versione di UML è stata sviluppata nel 1994 con lo scopo di fornire uno strumento unico di descrizione a supporto del lavoro dell'ingegnere del software. UML è continuamente aggiornato ed arricchito: la versione più recente è la 2.0.

È nata così l'idea di un formalismo estremamente flessibile e completo impiegabile mediante un qualsiasi software di image o process editing (nello specifico è stato impiegato Microsoft Visio), con il quale l'azienda può essere rappresentata come collezione di processi, costituiti da una sequenza di elementi base collegati tra di loro mediante relazioni di precedenza e relazioni tra dati, prevedendo di adottare una base di dati per salvare tutti gli elementi di un processo e le relazioni che li legano tra di loro e con i dati, cosicchè la navigazione di un processo si possa ridurre all'esecuzione di semplici query. L'impiego di un database comporta un ulteriore vantaggio: è, infatti, possibile instaurare una relazione diretta tra i dati strutturali e quelli runtime (informazioni che sono generate durante l'attraversamento di un processo). Si capisce, quindi, che la base di dati, che verrà illustrata nei capitoli successivi, è sostanzialmente divisa in due parti: una dedicata alla struttura dell'azienda e un'altra impiegata come punto di salvataggio di tutte le informazioni create durante l'utilizzo del sistema di gestione della qualità.

3.1.1 Descrizione del formalismo

La rappresentazione di un processo aziendale dev'essere realizzata mediante la stesura di un diagramma di control flow utilizzando gli elementi e le regole che seguono.

Start

Lo *Start* viene impiegato per identificare il punto di inizio del processo ed è fornito di più uscite. Nello stesso diagramma possono esistere più copie del blocco start così da facilitare la rappresentazione di processi molto complessi anche se esse vanno intese come un'unica entità. Sebbene vi possano essere più rappresentazioni di tale elemento l'abilitazione di uno di essi causerà l'attivazione di tutte le copie. Si può, quindi, dire che esso è logicamente unico.

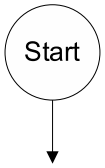

Figura 3.1. Rappresentazione grafica del blocco *Start*.

Attività

Il blocco *Attività* è un elemento fondamentale perché rappresenta l'azione che l'utente è chiamato a compiere per far si che l'attraversamento del diagramma

possa avanzare. Un'attività può produrre dati sensibili per il sistema, come, ad esempio, documenti, ed è, quindi, molto importante specificare quali dati sono ingresso e quali invece vengono prodotti. Si può inoltre intuire che una parte consistente di uno schema di processo aziendale è rappresentato da una "catena" di attività collegate tra loro da archi di precedenza, i quali, oltre a definire i percorsi, sottolineano il vincolo che impedisce di abilitare un'attività se quelle che la precedono non hanno terminato. Graficamente un'attività è rappresentata da un rettangolo dotato di:

- ingressi di flusso multipli;
- uscite di flusso multiple;
- dati in ingresso multipli;
- dati in uscita multipli.

La produzione di dati da parte di un'attività è un aspetto molto importante per l'evoluzione del processo e per questo essa non può essere considerata terminata se non ha prodotto tutti i dati in uscita; viceversa, un'attività non può essere abilitata se non dispone di tutti i dati in ingresso. Tali vincoli permettono di evitare possibili situazioni di stallo dovute alla mancanza di dati. Si crea, di conseguenza, oltre a un vincolo di precedenza, un vincolo sui dati.

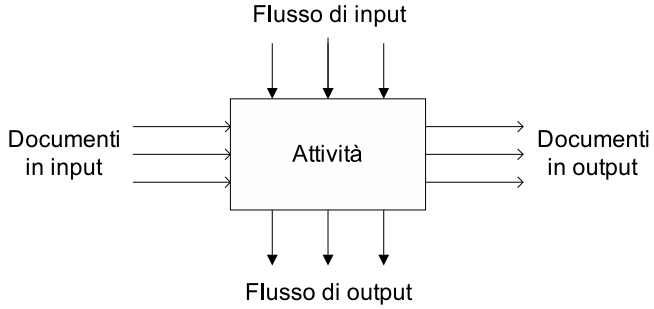

Figura 3.2. Rappresentazione grafica del blocco *Attività*.

Scelta

Gli oggetti *Scelta* vengono impiegati per rappresentare un punto di decisione. Questo elemento permette di decidere il percorso di attraversamento del diagramma sulla base di una scelta; una volta che questa è stata presa, solo l'uscita di flusso ad essa associata sarà abilitata. Il blocco di scelta è rappresentato attraverso un rombo dotato di un unico ingresso di flusso e n uscite, una per ogni possibile esito della decisione.

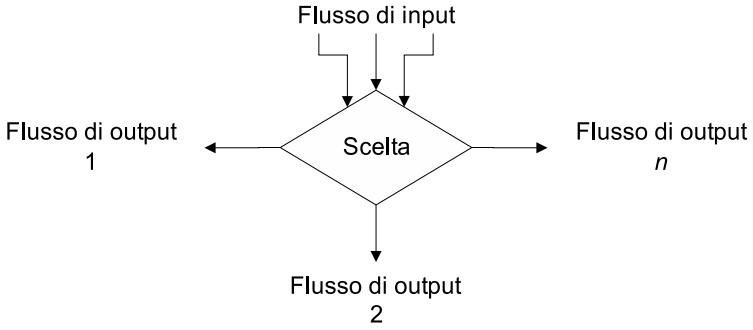

Figura 3.3. Rappresentazione grafica del blocco *Scelta*.

Or

Il blocco *Or* permette di fondere due o più archi di precedenza per formarne uno unico. Tale blocco ha n ingressi e un'unica uscita; il nome ne rappresenta la caratteristica base, esso, infatti, rispetta le regole dell'omonimo operatore logico, ovvero viene abilitata l'uscita quando almeno un ingresso è attivo.

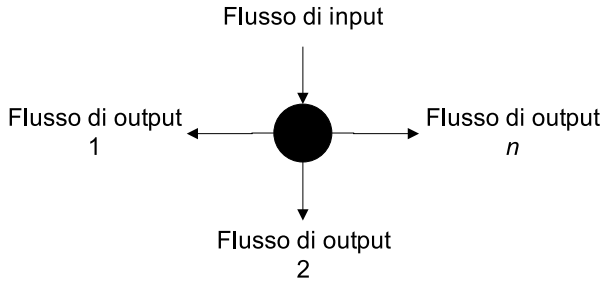

Figura 3.4. Rappresentazione grafica del blocco *Or*.

Finish

Il *Finish* rappresenta il concetto di conclusione con successo che si manifesta solamente se il processo termina raggiungendo lo scopo per il quale è stato creato. Il blocco finish è dotato di n ingressi e nessuna uscita e, come per il blocco *Start*, è possibile duplicarlo durante la stesura del diagramma per facilitare la schematizzazione dei processi complessi, ricordando, però, che esso è logicamente unico.

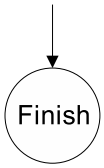

Figura 3.5. Rappresentazione grafica del blocco *Finish*.

Stop

Diversamente dal finish, il blocco *stop* viene impiegato nel momento in cui il flusso di attraversamento incorre in un'eccezione che ne provoca la conclusione senza successo. Il blocco stop è dotato di n ingressi e nessuna uscita e, come per il blocco *Start*, è possibile duplicarlo durante la stesura del diagramma per facilitare la schematizzazione dei processi complessi sempre ricordandone l'unicità logica.

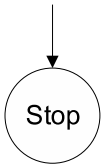

Figura 3.6. Rappresentazione grafica del blocco *Stop*.

Subprocess

Questo elemento consente di passare ad un livello inferiore (livello attuale +1). Esistono, infatti, casi in cui un processo, per poter terminare, necessiti di dati che non gli competono dovendo, quindi, avviare un sottoprocesso che glieli possa fornire. Per questo motivo, la rappresentazione grafica di un processo aziendale può essere distribuita su più livelli. Il blocco è rappresentato da un rettangolo con due fasce laterali dotato di n ingressi, logicamente in *and* tra di loro e un'unica uscita; all'interno del rettangolo sarà riportato il nome del sotto processo da avviare. Il passaggio ad un livello superiore (livello attuale -1) è implicitamente rappresentato dal blocco finish, infatti, a processo ultimato, per conoscere il percorso da seguire nella "catena di produzione" è necessario passare ad un livello meno dettagliato in cui sono descritte le relazioni tra processi (livello zero).

3.1.2 Attività dettagliabile

Questo elemento rappresenta un tipo di attività che può essere espansa ricorsivamente, di cui al momento del suo disegno non si conoscono ancora le azioni che lo

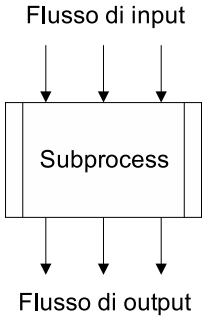

Figura 3.7. Rappresentazione grafica del blocco *Subprocess*.

"popoleranno". L'unico vincolo che si pone al disegno di questo tipo di attività è che al momento del disegno, essa deve stabilire quali documenti dovranno essere prodotti e quali documenti dovranno essere consultati per la sua esecuzione. Si è ritenuto importante avere questo elemento per offrire la necessaria flessibilità a processi complessi di cui non si conoscono a priori i dettagli, ovvero il flusso di esecuzione e i documenti prodotti.

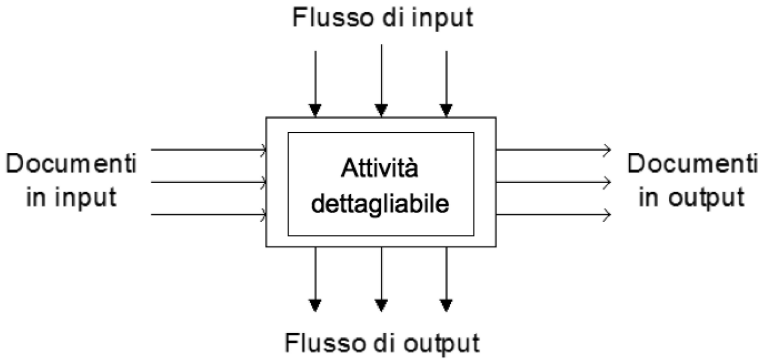

Figura 3.8. Rappresentazione grafica del blocco *Attività dettagliabile*.

Oltre ai blocchi che compongono il flusso del processo, sono stati definiti anche degli oggetti fondamentali per l'interazione tra attività e utenti.

Documento

Rappresenta un elemento che ha sede in un repository oppure, più semplicemente, in un archivio interno all'azienda. Un documento può essere soggetto a revisioni ognuna identificata da un numero sequenziale.

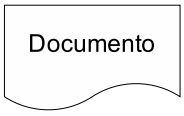

Figura 3.9. Rappresentazione grafica del blocco *Documento*.

Registrazione

Rappresenta un elemento destinato a un supporto digitale, ad esempio un database, e per il quale non è prevista la revisione.

Figura 3.10. Rappresentazione grafica del blocco *Registrazione*.

Notifica

È un documento che non può essere soggetto a revisione. Un esempio di notifica sono le e-mail inviate ai clienti per comunicare l'avvenuta apertura di un ticket.

Figura 3.11. Rappresentazione grafica del blocco *Notifica*.

Documento multiplo

È un documento del quale non si conosce a priori il nome o la quantità di documenti che rappresenta. Esempio di documento multiplo sono i file sorgenti oppure la documentazione tecnica.

Figura 3.12. Rappresentazione grafica del blocco *Documento multiplo*.

Gli elementi descritti devono essere direttamente collegati alle attività utilizzando degli archi orientati grazie ai quali è possibile specificare se sono prodotti (Attività —> Documento/Registrazione/Notifica/Documento Multiplo) o utilizzati (Attività <— Documento/Registrazione/Notifica/Documento Multiplo). Nel diagramma di schematizzazione di un processo aziendale va riportato il responsabile di processo. In aggiunta a ciò vanno esplicitati i responsabili di ogni elemento del diagramma, cioè coloro che possono compiere l'azione riportata nelle attività o prendere decisioni nel caso di una scelta. La specifica del responsabile di elemento deve avvenire mediante delle bande funzionali, le quali riporteranno in una fascia laterale il ruolo del responsabile di elemento. La stesura del diagramma di control flow deve inoltre rispettare le seguenti scelte stilistiche:

- gli archi di precedenza devono avere uno spessore maggiore rispetto a quelli che collegano documenti, registrazioni e notifiche con le attività;
- il testo deve avere una dimensione che permetta una facile lettura senza quindi utilizzare font o dimensioni non adatte ad un documento tecnico;
- le bande funzionali devono essere disegnate impiegando una linea tratteggiata con spessore inferiore rispetto a quello scelto per gli archi di precedenza.

3.1.3 Esempi di applicazione

Per fare chiarezza sul formalismo appena descritto vengono mostrati alcuni esempi in vengono mappate, tramite le regole sopra definite, alcune sezioni dei processi aziendali che verranno illustrati in maniera approfondita nella prossima sezione.

In figura 3.13 si ha una sezione del processo *Product Management*, dove si osservano quattro tipi di oggetti differenti:

- subprocess;
- scelta;
- or;
- stop.

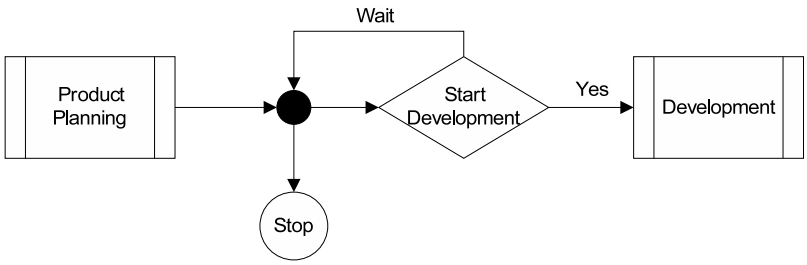

Figura 3.13. Esempio di utilizzo del formalismo tratto dal processo *Product Management*.

La sezione comincia con il sottoprocesso *Product Planning*: in questo caso il processo chiamante non avanza finchè il sottoprocesso non ha terminato la sua esecuzione. Successivamente l'Executive Board deve validare il progetto in via di sviluppo. Tale approvazione è rappresentata da un blocco di scelta con tre possibili uscite:

- se al momento non è possibile procedere allo sviluppo per qualche motivo il progetto viene messo in attesa;
- se non ci sono problemi si procede con il sottoprocesso *Development*;
- se l'Executive Board decide di non validare il progetto il processo termina prematuramente.

Analizzando in dettaglio le uscite dell'oggetto appena considerato si notano due particolari casi. Il primo è rappresentato dal ciclo nel diagramma di control flow: in questa circostanza, per ricongiungersi alla freccia di flusso entrante nel blocco *Scelta* viene utilizzato un oggetto di tipo *Or*. Di fatto, questo elemento di congiunzione è molto usato per creare iterazioni nell'esecuzione dei processi. Il secondo caso particolare è rappresentato dalla terminazione del processo tramite blocco *Stop*.

In figura 3.14 si ha una sezione del processo Development, dove si notano cinque tipi di oggetti differenti:

- attività;
- scelta;
- documento;
- registrazione;
- or;

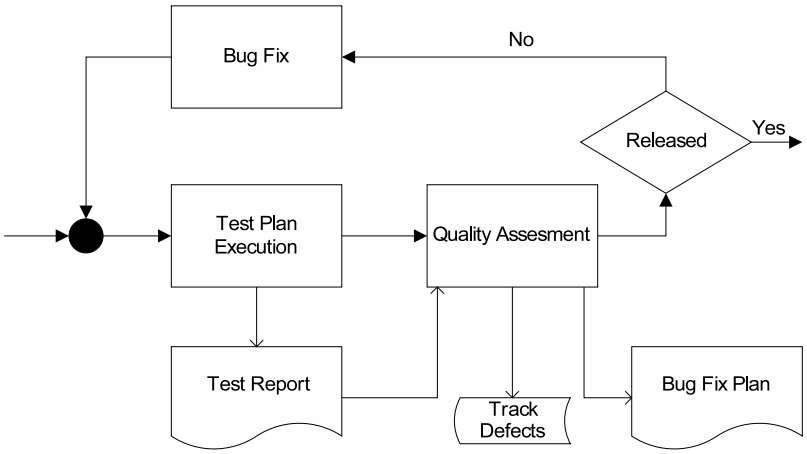

Figura 3.14. Esempio di utilizzo del formalismo tratto dal processo *Development*.

La sezione comincia con l'attività *Test Plan Execution*, che produce il documento *Test Report*. La porzione di processo continua con l'attività *Quality Assessment*, che riceve in ingresso il documento appena prodotto e restituisce in uscita *Bug Fix Plan* e *Track Defects*. Successivamente, si arriva ad un blocco di scelta dove si decide se rilasciare il prodotto o eseguire delle correzioni. In quest'ultimo caso viene avviata l'attività *Bug Fix* e la sezione ricomincia dall'inizio tramite un elemento di tipo *Or*.

3.1.4 Considerazioni

Durante la fase di ricerca del tool per il supporto al formalismo ed anche nella vera e propria ricerca di un modello da utilizzare sono state trovate molte soluzioni valide. Esse, però, si rivelavano, spesso, troppo ricche di informazioni rendendole, di conseguenza, difficili da apprendere. La volontà era quella di realizzare un sistema che non crei disagi all'azienda che decide di utilizzarlo; sulla base di ciò, non si poteva sviluppare una soluzione che richiedesse uno step iniziale troppo impegnativo o che andasse a modificare la struttura aziendale. Per questi motivi, l'utilizzo di un modello generico, ma estremamente semplice (pochi elementi regolati da semplici vincoli), e di un sistema di salvataggio privo di regole (si possono creare tutte le relazioni desiderate una volta inseriti gli elementi da utilizzare) si rivela la soluzione più adatta. È importante ricordare che il lavoro di modellizzazione si è svolto dandogli il giusto tempo nella speranza di aver coperto e risolto tutti i punti critici che possono incorrere nella descrizione formale dei processi di un'azienda.

3.2 Esempi di processi aziendali

Nel seguito si possono trovare esempi di processi disegnati con il formalismo adottato, tratti da situazioni concreti aziendali.

3.2.1 Sales Management

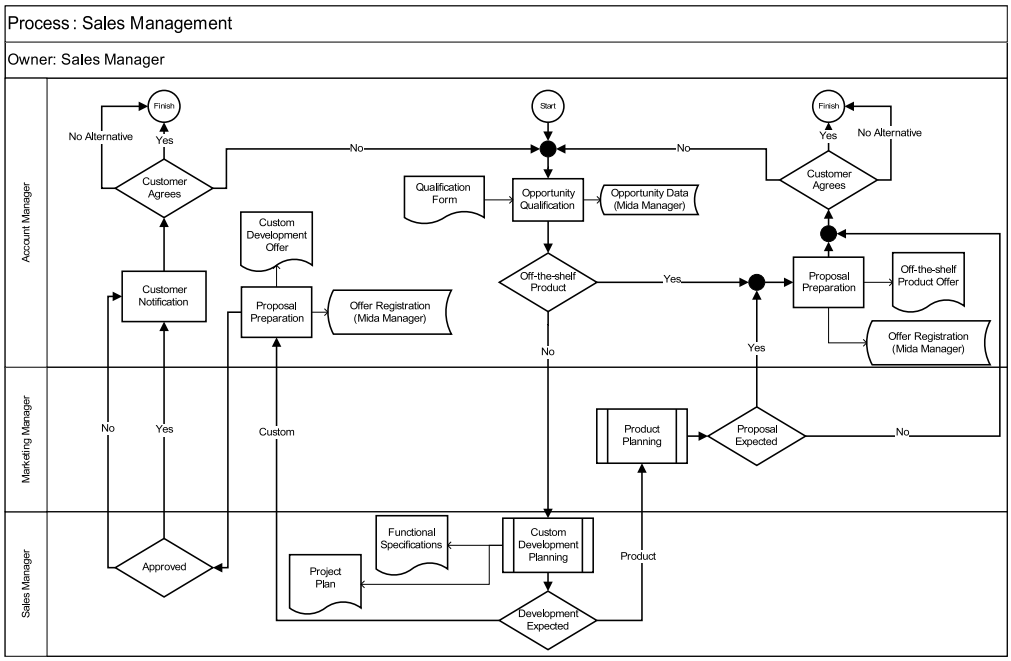

Figura 3.15. Rappresentazione grafica del processo Sales Management.

Il processo *Sales Management* (vedi figura 3.15) ha lo scopo di generare le offerte di prodotto sulla base delle richieste avanzate dai clienti. La fase di valutazione di quest'ultime può intraprendere percorsi diversi, in quanto le funzionalità richieste dal cliente potrebbero essere già offerte da un prodotto esistente, oppure semplicemente richiedere una personalizzazione se non come ultimo lo sviluppo di un nuovo prodotto. Al termine della prima fase, nella quale vengono definite le funzionalità che il prodotto dovrà fornire, si può passare direttamente all'emissione dell'offerta se la richiesta del cliente ha permesso di identificare un prodotto già disponibile. In caso contrario, viene fatta una valutazione sul piano economico e tecnico, la quale prevede un'analisi di opportunità e di fattibilità, con cui viene deciso se realizzare un nuovo prodotto oppure apportare delle modifiche ad uno già esistente; in entrambi i casi sarà prodotta un'offerta per il cliente.

Documenti in ingresso e in uscita

In ingresso al processo si ha:

- *Qualification Form*: è il documento che viene compilato, sulla base di un template, nel momento in cui arriva la richiesta del cliente e che contiene tutte le informazioni utili per valutare le funzionalità del prodotto richiesto.

In uscita al processo si ha:

- *Off-the-shelf Product Offer*: è il documento che contiene l'offerta di prodotto, nel caso in cui il cliente abbia richiesto un prodotto già esistente;
- *Custom Development Offer*: è il documento che contiene l'offerta di prodotto, nel caso in cui venga realizzato un nuovo prodotto;
- *Functional Specifications*: le specifiche funzionali per lo sviluppo del prodotto; esso viene generato solamente se la richiesta del cliente non ha identificato un prodotto già esistente;
- *Project Plan*: è il documento che contiene il piano di progetto per lo sviluppo del prodotto; esso viene generato solamente se la richiesta del cliente non ha identificato un prodotto già esistente.

Particolarità del processo

La particolarità del processo si ha nella parte finale; si può notare che un'offerta "respinta" dal cliente non porta ad una modifica di quest'ultima, ma provoca la terminazione e la conseguente riattivazione del processo con in ingresso le nuove richieste. In questo modo ogni offerta di prodotto, anche se riferita alla stessa richiesta dello stesso cliente, non verrà mai perduta.

3.2.2 Custom Development Planning

Il processo *Custom Development Planning* (figura 3.16) ha la funzione di valutare le commesse, ovvero dei prodotti personalizzati su richiesta diretta dei clienti, sul piano economico e tecnico, prevede infatti un'analisi di opportunità e di fattibilità. Il processo nasce da una richiesta specifica di un cliente. A questo punto, viene assegnato un projet manager per gestire la commessa. Questo, preso contatto con il cliente, fa un'analisi tecnica-economica da cui nasce l'*Opportunity plan*. Mediante quest'ultimo documento la direzione decide se procedere oppure terminare il processo. In caso positivo, il project manager, conciliando le richieste del cliente con le sue possibilità in campo tecnico, produce un documento contenente i requisiti della propria soluzione. A questo punto, ripresi i contatti con il cliente, negozia i requisiti definitivi e i tempi di sviluppo. Superata l'analisi di fattibilità con le condizioni del cliente, la decisione finale spetta all'executive board : sia in caso che si scelga di procedere con lo sviluppo o meno, il processo

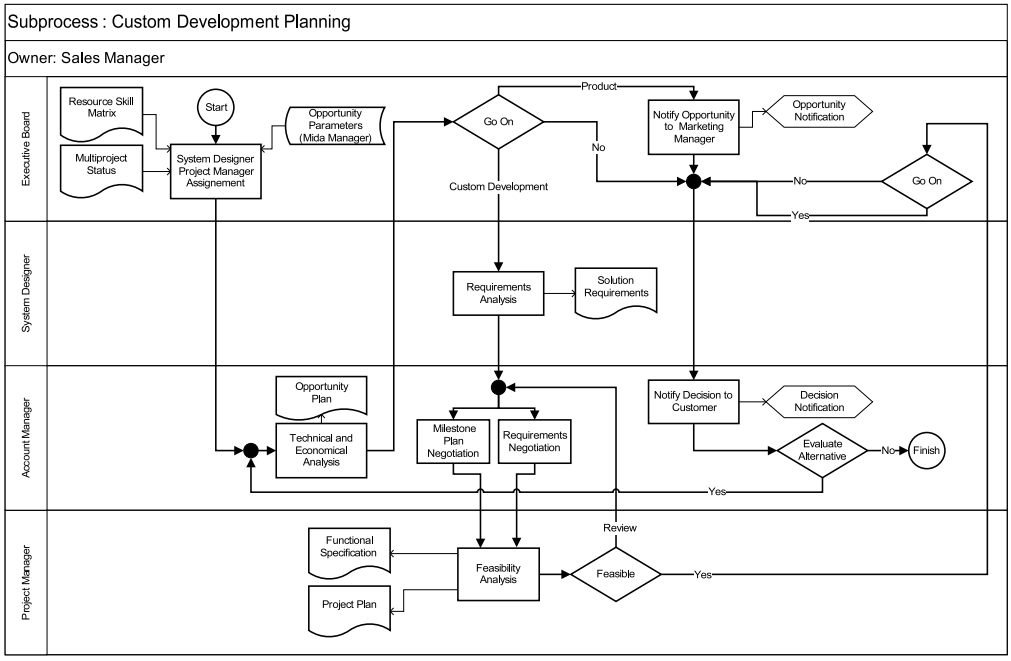

Figura 3.16. Rappresentazione grafica del processo Customer Development Planning.

termina con una notifica al cliente, il quale potrà, in caso di rifiuto, rivedere la sua richiesta.

Documenti in ingresso e in uscita

I documenti in ingresso sono :

- *Resource Skill Matrix*: contiene informazioni a proposito delle capacità specifiche dei singoli sviluppatori
- *Multiproject Status*: contiene informazioni sullo stato dei progetti in corso d'opera
- *Opportunity Parameters*: contiene informazioni a proposito dei prodotti già presenti in listino

I documenti in uscita sono :

- *Opportunity Plan*: contiene un'analisi economica e tecnica della richiesta del cliente
- *Solution Requirements*: contiene i requisiti della *bozza* di prodotto, in quanto questi verranno ridiscussi con il cliente

- *Functional Specification*: contiene le specifiche funzionali per lo sviluppo del prodotto custom

- *Project Plan*: contiene il piano di progetto per lo sviluppo del prodotto custom

Particolarità del processo

A differenza dello sviluppo prodotti, queste soluzioni su misura per clienti con specifiche richieste sono portate a termine in genere da un solo sviluppatore che viene scelto in base alla sua disponibilità e alle sue capacità. Dopo l'approvazione dell'*opportunity plan*, può capitare che venga alla luce una buona idea per un prodotto : in questo caso il processo termina con una notifica di rifiuto al cliente che ne aveva effettuato la richiesta e un'altra al marketing che segnala l'opportunità di valutare un nuovo prodotto generico. La produzione dei requisiti del prodotto custom non si basa solo sulle richieste del cliente, ma su un compromesso tra queste e la fattibilità di esse sul piano tecnico. Dopo aver definito le caratteristiche della soluzione proposta dallo sviluppatore, queste vengono discusse e concordate con il cliente.

3.2.3 Product Planning

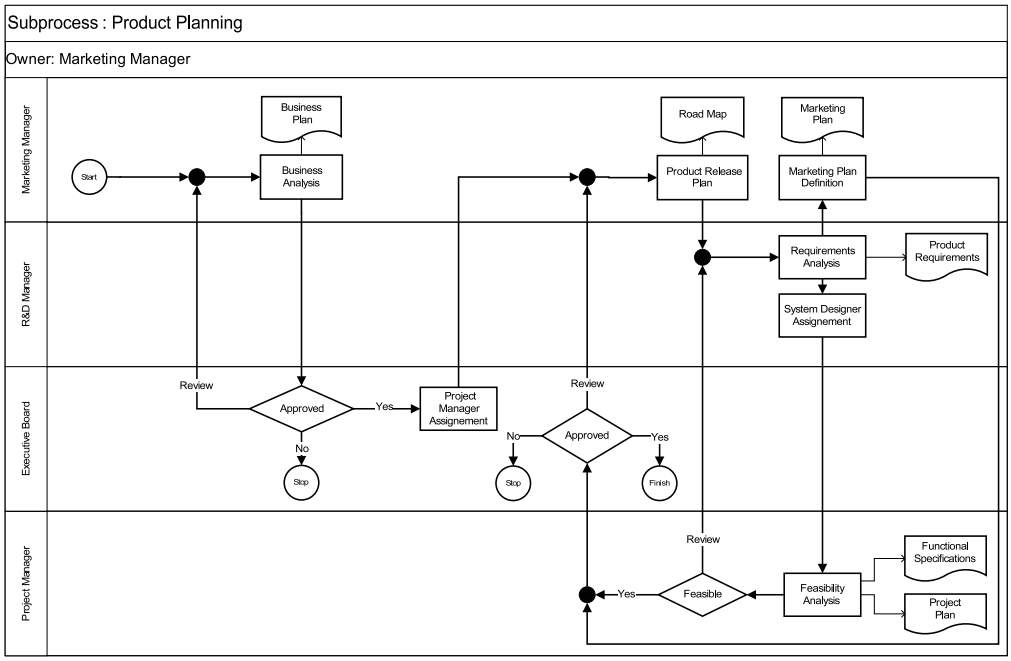

Figura 3.17. Rappresentazione grafica del processo Product Planning.

Il processo *Product planning* (figura 3.17) si occupa di effettuare un'analisi approfondita sull'opportunità di creazione di un nuovo prodotto. Se questa mette in evidenza buone possibilità di vendita e guadagno, al termine del processo vengono definiti i parametri per lo sviluppo. Il processo comincia con la redazione del *business plan*, il quale dà un'indicazione all'executive board se continuare, magari rivedendo alcuni dettagli, con l'assegnazione del project manager. Dopo la creazione della *road map*, vengono definiti i requisiti del prodotto, ovvero le funzioni che questo dovrà implementare. A questo punto il lavoro procede sul piano del marketing con la creazione del *marketing plan*, e sul piano tecnico con la redazione di *specifiche funzionali* e *piano di progetto*. L'executive board, basandosi su questi ultimi tre documenti, decide se cominciare lo sviluppo anche revisionando certi documenti e procedure, oppure se fermare il processo e non procedere alla realizzazione del prodotto.

Documenti in ingresso e in uscita

Per questo processo non sono previsti documenti in ingresso. In uscita ne troviamo diversi :

- *Business Plan*: contiene una previstione dei tempi di sviluppo, dei costi e dei possibili profitti
- *Road Map*: contiene un diagramma temporale che dà un'indicazione sui tempi di sviluppo del prodotto
- *Marketing Plan*: contiene i prezzi, il piano di comunicazione e la definizione dei canali di vendita
- *Product Requirements*: contiene i requisiti che il prodotto deve soddisfare
- *Functional Specification*: contiene le specifiche funzionali per lo sviluppo
- *Project Plan*: contiene il piano di progetto per lo sviluppo

Particolarità del processo

La *Road map* è uno documento molto utile, perchè permette sia al marketing, che deve presentare il prodotto sul mercato, sia agli sviluppatori, che devono lavorare su diverse parti del progetto, di avere delle scadenze temporali abbastanza precise. In realtà, durante l'avanzamento del progetto, questo documento viene revisionato molte volte, anche in base a cambiamenti in corso d'opera sul prodotto stesso. Un'altro aspetto da evidenziare è una delle fasi finali del processo : prima di sottoporre il progetto al giudizio finale dell'executive board, sono i tecnici stessi che ne fanno un'analisi di fattibilità e possono richiedere una revisione dei requisiti di prodotto, senza interpellare direttamente la direzione, nel caso il progetto da sviluppare non sia fattibile.

3.2.4 Product Management

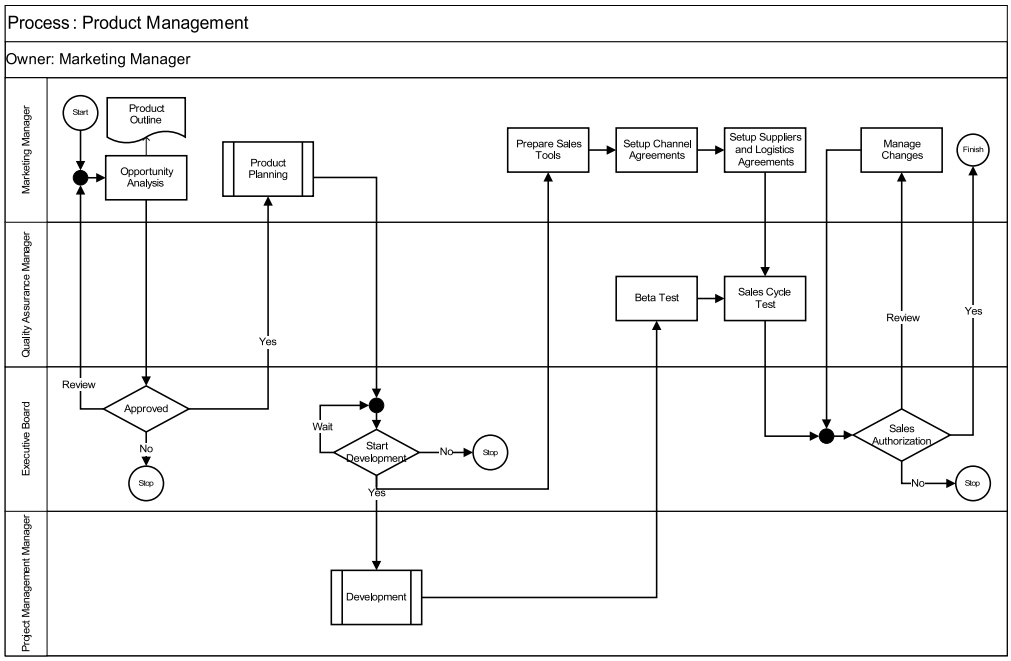

Figura 3.18. Rappresentazione grafica del processo Product Management.

Il processo *Product management* (figura 3.18) descrive l'intero ciclo di creazione di un prodotto, dall'analisi di opportunità al lancio sul mercato. Il processo comincia con un'analisi d'opportunità, ovvero una valutazione generale di un'*idea* per un nuovo prodotto. Se questa viene accettata viene eseguito il sottoprocesso *Product planning* che esamina più a fondo le caratteristiche che deve avere il nuovo prodotto e dà le direttive per il suo sviluppo. Dopodichè cominciano simultaneamente i beta test sul piano tecnico, mentre il marketing si occupa di accordarsi con i canali di vendita per il lancio del prodotto. Infine, dopo l'approvazione dell'executive board, il prodotto viene rilasciato nel mercato. In caso di revisione da parte della executive board al momento dell'autorizzazione, il marketing manager deve effettuare i cambiamenti proposti e far rivalutare il prodotto prima di procedere ad un eventuale rilascio nei canali di distribuzione.

3.2.5 Development

Il processo *Development* (figura 3.19) ha come scopo lo sviluppo di un prodotto e quindi la realizzazione del software corredato dalla documentazione tecnica e di release. La fase iniziale del processo prevede un'analisi del personale, cioè una valutazione delle competenze e della disponibilità di ogni sviluppatore, così da poter selezionare il personale che costituirà la risorsa umana del progetto.

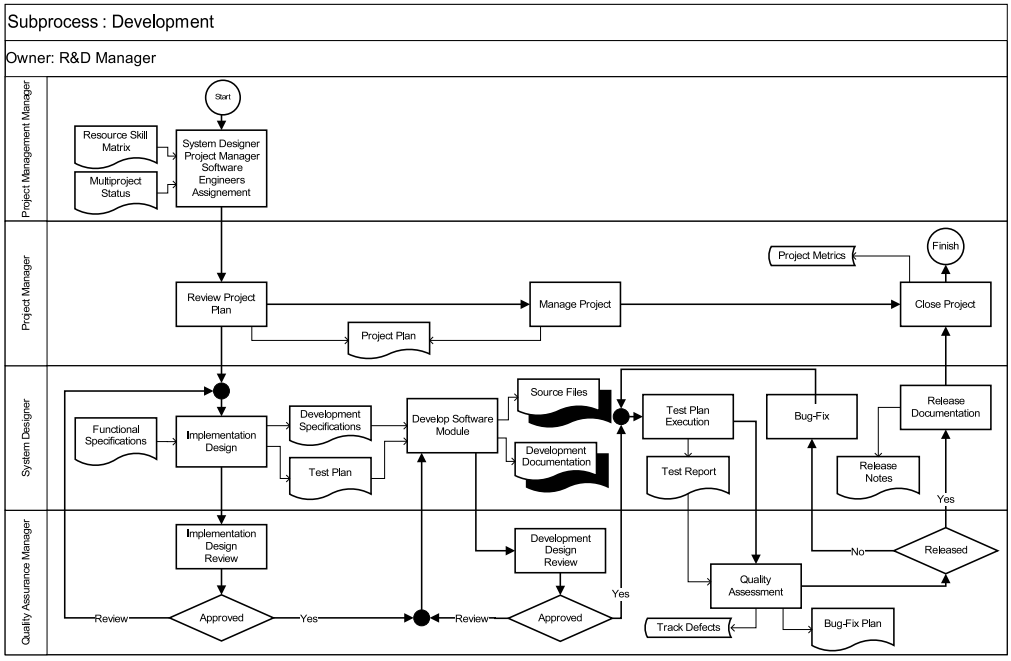

Figura 3.19. Rappresentazione grafica del processo Development.

A seguito di questa prima fase, vi è la definizione delle specifiche di sviluppo (costituisce la suddivisione, per priorità, delle funzione da sviluppare che verranno successivamente distribuite sull'asse temporale) e la pianificazione della fase di test. Terminata l'analisi, si passa allo sviluppo del prodotto durante il quale viene realizzata, oltre al codice sorgente, la documentazione tecnica. Conclusa quest'ultima fase, viene compiuto un lavoro di test sul materiale prodotto generando, a bug risolti, release del prodotto le quali saranno completate dalla documentazione di release prima del rilascio.

Documenti in ingresso e in uscita

In ingresso al processo si ha:

- *Resource Skill Matrix*: è il documento che contiene una matrice di competenze la quale riporta tutte le competenze di ogni individuo del personale;
- *Multiproject Status*: è il documento dal quale è possibile conoscere lo stato d'avanzamento di ogni progetto;
- *Functional Specifications*: è il documento che contiene tutte le informazioni utili sulle funzionalità che il prodotto dovrà fornire.

In uscita al processo si ha:

- *Project Plan*: è il documento che contiene il piano di progetto per lo sviluppo del prodotto;
- *Development Specifications*: è il documento che contiene il piano di sviluppo e quindi la suddivisione, per priorità, delle funzione da sviluppare e con relativa distribuzione sull'asse temporale;
- *Source Files*: costituiscono i file sorgenti del porgetto;
- *Development Documentation*: costituisce la documentazione tecnica del progetto;
- *Test Plan*: è il documento che contiene il piano di test e quindi la divisione temporale delle funzionalità da verificare;
- *Test Report*: è il documento che contiene il report di ogni test così da conservare lo storico del testing;
- *Bug-Fix Plan*: è il documento che contiene la suddivisione, per priorità, dei bug da risolvere e con relativa distribuzione sull'asse temporale;
- *Release Notes*: è il documento che contiene tutte le informazioni della release alle quali sono associate.

3.2.6 Customer Support

Il processo *Customer Support* (figura 3.20) ha lo scopo di dare assistenza sui prodotti una volta che questi vengono venduti al cliente; l'assistenza viene garantita solo in presenza di un contratto di assistenza valido. Quindi, tutte le segnalazioni di mal funzionamenti o richieste di assistenza rappresentano l'input di questo processo; da notare che quasi sempre l'azienda non tratta direttamente con il cliente ma con i canali di vendita intermedi. Una volta che la segnalazione è giunta, viene fatta una prima analisi per cercare di individuare il probema; fatto questo, viene aperto un ticket all'interno di un sistema dedicato in modo da tracciare il lavoro che verrà eseguito. Se il contratto di assistenza è presente e ancora valido, si cerca di risolvere il problema e una volta risolto se ne chiede una conferma da parte del cliente; se tutto è a posto si chiude il ticket aperto all'inzio del processo e questo termina.

Documenti in ingresso e in uscita

In ingresso al processo si hanno due documenti:

- *Signaling* : è il documento contenente la segnalazione da parte del cliente (o più spesso dal canale di vendita) di un guasto o comunque di un qualcosa che richiede l'intervento dei tecnici dell'azienda. La segnalazione generalmente arriva via mail oppure via fax, l'importante ovviamente è che sia una segnalazione scritta;

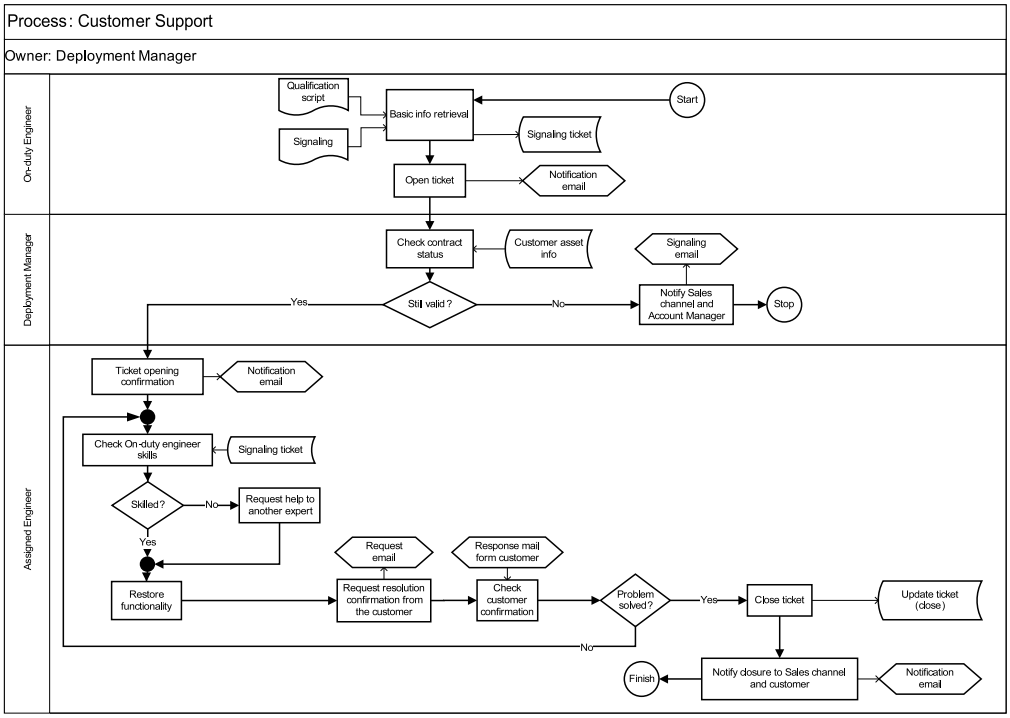

Figura 3.20. Rappresentazione grafica del processo Customer Support.

- *Qualification Script* : è un documento contenente una serie di domande standard che hanno l'obiettivo di ottenere quante più informazioni possano essere utili ad individuare e risolvere il problema riscontrato (alcune informazioni tipiche che si cercano di ottenere attraverso il questionario sono il riferimento del cliente, il tipo di problema riscontrato oppure che genere di informazioni si vuole).

In uscita non si ha nessun documento ma solo l'aggiornamento del ticket chiudendolo e la mail di notifica verso il cliente per notificargli l'avvenuta chiusura della sua segnalazione.

Particolarità del processo

I passaggi importanti di questo processo si hanno all'inizio quando viene aperto il ticket della segnalazione; il ticket permetterà di tracciare la segnalazione e tutto il lavoro che ne deriverà; inoltre, l'identificativo del ticket viene comunicato anche al cliente. Altro punto molto importante, dato che determina se un cliente ha diritto ad avere assistenza oppure no, è il controllo di validità del contratto di assistenza: se questo è ancora valido allora si procede con la ricerca di una soluzione al problema, altrimenti verrà fatta una segnalazione al canale di vendita

(che tratta con il cliente) e al commerciale interno all'azienda. Ultimo passaggio importante è quello alla fine del processo, una volta che è stato risolto il problema; infatti viene richiesta una conferma da parte del cliente della reale risoluzione del problema e in caso affermativo si comunica a quest'ultimo la chiusura del ticket relativo alla sua segnalazione.

3.2.7 Order Management

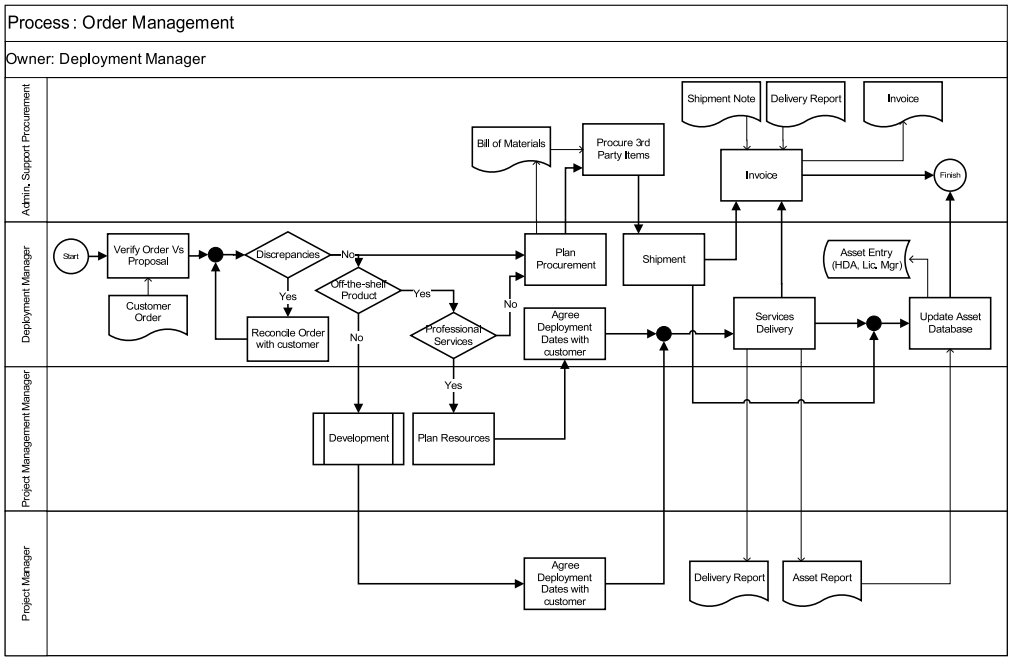

Figura 3.21. Rappresentazione grafica del processo Order Management.

Il processo *Order Management* (figura 3.21) ha lo scopo di predisporre la spedizione dei prodotti e organizzare gli eventuali servizi preofessionali verso i clienti che ne fanno richiesta; per servizi professionali si intendono ad esempio dei corsi tenuti da tecnici dell'azienda per insegnare a usare un particolare prodotto oppure la disponibiltà di tecnici per un certo periodo verso il cliente. Una volta ricevuto l'ordine si fa un controllo sulla correttezza di questo ed eventuali differenze vengono discusse con il cliente; una volta che l'ordine è a posto si passa al reperimento del materiale se questo è disponibile oppure al suo sviluppo se così non fosse ed inoltre, se richiesti, si procede anche nella pianificazione del calendario per i servizi professionali (calendario che verrà poi discusso anche con il cliente). Dopo aver preparato tutti i documenti si passa alla spedizione e all'erogazione dell'eventuale servizio richiesto; infine, si avranno di ritorno la fattura e un resoconto su quello che è stato fatto durante il servizio.

Documenti in ingresso e in uscita

In ingresso al processo troviamo i seguenti documenti:

- *Customer Order* : è il documento contenente la lista dei prodotti richiesti dal cliente; una volta verificata la sua correttezza si passa alla fase successiva, ovvero al reperimento del materiale vero e proprio.

- *Bill of Materials* : viene prodotto e poi usato all'interno del processo, per questo motivo si trova sia in ingresso che in uscita ad esso. Questo documento (derivato dal documento precedente ovvero *Customer order*) contiene la lista definitiva dei prodotti richiesti dal cliente; ci si basa su di esso per il reperimento del materiale vero e proprio;

- *Shipment Note* : documento contenente le note riguardanti la spedizione del materiale; esso è utile in fase di fatturazione del materiale;

- *Delivery Report* : viene prodotto e poi usato all'interno del processo, per questo motivo si trova sia in ingresso che in uscita ad esso. Questo documento è la ricevuta che attesta che il materiale è stato correttamente consegnato al cliente; inoltre, nel caso in cui sia stato anche richiesto un servizio professionale, in questo documento è anche presente un resoconto su che cosa è stato fatto.

In uscita al processo invece possiamo trovare più documenti:

- *Bill of materials* : viene prodotto e poi usato all'interno del processo, per questo motivo si trova sia in ingresso che in uscita ad esso. come spiegato prima, questo contiene la lista definitiva dei prodotti richiesti dal cliente; ci si basa su di esso per il reperimento del materiale vero e proprio.

- *Delivery report* : viene prodotto e poi usato all'interno del processo, per questo motivo si trova sia in ingresso che in uscita ad esso. Questo documento è la ricevuta che attesta che il materiale è stato correttamente consegnato al cliente; inoltre, nel caso in cui sia stato anche richiesto un servizio professionale, in questo documento è anche presente un resoconto su che cosa è stato fatto;

- *Invoice (Fattura)* : è un documento fiscale che viene emesso dall'azienda e che attesta che è stato venduto un qualcosa ad un particolare cliente e che quindi essa ha il diritto ad essere pagata per questo;

- *Asset report* : documento contenente la lista dei software che sono stati installati sulla macchina del cliente da parte del tecnico; questo documento è relativo solo al caso in cui ci sia stato un servizio professionale.

Oltre a questi documenti, prima della conclusione del processo, avviene l'aggiornamento o la creazione di un entry su database dedicati per tenere traccia del software installato sulla macchina del cliente nel caso in cui questo sia stato installato dai tecnici.

Particolarità del processo

Dopo che la lista degli acquisti è stata confermata può capitare che uno o più prodotti non siano disponibili o perchè sono esaurite le scorte oppure perchè sono prodotti ancora non esistenti; nel secondo caso si passa al processo di *Development* nel quale si andrà a progettare e poi, dopo tutte le valutazioni del caso, a produrre il prodotto richiesto.

3.2.8 Bug Management

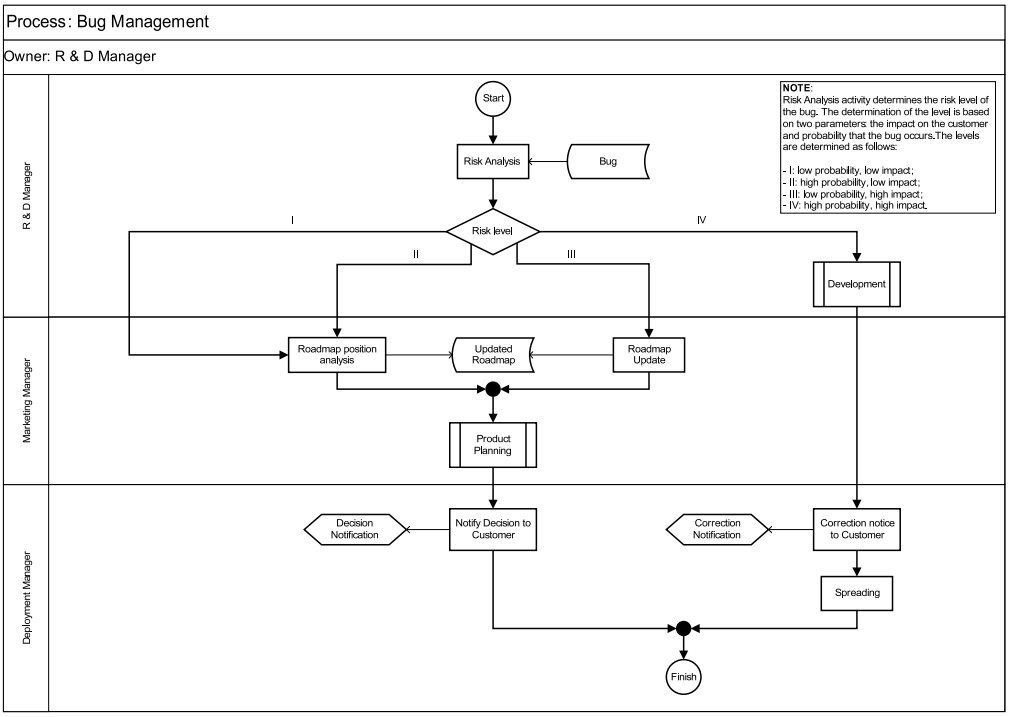

Figura 3.22. Rappresentazione grafica del processo Bug Management.

L'obiettivo del processo *Bug Management* (figura 3.22) è gestire la soluzione dei bug software che vengono segnalati dall'assistenza clienti o dal reparto di sviluppo. Il bug da correggere viene sottoposto ad un'analisi di rischio (vedi figura 3.23) che si basa essenzialmente su due parametri: la probabilità che il bug si verifichi (intesa come frequenza) e l'impatto che il bug ha sul cliente.

Come risultato dell'analisi, viene associato al bug un livello di rischio in base al quale si deciderà come operare. I livelli vengono determinati come segue:

- *Livello I*: probabilità e impatto bassi. I bug in questa categoria hanno un livello di rischio minimo e, quindi, richiedono un intervento a bassa priorità. Questi verranno, quindi, risolti solamente quando saranno concluse le attività di sviluppo a priorità maggiore.
- *Livello II*: alta probabilità ma basso impatto. Anche se l'impatto di questi bug sul cliente non è alto, l'alta frequenza con cui essi si presentano suggerisce un intervento non troppo lento. In questo caso viene, quindi, aggiornata la *Roadmap* stabilendo quando si andrà ad intervenire per risolvere il problema.
- *Livello III*: probabilità bassa con alto impatto. Come con il livello di rischio precedente, si procede con l'aggiornamento della *Roadmap*, ma, in questo caso, la volontà è quella di risolvere il bug entro la data di rilascio della release successiva del prodotto.
- *Livello IV*: probabilità e impatto alti. I bug che ricadono in questa categoria richiedono un intervento immediato da parte dell'azienda. Viene subito attivato un processo di sviluppo della patch correttiva che andrà, poi, distribuita ai clienti.

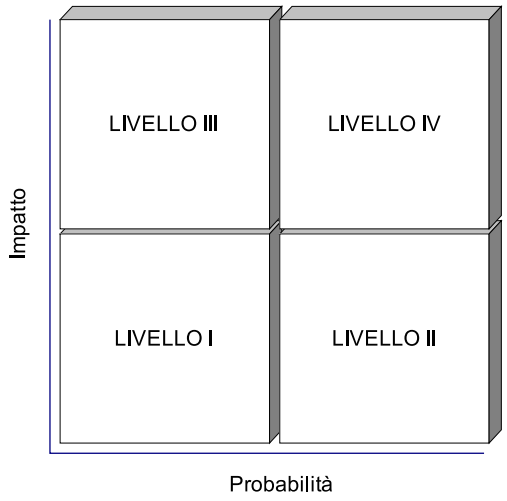

Figura 3.23. Schema per la determinazione del livello di rischio di un bug.

Ritornando alla procedura seguita per la risoluzione dei bug con i primi tre livelli di rischio, in seguito all'aggiornamento della *Roadmap*, viene attivato il processo di *Project Planning*, in seguito al quale viene notificata al cliente la decisione presa dall'azienda.

3.2.9 Supply Management

Il processo di *Supply Management* (figura 3.24) si occupa della gestione degli approvvigionamenti. In seguito all'evasione di un ordine può emergere la necessità di effettuare degli acquisti che porterà, quindi, alla stesura di una *Purchasing list*. Questa lista viene visionata dalla direzione che decide se approvarla o meno. In caso di approvazione viene effettuato un ordine al/ai fornitore/i. Nel caso un fornitore segnali dei possibili ritardi, viene avvisato il responsabile della commessa da evadere che si occuperà di effettuare le opportune azioni correttive (ad esempio contattare il cliente dell'azienda interessato per negoziare su una variazione delle scadenze). In caso contrario, invece, al ricevimento dei prodotti dei fornitori, si procede con il controllo dei documenti di trasporto e della fattura (documenti che verranno, poi, archiviati) ed, in seguito, con il controllo della merce. Se si riscontrano dei problemi in queste due attività, si procede contattando il fornitore, altrimenti si avvisa il responsabile della commessa dell'avvenuto approvvigionamento e si chiude il processo.

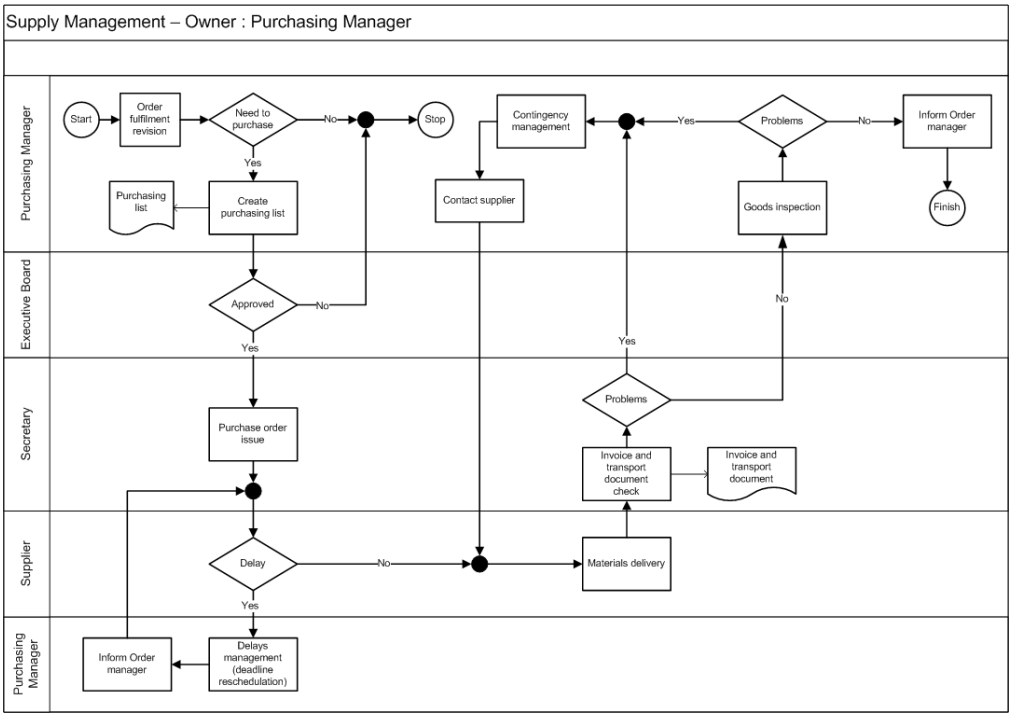

Figura 3.24. Rappresentazione grafica del processo Supply Management.

Capitolo 4

Il progetto WQF

Questa parte di lavoro è stata dedicata allo sviluppo di un'analisi dei requisiti e alla conseguente implementazione di un software per il sistema di gestione della qualità (SGQ) sulla base del formalismo definito. Il lavoro è iniziato con riferimento alla norma ISO 9001 ed è, poi, proseguito verso aspetti più pratici, legati, principalmente alle esigenze reali. La disponibilità di software (commerciali o meno) per SGQ non è sufficiente ad affermare che essi si possano adottare in tutte le realtà aziendali. Nella maggior parte dei casi si è osservato che questi software sono stati progettati e sviluppati avendo come riferimento una *generica* realtà; per questo motivo, tali tools, sebbene ricchi di funzionalità, non possono essere applicati all'ambito di interesse di questo lavoro. In particolare, le analisi effettuate hanno rilevato che non è possibile utilizzare software commerciali senza dover rivedere le esigenze aziendali, non potendo, quindi, sfruttare appieno le caratteristiche che si desiderano riguardo al software di SGQ.

Onde evitare, quindi, che l'introduzione di un sistema per la qualità imponga una *sovrastruttura* aziendale, si è deciso di intraprendere un percorso, sicuramente più ostico, ma anche più flessibile, di sviluppo *ad hoc* del software di ausilio al suddetto sistema.

4.1 Gli obbiettivi

Le funzioni richieste dal sistema sono riunite come segue. Lo strumento da analizzare deve:

- permettere di disegnare i processi aziendali e memorizzarne le istanze create;
- tenere traccia dei dati prodotti;
- produrre la documentazione per il manuale della qualità nel momento in cui questa venga richiesta;

- mantenere lo storico delle revisioni dei processi, per poter sempre verificare quale versione di un processo fosse attiva in un certo momento;

- mantenere lo storico documenti prodotti in passato;

- fornire un motore di workflow, ovvero uno strumento che, seguendo passo passo i grafi dei processi, guidi le attività del personale in modo da aiutarlo ad eseguile rispettando le procedure aziendali. In riferimento all'esecuzione dei processi, il sistema deve permettere di mantenere informazioni sulle attività svolte, in modo da poterle tracciare con semplicità. In linea di principio il funzionamento di quest'ultimo è piuttosto semplice: un processo è composto da varie attività di tipologie differenti (ad es. start, finish, scelta). Ad ognuna di queste attività viene associata una vista per l'utente in funzione delle caratteristiche dell'attività stessa. Eseguire un processo significa crearne un istanza attivando, quindi, il corrispondente nodo start, che genererà la pagina corrispondente all'attività immediatamente successiva. Una volta conclusa ogni attività il sistema si occuperà di fornire a chi di dovere la vista dell'attività che segue;

- gestire gli accessi degli utenti a dati e attività, in base al ruolo di appartenenza all'interno dell'azienda.

È risultato chiaro, sin dai primi colloqui con l'azienda, che lo sforzo nella realizzazione del software debba essere messo non tanto nella generalità dei moduli e delle funzionalità (caratteristica già presente nei software commerciali), quanto nella capacità del sistema di mantenere tutti i dati ed i documenti necessari al SGQ. Il software deve essere, infatti, capace di descrivere la realtà aziendale (intesa come insieme di processi, procedure, ecc.) anche a fronte di cambiamenti repentini e di riuscire a tener traccia dei cambiamenti avvenuti nel tempo in modo semplice ed efficace.

Come già detto, uno degli aspetti chiave di un software per SGQ è la gestione della documentazione. Questo aspetto risulta tanto importante per l'ottenimento della certificazione, quanto spesso critico per le varie figure aziendali. La produzione dei documenti del SGQ è spesso vista dal personale come un inutile (o peggio dannoso) aumento del carico di lavoro. Per questi motivi spesso ci si scontra con realtà che si sentono eccessivamente vincolate dalla gestione della documentazione e, in alcuni casi, questa gestione viene posticipata fintantochè non risulti improrogabile. Il risultato effettivo di un tale approccio è la presenza di incoerenze nei dati presentati all'eventuale ispettore, con il rischio di ricevere un ammonimento o, in casi estremi, la revoca della certificazione. Spesso una gestione non corretta della documentazione non può essere attribuita unicamente al personale aziendale: in molti casi, infatti, l'aumento del carico di lavoro non è attribuibile unicamente alle prescrizioni normative, ma anche al tipo di

ausilio (ad esempio il software) che si utilizza per la gestione degli aspetti relativi alla qualità. Questa affermazione risulta tanto più verosimile se si considera una realtà aziendale in cui si utilizzano software commerciali per il SGQ. Spesso questi pacchetti "impongono" un modo di lavorare e, di conseguenza, di produrre documenti, che non risulta coerente con la prassi quotidiana dell'azienda. Nella fase di progettazione di un nuovo software per il sistema di gestione della qualità, questo è un aspetto fondamentale e va, perciò, analizzato in dettaglio. Il software da realizzare deve, quindi, avere la capacità di effettuare quella che viene chiamata *autodocumentazione*, ovvero deve essere in grado di produrre in modo automatico documenti e registrazioni coerenti con le specifiche normative. L'espressione "in modo automatico" sta a significare che il carico di lavoro svolto dal personale dipendente deve essere minimo.
Per meglio comprendere come sia possibile sviluppare un software in grado di presentare una forte proprietà di autodocumentazione, si deve prima di tutto capire come i documenti debbano essere redatti, promulgati, revisionati e distrutti in modo da ottemperare ai requisiti normativi. Uno dei concetti fondamentali espressi dalla norma è la *rintracciabilità* dei documenti. Ciò significa che ogni singola registrazione deve essere sempre accessibile in modo immediato e definito. Devono essere, quindi, note in ogni momento la collocazione del documento, la versione e/o revisione e l'ambito di applicabilità (chi deve conoscere il documento, chi può leggerlo, ...). I principali problemi relativi alla realizzazione di un'architettura software in grado di implementare queste funzionalità risiedono nel riuscire a rendere il meccanismo di rintracciabilità trasparente ed efficace. È, infatti, inutile prevedere delle procedure atte alla redazione e alla pubblicazione di documenti se, successivamente, la loro consultazione risulta complicata. L'ovvia conseguenza sarebbe quella di produrre documenti che non vengono mai consultati, oppure di scoraggiare la produzione dei documenti stessi. Un risultato frequente consiste nel produrre documentazione che presenta delle carenze (mancano i documenti al momento della visita ispettiva) o delle incongruenze (i documenti prescrivevano una certa procedura che, però, non è stata eseguita in quanto il documento non è stato consultato).

Capitolo 5

Definizione del piano di lavoro

La realizzazione di un *progetto*[1] ne prevede l'attuazione in un tempo relativamente limitato, e comunque ben definito, che può andare da qualche settimana a un paio di anni; qualora, invece, il tempo richiesto per il raggiungimento dell'obiettivo sia superiore, è allora più corretto parlare di *programma*, con obiettivi più complessi e di più ampia portata rispetto a quelli di un progetto che, nel loro insieme, costituiscono generalmente realizzazioni di tipo strategico e con effetti di lungo termine.

La definizione dettagliata di un modello metodologico di riferimento è essenziale per costituire una struttura ordinata e standardizzata, nell'ambito di un'organizzazione, che permetta, fra l'altro, la definizione degli aspetti organizzativi, gestionali e dei documenti da produrre per l'alimentazione del *database di progetto*. In generale, si definisce *ciclo di vita* di un progetto l'insieme delle fasi che, succedendosi nel tempo, danno luogo alla realizzazione di quanto previsto dal progetto; un esempio di fasi relative a un progetto informatico può essere scomposto è:

- definizione dei requisiti utente;
- stesura delle specifiche funzionali;
- disegno di primo livello;
- disegno dettagliato;
- produzione e integrazione;
- test di sistema;
- test di accettazione;

[1] Una definizione generale di progetto è quella di un insieme di attività finalizzate al raggiungimento di un determinato obiettivo, univocamente definito, attraverso l'impiego di risorse umane, materiali, tecnologiche, temporali e finanziarie, nel rispetto di prefissati vincoli in termini di tempi, costi e qualità.

- operatività e manutenzione.

In questo contesto, per *controllo* si intende un adeguato governo del processo realizzativo: il Project Manager[2] ha, in generale, la responsabilità sia del "prodotto" che del "processo di produzione", il cui controllo può essere attuato tramite idonea attività di pianificazione e reporting. Il controllo viene, infatti, assicurato tramite la pianificazione delle attività e il monitoraggio periodico del relativo stato di avanzamento, al fine di individuare possibili azioni correttive atte a recuperare eventuali scostamenti dagli obiettivi attesi ovvero a effettuare una ripianificazione. Nella pianificazione, d'altra parte, occorre tener conto che nessun piano è in grado di focalizzare a priori tutti i problemi che verranno incontrati o il tempo preciso che verrà richiesto da ogni singola attività: per questo è previsto che il piano venga periodicamente riemesso, e può costituire uno sforzo inutile scendere nella sua definizione ad un livello di dettaglio troppo spinto. La prima emissione del Piano di Progetto dovrebbe, tuttavia, essere sufficientemente valida e i tempi e i costi totali pianificati non dovrebbero, in genere, subire variazioni molto significative, se non in casi eccezionali ed opportunamente motivabili e documentabili. Una buona tecnica di gestione del controllo è la scomposizione del progetto in attività di complessità inferiore; un esame più approfondito può rilevare che il sistema da realizzare può essere opportunamente segmentato e, per progetti di dimensioni rilevanti, se ne possono in tal modo ridurre i rischi.

In dipendenza delle dimensioni e della durata temporale di un progetto, l'insieme delle attività che lo costituiscono e che concorrono alla sua definizione possono essere, quindi, scomposte in attività di livello gerarchicamente inferiore, più semplici e quindi più facilmente definibili e controllabili sul piano realizzativo: quanto più ciascuna attività risulta chiaramente definita nei suoi aspetti tecnici, temporali ed economici tanto più sarà semplice valutarne l'evoluzione progettuale e l'eventuale scostamento dei suoi parametri di valutazione dai valori attesi. Una delle più importanti fasi nella definizione di un progetto è, quindi, l'identificazione e la descrizione di ciascuna attività che lo costituisce e l'ulteriore suddivisione in attività più semplici, fino ad un livello tale per cui ciascuna attività elementare, o compito, è chiaramente ed univocamente identificata, in termini di prodotto da realizzare e di tempi e costi di esecuzione. L'attività di individuazione delle componenti elementari deriva dalla necessità di disporre di un procedimento ordinato e sistematico per una definizione che assicuri una corretta interrelazione fra tutti gli elementi, attraverso la creazione di una *struttura analitica di progetto* o *struttura di scomposizione del lavoro*. La struttura che ne deriva è detta *Work Breakdown Structure* (WBS) o *Project Breakdown Structure* (PBS) e costituisce una rappresentazione del progetto, in forma grafica e/o descrittiva che, suddividendo le attivitá in livelli, consente un'analisi di dettaglio indispensabile per una corretta identificazione delle attività elementari la cui ese-

[2]Il Project Manager è la figura centrale e di riferimento per qualunque progetto che riveste il duplice ruolo di responsabile unico del buon esito del progetto e punto riferimento nei confronti del committente, del management aziendale e dell'intero team di progetto.

cuzione integrata conduce alla realizzazione dell'intero progetto. Una struttura WBS consente di evidenziare gli elementi oggetto di consegna al cliente (detti anche *deliverables*) e i principali compiti funzionali da eseguire per la realizzazione di ciascuno di essi, attraverso la definizione e l'esecuzione di test, intesi come risultati materiali o immateriali delle attività di progetto. É bene evidenziare che la struttura di una WBS non ha alcuna relazione con lo sviluppo temporale delle attività di progetto, mentre è la successiva attività di pianificazione a dover ordinare cronologicamente fasi e compiti; le caratteristiche fondamentali della metodologia derivate dall'impiego di una struttura WBS sono:

- il collegamento fra loro e al prodotto finale i singoli compiti;
- la scomposizione dell'elemento di più alto livello (progetto) nei principali elementi costitutivi: sistemi, facilities (risorse ed accessori), deliverables;
- la scomposizione di ciascun elemento costitutivo in oggetti di entità più semplici (in termini di entità e complessità) e costo inferiore, fino all'identificazione di un oggetto ben definito da consegnare;
- la visualizzazione del progetto nella sua interezza, evidenziando la sua complessità e i collegamenti fra i vari elementi.

Dalla costruzione della WBS, scaturisce l'individuazione dei principali compiti da eseguirsi dalle singole funzioni per la realizzazione dell'oggetto: la scomposizione di un progetto in una Work Breakdown Structure, infatti, consente di identificare elementi e compiti chiaramente gestibili e attribuibili a specifiche responsabilità, la cui attuazione possa essere pianificata, valutata, schedulata e controllata: in tal modo il progetto viene definito in ogni sua parte e consente la realizzazione di riepiloghi sulla realizzazione del progetto stesso. Una WBS, pertanto, costituisce un valido elemento di comunicazione, evolvendo e riflettendo i piani correnti in base al reale sviluppo del progetto ed esplicitando maggiori dettagli in vista dell'esecuzione del progetto; costituisce, inoltre, un efficace ausilio per individuare tutti gli elementi fondamentali per il buon esito del progetto, evitando di trascurare aspetti meno evidenti e permettendo di chiarire ruoli e impegni del responsabile dell'esecuzione di ciascun compito. L'analisi di una struttura WBS deve essere condotta con tutti gli attori coinvolti nella realizzazione del progetto, allo scopo di raggiungere alla piena condivisione della sua validità, identificando e condividendo la strutturazione in work packages (compiti e connesse attività per il loro completamento) che devono essere opportunamente pianificati, valutati, budgetati, schedulati e controllati.

Il diagramma di Gantt costituisce invece, un efficace strumento di ausilio alla pianificazione e consente di disporre in forma integrata e visuale, di semplice e immediata comprensibilità, l'evoluzione temporale complessiva del progetto e l'interrelazione fra tutte, o le principali, attività in cui il progetto stesso è stato scomposto. Un diagramma di Gantt è costituito da un elenco delle principali

attività e delle sottoattività, generalmente coincidenti con quelle definite nella costruzione della WBS, disposte in forma tabellare e collegate temporalmente in coerenza con con quanto definito in precedenza. In un diagramma di Gantt, per ciascuna attività possono essere specificate varie informazioni che la caratterizzano; generalmente è specificato un identificativo univoco, la data di inizio e di conclusione, la durata prevista, le risorse allocate, le attività da cui questa dipende e quelle ad essa subordinate. La figura 5.1 mostra il diagramma di Gantt per il progetto WQF.

Quanto più la pianificazione del un progetto è effettuata in modo dettagliato ed integrato a livello di attività elementari, tanto più preciso risulta il controllo della sua evoluzione; in conseguenza di ciò, la possibilità di assicurare il buon esito di un progetto, ovvero di minimizzare gli scostamenti dei risultati rispetto agli obiettivi attesi, può conseguirsi attraverso una schedulazione dettagliata fino al livello del singolo compito ed opportunamente integrata. La tecniche introdotte in precedenza, se correttamente applicate, possono agevolare il raggiungimento di tale obiettivo, in quanto è in grado di assicurare un controllo puntuale ed istantaneo dello svolgimento di ogni attività, attraverso i compiti elementari in cui è stata scomposta, e di verificare, tramite l'interazione fra le attività e i compiti, gli impatti che uno slittamento temporale sul completamento di un'attività possono indurre sul progetto o su sue singole parti. D'altro canto, nella scomposizione delle attività e, quindi, nella definizione dei reticoli, occorre tener conto che se questi risultano troppo dettagliati, il reticolo integrato del progetto può divenire complesso, di difficile immediatezza descrittiva e richiedere, inoltre, strumenti elettronici di ausilio potenti, in grado di elaborare i dati e fornire risposte in tempi rapidi. Occorre, pertanto, individuare una soluzione di compromesso nella scomposizione delle attività, perché se da un lato quanto più questa è dettagliata più risulta semplice assicurare il controllo puntuale di ogni singola attività, dall'altro una eccessiva frammentazione può renderne più complessa e onerosa la gestione, complicando oltremodo l'analisi dell'evoluzione del progetto e l'eventuale esecuzione di simulazioni a supporto della pianificazione e delle scelte progettuale.

Per garantire il buon esito del progetto, in riferimento ai vincoli da rispettare, è necessario mantenere sotto costante controllo gli indicatori di tempi, costi e qualità precisati in fase di definizione e pianificazione. Tale attività di controllo costituisce uno dei momenti più critici dell'intero processo di Project Management in quanto dall'osservazione puntuale degli indicatori citati è possibile intraprendere eventuali azioni correttive ovvero procedere a una ripianificazione del progetto, in parte o in toto, per raggiungere l'obiettivo prefissato e per minimizzare l'eventuale scostamento dovuto ad iniziali errori di valutazione o fenomeni intervenuti successivamente, che possano compromettere il buon esito dell'intero progetto o dar luogo al mancato rispetto dei vincoli inizialmente prefissati.

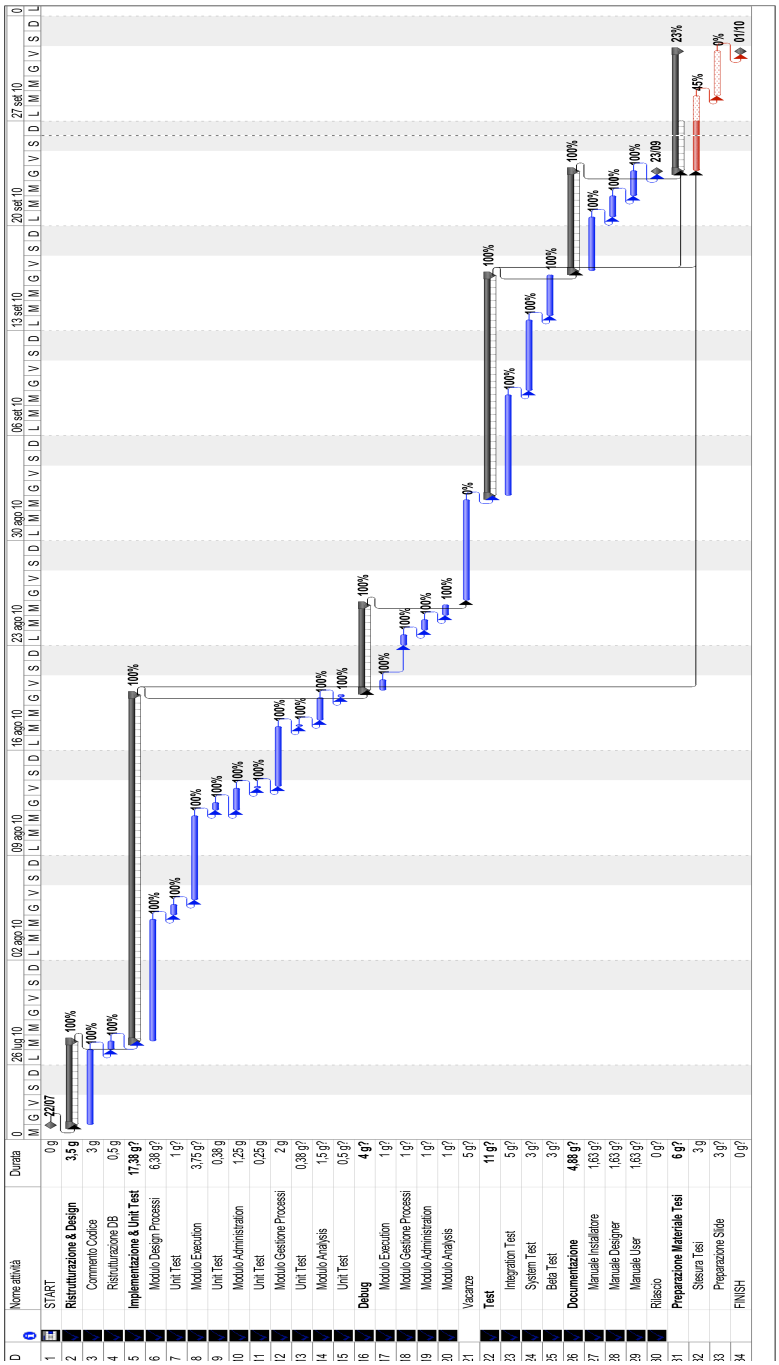

Figura 5.1. Diagramma di Gantt del progetto WQF.

Capitolo 6

Analisi Funzionale

Con l'introduzione del concetto di *indipendenza dei processi* non più vincolati alla *baseline* e *attività dettagliabili*, si è arrivati alla ridefinizione radicale del sistema di gestione dei processi, in cui la centralità del concetto di *baseline*[1] sparisce per far posto alla lista di processi attivi.

Si ricorda che la politica di gestione dell'esecuzione di sottoprocessi rimane invariata rispetto a quella descritta nel capitolo 8 della tesi di laurea citata in bibliografia [2].

6.1 Processi statici

Per *Processi statici*, si intendono quei processi disegnati a tempo di design, in cui un login con ruolo di Designer, stabilisce il loro contenuto formato da attività. Si differenziano dai *Processi dinamici* (vedi il paragrafo 6.3) per il fatto che:

- Sono immutabili una volta che *vengono immessi* nel sistema ovvero quando passano nello stato *Available* (si veda il diagramma degli stati e delle transizioni 6.1).
- Valgono per ogni loro istanza di processo creata.

6.1.1 Ciclo di vita di un processo statico

La figura 6.1 descrive gli stati che possono essere assunti da un processo statico e le sue modalità di transizione. Un processo P nuovo al sistema può essere creato tramite *create* che lo porta nello stato di *development*. Permanendo in questo stato, P può essere disegnato passando nel sotto-stato *activity development* tramite la funzione *modify process* in cui il designer può disegnare le attività associate a P, con i relativi documenti, creare i connettori, modificare connettori ecc. (come descritto nella sezione riguardanti le funzionalità 6.4) In qualsiasi momento il

[1] definita come: snapshot del sistema aziendale in un determinato istante. Si veda [2] indicata in bibliografia per ulteriori dettagli.

designer può cancellare il processo in sviluppo tramite *delete*, operazione che di fatto conclude il ciclo di vita di P.

Una volta che il disegno di processo è completato, P può essere portato in stato di *Available* tramite *commit*. Se l'operazione ha successo, P non potrà in qualunque modo essere cancellato dal sistema. In *Available*, P può essere portato in *Obsolete* tramite *make Obsolete* oppure in *Scheduled* tramite *schedule*, qualora il designer decida di schedularlo per renderlo attivo ad una certa fissata.

Denotiamo di seguito con P.n , la revisione n-esima di P, per n maggiore 0.

Il processo P per arrivare allo stato *Active* , deve passare per lo stato *Loading* : nell'ipotesi che P sia un processo nuovo al sistema, cioè con n=1, questo stato è fittizio; nel caso invece sia già presente nel sistema un processo P.m in *Active* , con m diverso da n, il passaggio di P.n allo stato *Loading* è in contemporanea al passaggio allo stato *Unloading* di P.m, ovvero i passi seguiti avvengono all'interno di una transazione. In definitiva in *Active* la coppia P.n deve essere unica, ovvero in *Active* ci deve essere presente un'unica revisione di uno stesso processo. Dal momento in cui viene caricato la nuova revisione di P, tutte le istanze di quest'ultime che verranno create saranno della revisione n, mentre si permetterà alle istanze di P già avviate di essere concluse dai leggittimi *owner* del flusso di esecuzione delle attività associate.
Da *Active* P può essere reso di nuovo *Available* tramite l'*Unload* e successivamente il *make Available*. Con questa operazione si dice in pratica al sistema che P non è più attivo, ma è disponibile per essere di nuovo schedulato oppure per essere reso obsoleto.

Ipotizzando che nel sistema esista già un processo P, si supponga che lo si voglia revisionare -si immagini inoltre di avere P.m in active, P.j in Obsolete, P.k in scheduled, il caso di P in *development* non è contemplato, poiché si suppone che si possa intervenire direttamente sulla revisione di P in *development* -. L'operazione di revisione è implementata da *copy*, il quale crea un nuovo processo con revisione z, dove z= $max(m, j, k) + 1$. Eventuali sottoprocessi non vengono copiati: sarà responsabilità del designer eventualmente di ricollegarli.

Un altro modo per creare un processo statico nuovo al sistema è dato da *create from dynamic*: in pratica questa funzione converte un processo dinamico, valido solo, come si vedrà nel paragrafo 6.3, per un'istanza di attività dettagliabile di un'istanza di processo, in uno statico. Anche in questo caso tutti gli eventuali sottoprocessi non vengono copiati, sarà il designer che li dovrà ricollegare.

6.2 Gestione sottoprocessi

Si immagini di avere il seguente scenario: si vuole riportare un processo da *Active* a *Available* usato come sottoprocesso da un altro del quale possono esserci delle istanze attive (sia all'interno di processi dinamici che processi statici).
Detto P1 una istanza attiva del processo P che vuole usare il processo S, si vuole modificare S in S'. Al momento dell'editing, il designer tratta S e S' come due

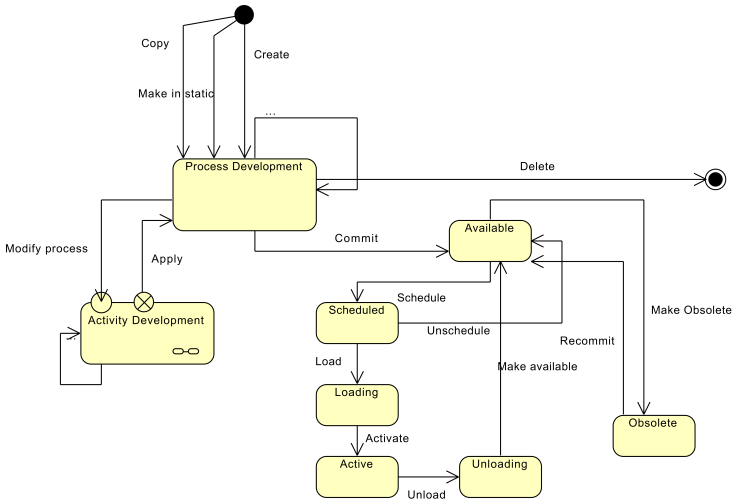

Figura 6.1. Diagramma di stato e transizioni di un processo

processi distinti, in cui S è da scaricare e S' da caricare. Nella maschera di interazione di carico e scarico processi, può darsi il caso di voler disattivare S (in modo da non poterlo eseguire), anche se P1 prima o poi lo vorrebbe eseguire, per risolvere l'inconsistenza che si verrebbe a creare, il sistema consente di disattivare S, epperò prima di farlo cerca tutte le istanze di processi attivi che lo potrebbero utilizzare e, attraverso una maschera di interazione a forma di tabella, chiede per ogni istanza qual'è il nuovo processo che sostituisce S (S' oppure un qualunque altro).

Al momento del caricamento di S' ci sarà un check sintattico dei morsetti, che devono essere compatibili con tutte le istanze che andranno a utilizzare S'. Una volta che S è stato scaricato, le sue istanze vengono comunque finite.
In fase di schedulazione è necessario preventivamente verificare che il processo interessato non utilizzi sottoprocessi, in caso affermativo, il designer deve schedulare anche quest'ultimi e analogamente prima di far tornare in *Available* un processo *Scheduled*, bisogna verificare che non sia utilizzato da un processo *Scheduled*.

Nell'ipotesi che un processo P sia stato schedulato utilizzando (come sottoprocesso) un processo S che in quell'istante è in stato di *Active*, è ragionevole che si debba rendere S non scaricabile dal sistema - cioè far passare S da *Active* a *Available* -. Questo per evitare che, nel momento in cui P passa in *Active*, il sistema diventi inconsistente, se precedentemente fosse stato scaricato S.

6.3 Attività dettagliabili e Processi dinamici

Un'*attività dettagliabile* è un particolare tipo di attività che

1. a tempo di design è trattata come un'attività "normale".
2. a tempo di runtime deve essere dettagliata trasformandola in un processo dinamico.

Per *processo dinamico*, si intende un particolare tipo di processo associato a una particolare istanza di *attività dettagliabile* e che ha validità solo per quell'istanza di *attività dettagliabile*. Come accennato prima, l'*attività dettagliabile* a tempo di design è trattata come un qualunque altro tipo di attività, a cui si associano eventuali documenti di input ed output, ma che a tempo di esecuzione provoca il lancio della funzione di editing. Si può pensare alla funzione di edizione di una attività dettagliabile AD, come a una funzione disponibile nella vista del ruolo che ha diritto di editing su AD (ownership).

6.3.1 Ciclo di vita di un processo dinamico

Di seguito si indica con *editor* il ruolo aziendale che ha l'ownership dell'*attività dettagliabile* da espandere. Un processo dinamico PD può essere creato lanciando la funzione *create* che lo fa passare in *development*; in questo stato PD può essere disegnato in modo simile se non uguale a quello dei processi statici. Quindi in questa fase l'editor(o il designer) può aggiungere attività, creare connettori, associare documenti. Il salvataggio di PD avviene automaticamente in quanto ogni azione atomica ha subito effetto sul database. Se viene finita la fase di disegno, è possibile rendere attivo il processo dinamico associato tramite il comando *save and activate*. Vengono fatti i controlli di consistenza sul processo e se li supera, il processo può essere reso disponibile a coloro che lo devono eseguire. Dato che si vuole modellare la natura flessibile dell'*attività dettagliabile*, è data la possibilità al suo editor di poter modificare porzioni non ancora eseguite del processo dinamico associato: egli può modificare *tutte quelle attività che sono a valle di archi non ancora percorsi*. La funzione che implementa questo comportamento è il *modify* che riporta il processo dinamico in *Development*. Nel momento in cui il processo viene riportato in sviluppo tutte le attività ad esso associate non ancora eseguite vengono bloccate.

Per quanto riguarda il controllo dei documenti di input ed output, la loro coerenza viene effettuata ogni volta che l'editor lancia il comando *verify process* oppure quando cerca di attivarlo lanciando la funzione *activate*. Quando questo accade, il sistema, oltre a controllare la consistenza del processo dinamico - tutti gli stessi controlli effettuati per un processo statico-, controlla anche che *almeno una* delle attività del processo utilizzi i documenti di input dell'attività associata e *almeno una* attività del processo dinamico produca i documenti di output dell'attività associata, definiti a tempo di design.

Quando PD associato conclude, va in stato di *Finished*.

C'è da ricordare che PD può essere usato come base per la creazione di un altro processo dinamico associato a una nuova istanza della stessa attività dettagliabile: quando l'editor chiama *redevelop* viene creata una copia PD' di PD, a cui si possono apportare eventuali modifiche.

Come descritto precedentemente 6.1, un *processo dinamico* può essere riconvertito in statico. La conversione viene fatta solo per processi che si trovano nello stato di *Finished* ed è comunque una copia.

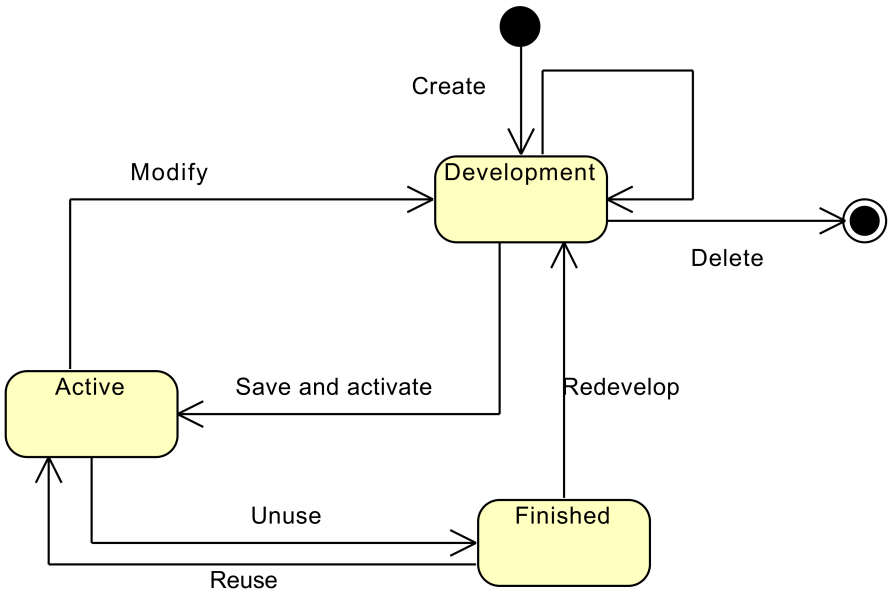

Figura 6.2. Diagramma di stato e transizioni di un processo dinamico

6.4 Funzionalità

Si è ritenuto opportuno modellare le funzionalità del sistema tramite digramma degli *Use Case*, i vantaggi rappresentati da tale modellazione sono il raggruppamento delle funzioni per ruolo e per area concettuale; infatti è subito possibile visualizzare le funzioni avviabili da ogni attore che andrà ad interagire col sistema.

La figura 6.3, sintetizza che le funzionalità attivabili dagli attori che andranno a interagire col sistema:

1. *User*: è l'utente che può:

   - eseguire i processi (si veda il paragrafo 6.4.2).
   - disegnare i processi dinamici, che nello *Use Case* sono modellati come generalizzazione del disegno di processi.
   - interrogare il database per effettuare analisi di dati

2. *Process Designer*: è un ruolo che può:

   - gestire i processi, sia statici che dinamici. La gestione del processo è l'insieme delle operazioni che comprende la creazione, la modifica, il disegno, la cancellazione e altre operazioni quali visione, schedulazione dei processi. Come si nota dal diagramma dello *Use Case*, la gestione di un processo *include* la gestione di attività, ciò vuol dire che *Gestione Processi* può effettuare una chiamata a *Gestione Attività*, quando necessario, ovvero quando egli deve creare o modificare il disegno di un processo.

3. *System Administrator* è un ruolo che può:

   - gestire gli utenti, creandoli, modificandoli, cancellandoli, creare ruoli aziendali, assegnare ruoli.

4. *Observer* è un ruolo che può:

   - eseguire analisi di dati, anche a livello avanzato.

   Lo si può pensare a un certificatore che può interrogare la base di dati sulla gestione aziendale tramite opportune maschere.

5. *Guest* è un ruolo che può effettuare solo limitate interrogazioni di dati.

Il *System Administrator* deve avere l'accortezza di assegnare almeno un ruolo aziendale ad ogni Login(con ruolo di sistema diverso da *Process Designer* e *System Administrator*) che crea. In caso contrario il sistema, nel caso la Login che cerchi di loggarsi, come verrà spiegato più avanti, ragionevolmente, viene "degradato" a ruolo di sistema *Guest*.

6.4.1 Gestione Processi

In seguito definiamo con *DPRFA* (Dynamic Process Returned From Active) un processo che soddisfa ambe due le seguenti condizioni:

- è un processo dinamico

- è passato da *Active* a *Development*

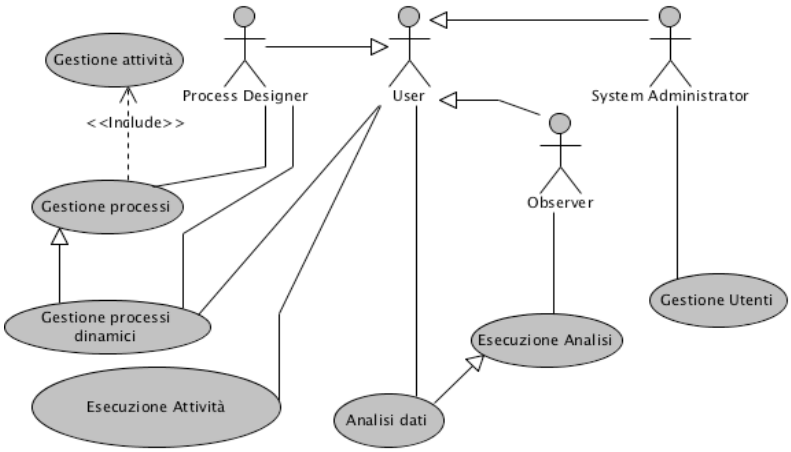

Figura 6.3. Diagramma degli *Use Case*

In realtà la prima condizione risulterebbe ridondante -la si lascia per chiarezza espositiva- in quanto solo un processo dinamico può passare da *Active* a *Dvelopment*, mentre un processo statico in nessun modo può ritornare allo stato *Development*, una volta che è stato attivato.

Create Process Il login che ha ruolo di process designer, deve immettere il nome, che non deve essere presente già in sistema, del processo che vuole creare. Il sistema assegna automaticamente al processo la revisione numero 1. Il processo creato è vuoto e se lo si vuole disegnare, bisogna sceglierlo tra la lista dei processi da modificare tramite la funzione *Modify Process*.

Modify process Il designer visualizza tutti i processi che sono in stato di Development e seleziona il processo da modificare. Si rende disponibile la visualizzazione del contenuto del processo selezionato. Inoltre si rende disponibile la modifica del nome del processo selezionato. Cliccando su *Modify Process*, il designer può procedere al disegno (o alla modifica del disegno) del processo selezionato. L'azione porta il processo selezionato nel sottostato *Activity development*.

Create Activity La creazione delle attività è nel contesto del processo selezionato. Con l'introduzione delle attività dettagliabili, il sistema rende disponibile al designer la creazione di questo un nuovo tipo di attività. Quest'ultima, in definitiva, è marcata come attività dettagliabile, ma nel complesso viene disegnata

come una attività "nomale": quindi ha documenti opzionali di input ed output, connettori di ingresso e uscita, ecc. La caratteristica particolare dell'attività dettagliabile risalta solo nel momento in cui viene creata l'istanza del processo che la contiene.

Delete Activity Vengono visualizzate tutte le attività disegnate per il processo corrente e selezionando quelle desiderate, le si può cancellare. Vengono inoltre cancellati gli archi entranti e uscenti dell'attività. Nel caso l'attività fosse contenuta in un *DPRFA*, l'attività è cancellabile solo se non è aperta e non è stata eseguita.

Crea Connettore Il designer deve selezionare quali attività connettere, tramite una lista di attività di partenza e una di arrivo. Confermando il sistema memorizza il connettore sul database. Nel caso l'attività di arrivo (AA) o di partenza (AP) fosse contenuta in un *DPRFA* si possono distinguere i seguenti casi:

- se AP non è stata eseguita o il controllo di flusso si è fermato su AP e AA non è stata eseguita, si permette la creazione del connettore. Se AA è stata eseguita o il controllo di flusso si è fermato su AA, la creazione del connettore viene negata.
- se il controllo di flusso si è fermato su AP e AA non è stata eseguita, la creazione del connettore è permessa. Se AP è stata eseguita, la creazione è negata.
- non ci sono restrizioni nel caso AA e AP non sono state eseguite.

I precedenti vincoli sono stati introdotti per rispettare il vincolo più forte che "sono modificabili solo attività a valle di archi non ancora percorsi".

Cancella connettore Selezionando la coppia attività partenza(AP)-attività arrivo(AA), si cancellano i connettori associati. Nel caso la modifica riguardasse un *DPRFA*, il connettore è cancellabile solo se AA non è stata eseguita e AP non è stata eseguita; se il controllo di flusso si è fermato su AP e AA non è stata eseguita, la cancellazione è possibile.

Define ACL I meccanismi di assegnazione delle risorse e di escalation per quanto concerne con l'introduzione delle attività dettagliabili, rimane immutata, essendo quest'ultime a tempo di design gestite come attività "normali".

Schedule processes (Batch Schedule) Per una corretta schedulazione di processi, si dovrebbe schedulare per blocchi, e non per singolo processo, in quanto questo ultimo approccio potrebbe causare avvisi di errore in fase controllo di

consistenza. La funzione permette comunque di trovare automaticamente eventuali dipendenze, ovvero processi che puntano ad altri sottoprocessi e di risolverle automaticamente. La schemata appare al designer come a delle maschere di carico scarico di processi, in cui processi disponibili (in stato di *Active*, *Scheduled*, *Available* o *Obsolete*) vengono schedulati a sostituzione di quelli attualmente in stato di *Active* in sistema. A partire dalla data indicata, i processi schedulati passano in *Active*, mentre vengono scaricati eventuali processi attivi, sostituiti dai primii. Si noti che per far sì che un processo resti in *Active*, il *Designer* deve schedularlo, altrimenti il sistema lo "scarica". Per come viene gestito il carico e scarico dei sottoprocessi -vedi la gestione dei sottoprocessi al paragrafo 6.2-, a regime avremo processi che non possono essere più attivati, mentre può darsi che le loro istanze debbano essere ancora concluse. Questa tipologia di processi denominata *zombie* sarà resa visualizzabile da questa funzione.

Infine l'*Unschedule* dei processi (si veda l'*Usecase Diagram* in figura 6.3), viene cablata all'interno di questa funzionalità, in quanto nel caso il designer decida di rischedulare un insieme di processi, questi non fa altro che fare l'*Unschedule* del vecchio insieme e *Schedule* del nuovo insieme.

Unload e Make Available Si supponga di voler scaricare un processo P da *Active* per portarlo in *Available*. Per come vengono gestiti i sottoprocessi, in caso un P sia puntato da un altro processo P' *active*, il sistema impedirà che P venga riportato in *Available*. Questo vincolo viene proposto per prevenire inconsistenze al sistema.

Make Obsolete e recommit Per far sì che la lista di processi *Available* non cresca in modo abnorme, rendendo l'operazione di schedulazione di processi un compito arduo e confuso, si è pensato di rendere possibile per il designer "accantonare" processi che si reputa che non vengano più utilizzati facendolo passare allo stato di *Obsolete*. Si nota che un processo *Obsolete* può puntare a sottoprocessi in *Active*.

E' possibile in modo complementare, far tornare in auge una revisione di un processo *Obsolete* tramite *reload* che lo fa (ri)passare in *Available*.

Queste due funzioni, sono implementate come maschere di carico scarico. Non ci sono vincoli alle operazioni appena descritte.

Dynamic Process Management Come si nota dal diagrama dello *Use Case*, si rende disponibile al designer la gestione dei processi dinamici: egli può visualizzare tutte le attività dettagliabili e i loro processi dinamici associati presenti a sistema classificati per stato di esecuzione e ownership, modificare i processi dinamici non ancora terminati, creare nuovi processi dinamici associati alle attività dettagliabili. Per i processi dinamici *Finished*, il designer può tramutarli in statici.

6.4.2 Esecuzione Processi

Le funzioni che fanno parte di questo raggruppamento sono le seguenti:

- *Start process*
- *Execute activity*
- *Modify dynamic process*

Start process Ipotizzando di avere un processo P *Active* che ha nel suo flusso di attività un'attività dettagliabile e ricordando che è l'*owner* dell'attività di *start* che può far partire un'istanza I di P, è ragionevole che in conseguenza all'avvio di I, le attività dettagliabili contenute al suo interno siano rese editabili. Questa politica rende più snello il disegno di eventuali processi dinamici associati a I (o meglio alle istanze di attività dettagliabili associate a I). Tuttavia l'editabilità delle attività dettagliabili dipende dalla *policy*[2] assegnata a tempo di design:

- se ha *policy* FIFW (esclusivo o non) l'attività è editabile non appena il processo P viene avviato e la prima login che ne acquisisce l'ownership può cominciare il disegno del processo dinamico associato.
- se ha *policy* Ownership escalation, sarà editabile non appena l'Assigner la assegna a una Login.
- se la sua ownership deve essere assegnata da una *GAO*, essa sarà editabile non appena l'esecutore della *GAO* la assegna a una Login.

Execute activity e Dynamic Process Design Ricordiamo che l'esecuzione di un'attività dettagliabile AD consiste nel suo disegno: quando c'è semaforo verde per l'edizione di AD, l'*owner* di quest'ultima, procede al suo disegno e non può "terminarla" finchè non lo completa. Di seguito un possibile flusso di esecuzione dell'edizione di AD:

1. Dalla lista di attività editabili, l'*owner* di AD clicca su *Edita*
2. All'*owner* vengono proposte due opzioni:
   - *Crea un nuovo processo dinamico*
   - *Crea da un processo dinamico esistente*

   Nell'ultimo caso vengono proposti solo processi dinamici associati (in *Finished*) ad AD e al login dell'*owner*

[2] per la sua definizione si veda il capitolo 8 della tesi [2] citata in bibliografia

3. Si immagini che l'*owner* abbia finito il disegno di un nuovo processo dinamico, quindi primo caso del punto precedente, creando e collegando attività associate, carichi a sistema il processo dinamico, ovvero chiami la funzione *make active*.

4. Lo start del processo dinamico avviene in automatico (nel caso tutte le attività precedenti a AD siano state eseguite) e il flusso di esecuzione dell'istanza del processo che contiene l'attività dettagliabile riparte proprio in cascata allo start del processo dinamico.

5. Una volta concluso il flusso di esecuzione del processo dinamico, questi punterà all'attività successiva a AD.

Per quanto riguarda la funzione *Crea da un processo dinamico esistente*, il processo dinamico selezionato viene copiato, associandogli un nuovo nome, privo di eventuali link a sottoprocessi, e rimesso in *Development*. Si noti che similarmente a un sottoprocesso, una attività non viene effettivamente chiusa: viene considerata chiusa nel momento in cui viene finita l'attività di *finish* del processo dinamico associato.

Modify dynamic process Durante il flusso di esecuzione descritto nel paragrafo 6.4.2, può darsi il caso che si voglia modificare un processo dinamico già avviato, come già descritto nel paragrafo 6.3. Questa funzione permette all'*owner* delle attività dettagliabili di:

- Visualizzare tutti i processi dinamici associati alle attività dettagliabili avviati e ancora non conclusi.

- Scegliere il processo dinamico da modificare.

Supponendo che *owner* abbia scelto di modificare un processo PD non ancora concluso. Tutte le attività a valle di archi non ancora eseguiti, verranno inibiti dall'iniziare, mentre istanze già create di attività facenti parte di PD -si legga attività già eseguite *e* attività attualmente in esecuzione- non possono più essere modificate. Tutte le istanze di attività che risultino in quell'istante aperte vengono, marcate come *holding* e le attività in cascate vengono inibite ad essere aperte. Solo quando l'*owner* procede a riattivare PD, lanciando *activate*, e dopo che rivengono effettuati i controlli di consistenza del processo, così come viene controllata la compatibilità della porzione modificata di PD e quella già eseguita, il flusso di esecuzione di PD riprende.

Capitolo 7

Analisi tecnica

7.1 Architettura dell'applicazione

La progettazione di un'applicazione richiede sempre di affrontarne lo studio dell'architettura complessiva. Chiarire tale tema consente una maggior coordinazione del codice con un complessivo vantaggio nella comprensione e nella leggibilità dell'applicazione stessa. Inoltre, ciò consente l'uso di un linguaggio unificato tra gli sviluppatori che permette loro una maggiore comprensione degli intenti reciproci. Infine, l'architettura scelta impone un orientamento alle successive implementazioni e sviluppi e garantisce un'omogeneità sempre desiderabile quando un lavoro viene svolto in gruppo.

7.1.1 I livelli di astrazione

Quando si costruisce un'applicazione, nella fattispecie una *web application*, è bene distinguere diversi livelli di astrazione (in inglese *layer*). Si tratta di astrazioni logiche che interagiscono una con l'altra, ma che sono isolate al tempo stesso. Un layer può essere, ad esempio, la struttura che si occupa di organizzare la persistenza dei dati (*persistence layer*) che avrà a che fare solamente con il layer che si occupa delle operazioni di servizio (*service layer*). Questi due layer sono collegati tra loro, ma il service layer isola il persistence layer dagli altri. Non sempre però si verifica tale isolamento, come nel caso del *domain model layer*. Ciò sarà più chiaro nel proseguio di questo capitolo. Tipicamente un'applicazione web viene suddivisa in tre livelli:

- il *Top layer* che si occupa delle interazioni con l'utente;
- il *Middle layer* che fornisce i servizi di base;
- il *Bottom layer* che si occupa della persistenza dei dati;

ciascuno dei quali ha diretto rapporto con il solo suo superiore in una struttura come quella illustrata in fig. 7.1.

Figura 7.1. Tradizionale suddivisione dei layer in un'applicazione web.

Con uno sguardo più approfondito si possono identificare cinque layer di astrazione:

- lo *User Interface layer*;
- il *Web layer*;
- il *Service layer*;
- il *Domain model layer*;
- il *Persistence layer*;

anch'essi collegati in una struttura dove *User Interface layer* e *Web layer* corrispondono al *Top layer*, il *Service layer* al *Middle layer* e il *Persistence layer* al *Bottom layer*. Il *Domain Model layer* può essere considerato trasversale agli altri quattro. La figura 7.2 illustra la situazione appena descritta.
Come appena detto i layer sono tra loro isolati, nel senso che ciascuno ha diretto

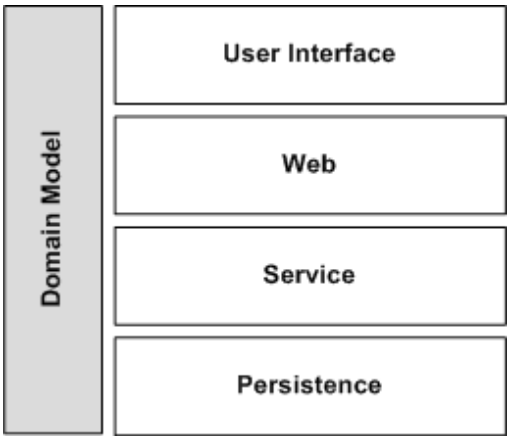

Figura 7.2. Suddivisione dettagliata dei layer in un'applicazione web.

rapporto solo con il suo immediato superiore. Ciò consente di separare il dominio dei problemi da affrontare (ad esempio la persistenza dei dati, la navigazione web o la realizzazione dell'interfaccia utente) e conduce verso un'applicazione più flessibile e testabile. Così facendo, infatti, l'implementazione di ciascun layer può variare in modo indipendente, con evidente beneficio per la flessibilità, favorendo, quindi, il disaccoppiamento delle relazione di dipendenza tra di essi (da qui la maggior testabilità). Favorire l'isolamento significa ridurre le dipendenze tra i layer: minori sono le dipendenze, meno risulta costosa la modifica di un layer. Per questo motivo si deve fare in modo che ogni layer comunichi solo con uno o due livelli, il Domain Model layer rappresenta, ovviamente, un'eccezione.
Per capire quanto detto detto finora, è necessario approfondire il significato dei cinque layer, le loro interazioni e funzioni specifiche. Si dedica, perciò, ancora del tempo a questo tema, anche per sottolinearne l'importanza.

User Interface layer

Lo User Interface layer si occupa di presentare le risposte generate dal web layer all'utente. Tali informazioni si possono essere fornite in varie forme come, ad esempio, un file `PDF` o `XML`, oppure una pagina `XHTML` per un browser. In linea generale questo layer deve essere tenuto quanto più possibile separato dal web layer in modo da riutilizzare quest'ultimo quanto più possibile. Esso rappresenta l'ultimo livello raggiunto dall'applicazione una volta che sono state raccolte e organizzate tutte le informazioni necessarie. Tenere lo User Interface layer come ultimo livello e separato dal resto dell'applicazione consente di non vincolare, per esempio, le altre risorse ai tempi di connessione e trasmissione dei dati, lasciandole, quindi, libere per altri servizi. Inoltre, si può rendere più agevole la migrazione tra varie tecnologie di visualizzazione come `FreeMarker`, `JSP`[1], `Velocity`, `XSLT` o `Tiles`, poiché tale passaggio non influenza gli altri layer. Ciò rende anche più facile la collaborazione tra sviluppatori che si specializzano in aree diverse, nella fattispecie nel design di pagine web, consentendo a questi ultimi uno sviluppo parallelo con l'utilizzo degli strumenti ad essi più congeniali.

Web layer

Il Web layer ha due funzionalità principali. Innanzitutto esso dirige il flusso di navigazione dell'utente in modo da rappresentare nella sequenza coretta la logica di navigazione tra le varie viste; in secondo luogo stabilisce il collegamento tra il Service layer e il mondo dell'`HTTP`. Il Web layer delega ogni logica di servizio al Service layer e si occupa di gestire i parametri di richiesta e di risposta, gestisce le sessioni `HTTP` e, in generale, interagisce con le `Servlet API`. Il Web layer chiama, quindi, i metodi del Service layer e gestisce gli errori da inviare all'utente, cosa che rende molto più semplice la navigazione allo stesso.

[1] http://java.sun.com/products/jsp/

Service layer

Il Service layer occupa un ruolo importante sia per il client, ovvero chi ne invoca i metodi, sia per il sistema. Al primo fornisce funzionalità a grossa granularità per un utilizzo facile, non imponendogli di dover invocare una serie complessa e intricata di metodi e, al tempo stesso, limitandone al massimo le interazioni con il resto del sistema. Per grossa granularità si intende che il client, con una chiamata a un metodo dell'interfaccia di servizio, compie tutto il lavoro di cui ha bisogno senza doversi occupare di quali metodi ausiliari siano da invocare e quale sia il loro ordine corretto. Realizzare tutto il lavoro a livello del Service layer consente di ridurre al minimo le comunicazioni tra client e sistema. Un'ulteriore caratteristica di questo layer è l'essere *stateless*, cioè ciascuna chiamata a un metodo del Service layer non deve riferirsi a precedenti chiamate a esso. Ogni stato tra chiamate di metodi va tenuto all'interno del Domain Model layer. Mantenedo tali principi si ottiene un buon incapsulamento di tutti i casi d'uso del sistema. Diventa, quindi, anche più facile aggiungere meccanismi di comunicazione diversi a una singola interfaccia di sevizio. Si ottiene, così, una migliore adesione al principio DRY[2] e al conseguente beneficio in fatto di riutilizzo del codice.

Domain Model layer

Nel Domain Model layer si colloca la *business logic* dell'applicazione. Questo layer contiene la logica degli stati e dei casi di uso. Esso raccoglie gli oggetti da manipolare con il loro stato ed il loro comportamento. Se esso non contenesse tali informazioni, si andrebbe incontro alla costruzione di un cosidetto *Anemic Domain Model*. Si farà abbondante uso di tutte le regole di OOP (Object Oriented Programming) come polimorfismo ed ereditarietà. Si sottolinea che se nel Service layer si è incoraggiati a fare uno spinto utilizzo di iterfacce, nel Domain Model layer tale utilizzo, sebbene sempre apprezzabile, deve essere realizzato secondo necessità, evitando, quindi, un ploriferare di interfacce non utili. Il significato del termine *business logic* è ampio: si tratta di ogni regola o specifica richiesta dal committente e spazia da un complesso sistema di controlli di stato a una semplice regola di validazione dei dati inseriti. Si noti che, in alcuni casi, tale logica può essere affidata all'esterno del layer, andando in deroga al principio di incapsulamento di tali servizi all'interno del Domain Model layer. Per fare un

[2]DRY (Don't Repeat Yourself), anche conosciuto come "Single Point of Truth", è un principio secondo il quale l'informazione non debba essere ripetuta e ridondante e non si debba esprimere lo stesso concetto più di una volta, specie se in forma diversa. Di particolare importanza, nell'informatica, diventa un Design pattern della programmazione secondo il quale bisogna evitare il più possibile la duplicazione del codice, poiché questa complica la mantenibilità e la leggibilità del codice stesso. DRY è uno dei concetti fondamentali espressi nel libro *The Pragmatic Programmer*. Un codice DRY riduce al minimo le informazioni ridondati e le duplicazioni, risulta molto più pulito, mantenibili e leggibile e ricorre, dove possibile, all'utilizzo di funzioni per accorpare in un unico punto funzionalità usate più volte.

esempio, un controllo di unicità di un campo all'interno del database conviene sia affidato al DBMS, pena un notevole calo delle perfomance. Si deve avere, però, cura nel gestire le eccezioni dovute a eventuali errori di accesso alla base di dati, non semplicemente mandando all'utente un messaggio di errore, ma specificandone il tipo, così da rendere più comprensibile l'utilizzo dell'applicazione. Infine, il Domain Model layer non deve avere alcuna dipendenza dai framework e dai container, così che possa essere testato al di fuori di essi.

Persistence layer

Il Persistence layer è responsabile dell'interfacciamento con il meccanismo preposto alla persistenza dei dati al fine di ritrovarli o salvarli. Tipicamente questo layer contiene i metodi CRUD (Create Read Update Delete). Il beneficio di isolare questo layer è quello di proteggere il sistema dai cambiamenti. Supponiamo, ad esempio, di voler passare dall'utilizzo di un DBMS ad un altro. Se non si fosse isolata in tale layer la logica di persistenza dei dati, si dovrebbe spendere parecchio tempo nel cercare e modificare le classi del sistema che hanno accesso al database; inoltre andrebbero ripensate le classi di test. Tale isolamento consente, inoltre, di testare l'applicazione indipendentemente dal database, accelerando considerevolmente i tempi di sviluppo. Certamente non si può fare a meno di verificare che il sistema funzioni con il database di utilizzo ma, come si vedrà in seguito, si tratta di integration testing.

7.1.2 Breve storia delle Web Application

Inizialmente i siti Web erano del tutto statici, in quanto presentavano delle semplici pagine `HTML`. Inoltre, il protocollo servente, ovvero l'`HTTP` (Hypertext Transfer Protocol), era un semplice protocollo *stateless*. Questa situazione non durò a lungo. Ben presto, infatti, nacque l'esigenza di mostrare nei siti delle informazioni che potessero cambiare nel tempo. Le persone, inoltre, desideravano compiere operazioni sempre più complesse con l'utilizzo dei siti Web, per esempio tenere traccia di ciò che un utente aveva fatto all'ultimo accesso al sito, in modo da consentire relazioni commerciali, come aggiungere oggetti ad un carrello. Si necessitava, quindi, di un meccanismo che fornisse contenuti dinamici e che desse la possibilità di memorizzare uno stato con un protocollo *stateless* come `HTTP`.

Script CGI: il primo meccanismo per i contenuti dinamici

Il primo meccanismo che mise a disposizione degli utenti contenuti dinamici è stata la Common Gateway Interface (`CGI`). Applicazioni eseguibili (di solito, ma non necessariamente, scritte in `Perl` o `C`) venivano fornite con un'interfaccia che abilitava i client ad accedervi attraverso il protocollo `HTTP`. Nonostante tutto, `CGI` soffriva di numerosi svantaggi:

- Ogni richiesta `CGI` necessitava dell'avvio di un processo del sistema operativo.
- Ognuno dei processi creati, quindi, caricava ed eseguiva un programma `CGI`.
- Una codifica noiosa e ripetitiva era necessaria per gestire il protocollo di rete e per interpretare le richieste.

Le prime due operazioni dell'elenco precedente potevano utilizzare un gran numero di cicli della CPU ed una grande mole di memoria. Poichè entrambe le operazioni dovevano essere eseguite per ogni richiesta, vi era la possibilità che un server si sovraccaricasse nel caso in cui esso ricevesse un gran numero di richieste in un breve lasso di tempo. Inoltre, poichè i programmi `CGI` erano mutuamente indipendenti, e spesso scritti in linguaggi tra loro incompatibili, non era possibile riutilizzare il codice per la gestione della rete e per l'interpretazione delle richieste. Si noti che `CGI` descriveva solo il contratto tra il server web e il programma. Non erano, infatti, servizi per contribuire a implementare un servizio *user-centric* (centralizzato sull'utente). Era, quindi, difficile memorizzare l'identità dell'utente attraverso le varie richieste, limitare l'accesso all'applicazione ai soli utenti autorizzati o memorizzare informazioni *runtime* nell'applicazione. Per questo motivo, gli script `CGI` non vengono utilizzati nei moderni siti Web.
Si è reso, quindi, necessario poter lavorare con un *framework* per la creazione di applicazioni Web in grado di fornire le funzionalità appena citate e di risolvere il problema delle scarse prestazioni della scalabilità.

Java lato server: Servlet

Le `Servlet` sono porzioni di codice scritte in `Java` che hanno una forma definita e vengono richiamate per generare dinamicamente contenuti o eseguire alcune azioni. Le `Servlet` superano i problemi di `CGI` precedentemente menzionati in quanto:

- L'`overhead` dovuto all'avvio di un processo del sistema operativo per ogni richiesta è completamente azzerato. Una Java Virtual Machine (`JVM`), infatti, è attiva in background e tutte le richieste sono da essa gestite.
- Le classi `Java` vengono caricate dalla `JVM` per elaborare le richieste in entrata; se più di una richiesta richiede il medesimo trattamento, la classe già caricata può essere utilizzata per gestirla. Questo elimina l'overhead del caricamento delle classi per tuttie le richieste tranne la prima.
- Il problema della gestione dello stato al di sopra di un protocollo *stateless* come `HTTP` è risolto, come verrà successivamente spiegato.
- Il codice che gestisce il protocollo di rete e che decodifica le richieste in arrivo può essere condiviso da tutte le classi Java che elaborano le richieste.

Come detto, le *Servlet* hanno una forma ben definita: questo significa che tutte le *Servlet* implementano un'interfaccia chiamata `Servlet`, che definisce il loro ciclo di vita standard, ovvero una lista di metodi che verranno chiamati in modo prevedibile. L'inizializzazione è agevolata dall'uso del metodo `init()`. Tutte le risorse necessarie per la *Servlet*, insieme alle eventuali inizializzazioni che la servlet deve fare prima di poter servire le richieste dell'utente, sono fornite da questo metodo, che viene chiamato una sola volta per ogni istanza della *Servlet*. Ogni *Servlet* può gestire più richieste da più utenti. La prima volta che si effettua la richiesta del file, quest'ultimo viene compilato, creando una *Servlet*, che sarà archiviata in memoria (per servire le richieste successive); solo dopo questi passaggi viene elaborata la pagina `HTML` che viene mandata al browser. Ad ogni richiesta successiva alla prima, il server controlla se sulla pagina richiesta è stata effettuata qualche modifica, in caso negativo richiama la *Servlet* già compilata, altrimenti si occupa di eseguire nuovamente la compilazione e memorizzare la nuova. L'interfaccia `Servlet` definisce, inoltre, un metodo `service()` che viene chiamato ad ogni richiesta utente. Tale metodo controlla l'elaborazione e la generazione della risposta da mandare al *client*. Quando una richiesta viene servita, la *Servlet* resta in attesa della successiva richiesta. Il metodo `service()` si occupa, inoltre, di verificare che tipo di richiesta `HTTP` è stata fatta (per esempio `GET` o `POST`), ed inoltra la richiesta al metodo opportuno. Infine, un metodo chiamato `destroy()` viene invocato per "smaltire" la classe *Servlet* (vedi figura 7.3). Questo metodo si occupa, inoltre, di liberare le risorse acquisite dal metodo `init()`.

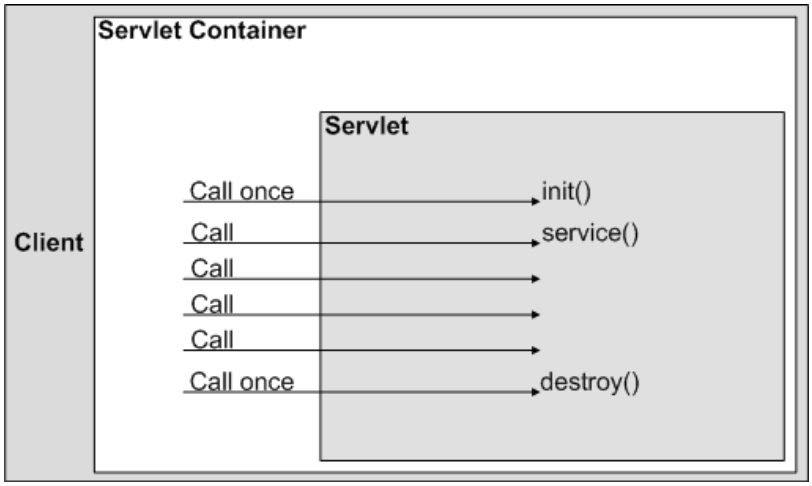

Figura 7.3. Metodi delle Servlet.

Molti fornitori sviluppano ambienti di esecuzione per le *Servlet* noti come *Servlet Container*. Ciascuno di essi deve garantire che vengano seguite le specifiche delle *Servlet*, in modo che una *Servlet* scritta secondo tali specifiche possa

essere eseguita in diversi ambienti senza dover apportare delle modifiche. I *containers* forniscono, inoltre, dei servizi in aggiunta a quelli per la gestione del ciclo di vita di una *Servlet*. Tra di essi si possono menzionare quelli che mettono a disposizione i parametri di inizializzazione, le connessioni con le basi di dati e quelli che permettono la gestione delle sessioni. Per risolvere il problema del mantenimento della sessione dell'utente, il *container* mantiene l'identità del client tramite *cookies* temporanei che memorizzano un *token* che fa riferimento all'utente. in questo modo il *container* è in grado di identificare un client attraverso più richieste.

Sebbene le *Servlet* offrano un notevole miglioramento rispetto a *CGI*, in particolare per quanto riguarda le prestazioni ed il carico del server, anch'esse presentano alcuni inconvenienti. Esse sono state ideate per l'elaborazione della logica, ma risultano meno utilizzabili per la presentazione dei contenuti (ad esempio per l'`HTML`). La codifica dell'output testuale nel codice, compresi i tag `HTML`, rende l'applicazione meno gestibile perchè, quando il testo in `HTML` deve essere cambiato, le *Servlet* devo essere ricompilate. In secondo luogo, è necessario che il designer `HTML` abbia delle buone conoscenze di `Java` per evitare di generare bachi nella *Servlet*. Spesso capita che il programmatore dell'applicazione riceva del codice `HTML` da un altro designer e che lo incorpori nel codice della *Servlet* in corso d'opera. Questo, però, è una procedura incline agli errori.

Per risolvere questo problema Sun Microsystems ha introdotto le *JavaServer Pages* note anche con l'acronimo *JSP*.

JavaServer Pages

Le specifiche della prima edizione delle *JavaServer Pages* (*JSP*) presentavano molte somiglianze con *Active Server Pages* (*ASP*), una tecnologia Microsoft. Entrambe si sono, però, molto evolute da allora, tanto che la loro somiglianza è, oggi, molto ridotta. La tecnologia `JSP` è stata migliorata con l'introduzione delle *Tag Libraries*. Queste librerie di tag sono raccolte di tag personalizzati; ogni tag corrisponde ad un modulo `Java` altamente riutilizzabile.

Dietro le quinte, la `JSP` viene compilata in una classe *Servlet* la prima volta che essa viene invocata. Questa *Servlet* viene, quindi, richiamata ad ogni successiva richiesta, evitando, perciò, l'analisi e la compilazione della `JSP` ogniqualvolta l'utente accede al sito. Le `JSP` si diffusero notevolmente grazie alla loro grande idoneità alla creazione di contenuti visuali dinamici, in un momento in cui Internet moltiplicava la sua popolarità. Come le *Servlet*, anche le `JSP` operano all'interno di un *container*. Il *JSP container* offre gli stessi servizi di un *Servlet Container*, ma richiede, in aggiunta, il passo supplementare per la conversione in *Servlet* della `JSP` e per la compilazione del codice prima della sua esecuzione. Come si vedrà nelle sezioni successive, Apache Tomcat contiene sia il *Servlet container* (chiamato Catalina), che esegue le *Servlet* e le `JSP` compilate, sia il compilatore per le `JSP` (chiamato Jasper). La combinazione di un compilatore per `JSP` ed un *Servlet container* prende il nome di *Web Container* (un conteni-

tore in grado di ospitare applicazioni Web `Java`). Una differenza pratica tra le *Servlet* e le `JSP` sta nel fatto che le prime sono fornite già compilate, cosa che non avviene necessariamente per le seconde. Per un amministratore di sistema questo significa che i file delle *Servlet* devono essere memorizzati in un'area riservata del server, mentre le pagine `JSP` possono tranquillamente essere mescolate alle pagine `HTML` statiche, alle immagini e alle altre risorse della sezione pubblica. In ogni caso, se non viene seguita una buona politica di sviluppo, questo può condizionare la mantenibilità di un sito.

Con l'avvento delle *Servlet* e delle `JSP`, sono state sviluppate moltissime applicazioni Web risultate, poi, poco mantenibili. Le cause di tutto ciò erano le seguenti:

- il flusso di controllo dell'applicazione Web, ovvero quali contenuti devono essere visualizzati e in quale ordine, era spesso codificato all'interno delle pagine Web stesse;
- la *business logic* del sito Web era strettamente associata alla presentazione dell'interfaccia utente.

Questa tipologia di architetture è, oggi, nota come *Model 1 architecture*. Essa si adatta solamente a siti di piccole dimensioni e con funzionalità limitate, o con pagine Web con limitati requisiti di espansione. Realizzare siti Web in questo modo è molto semplice e, quindi, la produttività aumenta quando la complessità è bassa. Questo modello non è, però, raccomandato per siti di maggiori dimensioni, in quanto il tempo risparmiato in fase di sviluppo verrebbe poi perso in fase di debug. La struttura di un'applicazione Web con architettura Model 1 è rappresentata in figura 7.4. Osservando la figura, è semplice capire perchè tale approccio può presto diventare ingestibile con l'aumento della complessità del sito Web e rende difficili anche i cambimenti nel controllo del flusso. Come detto in precedenza, uno dei problemi più frequenti consisteva nel mischiare logica dell'applicazione con presentazione dei dati. L'alternativa a tutto ciò consiste nel mantenere le pagine esenti da codice `Java` (ove possibile), e localizzare la logica in classi `Java`. Lo sviluppo delle moderne applicazioni Web include alcune regole di base che rendono i siti e le applicazioni più semplici da gestire ed, eventualmente, da estendere:

- Non incorporare la logica della gestione delle richieste degli utenti ed il flusso di controllo all'interno delle stesse pagine `JSP`. Questo rende difficile la manutenzione del codice.
- Non mescolare la logica applicativa con la logica dell'interfaccia utente (meglio nota come logica di presentazione). L'architettura MVC, che verrà illustrata più avanti, spiega come questo è fatto. Questa architettura è, talvolta, chiamata *Model 2 architecture* in contrasto con quella *Model 1* menzionata in precedenza.

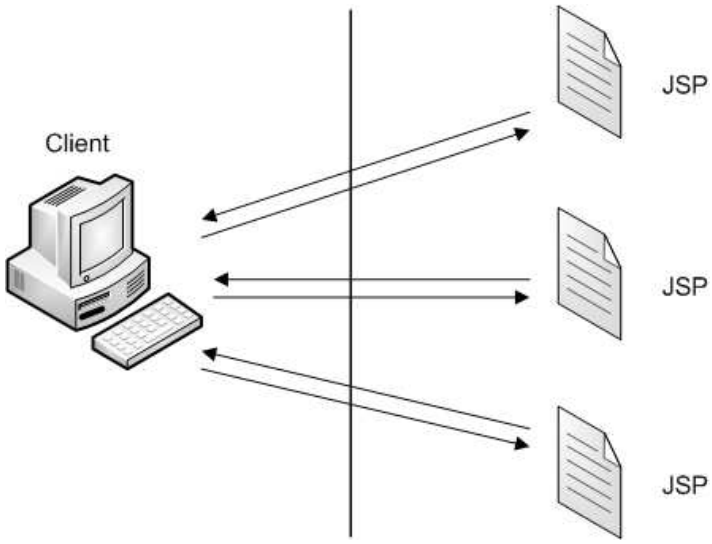

Figura 7.4. Struttura di un'applicazione Web con architettura Model 1.

- Tenere il codice `Java` fuori dalle pagine `JSP`. Le *Tag libraries* aiutano gli sviluppatori a fare tutto ciò.

- Un'altra buona pratica è quella di usare le pagine `JSP` come dei template. Per esempio l'intestazione della pagina che include, per ipotesi, il logo dell'azienda può trovarsi in una pagina `JSP`, il menu principale del sito in un seconda e l'informazione da visualizzare in una terza. Quando l'utente effettua la richiesta, questi elementi separati vengono assemblati e visualizzati dall'utente come un'unica pagina.

7.1.3 Architettura MVC

Nella sezione precedente è stato mostrato un esempio di applicazione con architettura *Model 1*, in cui le diverse pagine `JSP` devono sapere a quale pagine indirizzare l'utente e da quale esso proveniva. Tutto ciò diventa presto molto complicato e difficile da gestire per applicazioni sufficientemente complesse. L'architettura *Model 2* o *MVC* (*Model View Controller*) aiuta a risolvere il problema permettendo la separazione della logica applicativa dalla presentazione del contenuto `HTML`. Il *Model* corrisponde alla logica dell'applicazione, ovvero alle regole che determinano ciò che viene mostrato e a chi. Il componente *View* è dato, invece, dall'insieme delle pagine (nel nostro caso `JSP`) che visualizzano il contenuto creato. Il *Controller*, infine, determina quale parte del Model viene invocato e quale pagina `JSP` viene caricata per effettuare il rendering dei dati. In altre parole, il Controller definisce la struttura dell'applicazione e la logica del flusso delle pagine. Queste informazioni vengono lette da dei file di configu-

razione e non sono, quindi, "sepolte" nel codice delle pagine Web, a differenza di quanto avveniva con le applicazioni *Model 1*. La figura 7.5 mostra lo schema dell'architettura MVC.

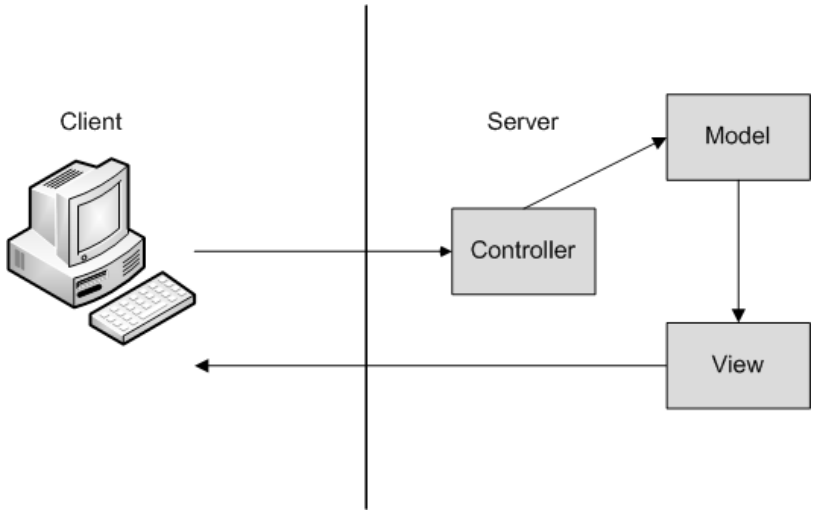

Figura 7.5. Struttura di un'applicazione Web con architettura MVC.

7.1.4 La struttura dell'applicazione WQF

Quanto emerso dall'analisi funzionale e dai colloqui con l'azienda, ha portato ad individuare nell'MVC un *pattern di progettazione* (*design pattern*[3]) quale punto di partenza nello sviluppo di un software in grado di soddisfare i requisiti espressi. Prima che il pattern MVC venisse definito, le interfacce grafiche tendevano a "mischiare" questi tre oggetti.

L'utilizzo di questo pattern permette, quindi, di disaccoppiare memorizzazione, manipolazione e presentazione dei dati. L'effetto finale è quello di permettere uno sviluppo e una gestione separata dei vari blocchi che compongono un software.
L'applicazione di questo design pattern nell'ambito dello sviluppo di un software per il sistema di gestione della qualità risulta molto interessante. È importante sottolineare la necessità di utilizzare un supporto informativo che sia completamente indipendente dalle viste presentate ai vari attori. Tutto ciò permette la massima integrazione con strumenti software già in uso, senza la necessità di dover cambiare il metodo con cui questi vengono gestiti ed utilizzati. È, infatti, esclusivo compito della porzione Control del software effettuare la traduzione tra dati grezzi e viste (o viceversa). Tra l'altro, Control non è necessariamente vincolato ad un unico modello di presentazione dei dati, ma può essere facilmente esteso per supportare diverse viste.

La figura 7.6 mostra lo schema completo dell'architettura MVC per l'applicazione WQF. Nel disegno sono, inoltre, indicate le tecnologie utilizzate, le quali verranno approfondite nel prossimo capitolo.

7.2 Tecnologie e software utilizzati

7.2.1 Il linguaggio di programmazione: `Java`™

`Java`[4] fu concepito da James Gosling, Patrick Naughton, ChrisWarth, Ed Frank e Mike Sheridan in Sun Microsystems Inc. nel 1991 e richiese 18 mesi per sviluppare la prima versione funzionante. Inizialmente il linguaggio fu battezzato "OAK" ma venne rinominato "Java" nel 1995. Tra la versione iniziale di Oak nell'autunno 1992 e l'annuncio al pubblico di `Java` nella primavera del 1995, molte altre persone avevano contribuito al progetto e all'evoluzione del linguaggio. Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin e Tim Lindholm diedero un contributo importantissimo per la maturazione del prototipo originale. La spinta iniziale per `Java` non venne da Internet, bensì dal bisogno di un linguaggio

[3] Nell'ingegneria del software, un design pattern (struttura di progettazione) può essere definito "una soluzione progettuale" generale a un problema ricorrente. Esso non è una libreria o un componente di software riusabile, quanto una descrizione o un modello da applicare per risolvere un problema che può presentarsi in diverse situazioni durante la progettazione e lo sviluppo del software. La differenza tra un algoritmo e un design pattern è che il primo risolve problemi computazionali, mentre il secondo è legato agli aspetti progettuali del software.

[4] `http://java.sun.com`

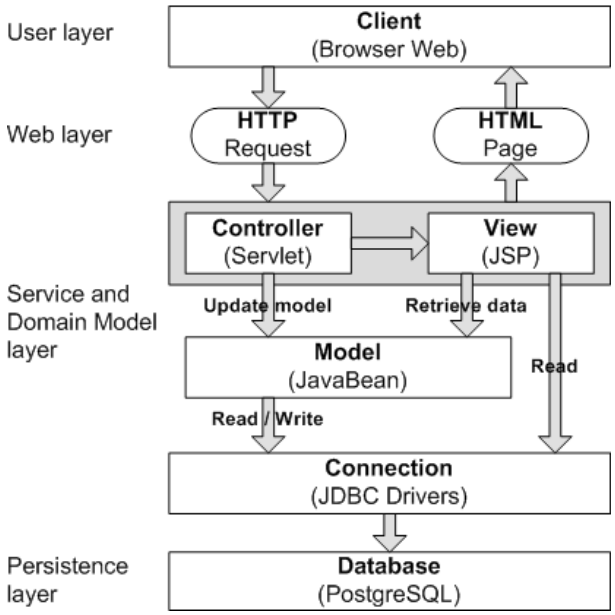

Figura 7.6. Schema completo dell'architettura MVC per l'applicazione WQF.

indipendente dalla piattaforma, cioè neutro rispetto all'architettura, che avrebbe potuto essere impiegato per creare software da incorporare in diversi dispositivi elettronici commerciali come forni a microonde e telecomandi.

`Java` è un linguaggio di programmazione *general-purpose*, concorrente, basato su classi e orientato agli oggetti; deriva dal `C` e dal `C++` ma è organizzato in modo differente, con molti aspetti omessi e nuove idee prese da altri linguaggi. Ecco i punti di forza di questo linguaggio:

- *semplice*: è stato concepito per essere semplice da apprendere, efficace da usare e rivolto ai programmatori professionisti; pur dovendo molto a `C++`, `Java` risulta essere molto più semplice e tollerante grazie anche alla mancanza di puntatori;

- *orientato agli oggetti*: implementa questo paradigma in modo esteso con l'eccezione, per ragioni di efficienza, dei tipi semplici di dati;

- *robusto*: è fortemente tipizzato, quindi controlla il codice già durante la compilazione; inoltre non c'è la necessità di dover liberare la memoria quando si è terminato di usare un oggetto perchè questo compito è demandato al *garbage collector*; anche la gestione delle eccezioni è orientata agli oggetti;

- *multiprocesso*: il sistema *run-time* di `Java` fornisce una soluzione elegante, anche se sofisticata, per la sincronizzazione di multiprocessi che mette in grado di costruire sistemi interattivi funzionanti e affidabili;
- *indipendente dall'architettura*: l'obiettivo dei progettisti era "*scrivi una volta per tutte, esegui ovunque, in ogni momento, per sempre*" e per gran parte è stato raggiunto;
- *interpretato ad alte prestazioni*: `Java` consente la creazione di programmi multipiattaforma attraverso la compilazione in una forma intermedia chiamata bytecode; questo codice è stato pensato per ottenere alte prestazioni e può essere interpretato su ogni sistema che disponga di una Java Virtual Machine;
- *grande supporto*: `Java` dispone di varie API (*Application Program Interface*) per quasi ogni ambito, molto utili per lo sviluppo di applicazioni.

Lo scotto maggiore per questa serie di vantaggi lo si paga sul lato delle prestazioni visto che, a differenza dei linguaggi compilati, `Java` viene interpretato al momento dell'esecuzione.

7.2.2 Il Web Container: Apache Tomcat

Ai tempi della definizione delle specifiche `JSP 1.0` e `Servlet 2.0`, la Sun sviluppò il suo *Java Server Web Development Kit poi* rinominato *Java Servlet Development Kit* (*JSDK*), che utilizava un Web Server interamente realizzato in `Java`. Nel frattempo, un gruppo esterno di sviluppatori Open-Source, l'*Apache Java Group*, stava lavorando su un motore `JSP/Servlet` che fosse compatibile con le ultime API dei servlet e delle `JSP` ma che avrebbe dovuto integrarsi con i server Apache largamente diffusi; il motore prese il nome di *Apache JServ*. Realizzarono *JServ* in due parti: la prima era scritta in `C` e doveva fungere da modulo caricabile per il server Apache, l'altra era una implementazione in `Java`, di un *servlet container standalone*. Le due parti comunicavano per mezzo di un protocollo privato. Mentre il progetto *JSDK* della Sun era focalizzato sull'adesione alle specifiche, l'*Apache JServ* si concentrò sulle performance e sugli aspetti più pragmatici. Diventò presto evidente che i due progetti si sovrapponevano e che gli sforzi si sarebbero dovuti unire. Nel 1999 la Sun donò alla Apache Software Foundation il codice sorgente dell'implementazione di riferimento per le `JSP` e le `Servlet`; per integrare i due progetti sorse il gruppo di collaborazione *Jakarta* che finì, però, per includere tutti i progetti *open-source* dell'*Apache Java Group*. Tomcat[5] è uno di questi progetti: la serie 3.x discende direttamente dal progetto della Sun mentre la versione 4 impiega una nuova architettura ad alte prestazioni. L'attuale release di Tomcat è la 7 che implementa le specifiche per `Servlet 3.0` e `JSP 2.2`.

[5] `http://tomcat.apache.org`

Tomcat è un *Web Server* completamente scritto in `Java`. Più precisamente, si tratta di un *servlet container standalone* e rappresenta l'implementazione di riferimento per le tecnologie `JSP` e `Servlet`, con l'aggiunta di alcune caratteristiche che lo rendono più efficiente e ne fanno un'utile piattaforma per sviluppare e far girare applicazioni Web. Il servlet container gira su una Java Virtual Machine e può essere affiancato da un server Web in tre modi:

- *in-process*: il servlet container è legato al server Web da un *plug-in* che fa un po' da mediatore tra i due; *plug-in* e container si trovano nello stesso spazio di memoria del server così come la JVM che li esegue; questa configurazione garantisce le massime prestazioni dovute alla condivisione della memoria ma, d'altro canto, limita la scalabilità e l'affidabilità (il crash di un qualsiasi thread può portare al crash dell'intero server);
- *out-of-process*: a differenza del tipo precedente, qui si usano due spazi di memoria distinti: in uno gira il server con il *plug-in* `Java`, nell'altro si trova la JVM con il *servlet container*; solitamente plug-in e *container* comunicano usando il protocollo TCP-IP;
- *stand-alone*: in questo caso il *servlet container* funge da server Web vero e proprio e risponde direttamente alle richieste dei clients.

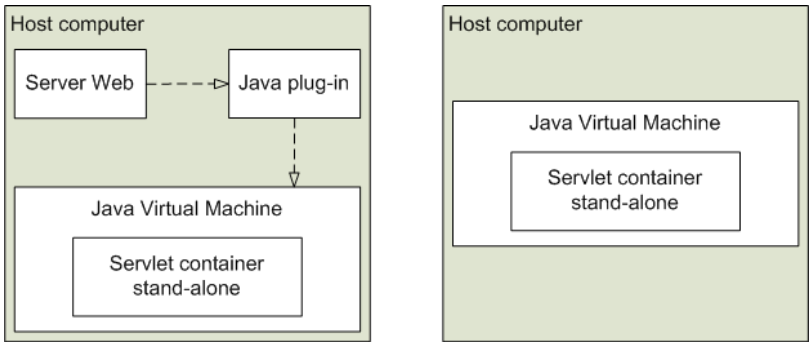

Figura 7.7. Un servlet container di tipo *out-of-process* (a sinistra) ed uno di tipo *stand-alone* (a destra).

La figura 7.7 mostra la struttura dei servlet container di tipo *out-of-process* e *stand-alone*.

Tomcat fornisce un servlet container di tipo *stand-alone*. Quest'ultimo, infatti, mette a disposizione i servizi di rete necessari a ricevere le richieste, decodificarle in base al protocollo `HTTP` e inviare le risposte. Inoltre ospita e gestisce le servlet durante tutto il loro ciclo di vita. Il flusso dei messaggi generato da una richiesta `HTTP` dell'utente (il client) si articola come segue:

- la richiesta viene ricevuta dal server Web e consegnata al *servlet container* che, come si è detto, può essere fisicamente separato;

- il *servlet container* determina, in base alla sua configurazione e ai parametri della richiesta, a quale servlet dovrà passare il controllo;
- il servlet usa l'oggetto che rappresenta la richiesta `HTTP` per individuare l'utente remoto e ricavarne il contenuto; esegue le operazioni per cui è stato programmato e prepara i dati da inviare al client;
- non appena il controllo ritorna al *servlet container*, questo si assicura che la risposta venga interamente inviata al client.

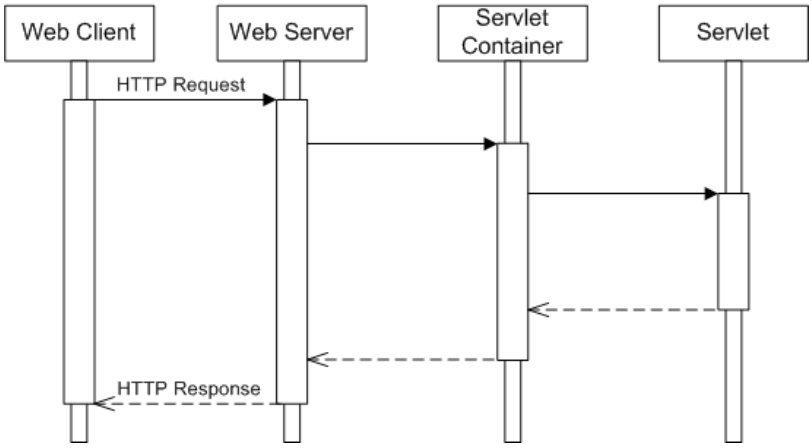

Figura 7.8. Flusso dei messaggi in un server Web dotato di un *servlet container* `Java`.

I parametri di configurazione e di inizializzazione di Tomcat, come quelli di ogni singola applicazione Web installata, vengono specificati da alcuni file in formato XML:

- `server.xml` viene letto ad ogni avvio del servlet container e contiene informazioni come il nome del server, la porta per la connessione `HTTP`, la directory di installazione delle applicazioni, i timeouts;
- `web.xml` può essere presente nella directory /WEB-INF/ di ciascuna applicazione e specifica:
  - quale meccanismo di autenticazione usare;
  - i parametri di inizializzazione dei servlet;
  - eventuali filtri che mappano le richieste `HTTP` e le inoltrano al servlet corretto;
  - eventuali pagine di errore;
  - la configurazione delle sessioni di lavoro;
  - l'uso di eventuali librerie di tag personalizzati.

7.2.3 I Java Beans

Come si è accennato non è molto comodo e conveniente inserire troppo codice `Java` (cioè *scriptlet*) all'interno delle pagine `JSP`. Può, invece, essere molto utile racchiuderlo in una classe apposita che viene chiamata al momento del bisogno; per questo esistono i `JavaBeans`[6] ovvero l'architettura a componenti di `Java`. I vantaggi offerti da questi componenti software nell'ambito delle applicazioni Web sono quelli di permettere la riusabilità del software (*Write Once Run Anywhere*, ovvero "scrivi una volta, esegui dappertutto"), la separazione tra contenuto e codice, l'implementazione di uno stato dell'applicazione (compensando la natura "senza stati", *stateless*, del protocollo `HTTP`). La specifica relativa ai `JavaBeans` è molto complessa, ma vale la pena di elencarne alcune caratteristiche:

- non devono avere alcuna proprietà pubblica;
- le proprietà che devono essere "esposte" devono avere metodi `set` e `get` secondo necessità;
- devono avere almeno un metodo costruttore senza argomenti per poter caricare il bean a piacimento.

Le `JSP` mettono a disposizione alcune direttive per accedere alle proprietà dei beans.

7.2.4 Spring Framework

Affrontare la progettazione di un'applicazione comporta la scelta quasi obbligata di un *framework*. Sebbene si possa pensare di poter realizzare da soli ogni aspetto del proprio lavoro, è saggio ricorrere a soluzioni già sperimentate e ampiamente testate offerte da numerosi progetti open source che forniscono gli strumenti di base per lo sviluppo veloce del proprio codice. La scelta è, fortunatamente, molto ricca. Probabilmente il *framework* di maggior successo è *Spring Framework*[7] e, questo, è stato scelto per lo sviluppo del progetto WQF. Il principio guida di tale *framework* è quello di essere leggero; tale leggerezza non si riferisce al numero di classi o alle dimensioni della distrubuzione, quanto, piuttosto, al suo minimo impatto. Ottenere i benefici del nucleo fondamentale di *Spring Framework* richiede, infatti, minimi cambiamenti nel codice della propria applicazione. Il nucleo principale appena citato di *Spring Framework* è basato sull'*Inversion of Control* altrimenti detto *Dependency Injection*; si tratta di una tecnica che estrae la creazione e la gestione delle dipendenze dei componenti. Si consideri, ad esempio, una classe `Alpha` che dipende da un'istanza della classe `Beta` per realizzare un determinato compito; tipicamente `Alpha` dovrebbe creare un'istanza di `Beta` utlizzando l'operatore `new` o una classe `factory`; l'approccio

[6] `http://java.sun.com/javase/technologies/desktop/javabeans/`
[7] `http://www.springsource.org/`

della *Dependency Injection* permette di fornire l'istanza necessaria a `Alpha` in *runtime*. I benefici di tale approccio si possono sintetizzare nei seguenti punti:

Ridurre il codice di legame : uno dei punti di maggior valore della *Dependency Injection* è la capacità di ridurre il codice da scrivere per legare i differenti componenti di un'applicazione. Spesso ciò può essere banale riducendosi alla creazione di una nuova istanza, ma ciò può risultare complesso nel caso di risorse remote.

Estrarre le dipendenze : ciò consente di passare dall'implementazione di una dipendenza a un'altra più facilmente.

Gestire le dipendenze in un unico luogo : un'applicazione di una certa complessità ha tipicamente le sue dipendenze diffuse su tutto il codice che la costituisce. Usando la *Dependency Injection* tutte le informazioni riguardanti le dipendenze sono contenute in un unico luogo rendendo la gestione di tali dipendenze meno problematica.

Aumentare la testabilità : programmando una classe in funzione della *Dependency Injection* implica la possibilità di sostituire le dipendenze facilmente, aspetto particolarmente utile in fase di test. Se un'applicazione necessita dell'accesso a un database, ma si vuole fare un test indipendentemente da esso, sarà sufficiente creare un'implementazione fittizia della dipendenza dal database che ritorni una serie di dati da passare all'oggetto sotto test; tale meccanismo favorisce anche il test di applicazioni web, basterà usare dei *mock object*[8] di `MockHttpServletRequest` e `MockHttpServletResponse`.

Promuovere una buona progettazione : progettare per la *Dependency Injection* significa, essenzialmente, programmare per interfacce: tipicamente i maggiori componenti sono definiti da interfacce le cui concrete implementazioni sono create e collegate dalla *Dependency Injection*.

Numerose sono le possibilità offerte da *Spring Framework* come si può vedere dall figura 7.9. Innanzitutto, il supporto dell'AOP (*Aspect Oriented Programming*) fornisce la possibilità di realizzare una logica di controllo trasversale, cioè applicabile a più parti di un'applicazione automaticamente. Vi sono due principali tipi di AOP:

- statico: fornisce la possibilità, durante la compilazione, di costruire la logica AOP con cui arricchire l'applicazione;

[8] Nella programmazione orientata agli oggetti, i *mock objects* sono oggetti simulati che imitano il comportamento degli oggetti reali in modo controllato. Un programmatore di computer crea, in genere, un oggetto fittizio per testare il comportamento di qualche altro oggetto, più o meno allo stesso modo in cui il progettista di una vettura utilizza un manichino da crash test per simulare il comportamento dinamico di un umano negli impatti del veicolo.

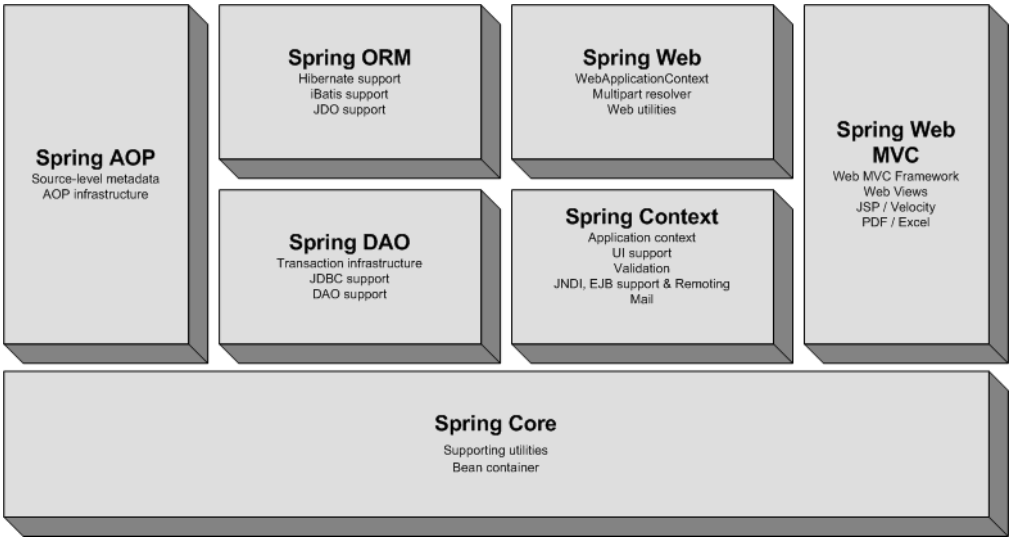

Figura 7.9. Struttura di *Spring Framework*.

- dinamico: è il caso di *Spring Framework* in cui la logica trasversale è applicata in *runtime*.

Spring Framework supporta anche numerose tecnologie DAO (*Data Access Object*) che possono essere integrate simultaneamente. Oltre alla JDBC (*Java DataBase Connectivity*), *Spring Framework* consente l'utilizzo di diverse tecnologie di ORM (*Object Relational Mapping*) tra cui Hibernate, JDO (*Java Data Objects*), Oracle TopLink, OJB (*Apache Jakarta's ObJect Relational Bridge*), iBATIS SQL e JPA (*Java Persistence API*). *Spring Framework* consente, inoltre, di relizzare transazioni sia dichiarative che programmatiche. Sebbene questo framework consenta lo sviluppo di applicazioni desktop, esso ha un supporto particolarmente ricco per le web application di cui si sono stati trattati vari aspetti nel corso della presente tesi. Un altro ricco settore di sviluppo con *Spring Framework* è quello del RMI (*Remote Method Invocation*) di cui supporta `Java` RMI, Caucho Hessian, Caucho Burlap e, oltre a tali protocolli, anche un protocollo basato su `HTTP` che sfrutta la serializzazione di Java™. Un altro campo in cui si distingue questo framework è la gestione dell'invio di email, esso fornisce ricche API e la possibilità di configurare due implementazioni: `JavaMail` e `MailMessage`. Molte applicazioni spesso necessitano di operazioni di avvicendamento come l'invio di informazioni a clienti o l'avvio, in dati momenti, di funzionalità dell'applicazione; anche in questo campo *Spring Framework* offre diverse possibilità: una utilizza la classe `Timer` di `Java`™, l'altra `Quartz Scheduler`. Sebbene vi siano ancora diverse funzionalità supportate da *Spring Framework* si conclude questa breve presentazione citando l'opportunità, offerta dallo stesso, di evitare la scrittura di molto codice per gestire le eccezioni fornendo, allo stesso tempo, una gerarchia

di eccezioni veramente granulare.

7.2.5 L'ambiente di sviluppo: Eclipse

Eclipse[9] è un progetto open source legato alla creazione e allo sviluppo di una piattaforma di sviluppo ideata da un consorzio di grandi società quali *Ericsson*, *HP*, *IBM*, *Intel*, *MontaVista Software*, *QNX*, *SAP* e *Serena Software*, chiamato *Eclipse Foundation*, e creata da una comunità strutturata sullo stile dell'open source. Pur essendo orientato allo sviluppo del progetto stesso, questo IDE (*Integrated Development Environment*, in italiano "ambiente di sviluppo integrato") è utilizzato anche per la produzione di software di vario genere. Si passa infatti da un completo IDE per il linguaggio `Java` (JDT, *Java Development Tools*) ad un ambiente di sviluppo per il linguaggio `C++` (CDT, *C/C++ Development Tools*) e a plug-in che permettono di gestire `XML`, `PHP` e persino di progettare graficamente una GUI per un'applicazione `Java` (Eclipse VE, *Visual Editor*), rendendo di fatto *Eclipse* un ambiente RAD. Il programma è scritto in linguaggio `Java`, ma anziché basare la sua GUI su Swing, il toolkit grafico di Sun Microsystems, si appoggia a `SWT`, librerie di nuova concezione che conferiscono ad Eclipse una straordinaria reattività. La piattaforma di sviluppo è incentrata sull'uso di plug-in, delle componenti software ideate per uno specifico scopo, per esempio la generazione di diagrammi `UML`, ed in effetti tutta la piattaforma è un insieme di plug-in, versione base compresa, e chiunque può sviluppare e modificare i vari plug-in. Nella versione base è possibile programmare in `Java`, usufruendo di comode funzioni di aiuto quali: completamento automatico (*Code completion*), suggerimento dei tipi di parametri dei metodi, possibilià di accesso diretto a CVS e riscrittura automatica del codice (funzionalità questa detta di *Refactoring*) in caso di cambiamenti nelle classi. Essendo scritto in `Java`, *Eclipse* è disponibile per le piattaforme Linux, HP-UX, AIX, Mac OS X e Windows.

7.2.6 Il DBMS: PostgreSQL

PostgreSQL[10] ebbe inizio come *Ingres*, sviluppato all'università di Berkeley in California tra il 1977 e il 1985. Il sorgente di *Ingres* fu preso e migliorato dalla *Relational Technologies/Ingres Corporation* che produsse i primi server database che ottennero successo commercialmente (*Ingres Corp.* fu, in seguito, acquistata dalla *Computer Associates*). Nel 1986, a Berkeley, Michael Stonebraker capeggiò un gruppo per sviluppare un database server chiamato *Postgres* (1986-1994) sponsorizzato dalla *Defense Advanced Research Projects Agency* (*DARPA*), dall'*Army Research Office* (*ARO*), dalla *National Science Foundation* (*NSF*), e da *ESL Inc.*. Il sorgente di Postgres fu preso dalla *Illustra* e sviluppato come prodotto commerciale (l'*Illustra* fu più tardi comprata da *Informix* e integrata nel *Informix's Universal Server*). Due studenti laureati a Berkeley,

[9] `http://www.eclipse.org/`
[10] `http://www.postgresql.org/`

Jolly Chen e Andrew Yu, aggiunsero l'interfaccia `SQL` a *Postgres*, e lo chiamarono *Postgres95* (1994-1995). Essi lasciarono Berkeley, ma Jolly continuò ad aggiornare e supportare *Postgres95*, che ha ancora una mailing list attiva. Nell'estate 1996, divenne chiaro che la domanda di un database server `SQL` con tecnologia Open Source era forte e che, quindi, si doveva formare un team per continuare lo sviluppo. Marc G. Fournier a Toronto (Canada) si offrì di ospitare la mailing list e approntò un server per ospitarne i sorgenti. Il migliaio di iscritti alla mailing list furono spostati sul quella nuova ed il server fu configurato, fornendo alle persone gli account per apportare le modifiche ai files sorgenti usando *CVS*. Le persone maggiormente impegnate furono: Marc G. Fournier, Thomas Lockhart da Pasadena (California), Vadim Mikheev da Krasnoyarsk (Russia), e Bruce Momjian. Il loro primo obiettivo fu esaminare la vecchia mailing list, valutare le patch che erano state pubblicate per fisssare i vari problemi. Allora il sistema era molto fragile e non facilmente comprensibile. Pur parlando, già a quei tempi, di aggiungere delle caratteristiche, l'instabilità del sistema costrinse gli sviluppatori a focalizzarsi sulla soluzione dei problemi esistenti. Nel 1996 il team decise di cambiare il nome da *Postgres95* a *PostgreSQL* che evidenziava le doti `SQL` e rilasciarono le release ogni 3-5 mesi. L'ORDBMS *PostgreSQL* viene, oggi, distribuito sotto licenza BSD.

PostgreSQL è uno dei più avanzati database *Open Source* e la ricchezza delle sue funzionalità può essere paragonata a database commerciali quali *DB2* e *Oracle*. Di seguito si riporta una breve lista delle caratteristiche principali:

- **DBMS relazionale e a oggetti**: si tratta infatti di un database che oltre ad essere relazionale è anche a oggetti, cioè estende alla base di dati il paradigma di programmazione a oggetti;
- **elevata estendibilità**: l'utente può usare operatori, funzioni, metodi di accesso e tipi di dati definiti dall'utente stesso;
- **supporto completo di** `SQL`: *PostgreSQL* supporta le specifiche fondamentali di `SQL99` e funzionalità avanzate di `SQL92`;
- **integrità referenziale**: *PostgreSQL* implementa l'integrità referenziale per garantire la validità dei dati nel database;
- **API flessibili**: tra queste interfacce sono comprese quelle per `Object Pascal`, `Python`, `Perl`, `PHP`, `ODBC`, `JDBC`, `Ruby`, `TCL`, `C/C++` e `Pike`;
- **linguaggi procedurali**: supporta linguaggi procedurali interni come quello nativo `PL/pgSQL` e altri quali `Perl`, `Python` e `Tcl/Tk`;
- **Multi Version Concurrency Control (MVCC)**: tecnologia usata per evitare i lock non necessari. Normalmente l'utente in fase di lettura è bloccato da chi sta aggiornando i record. In *PostgreSQL* chi legge non viene mai bloccato da chi scrive perchè il database tiene traccia di tutte

le transazioni effettuate dagli utenti ed è in grado di gestire i record senza far attendere la loro disponibilità;

- **client/server**: *PostgreSQL* riserva un processo per ogni utente che cerchi di connettersi;
- **Write Ahead Logging (WAL)**: tecnologia che incrementa l'affidabilità registrando le modifiche prima che siano scritte nel database; ciò garantisce che, nel caso si verifichi un crash, esista un salvataggio delle transazioni con cui effettuare il rispristino.

7.3 *Annotations*

Le *annotations* sono state rilasciate formalmente con la versione 1.5.0 del JDK (alias JDK 5.0) nel 2004, dopo essere state introdotte nel 2002 dalla JCP[11] sotto il nome di JSR-175[12] .

Brevemente le annotazioni sono tags che riguardano porzioni di codice `Java` che vengono processati da tools complementari al compilatore standard `javac`. Il compilatore `Java` standard, infatti, riesce ad interpretare solo poche annotazioni predefinite e per un uso più avanzato occorre implementare un tool di processamento oppure utilizzare tool di terze parti (ad esempio Springframework).

Gli usi che si possono fare delle annotazioni sono per esempio le seguenti:

- Generazione automatica di file ausiliari, come le class di informazioni dei bean oppure i *deployment descriptor*.
- Generazione automatica di codice per il testing, il logging, la semantica delle transazioni ecc.

Come si vedrà più avanti, nel progetto WQF si è fatto largo uso delle seconde (ad esempio `@Autowired`, in cui un bean `Alpha` viene iniettato in un altro bean `Beta`, senza necessità di dichiararlo nel file `xml`). Esempi di annotazioni del primo tipo sono:

- `@Override` - che informa il compilatore che si sta facendo override di un elemento della superclasse-;
- `@Deprecated` indica che l'elemento marcato è deprecato e diverrà obsolescente a breve.

Formalmente una annotazione può essere definita, tramite una *annotation interface*, nel seguente modo:

[11] *Java Community Process*: programma *open*, nel senso che la partecipazione è aperta a tutti, che si occupa dello sviluppo e approvazione di nuove specifiche per future versioni e funzionalità della tecnologia Java

[12] *Java Specification Request*: documenti formali che descrivono specifiche di nuove funzionalità Java.

```
1 public @interface Esempio
2 {
3 //qui possono andare eventuali metodi...
4 }
```

Listing 7.1. Esempio di definizione di una *Annotation*

La dichiarazione di `@interface`[13], in pratica crea una Java `interface` e le classi che concretamente la implementano vengono date in pasto ai tools che le andranno a processare. Si noti che i tools possono inoltre chiamare gli eventuali metodi definiti nel corpo della annotazione.

L'uso della annotazione appena definita può essere utilizzata come un qualsiasi *modificatore*[14] per annotare un variabile di istanza, una classe, un metodo, una variabile locale o addirittura un'altra annotazione, a seconda degli, si veda un esempio in seguito, `ElementType`[15] definiti nella *meta-annotazione* `@Target`.

Di seguito vediamo un esempio di annotazione un po' più completa:

```
1 @Target(ElementType.METHOD)
2 @Retention(RetentionPolicy.RUNTIME)
3 public @interface AltroEsempio
4 {
5
6 String value() default "none";
7
8 }
```

Listing 7.2. Altro esempio una *Annotation*

`Target`[16] e `Retention`[17] sono *meta-annotazioni* che annotano `AltroEsempio`. La prima informa il compilatore che quest'ultima può essere applicata solo a metodi, pena l'errore in compilazione; la seconda invece, informa il compilatore che deve essere trattenuta a *runtime*, quando il file class è caricato nella Virtual Machine. Per maggiori dettagli si rimanda alle API `Java`[18]. La presenza del metodo value(), indica che l'uso dell'annotazione accetta un argomento di tipo `String` che ha come valore di default "none".

Altre due *meta-annotazioni* da menzionare sono: `Documented` e `Inherited`. Il primo aiuta nella produzione della documentazione -tramite Javadoc per esempio-

[13] dove `@` si legge AT, ovvero *Annotation Type*

[14] altri modificatori sono ad esempio: `public`, `protected` o `static`

[15] oggetto di tipo `enum`. Per una lista di valori completi si veda: `http://download-llnw.oracle.com/javase/6/docs/api/java/lang/annotation/ElementType.html`.

[16] se non viene definita, vuol dire che l'annotazione può essere utilizzata in qualunque elemento del programma

[17] quando non viene definita viene impostata al valore di default che è `RetentionPolicy.CLASS`, ovvero che la classe annotata viene registrata nel file `Class` e non trattenuta a runtime

[18] http://download.oracle.com/javase/6/docs/api/

, mentre il secondo si applica solo ad annotazioni per classi: per esempio supponendo di avere definito una annotazione `@Ereditabile`, annotata a sua volta da `Inherited`, la seguente classe `Figlio` è automaticamente annotata come `@Ereditabile`.

```
1 @Inherited
2 public @interface Ereditabile{ ... }
3
4 @Ereditabile class Padre{}
5
6 class Figlio extends Padre{}//eredita la annotazione
    @Ereditabile
```

Listing 7.3. Esempio di uso di `@Inherited`

7.3.1 *Annotations* utilizzate nel progetto WQF

Di seguito si illustrano tutte le annotazioni utilizzate nel progetto WQF. Come si vedrà, molte annotazioni sono state implementate per esemplificare la stesura del file di configurazione `xml`, mentre altre sono state usate per far fronte alla ridefinizione dei `Controller` ed altre ancora per la semantica delle *transazioni*.

`@Autowired` Questa annotazione è stata largamente utilizzata per la *Dependency Injection*. L'approccio delle annotazioni ha permesso il quasi azzeramento delle righe di configurazione `xml`. Per una completa comprensione, di seguito si illustra la definizione riportata nell'API `org.springframework.beans.factory.annotation.Autowired`:

```
1 @Retention(value=RUNTIME)
2 @Target(value={CONSTRUCTOR,FIELD,METHOD})
3 public @interface Autowired
```

Listing 7.4. Estratto del Javadoc di `@Autowired`

Come si nota, `@Autowired` può essere utilizzato per annotare metodi (ad esempio i setters), variabili d'istanza, oppure costruttori. Nello specifico in WQF si è fatto uso di questa annotazione sulle variabili d'istanza, come riportato nel seguente esempio:

```
1 @Controller
2 @RequestMapping("/processDevelopment/createProcess.htm")
3 public class CreateProcessController{
4     @Autowired
5     private ProcessDevelopmentService procDevService;
6     //...
7 }
```

Listing 7.5. Esempio uso di `@Autowired` in WQF

In Springframework, l'oggetto che si occupa di processare `@Autowired` è l'oggetto `AutowiredAnnotationBeanPostProcessor`, che implementa l'interfaccia `BeanPostProcessor`. Tutti gli oggetti o metodi annotati con `@Autowired` vengono processati iniettandovi il bean collegato non appena quest'ultimo viene creato. Si nota che l'oggetto `AutowiredAnnotationBeanPostProcessor` viene registrato implicitamente dal tag `xml <context:component-scan/>` e non c'è bisogno quindi di implementarlo programmaticamente.

`@Controller`,`@Service` e `@Repository` Queste tre annotazioni sono specializzazioni dell'annotazione `@Component` e servono per annotare classi che verranno poi individuati automaticamente durante lo scanning del classpath da parte del*bean* `ClassPathBeanDefinitionScanner` e registrato nell'`Application Context` per la *Dependency Injection*. In breve, come dice il loro stesso nome, le classi che interrogano il database vengono annotati con `@Repository`, le classi di servizio con `@Service` e quelle abilitate a controller del *design-pattern* MVC, con `@Controller`, si veda come esempio di uso il listato 7.5. Di seguito un esempio di uso di `@Service`:

```
1 @Service
2 public class ProcessDevelopmentServiceImpl implements
     ProcessDevelopmentService {
3     @Autowired
4     private ProcessDevelopmentDAO blDevDAO;
5     ...
6 }
```

Listing 7.6. Esempio uso di `@Service` in WQF

Esmpio di uso di `@Repository`

```
1 @Repository
2 public class ProcessDevelopmentDAOImpl implements
     ProcessDevelopmentDAO {
3     ...
4 }
```

Listing 7.7. Esempio uso di `@Repository` in WQF

`@RequestMapping`, `@ModelAttribute`, `@RequestParam`, `@InitBinder` L'annotazione `@RequestMapping` solitamente è utilizzata in modo congiunto alla già citata `@Controller`. Nelle versioni precendenti al 2.5 di Springframework, per mappare le richieste web al corrispettivo `Handler` si doveva esplicitamente dichiarare nel file di configurazione `xml` i rispettivi `HandlerMapping`. Mentre con l'uso delle annotazioni tutto ciò è implicitamente implementato da `@RequestMapping` e `@Controller`. Il loro uso permette al `DefaultAnnotationhandlerMapping`, abilitato di default dal `DispatcherServlet`, di mappare le richieste web provenienti dai client. In particolare quando si usa `@RequestMapping` a livello di

classe, essa prende come parametro una `String` che indica il criterio per il quale la richiesta deve essere gestita dal `Controller` associato. Con granularità più fine, poi, l'annotazione `@RequestMapping` viene utilizzata per la gestione del tipo di richiesta.

A titolo di esempio si riporta il seguente:

```
1  @Controller
2  @RequestMapping("/processDevelopment/createProcess.htm")
3  public class CreateProcessController{
4  //...
5  @RequestMapping(method=RequestMethod.POST)
6  public ModelAndView onSubmit(@ModelAttribute("newProcess")
       Process cp, BindingResult result)
7  {
8  //...
9  }
10 @RequestMapping(method=RequestMethod.GET)
11 public ModelMap view()
12 {
13 //...
14 }
```

Listing 7.8. Esempio uso di `@RequestMapping` in WQF

La semantica del precedente listato è la seguente:

- Tutte le richieste all'indirizzo `/processDevelopment/createProcess.htm` devono essere gestite da `CreateProcessController`
- Se la richiesta `http` è una `GET` allora invoca il metodo view()
- Se la richiesta `http` è una `POST` allora invoca il metodo onSubmit()

Da aggiungere che `@RequestMapping` offre una flessibilità elevata per la definizione della politica di gestione delle richieste al `Controller` associato, per ulteriori dettagli si rimanda alla documentazione ufficiale di Springframework riportata in bibliografia [**?**].

L'annotazione `@ModelAttribute` viene utilizzata per esporre un oggetto per una web view, oppure può essere utilizzata per il *binding* di un parametro in uno specifico attributo di modello.

`@RequestParam` viene utilizzato per indicare che il parametro di un metodo è il *binding* di un parametro di una richiesta -ad es. `HttpServletRequest`-.

`@InitBinder` viene utilizzato per inizializzare l'oggetto `WebDataBinder`, quest'ultimo è utilizzato per popolare gli oggetti form e command.

Nel progetto WQF `@InitBinder` è stato utilizzato per il *binding* corretto degli oggetti `Date`.

`@Transactional` Questa annotazione, nel progetto WQF, è stata utilizzata come base per la definizione di due annotazioni personalizzate: `@SerializableTx` e `@ReadTx`. Di seguito si riportano le classi che le definiscono, contenuti nel package `it.unipd.dei.wqf.annotation`:

```
1 @Transactional(value="mytrans",isolation=Isolation.
    SERIALIZABLE)
2 @Target({ElementType.METHOD,ElementType.TYPE})
3 @Retention(RetentionPolicy.RUNTIME)
4 public @interface SerializableTx {
5 }
```

Listing 7.9. Classe che definisce `@SerializableTx`

```
1 @Transactional(value="readtx",isolation=Isolation.
    READ_COMMITTED)
2 @Target({ElementType.METHOD,ElementType.TYPE})
3 @Retention(RetentionPolicy.RUNTIME)
4 public @interface ReadTx {
5
6 }
```

Listing 7.10. Classe che definisce `@ReadTx`

Come facilmente si può intuire dal nome la prima può essere utilizzata per annotare un metodo transazionale che richieda il più alto grado di isolamento[19]per evitare le anomalie quali *dirty read*[20], *nonrepeatable read*[21] e *phantom read*[22], di due o più transazioni concorrenti. In particolare, con questa annotazione sono state segnati tutti quei metodi del livello Service che comportavano una qualche modifica di qualunque tabella del database; questo per prevenire il caso di inconsistenze dovute a più transazioni concorrenti che modificano le stesse porzioni di database.

Anche se PostgreSQL ha come livello di isolamento di default in READ COMMITTED e quindi sarebbe bastato annotare un metodo con `@Transactional` per avere un livello di isolamento adeguato, si è ritenuto opportuno esplicitare con l'annotazione `@ReadTx` la semantica delle transazioni di certi metodi. In particolare il livello di isolamento READ COMMITTED evita solo i *dirty read*.

Si noti che solo i metodi del *Service Layer* sono stati annotati come `@ReadTx` e `@SerializableTx`, in quanto sono complessi a sufficienza da formare una transazione logica. Nel `Dao Layer` i metodi si riducono a query isolate su por-

[19] `SERIALIZABLE`

[20] fenomeno per cui una transazione A legge dati di una transazione B concorrente che non ha ancora effettuato il *commit*

[21] fenomeno per cui una transazione A, rileggendo dati che aveva precedentemente letto, li trova modificati da una transazione B concorrente che nel frattempo ha effettuato *commit*

[22] una transazione A riesegue una query restituendo un insieme di righe che soddisfa la condizione di ricerca del `where`, trova che l'insieme delle righe che soddisfano la condizione è cambiato, a causa di una transazione B concorrente che ha effettuato *commit* nel durante

zioni limitate sul database e non sono complessi abbastanza da richiedere una transazione.

7.3.2 Implementazione dello Schedulatore

Nella logica del sistema WQF, lo schedulatore è un componente attivo del sistema che esegue una routine in un fissato istante. Il suo compito è quello di andare a controllare se ci sono processi schedulati; in caso affermativo l'algoritmo che esegue è il seguente:

1. controlla che nel sistema ci siano processi in *Scheduled* o processi *Active* con il campo `planned` settato a `true`;

2. per ogni processo P dell'insieme controlla che abbia il campo `schedulationDate` sia uguale alla data attuale;

3. se la condizione è verificata, P passa da *Scheduled* a *Active* (o rimane in *Active* se P era in *Active*);

4. se almeno un processo dell'insieme definito al punto 1. è passato (o rimasto) in *Active*, tutti gli altri processi che erano in *Active* prima dell'inizio della routine passano in *Available*.

L'algoritmo appena descritto è stato implementato dalla classe `ProcessSchedulerImpl`, in cui il metodo process() cabla tutta la logica dell'algoritmo.

Dal punto di vista implementativo, per far sì che la routine process() venga eseguita periodicamente, il metodo è stato annotato come `@Scheduled`, la quale prende come parametro l'espressione `cron=1 0 0 * * ?`[23], ovvero esegui la routine process() ogni giorno, un minuto dopo mezzanotte.

Il *bean* che processa l'annotazione `@Scheduled` è `ThreadPoolTaskScheduler` che implementa l'interfaccia `TaskScheduler`, il quale viene creato dal tag `xml <task:annotation-driven/>` nel file di configurazione `WQF-servlet.xml`.

7.4 Flusso di una richiesta `HTML`

Una richiesta (*request*) `HTML` è una serie di pacchetti di dati inviati dal browser dell'utente al server Tomcat attraverso la rete. In questa richiesta sono contenuti, tra le altre informazioni, il nome della pagina che si vuole visualizzare e gli eventuali campi inseriti nei form, in forma di stringa o binaria, a seconda del tipo di form dichiarato.

[23] una cron expression e formata da una stringa di 6 o 7 campi separata da spazi bianchi. Sono espressioni temporali altamente "espressivi" con cui si possono facilmente creare eventi quali: "ogni ultimo venerdì del mese alle 1.30 ". Vengono utilizzate dal tool Unix chiamato `cron`.Per maggiori dettagli si veda `http://wiki.opensymphony.com/display/QRTZ1/CronTriggers+Tutorial`

Nel file web.xml si definisce il meccanismo che caricherà le istanze dei componenti che gestiscono le richieste e le risposte HTTP. Il DispatcherServlet cercherà un file WQF-servlet.xml, inoltre si dichiara che ciascuna URL terminante con .htm andrà gestita da tale servlet.

```
<servlet>
    <servlet-name>WQF</servlet-name>
    <servlet-class>org.springframework.web.servlet.
       DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>WQF</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Listing 7.11. Frammento del file web.xml

7.4.1 Tre oggetti fondamentali

Con l'introduzione dei Controller annotati, l'associazione di una richiesta di una pagina tramite URL e il relativo Controller è stata semplificata al massimo: non c'è più bisogno di configurare tramite file xml esplicitamente i bean e le loro associazioni, ma viene tutte gestito da DefaultAnnotationHandlerMapping, AnnotationMethodHandlerAdapter e l'annotazione @RequestMapping. La prima classe gestisce le annotazioni a livello di classe, mentre la seconda classe restringe la mappatura a livello di metodi.

Gli appena citati bean, inoltre vengono creati automaticamente ponendo la seguente riga nel file di configurazione xml:

```
<context:component-scan base-package= "it.unipd.dei.wqf.
   service"/>
```

Listing 7.12. Frammento del file WQF-servlet.xml

Quando il Controller ha finito le sue elaborazioni, richiede di visualizzare una pagina; ed allora che interviene l'interfaccia ViewResolver, tramite una delle sue implementazioni.
Nel file WQF-servlet.xml (si veda il listato 7.13) è stato utilizzato InternalResourceViewResolver .

```
<bean id="jspViewResolver" class="org.springframework.web.
   servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.
       web.servlet.view.JstlView" />
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
</bean>
```

Listing 7.13. Frammento del file WQF-servlet.xml

Il bean `InternalResourceViewResolver` cerca una `JSP` il cui nome logico corrisponde all'`URL` di tale vista il cui suffisso deve essere `.jsp`, mentre il prefisso deve essere `/WEB-INF/jsp/`. Nel caso si volesse formare una catena di `ViewResolver`s, si noti che il `ViewResolver` appena definito deve essere collocato come ultimo ovvero avere un ordine minore di tutti gli altri, pena l'oscuramento del funzionamento di quelli dopo di lui.

7.5 Aggiungere una nuova JSP

In questa sezione viene presentata una guida che illustra il procedimento di inserimento di una nuova pagina `JSP` all'interno di un progetto Tomcat.

Si supponga di voler creare una nuova `JSP`, chiamata `newUser.jsp`, con la quale inserire un nuovo utente nel database. La pagina conterrà un form per l'inserimento dei dati richiesti. Le operazioni da effettuare sono le seguenti:

- creare la pagina `newUser.jsp`, all'interno della cartella `/WQF/WebContent/WEB-INF/jsp` o di una sua sottocartella, ad esempio `/WQF/WebContent/WEB-INF/jsp/subDir/`;

- creare il controller per la pagina. Per fare ciò è sufficiente realizzare una classe `Java` venga annotata con `@Controller`. In particolare, nel caso dell'esempio in questione, si deve creare il file `NewUserController.java` nella cartella `/WQF/src/`. Il listato 7.14 riporta un esempio dell'implementazione del controller.

```
1  @Controller
2  @RequestMapping("/subDir/newUser.htm")
3  public class NewUserController{
4  @RequestMapping(method=RequestMethod.GET)
5  public ModelMap get(ModelMap model)
6  {
7      User user=new User();
8      model.addAttribute("newUser", user);
9      return model;
10 }
11 @RequestMapping(method=RequestMethod.POST)
12 public String process(@ModelAttribute("newUser") User
       user, BindingResult result)
13 {
14 //eventuale validazione di user
15 //...
16 //aggiungi l'user
17 return "/sudDir/newUser";
18 }
19 }
```

Listing 7.14. Esempio di implementazione del file `NewUserFormController.java`

Alle righe 1 e 2 si sta dichiarando che la classe `NewUserController` è un `Controller` e se la richiesta è per l'indirizzo web `/subDir/newUser.htm`, essa deve essere presa in consegna da `NewUserController`. Nella fattispecie, come indicato dalla riga 4 alla 10, se la richiesta `HTTP` è una `GET`, allora il controller deve aggiungere attributo (un oggetto) `User` inizializzato al `Model` e chiamarlo `newUser`. Questo attributo sarà successivamente esposto alla `JSP` associata per essere visualizzata sotto forma di form `HTML`. Dalla riga 11 alla 18 vi è la logica di gestione qualora la richiesta `HTTP` sia un `POST`: il metodo prende come parametro l'attributo del modello "newUser", che avevamo precedentemente aggiunto al modello e che è stato eventualmente modificato nella pagina `JSP` e un oggetto `BindingResult` che viene utlizzato opzionalmente per esporre al `view` eventuali errori di validazione. Come si può notare il metodo process(), restituisce una stringa: essa è in realtà interpretata come il nome logico di una `view`. Se si decidesse di utilizzare un URL come link nelle `JSP` per richiedere la pagina, si può, ad esempio, usare `/subDir/newUser.htm`. Ciascun link nelle pagine avrà la seguente sintassi:

```
1 <a href="<c:url value="/subDir/newUser.htm"/>">nuovo
    utente</a>
```

Si noti che l'indirizzo scelto è all'interno del tag `<c:url>` fornito dalle librerie `JSTL`;

Capitolo 8

Applicazione di prova: autenticazione e gestione di permessi

Dopo aver configurato correttamente l'ambiente di lavoro, procurandosi tutti i tools richiesti, si è scelto di testare il sistema sviluppando una semplice applicazione di prova in cui utilizzare i meccanismi di autenticazione e autorizzazione forniti da Spring (Acegi) Security, poichè la sicurezza è una componente fondamentale per un'applicazione web ben fatta.

Il nome dato all'applicazione di test è `security_project` ed il suo funzionamento è illustrato in figura 8.1. In pratica, l'utente effettua l'accesso inserendo le proprie credenziali nel form della pagina di login. Nel caso in cui l'autenticazione vada a buon fine, l'utente accede alla pagina dei contenuti, in cui quest'ultimi variano in base ai permessi concessi all'utente. In caso contrario, invece, viene visualizzato un messaggio di errore e viene chiesto all'utilizzatore di reinserire le proprie credenziali di accesso.

8.1 Autenticazione utenti

All'applicazione `security_project` possono accedere solamente gli utenti elencati nella tabella 8.1. Quando si vogliono utilizzare le funzionalità per la

Login	Password	Ruolo	Permesso 1	Permesso 1
mario	rossi	admin	write	read
giuseppe	verdi	user	read	write
eva	adamo	progettista	write	null
dio	omni	universo	write	write

Tabella 8.1. Utenti dell'applicazione `security_project` e relativi permessi.

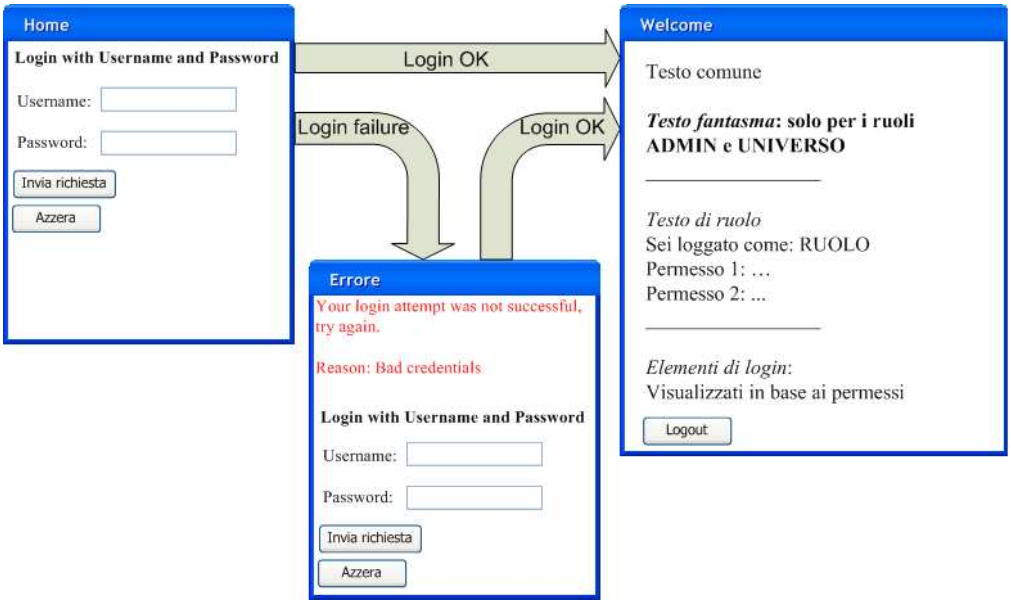

Figura 8.1. Rappresentazione grafica del funzionamento dell'applicazione `security_project`.

sicurezza messe a disposizione da Spring framework nella propria applicazione è necessario effettuare le opportune configurazioni. La prima cosa da fare è configurare il file `web.xml` inserendo il seguente frammento di codice:

```
1  <-- Estratto del file web.xml -->
2
3  <filter>
4      <filter-name>springSecurityFilterChain</filter-name>
5      <filter-class>org.springframework.web.filter.
           DelegatingFilterProxy</filter-class>
6  </filter>
7
8  <filter-mapping>
9      <filter-name>springSecurityFilterChain</filter-name>
10   <url-pattern>/*</url-pattern>
11 </filter-mapping>
```

Tutto ciò permette di utilizzare l'infrastruttura web di Spring Security.
`DelegatingFilterProxy` è una classe di Spring Framework che delega il lavoro a un'implementazione del filtro definita come un *bean* nel contesto dell'applicazione. In questo caso il *bean* si chiama `springSecurityFilterChain` ed è parte integrante dell'infrastruttura per gestire la sicurezza web. Una volta aggiunte queste righe al file `web.xml`, è possibile proseguire con la modifica del file di contesto dell'applicazione chiamato `applicationContext-security.xml`. Per abilitare il sistema di sicurezza web è necessario scrivere:

```
1 <-- Estratto del file applicationContext-security.xml -->
2
3 <http auto-config='true'>
4     <intercept-url pattern="/**" access="ROLE_USER,
          ROLE_PROGETTISTA,ROLE_ADMIN,ROLE_UNIVERSO" requires-
          channel="https" />
5     <form-login default-target-url="/welcome.htm" always-use
          -default-target="true" />
6 </http>
```

Il codice indica semplicemente che si vuole mettere in sicurezza tutti gli URL interni all'applicazione permettendo l'accesso solamente agli utenti autenticati con i ruoli elencati alla riga 4. Tutto ciò permette, quindi, di evitare accessi indesiderati effettuati conoscendo gli URL delle pagine protette. É possibile utilizzare più elementi `<intercept-url>` per definire differenti requisiti di accesso per diversi insiemi di URL, tenendo, però, conto del fatto che essi saranno valutati dal sistema nell'ordine in cui compaiono e che verrà, quindi, utilizzato il primo vincolo corrispondente all'URL richiesto. Se, a questo punto, si fa partire l'applicazione è possibile osservare immediatamente che il browser carica una pagina di login in cui si richiede di inserire le credenziali dell'utente, come raffigurato in figura 8.2. La cosa può stupire notevolmente, in quanto nessu-

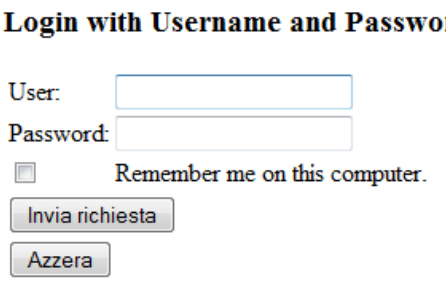

Figura 8.2. Pagina di login dell'applicazione `security_project`.

na pagina di login è stata precedentemente creata. La spiegazione è semplice e può anche sorprendere: a meno che non venga esplicitamente impostato l'URL della propria pagina di login, Spring Security si preoccupa di generarne una in automatico. Se, per esempio, si volesse effettuare il login attraverso la pagina `login.jsp` da noi creata, è sufficiente aggiungere il comando

```
1 login-page='/login.htm'
```

all'interno del tag `<form-login>` del listato precedente[1]. Il comando `default-target-url`, indica, invece, al sistema quale pagina caricare dopo aver effettuato

[1]Sarà, inoltre, necessario modificare il file `Security_project-servlet.xml` aggiungendo un tag `<bean>`. Per ulteriori approfondimenti rifarsi alla documentazione di Spring.

l'autenticazione dell'utente. La pagina di login creata in automatico da Spring Security mostra, inoltre, un messaggio di errore nel caso in cui l'autenticazione non vada a buon fine, vedi figura 8.3, come richiesto dai requisiti dell'applicazione. Se, invece, si volesse reindirizzare l'utente che inserisce credenziali non corrette ad una pagina di errore, ad esempio `loginError.jsp`, basta scrivere:

```
1 <form-login login-page="/login.htm" default-target-url="/
    welcome.htm" always-use-default-target="true"
    authentication-failure-url="/loginError.htm" />
```

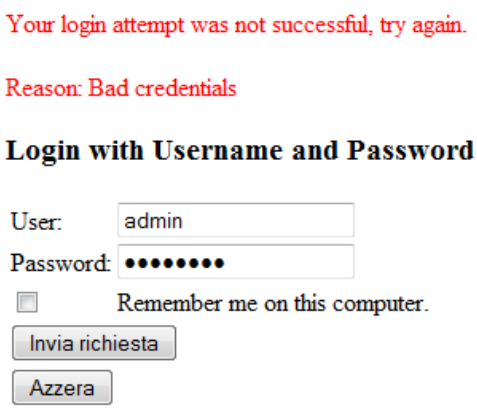

Figura 8.3. Errore di autenticazione nell'applicazione `security_project`.

All'interno del tag `<intercept-url>` della riga 4 è stato, inoltre, indicato di utilizzare il protocollo `HTTPS`. Un lettore attento avrà notato che, quando è stato indicato al sistema a quale pagina accedere in seguito ad un'autenticazione effettuata con successo (in questo caso `welcome.jsp`), si è usata l'estensione `.htm` al posto di `.jsp`. Il motivo di tutto ciò consiste nel fatto che si vuole fare in modo che il sistema intercetti un URL che termina con `.htm` e che richiami la pagina associata, con estensione `.jsp` ad esso associata. Le associazioni tra URL e pagine sono indicate nel terzo file di configurazione chiamato `Security_project-servlet.xml` (la convenzione definisce il nome come `nomeapplicazione-servlet.xml`) di cui si riporta un frammento di codice:

```
1 <-- Estratto del file Security_project-servlet.xml -->
2 <context:component-scan base-package= "myPack"/>
3 <bean id="viewResolver" class="org.springframework.web.
    servlet.view.InternalResourceViewResolver">
4     <property name="prefix" value="/WEB-INF/jsp/"/>
5     <property name="suffix" value=".jsp"/>
6 </bean>
```

L'handling del link `/welcome.htm` viene gestito dal controller `WelcomeController` che reindirizzerà, dopo aver eseguito il proprio lavoro, alla pagina `/WEB-INF/jsp/`

welcome.jsp. Il controller WelcomeController implementato per l'applicazione in esame ha un compito molto semplice, in quanto si limita a generare i permessi dei vari utenti e a passarli alla pagina welcome.jsp (come un oggetto perm di tipo Permessi) che li utilizzerà per filtrare i contenuti da visualizzare. Il codice di WelcomeController e di seguito riportato:

```
<-- Estratto del file WelcomeController.java -->

package myPack;

import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.
   SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.
   RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod
   ;
import org.springframework.web.servlet.ModelAndView;

@Controller

@RequestMapping("/welcome.htm")
public class WelcomeController{
    @RequestMapping(method=RequestMethod.GET)
    public ModelAndView handleRequest(@ModelAttribute("
       miopermesso") Permessi perm){
        Authentication auth = SecurityContextHolder.
           getContext().getAuthentication();
        if (auth.getName().equals("mario"))
        {
            perm = new Permessi(2,1);
        }
        if (auth.getName().equals("giuseppe"))
        {
            perm = new Permessi(1,2);
        }
        if (auth.getName().equals("eva"))
        {
            perm = new Permessi(2,0);
        }
        if (auth.getName().equals("dio"))
        {
            perm = new Permessi(2,2);
        }
        return new ModelAndView("welcome","miopermesso",perm
           );
    }
    @ModelAttribute("miopermesso")
```

```
38      public Permessi reference()
39      {
40          return new Permessi();
41      }
42  }
```

Il controller `WelcomeController` viene interpellato ogniqualvolta un utente si autentichi dalla pagina di login ed effettui, quindi, una richiesta della pagina `welcome.jsp`. Il controller si occupa della generazione dei permessi in base alle credenziali inserite, i quali vengono passati come parametro alla pagina `welcome` come un oggetto di tipo `Permessi`. Il flusso dei dati verrà, comunque, illustrato nei prossimi paragrafi.

Utilizzando Spring Security non è complesso recuperare le informazioni di autenticazione. Vengono, infatti, messe a disposizione un paio di implementazioni di base utili, che andremo a descrivere nelle prossime righe.

8.1.1 In-memory Authentication

Quando si inizia ad integrare Spring Security nella propria applicazione, spesso si desidera, per semplicità, evitare di dover prelevare le informazioni per l'autenticazione degli utenti da una fonte di persistenza (ad esempio una base di dati) sebbene, come vedremo in seguito, tutto ciò è tutt'altro che complicato. Per configurare l'applicazione in queste situazioni, non si deve fare altro che aggiungere il seguente frammento di codice nel file `applicationContext-security.xml` che viene caricato dal sistema nel momento in cui viene letto il file `web.xml` all'avvio dell'applicazione.

```
1  <-- Estratto del file applicationContext-security.xml -->
2
3  <!-- **********IN-MEMORY AUTHENTICATION********** -->
4
5  <authentication-provider>
6      <user-service>
7          <user name="mario" password="rossi" authorities="
               ROLE_ADMIN, ROLE_USER" />
8          <user   name="giuseppe" password="verdi" authorities
               ="ROLE_USER" />
9          <user name="eva" password="adamo"   authorities="
               ROLE_PROGETTISTA" />
10         <user name="dio" password="omni" authorities="
               ROLE_UNIVERSO" />
11         <user name="sadmin" password="sadmin" authorities="
               ROLE_USER,ROLE_UNIVERSO,ROLE_PROGETTISTA,
               ROLE_ADMIN" />
12     </user-service>
13 </authentication-provider>
```

8.1.2 JDBC Authentication

Spring Security offre anche la possibilità di leggere le informazioni per l'autenticazione degli utenti da una sorgente dati JDBC. I driver JDBC sono usati internamente a Spring e questo risparmia all'utente la fatica di dover realizzare un ORM (Object Relational Mapper) solamente per memorizzare i dettagli degli utenti. Le righe seguenti rappresentano la configurazione che serve a fare tutto ciò per l'applicazione `security_project` sempre nel file `applicationContext-security.xml`. Ovviamente il frammento di codice che riguarda l'autenticazione "in-memory" deve essere rimosso o commentato.

```
<-- Estratto del file applicationContext-security.xml -->

<!-- ************JDBC AUTHENTICATION************ -->

<authentication-provider user-service-ref="
   daoAuthenticationProvider">
</authentication-provider>

<beans:bean id="daoAuthenticationProvider" class="org.
   springframework.security.userdetails.jdbc.JdbcDaoImpl">
    <beans:property name="dataSource" ref="dataSource" />
</beans:bean>

<!-- Spring managed webcosts datasource -->
<beans:bean id="dataSource" class="org.springframework.jdbc.
   datasource.DriverManagerDataSource">
    <beans:property name="driverClassName" value="org.
       postgresql.Driver" />
    <beans:property name="url" value="jdbc:postgresql
       ://127.0.0.1:5432/sec_proj" />
    <beans:property name="username" value="postgres" />
    <beans:property name="password" value="admin" />
</beans:bean>
```

Non è necessario specificare alcun riferimento alla tabella del database nel caso in cui si utilizzi lo schema di default suggerito da Spring Security e di seguito riportato.

```
CREATE TABLE users (
username VARCHAR(50) NOT NULL PRIMARY KEY,
password VARCHAR(50) NOT NULL,
enabled BIT NOT NULL
);

CREATE TABLE authorities (
username VARCHAR(50) NOT NULL,
authority VARCHAR(50) NOT NULL
);

```

```
12 ALTER TABLE authorities ADD CONSTRAINT fk_authorities_users
       foreign key (username) REFERENCES users(username);
```

La tabella `users` contiene le informazioni riguardanti gli utenti, ovvero username, password e stato dell'account (*enabled* o *disabled*). La tabella `authorities` fornisce, invece, la lista dei ruoli di ciascun utente (un utente con più ruoli vedrà il suo username presente su più record).

È, comunque, possibile definire schemi personalizzati nella base di dati. È sufficiente, poi, indicare al sistema dove prelevare i dati degli utenti come fa il seguente frammento di codice di esempio:

```
1  <beans:bean id="daoAuthenticationProvider" class="org.
       springframework.security.userdetails.jdbc.JdbcDaoImpl">
2      <beans:property name="dataSource" ref="dataSource"/>
3      <beans:property name="usersByUsernameQuery">
4          <beans:value>
5              SELECT USERNAME, PASSWORD, ENABLED FROM USERS
                   WHERE USERNAME=?
6          </beans:value>
7      </beans:property>
8      <beans:property name="authoritiesByUsernameQuery">
9          <beans:value>
10             SELECT uat.USERNAME AS USERNAME, AUTHORITY FROM
                   AUTHORITIES uat, USERS ut WHERE uat.USERNAME
                    = ut.USERNAME AND uat.USERNAME=?
11         </beans:value>
12     </beans:property>
13 </beans:bean>
```

Il meccanismo usato per prelevare le informazioni è adattabile, quindi, a qualsiasi base di dati che disponga delle informazioni precedentemente elencate. La cosa fondamentale è modificare le query `usersByUsernameQuery` e `authoritiesBy UsernameQuery` adattandole al proprio database in modo che esse restituiscano rispettivamente `username-password-stato` e `username-ruolo`.

8.2 Autorizzazione sui ruoli

Ritornando al funzionamento dell'applicazione precedentemente introdotto, si è detto che vi sono degli elementi della pagina dei contenuti che vengono visualizzati in base al ruolo dell'utente che si è autenticato. La figura 8.4 mostra la differenza tra quello che vede un utente autenticato con ruolo ROLE_ADMIN, nella fattispecie l'utente "mario", ed uno con ruolo ROLE_PROGETTISTA, l'utente "eva". Si nota che l'utente con ruolo ROLE_ADMIN visualizza un contenuto che l'altro non vede, ovvero la scritta in grassetto. Per gestire le autorizzazioni sui contenuti, Spring Security mette a disposizione il tag `<authz:authorize>` che va utilizzato direttamente nelle pagine dell'applicazione. In particolare, nella

pagina `welcome.jsp` dell'applicazione `security_project`, ovvero la pagina dei contenuti, è presente il seguente frammento di codice:

```
1 <authz:authorize ifAnyGranted="ROLE_ADMIN,ROLE_UNIVERSO" >
2     <h3>
3         scritta riservata a chi ha permesso universo o admin
4     </h3>
5 </authz:authorize>
```

Nel tag è presente l'attributo `ifAnyGranted` seguito da una lista di ruoli. È sufficiente che l'utente appartenga ad uno dei ruoli elencati. Sono, inoltre, disponibili gli attributi `ifAllGranted` e `ifNotGranted` i cui significati sono facilmente intuibili. Il frammento di pagina racchiuso da questo tag viene, quindi, visualizzato solo se i requisiti di autorizzazione sono soddisfatti.

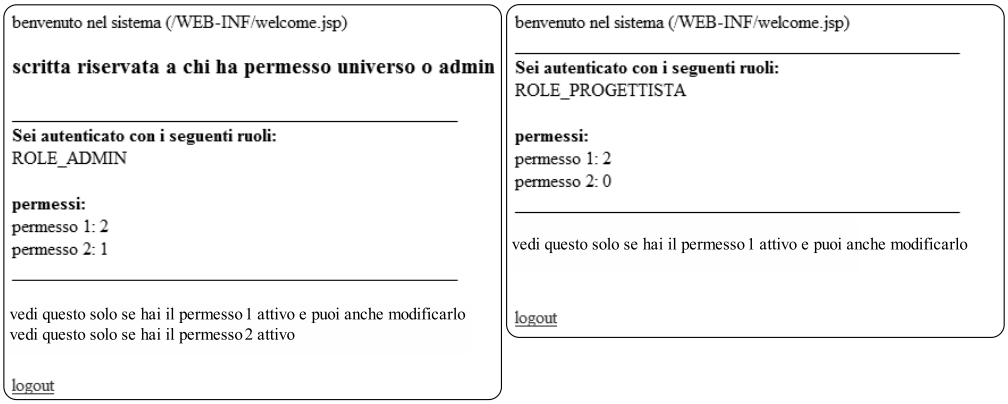

Figura 8.4. I contenuti visualizzati dall'utente "mario" (a sinistra) e quelli visti dall'utente "eva" (a destra).

8.3 Autorizzazione sui singoli permessi

Oltre alle autorizzazioni in base ai ruoli dell'utente autenticato, i contenuti dell'applicazione dovevano essere "filtrati" anche in base ai permessi specifici assegnati ai singoli utenti. Per semplicità i permessi sono stati indicati con dei numeri interi dove 0 corrisponde a "nessun permesso", 1 a "lettura" e 2 a "scrittura". Dopo aver effettuato l'autenticazione, Spring Security passa il controllo al controller `WelcomeController.java`, il quale restituisce un oggetto `perm` della classe `Permessi.java` indicante i permessi assegnati all'utente che ha effettuato il login e richiama la pagina `welcome.jsp`. È, inoltre, possibile restituire più oggetti alla pagina utilizzando l'interfaccia `java.util.map`[2]. Per differenziare

[2]Vedi documentazione `Java`

i contenuti in base ai permessi assegnati, sono stati creati due tag personalizzati[3], chiamati rispettivamente `permesso1.tag` e `permesso2.tag`, nella cartella `/WEB-INF/tags` dei quali riportiamo il codice.

```
<!-- permesso1.tag -->
<%@attribute name="perm_value" required="true "%>
<%
    if (Integer.parseInt(perm_value) > 0)
    {
        out.print("vedi questo solo se hai il permesso 1
           attivo");
    }
    if (Integer.parseInt(perm_value) == 2)
    {
        out.print("  e puoi anche modificarlo");
    }
%>
```

```
<!-- permesso2.tag -->
<%@attribute name="perm_value" required="true "%>
<%
    if (Integer.parseInt(perm_value) > 0)
    {
        out.print("vedi questo solo se hai il permesso 2
           attivo");
    }
    if (Integer.parseInt(perm_value) == 2)
    {
        out.print("  e puoi anche modificarlo");
    }
%>
```

Per utilizzare i tag nelle pagine dell'applicazione (per esempio in `welcome.jsp` è sufficiente scrivere:

```
<tags:permesso1 perm_value="${miopermesso.p1}"/>
<tags:permesso2 perm_value="${miopermesso.p2}"/>
```

dove `perm.p1` e `perm.p2` corrispondono ai valori dei permessi dell'utente autenticato restituiti da `WelcomeController`. Si osservi che la cartella `/WEB-INF/tags` è riconosciuta in automatico dal sistema che, quindi, sa dove andare a leggere il contenuto dei tag personalizzati inseriti.

Tornando all'esempio visualizzato in figura 8.4, è possibile osservare come il contenuto del tag `permesso2` non venga visualizzato dall'utente "eva" che non dispone di alcun permesso di tipo 2.

[3] In passato risultava abbastanza complicato scrivere tag `JSP` personalizzati. Con l'introduzione dei *tag files* nelle `JSP 2.0` le cose si sono notevolmente semplificate.

Capitolo 9

Definizione della base di dati

In questo capitolo si descrive la struttura della base di dati utilizzata dall'applicazione WQF il cui schema fisico è rappresentato in figura 9.1 (pag. 100). Si noti che le tabelle sono state poi suddivise per far risaltare le diverse funzionalità dell'applicazione.

La figura 9.2 (pag. 101) mostra la tabella preposta a salvare i *filtri di ricerca* creati dagli utenti per effettuare le ricerche sui dati storici conservati nella base di dati.

La figura 9.3 (pag. 105) mostra le tabelle per il disegno dei processi.

La figura 9.4 (pag. 105) mostra le tabelle per la memorizzazione di dati nell'esecuzione dei processi.

La figura 9.5 (pag. 106) mostra le tabelle preposte alla gestione degli accessi, dei ruoli aziendali e delle login.

Nelle sezioni seguenti verranno approfonditi i significati e le funzioni delle tabelle di ognuna delle parti appena elencate. Per non appesantire la descrizione della base di dati con dettagli troppo minuziosi, si è volutamente scelto di non descrivere le tabelle i cui attributi sono stati chiamati con nomi per quanto possibile esplicativi. Si porrà un accento solo su alcuni attributi che si ritengono fondamentali per la comprensione di alcune funzioni chiave.

9.1 Tabelle per il disegno dei processi

Le tabelle per il disegno dei processi (pag. 105) costituiscono la parte di dati qui definita *semi-statica*, in quanto quest'ultimi vengono modificati solamente nella fase di disegno dei processi da parte del process designer e non vengono alterati dall'esecuzione delle varie istanze. Nel caso però il disegno di processi contenga almeno una attività di tipo *expandible*, questa porzione di database viene modificata ricorsivamente durante l'esecuzione: in linea teorica, infatti, un processo che contenga almeno una attività *expandible* può avere associata un processo dinamico che ne contiene un'altra a sua volta e così via, modellando processi arbitrarialmente complessi.

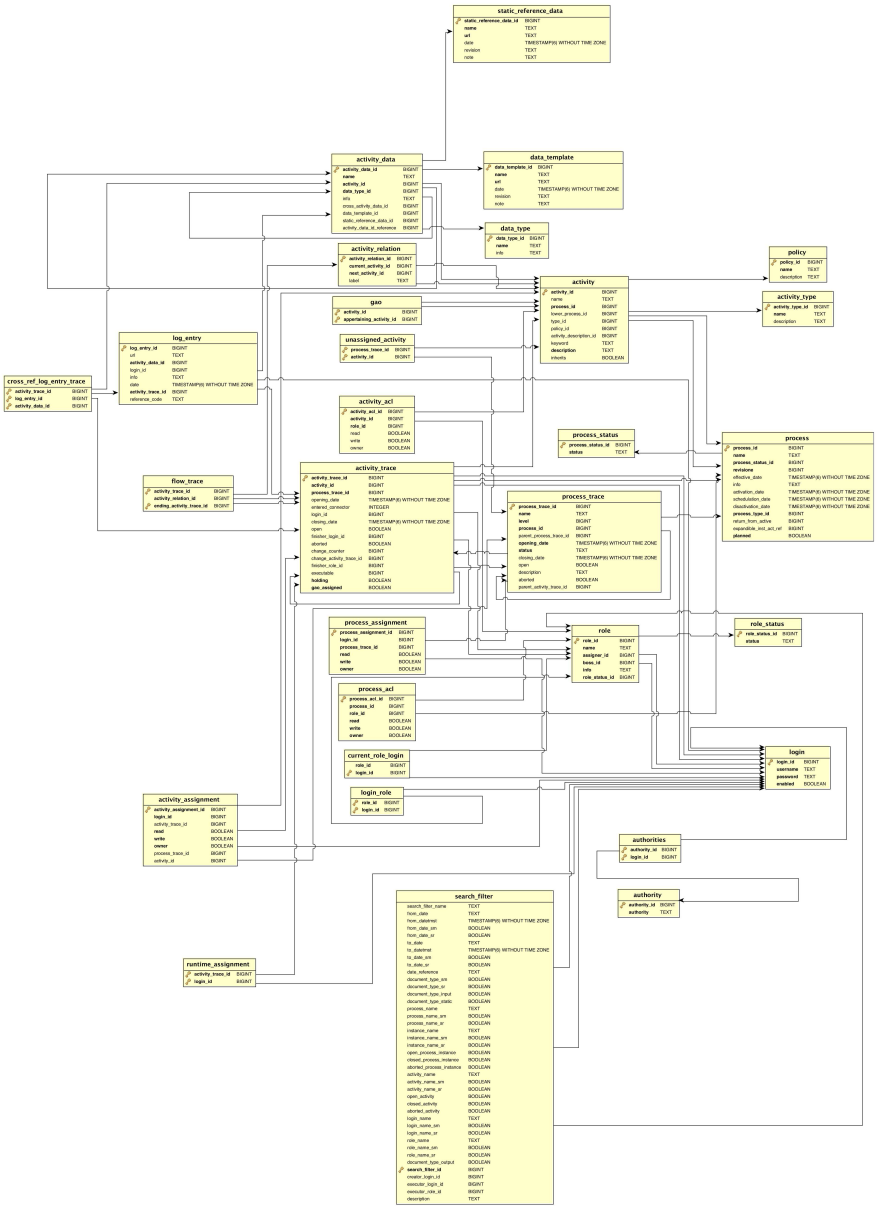

Figura 9.1. Schema fisico della base di dati utilizzata dall'applicazione WQF.

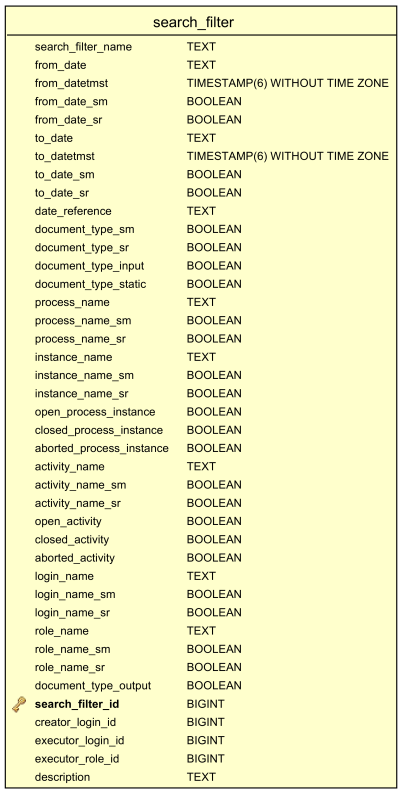

Figura 9.2. Tabella per la memorizzazione dei filtri di ricerca sui dati dell'applicazione WQF.

Le tabelle `process_status`, `process`, `activity` e `activity_type` contengono gli attributi che descrivono, rispettivamente, processi e relative attività (si veda l'analisi funzionale). La tabella `activity_relation` raccoglie le informazioni sulle connessioni tra le varie attività nel diagramma di flusso del processo. `policy` raccoglie la lista delle politiche di assegnazione delle risorse lavorative rese disponibili dall'applicazione, mentre la tabella `gao` indica quali attività "ordinarie" vengono assegnate dall'esecuzione di una particolare attività di tipo GAO[1]. La tabella `role` contiene l'elenco dei ruoli aziendali che compongono l'organigramma, mentre `login_role` la mappatura tra questi e gli utenti del sistema. Le tabelle `process_acl` ed `activity_acl` raccolgono le informazioni sui

[1]Si ricorda che GAO è l'acronimo di Group Activity Owner. Le attività di questo tipo comportano assegnazioni di risorse esplicite per un gruppo di attività. Vedi analisi funzionale al capitolo 8 della tesi citata in bibliografia [2].

permessi assegnati a ciascun ruolo per processi e attività.

Qualche parola va spesa su alcuni campi della tabella `process`:

- `process_type_id`: indica se il processo è *statico* o *dinamico*.
- `expandible_inst_act_ref`: è il campo che in cui viene memorizzato il riferimento all'istanza dell'attività *expandible*.
- `return_from_active`: nel caso di un processo di tipo *dinamico*, questo campo conta il numero di volte che il processo (ri)torna in *Development*.
- `planned`: è un booleano che ha la funzione di segnare se un processo è stato *pianificato per essere schedulato*. In particolare se i processi in *Active* devono essere schedulati vengono indicati con `planned=true`. Si noti che questo campo ha anche il compito di memorizzare lo stato intermedio tra lo stato attuale di un processo e il suo stato futuro *Scheduled*: infatti un processo non passa nello stato *Scheduled* finché il designer non ha premuto sul bottone `schedula`. Prima di tale azione, infatti, un processo, anche se risulta nella lista di processi *planned*, permane nel suo stato attuale.

9.2 Tabelle per la gestione dei documenti

La sezione della base di dati che gestisce i documenti usati o prodotti dalle attività, è costituita dalle tabelle `activity_data` in cui sono si mappano i documenti in ingresso/uscita, `data_type` che definisce i tipi di documento utilizzabili (ad esempio documento di input statico o upload con template), `data_template` in cui si salvano i riferimenti agli URL dei template ed, infine, `static_reference_data` che memorizza gli URL dei documenti statici disponibili.

9.3 Tabelle per la raccolta dei dati runtime

I dati contenuti in questa sezione del database sono quelli generati in *runtime* dall'esecuzione del motore di workflow che sta alla base dell'applicazione. La tabella `process_trace` memorizza le istanze di processo create; i campi `level`, `parent_process_trace_id` e `parent_activity_trace_id` servono a gestire i sottoprocessi permettendo al sistema di ritornare all'esecuzione del processo chiamante una volta che il sottoprocesso è stato concluso. La tabella `activity_trace` raccoglie i dati relative alle singole istanze di attività aperte. Un'istanza di attività viene aperta quando la precedente viene chiusa, ma essa non può essere eseguita fino a che tutte le attività che la precedono non sono concluse. Proprio per effettuare questa verifica è stato predisposto il campo `entered_connector` che memorizza il numero di connettori in ingresso percorsi. Il sistema si occuperà di confrontare il valore del campo con il numero totale

di connettori in entrata ed, eventualmente, di rendere disponibile all'esecuzione l'attività ponendo a `true` il valore del campo `open`. Il campo `holding`, viene utilizzato nel contesto di una attività afferente a un processo *dinamico* e serve per memorizzare lo stato del controllo di flusso, qualora l'owner o il designer decidano di riportare lo stato del processo da *Active* a *Development*. Le tabelle `process_assignment` ed `activity_assignment` contengono i record che documentano le persone (login) a cui i processi e le attività che li compongono sono stati assegnati. Quando una nuova istanza di processo viene avviata, viene generato un record nella tabella `unassigned_activity` per ognuna delle attività che lo compongono per indicare che esse non sono state, appunto, assegnate a nessuna persona. Una volta assegnata un'attività, il record corrispondente viene eliminato. La tabella può, quindi, risultare anche completamente vuota. Quando viene aperta un'istanza di attività con policy di assegnamento "Owner escalation", viene aggiunta una tupla alla tabella `runtime_assignment` che tiene, quindi, traccia delle escalation non ancora gestite; una volta effettuato l'assegnamento, la tupla in questione viene eliminata. La tabella `flow_trace` tiene traccia delle connessioni tra attività percorse durante il flusso di lavoro all'interno di un processo e risulta utile per capire quale cammino è stato eseguito per raggiungere una particolare attività nel caso essa sia raggiungibile da più vie. La tabella `log_entry`, infine, serve a memorizzare le informazioni (URL, creatore, data, ecc.) riguardanti i documenti prodotti dall'esecuzione delle attività (upload).

9.4 Tabelle per la gestione dei ruoli aziendali

In linea teorica, ad ogni login viene assegnato almeno un ruolo aziendale. Quindi una login può avere associato più ruoli aziendali e, tra questi, uno di default. Le tabelle che memorizzano le caratteristiche appena descritte sono le seguenti: `login_role`, `current_role`, `role`. La tabella `role_status` memorizza, come dice il nome, lo stato di un ruolo, che può essere *attivo* oppure *obsoleto*.

9.5 Tabelle per l'autenticazione degli utenti

Questa parte del database contiene le tabelle `login`, `authority` e `authorities` che identificano rispettivamente gli utenti dell'applicazione, i ruoli di sistema (differenti da i ruoli dell'organigramma aziendale) e la mappatura tra di essi. Le tre tabelle formano una struttura simile a quella suggerita da Spring Security illustrata nel capitolo 8.

9.6 Tabella per la memorizzazione dei filtri di ricerca

La tabella `search_filter` memorizza i filtri di ricerca creati dai singoli utenti per uso personale o dall'amministratore di sistema per renderli disponibili a gruppi di utenti (per esempio ad un particolare ruolo). Ogni singolo filtro è identificato, oltre che dall'`id` presente in tutte le tabelle, da un nome. I campi rimanenti sono quelli che determinano i criteri di ricerca che può essere suddivisa in tre categorie principali:

- ricerca per processo/attività;
- ricerca di documenti;
- ricerca per ruolo/persona.

Per permettere all'applicazione di appoggiarsi ad una sola tabella della base di dati, utilizzando, quindi, un unico filtro per le tre tipologie di analisi dei dati, è stato necessario creare un attributo `date_reference` che viene usato per eliminare l'ambiguità che producono i campi in cui vengono inserite le date per limitare la ricerca da un punto di vista temporale. Esse, infatti, si possono riferire alla data di apertura/chiusura di un'istanza di processo/attività o alla data di creazione di un particolare documento.

I campi booleani che terminano con i suffissi `_sm` e `_sr` (abbreviazioni di "show mask" e "show result") indicano, invece, quali campi del filtro devono essere mostrati nella maschera di ricerca e quali nella visualizzazione dei risultati prodotti.

Si noti che la tabella `search_filter` contiene le chiavi esterne `creator_login`, `executor_login` e `executor_role` che identificano creatore ed utilizzatori del filtro e che sono collegati rispettivamente alle tabelle `login` e `role` dello schema di fig. 9.1.

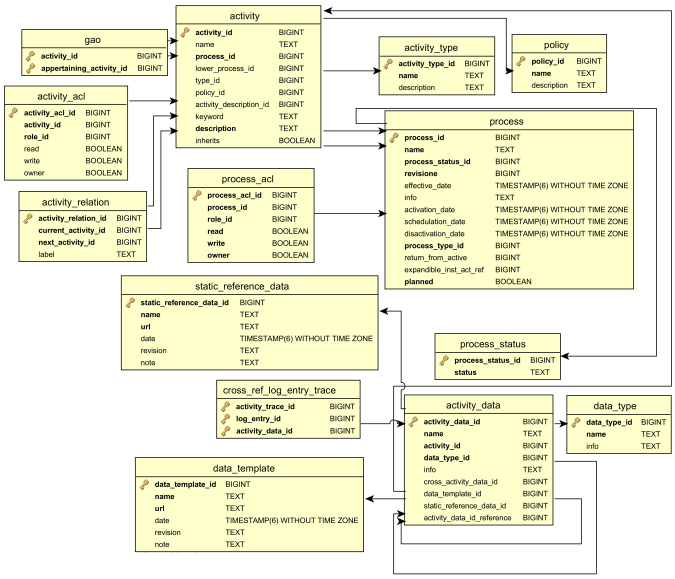

Figura 9.3. Tabelle per il disegno di processi.

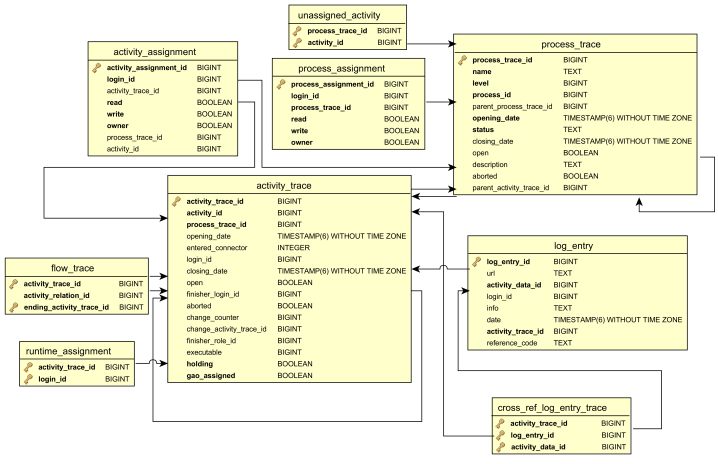

Figura 9.4. Tabelle per l'esecuzione di processi.

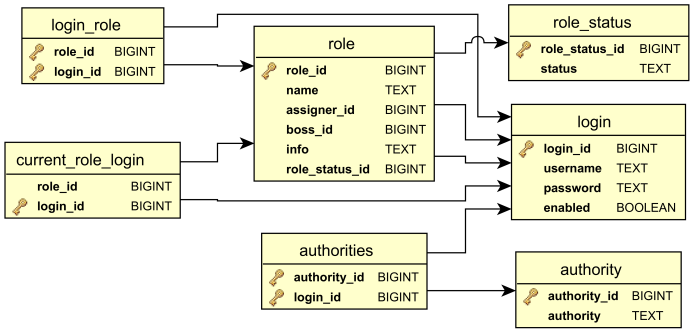

Figura 9.5. Tabelle per la definizione di ruoli aziendali.

Capitolo 10

Manuale dell'installatore

10.1 Installazione della Java Virtual Machine

Tomcat, come tutte le applicazioni *Java-based*, richiede una *Java Virtual Machine* (JVM) per funzionare. La Sun Microsystems distribuisce gratuitamente una JVM per Windows, Linux e Solaris. Vendors di terze parti e gruppi open-source distribuiscono JVM per altre piattaforme, non sempre gratuitamente. Prima di installare Tomcat 6 è necessario assicurarsi che la Java 5 JVM sia installata nel proprio sistema. Per verificare quale versione della JVM si ha, è sufficiente utilizzare il comando `java -version` come mostrato.

```
1 C:\> java -version
2 java version "1.6.0_16"
3 Java(TM) SE Runtime Environment, Standard Edition (build
     1.6.0_16-b01)
4 Java HotSpot(TM) Client VM (build 14.2-b01, mixed mode,
     sharing)
```

Una stringa di tipo `1.5.x`, o superiore, indica che si possiede una versione della JVM che permette di installare Tomcat correttamente. Il comando appena descritto presuppone che il percorso `JAVA_HOME/bin` sia contenuto nella variabile di sistema `PATH`. Le sezioni seguenti illustrano il procedimento per l'installazione della JVM in ambiente Windows e Linux. Nel caso si disponga già di una versione della JVM corretta è possibile passare direttamente alla sezione 10.2.

Le versioni di Tomcat precedenti alla `5.5` richiedevano l'installazione del *Java Development Kit* (JDK) e non solo del *Java Runtime Environment* (JRE). Il compilatore `Java` in esso contenuto era utilizzato dalle versioni precedenti di Tomcat per compilare le `JSP` in fase di *runtime* e, quindi, l'installazione del solo JRE non era sufficiente. Le versioni `5.5` e `6` di Tomcat includono un compilatore `Java` (*Eclipse JDT Java compiler*). Questo è usato per compilare le pagine `JSP` e, quindi, è possibile eseguire Tomcat `5.5` e `6`, solo con il JRE.

10.1.1 Installazione della JVM in ambiente Windows

Per prima cosa si deve scaricare l'ultima versione di JDK o JRE dal sito Web `http://www.oracle.com/technetwork/java/index.html`. A questo punto è sufficiente far partire l'eseguibile scaricato e, dopo aver seguito la semplice procedura di installazione, si avrà JVM/JRE installato nel proprio sistema. La cartella in cui sono stati installati i files è nota come *Java Home*. Essa si ramifica in numerose sottocartelle, ma quella di maggior interesse è `bin`, nella quale sono contnuti i vari eseguibili. Il passo successivo consiste nell'aggiungere la cartella *Java Home* in una variabile di sistema chiamata `JAVA_HOME`, in modo tale che essa possa essere trovata quando viene invocata. La sottocartella `bin` deve, invece, essere aggiunta alla variabile `PATH`. Per ottenere informazioni sul procedimento di creazione di una variabile di sistema si consulti il sito `http://support.microsoft.com`.

10.1.2 Installazione della JVM in ambiente Linux

Le versioni di JDK/JRE per Linux sono, a loro volta, disponibili, all'indirizzo `http://www.oracle.com/technetwork/java/index.html`. Una buona alternativa alla JVM offerta da Sun per Linux è l'implementazione di IBM[1]. Le istruzioni a seguire si riferiscono, comunque, alla distribuzione della Sun. La piattaforma ufficialmente supportata è Red Hat Linux, ma i pacchetti JDK e JRE della Sun funzionano senza apparenti problemi su tutte le altre distribuzioni.

Se si è scaricato un archivio tar/gzip, la procedura di installazione è la seguente. Per prima cosa, una volta scaricato il file si deve estrarne il contenuto. Per installare JDK per tutti gli utenti è necessario eseguire i comandi che seguono dopo essersi autenticati come `root` o utilizzando il comando `sudo`. A questo punto si deve spostare il file precedentemente estratto nella cartella in cui lo si vuole installare. Se si sta installando JDK (o JRE) per uno specifico utente, lo si deve installare nella cartella *Home* dell'utente stesso. In alternativa, la cartella di default è `/usr/java/jdk-[version number]`, dove `[version number]` corrisponde alla versione di JDK che si sta installando. Aggiungere, ora, i permessi di esecuzione al file scrivendo

```
1 # chmod o+x jdk-6u21-linux-i586.bin
```

ed eseguire il file con il comando

```
1 # ./jdk-6u21-linux-i586.bin
```

Prima di iniziare l'installazione, viene visualizzato il contratto di licenza che si deve accettare. Ad installazione completata, invece, si deve aggiungere al sistema la variabile `JAVA_HOME` con la posizione del pacchetto nel file system. Questo valore deve essere aggiunto al file `~/.bashrc` per la propria utenza, o a `/etc/profile` per tutti gli utenti.

[1] http://www.ibm.com/developerworks/java/jdk/

```
JAVA_HOME=/usr/java/jdk-6u21-linux-i586/
export JAVA_HOME
PATH=$JAVA_HOME/bin:$PATH
export PATH
```

Per testare l'installazione è sufficiente digitare `javac` nella shell. Questo dovrebbe produrre il seguente output (ritagliato per ragioni di brevità):

```
Usage: javac <options> <source files>
where possible options include:
-g                         Generate all debugging info
-g:none                    Generate no debugging info
-g:{lines,vars,source}     Generate only some debugging info
```

Per installare JVM usando un *installer* RPM, invece, si deve per prima cosa, scaricare il file nel formato `j2sdk-[version number]-linux-i586-rpm.bin`. Eseguendo questo file viene presentato il contratto di licenza per Apache e, al termine dell'esecuzione, viene creato un file RPM con lo stesso nome, ma senza il suffisso `.bin`. Per eseguire il file RPM ci si deve loggare come `root` o utilizzare il comando `sudo`, aggiungere i permessi di esecuzione e, quindi, eseguire il file:

```
# chmod o+x jdk-6u21-linux-i586-rpm.bin
# ./jdk-6u21-linux-i586-rpm.bin
```

Per permettere che l'installazione prosegua è necessario accettare i termini di licenza visualizzati. Una volta accettato il contratto di licenza, lo script di installazione andrà a creare il file RPM nella directory corrente. Per installare l'RPM scrivere:

```
# rpm -iv jdk-6u21-linux-i586-rpm.bin
```

Con questa procedura si è installato JDK in `/usr/java/jdk6-u21`.

10.2 Installazione di Tomcat

Nel proseguo del testo si indicherà la directory di installazione di Tomcat con `TOMCAT_HOME` o `CATALINA_HOME`. Le varie distribuzioni dei files di installazione di Tomcat per le diverse piattaforme possono essere reperite all'indirizzo web `http://tomcat.apache.org`. Il sito web di Tomcat elenca quattro differenti distribuzioni per ciascuna versione di Tomcat: *core*, *deployer*, *embedded* e *admin webapp*. Un utente che si avvicina per la prima volta a Tomcat può trovarsi, quindi, in serio imbarazzo nello scegliere la versione da scaricare. Le distribuzioni sono:

- Core: è la versione base do Tomcat ed è disponibile come file `tar.gz`, `ZIP` o come eseguibile per Windows (`.exe`).
- Deployer: si tratta del deployer per l'applicazione web Tomcat *standalone*.

- Admin Webapp: attualmente questa distribuzione non è disponibile per Tomcat `6.x`. Nelle versioni precedenti si trattava di un'applicazione web che veniva utilizzata per amministrare Tomcat e che si trovava in un package differente per motivi di sicurezza.

Per lo sviluppo di un'applicazione Web da caricare su un server sul quale è installato Tomcat, come nel caso dell'applicazione WQF, si deve scaricare la distribuzione *core*.

10.2.1 Installazione di Tomcat in ambiente Windows

La pagina di download presenta differenti link per Tomcat 6.x. Quella desiderata dagli utenti Windows presenta estensione `.exe`. Dopo aver salvato il file nella cartella desiderata, è sufficiente eseguire il file per far partire l'installazione. Come spesso accade, l'installer chiede di accettare la licenza Apache, dopodichè viene presentata una schermata in cui si chiede di scegliere i componenti da installare (vedi figura 10.1). É conveniente optare per un'installazione completa, con la

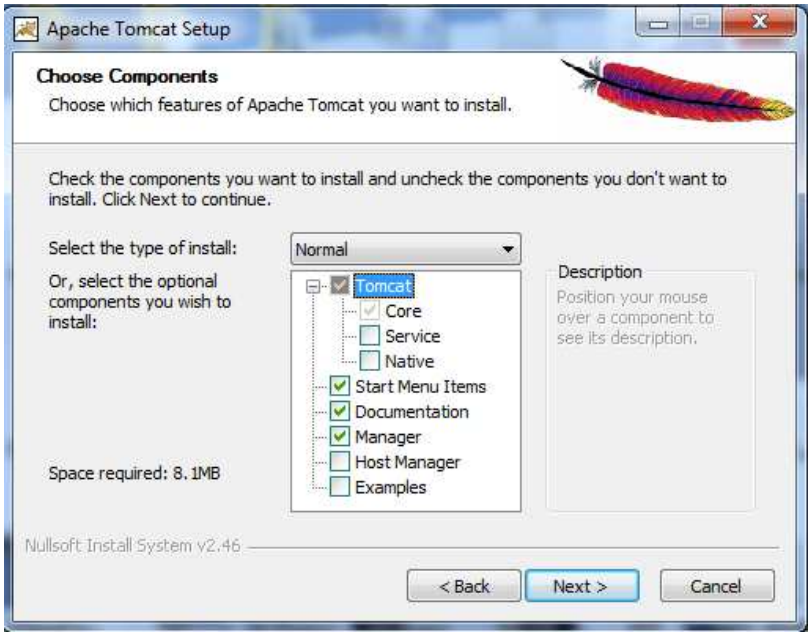

Figura 10.1. Scelta dei componenti da installare durante l'installazione di Tomcat.

quali verranno installati tutti i componenti di Tomcat. Alcuni di essi meritano una breve descrizione. Uno dei componenti che non verrebbe installato di default, ma che risulta essere molto utile, è il componente *Service* (selezionabile espandendo il sottomenu "Tomcat"). Questo componente, una volta installato, permette di avviare, terminare o far ripartire Tomcat come un qualsiasi servizio di Windows.

Uno dei vantaggi offerti da questa opzione è, quindi, la possibilità di far avviare Tomcat assieme al sistema operativo. L'nstallazione prosegue chiedendo di indicare la directory di installazione, la password di amministrazione che si desidera e la directory in cui si trova JDK (spesso viene rilevata in automatico). Al termine dell'installazione, anche se non è obbligatorio, è consigliabile aggiungere ad una nuova variabile di sistema chiamata `CATALINA_HOME` il path di installazione. Prima di dichiarare concluso il processo di installazione, è necessario verificare che tutto sia stato installato correttamente. Per fare tutto ciò, dopo aver avviato Tomcat (manualmente eseguendo `CATALINA_HOME/bin/tomcat6.exe` o, automaticamente, impostandolo come servizio di Windows), è sufficiente aprire un browser e accedere all'indirizzo `http://localhost:8080` e verificare che appaia una pagina come quella di figura 10.2.

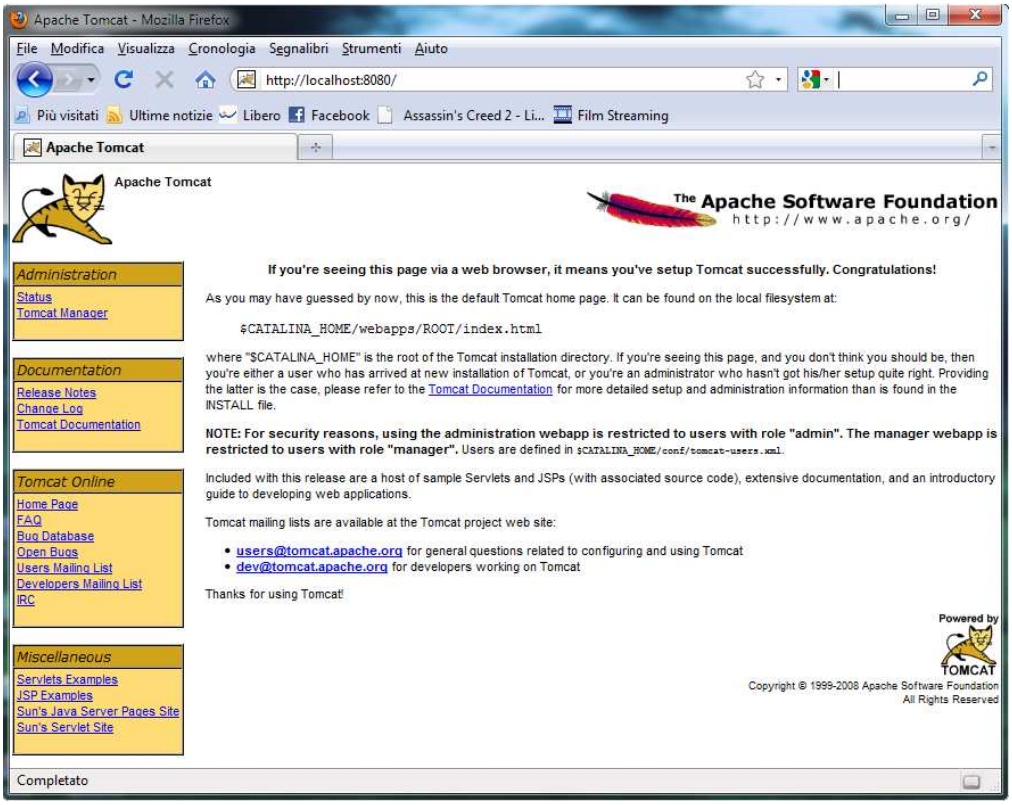

Figura 10.2. L'home page di Tomcat.

10.2.2 Installazione di Tomcat in ambiente Linux

L'installazione di Tomcat in ambiente Linux è molto semplice. Dopo aver scaricato il file `tar/gzip` dal sito di Tomcat, è sufficiente decomprimerlo nella cartella

in cui lo si desidera installare (ad esempio `/usr/java/tomcat-6`. A questo punto si deve esportare la variabile `$CATALINA_HOME` (in `bash`) utilizzando i seguenti comandi:

```
# CATALINA_HOME=/usr/java/tomcat-6
# export CATALINA_HOME
```

A questo punto è possibile avviare Tomcat scrivendo:

```
# CATALINA_HOME/bin/startup.sh
```

Un altro possibile modo per installare Tomcat è usare gli strumenti per la gestione dei pacchetti specifici delle diverse distribuzioni di Linux. Questo risulta comodo per molti utenti, ma va sottolineato che questi tools spesso posizionano i files di configurazione in locazioni non standard. É possibile verificare il buon esito dell'installazione con lo stesso metodo usato in ambiente Windows.

10.3 Configurazione dell'IDE Eclipse

Eclipse, essendo scritto in linguaggio Java, non richiede una vera e propria installazione. É sufficiente, infatti, scaricare l'archivio `Eclipse IDE for Java EE Developers` all'indirizzo `http://www.eclipse.org/download` (accertarsi di scaricare la versione *Galileo*), scompattarlo ed eseguire il file `eclipse.exe`. É possibile copiare la cartella di Eclipse da una macchina all'altra mantenendo la sua operativitá.

É disponibile un plugin per Eclipse che interfaccia l'IDE con Tomcat denominato *Sysdeo Eclipse Tomcat Launcher plugin* e reperibile all'indirizzo `http://www.eclipsetotale.com/tomcatPlugin.html`. Per installarlo è sufficiente decomprimere l'archivio nella cartella `plugins` di Eclipse. Tale plugin permette di:

- inizializzare e fermare Tomcat 4.x, 5.x e 6.x;
- registrare i processi Tomcat nel debugger di Eclipse;
- creare progetti WAR (modificando il file server.xml mediante l'uso di un wizard);
- aggiungere progetti Java al classpath di Tomcat;
- settare i parametri JVM di Tomcat, i classpath e i bootclasspath;
- esportare progetti Tomcat come file WAR.

Una volta installato correttamente il plugin, è dunque possibile realizzare progetti con la dicitura "Is a Tomcat Project", aggiungendo in automatico tutte le librerie necessarie al funzionamento Tomcat. Le librerie di Spring vanno, invece, aggiunte manualmente.

10.4 Attivazione del canale HTTPS

Per ottenere informazioni su come abilitare l'utilizzo di un canale di sicurezza tramite `HTTPS` per le applicazioni web che usano Tomcat, è possibile consultare la guida all'indirizzo `http://localhost:8080/docs/ssl-howto.html`. A titolo di esempio si riporta la configurazione dell'ambiente di lavoro in cui è stato sviluppato il progetto WQF. Da terminale eseguire:

```
%JAVA_HOME%\bin\keytool -genkey -alias tomcat -keyalg RSA
Immettere la password del keystore: password
Specificare nome e cognome:
 [Unknown]: Qifeng Zhu
Specificare il nome dell'unità aziendale:
 [Unknown]: DEI
Specificare il nome dell'azienda:
 [Unknown]: Università di Padova
Specificare la località:
 [Unknown]: Padova
Specificare la provincia:
 [Unknown]: Padova
Specificare il codice a due lettere del paese in cui si
   trova l'unità:
 [Unknown]: IT
Il dato CN=Qifeng Zhu, OU=DEI, O=Università di Padova,
L=Padova, ST=Padova, C=IT è corretto?
 [no]: si

Immettere la password della chiave per <tomcat>
    (Invio se corrisponde alla password del keystore):
```

Il comando sopra citato fa riferimento ad un ambiente Windows. In Linux basta sostituire `%JAVA_HOME%:` con `$JAVA_HOME/...` A questo punto si deve modificare il file `TOMCAT_HOME/conf/server.xml` come segue[2]:

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
<Connector port="8443" minSpareThreads="5" maxSpareThreads="
   75" enableLookups="true" disableUploadTimeout="true"
   acceptCount="100" maxThreads="200" scheme="https" secure
   ="true" SSLEnabled="true" keystoreFile="C:/users/qifeng
   /.keystore" keystorePass="password" clientAuth="false"
   sslProtocol="TLS" />
```

Per verificare che il certificato appena creato si integri correttamente con Tomcat, è sufficiente accertarsi che, digitando nel browser l'indirizzo `https://localhost:8443/`, venga visualizzata l'homepage di Tomcat (la stessa di figura 10.2).

[2]In ambiente windows il file `.keystore` viene creato nella cartella `C:/users/nomeutente/`, mentre in Linux il file va cercato in `HOME`

10.5 Installazione di PostgreSQL

L'installazione di PostgreSQL è molto semplice e non presenta alcuna differenza tra la versione per ambiente Windows e quella per Linux. L'ultima versione del software, al momento della scrittura della presente tesi, è la 8.4.2. L'archivio scaricato dal sito internet del produttore presenta un unico file che è sufficiente avviare per far partire la procedura di installazione. Dopo un messaggio di benvenuto, viene chiesto di specificare prima il percorso della directory in cui installare i files dell'applicazione, poi la cartella in cui verranno salvati i database creati. Proseguendo con l'installazione viene, quindi, richiesto l'inserimento della password che si vuole adottare per il *superuser* `postgres` del DBMS. Al passo successivo si richiede, invece, di specificare la porta di comunicazione del DBMS; il consiglio è quello di lasciare il valore predefinito `5432`. Prima di iniziare con la copia dei files su disco viene visualizzata una schermata nella quale scegliere delle opzioni avanzate per utenti esperti; anche qui si consiglia di lasciare i valori predefiniti. Al termine della copia dei files viene avviata un'applicazione che permette di installare alcuni pacchetti affini a PosgreSQL. Per poter utilizzare l'applicazione WQF è necessario installare i drivers JDBC che permettono a Java di Tomcat di comunicare con la base di dati.

10.5.1 Caricamento della base di dati

Il database per l'applicazione WQF è disponibile nel Cd-Rom allegato alla tesi come file `WQF.sql`. Esso è stato creato attraverso la procedura per effettuare i backup fornita dall'applicazione `pgAdmin`, che rappresenta l'interfaccia tra il DBMS e l'utente, e contiene solamente i dati minimi utili al funzionamento dell'applicazione.

La prima cosa da fare è creare un nuovo database vuoto di PostgreSQL chiamandolo, ad esempio, `wqf` ed indicandone l'utente proprietario (per quanto concerne questa guida si è assegnato il database all'utente `postgres`). A questo punto si devono caricare i dati nel database a partire dal file di backup. Per fare ciò, dalla versione 8.4, è sufficiente, una volta selezionato il database da popolare, cliccare sul pulsante `PSQL Console` nella barra degli strumenti e digitare, ad esempio, il comando:

```
1 \i C:/WQF.sql
```

Si deve, in pratica, solamente indicare il file da cui caricare i comandi SQL per "costruire" la base di dati. Nelle versioni più vecchie di PostgreSQL non è presente un collegamento diretto all'applicazione `psql` direttamente da `pgAdmin`. Per poter eseguire la procedura di ripristino da backup si deve, allora, avviare manualmente l'applicazione da riga di comando:

```
1 C:\Program Files (x86)\PostgreSQL\8.4\bin>psql -U postgres
    wqf
```

Il comando dice di avviare l'applicazione `psql` connettendosi al database `wqf` come utente `postgres`. Dopo aver inserito la password, se richiesta, è sufficiente digitare il comando descritto precedentemente.

10.6 Prima esecuzione dell'applicazione WQF

Per effettuare la prima esecuzione dell'applicazione WQF è necessario utilizzare Tomcat Manager, accessibile attraverso il link presente nel pannello di amministrazione dell'home page di Tomcat (figura 10.2). Dopo aver inserito le credenziali dell'amministratore di Tomcat, scorrere la pagina fino a trovare la voce "Select WAR file to upload" (vedi figura 10.3) dove si deve selezionare il file `.war` relativo all'applicazione; Eclipse permette, infatti, di esportare l'intero codice dell'applicazione in un unico file (con estensione `.war` appunto) caricabile direttamente da Tomcat per effettuare il *deploy* dell'applicazione. Una volta

Figura 10.3. Deploy di un file `.war` con Tomcat.

effettuata l'operazione di *deploy*, viene creata, in automatico, una cartella `WQF` in `TOMCAT_HOME/webapps` in cui è possibile trovare l'intera struttura dell'applicazione. L'ultima cosa da fare è modificare il file `dataAccessContext.xml` contenuto in `TOMCAT_HOME/webapps/WQF/WEB-INF` inserendo le credenziali per l'accesso al database come riportato nell'esempio seguente:

```
1 <bean id="dataSource"
2         class="org.springframework.jdbc.datasource.
             DriverManagerDataSource">
3         <property name="driverClassName" value="org.
             postgresql.Driver" />
4         <property name="url" value="jdbc:postgresql
             ://127.0.0.1:5432/wqf" />
5         <property name="username" value="postgres" />
6         <property name="password" value="password" />
7     </bean>
```

L'applicazione sarà, quindi, disponibile all'indirizzo `http://localhost:8080/WQF`.

Capitolo 11

Manuale del Process Designer

Le funzioni descritte in questo capitolo sono accessibili dal menu `Sviluppo processo` e . Si ricorda che questa sezione è visibile senza restrizioni agli utenti che presentano, tra i ruoli di sistema, quello di process designer (`ROLE_DESIGNER`). Quando un user (`ROLE_USER`), nel contesto di modifica di un processo dinamico, dovesse accedere a questa sezione, potrà visualizzare le sole pagine utili inerenti a processi dinamici.

11.1 Sviluppo processi

11.1.1 Crea processo

Figura 11.1. Creazione di un nuovo processo.

Per creare un nuovo processo, il disegnatore deve accedere alla pagina `Crea proc.` visualizzata in figura 11.1. La procedura prevede l'inserimento di un

nome del processo e di una descrizione (facoltativa). Il processo di inserimento termina con la pressione del pulsante `Aggiungi`. Se l'operazione di inserimento va a buon fine viene visualizzato un messaggio di conferma come, ad esempio, "`Processo ''Product Planning'' creato con successo.`". Il sistema si occupa di verificare l'unicità del nome inserito. Nel caso si tenti di creare un nuovo processo utilizzando il nome di un processo già esistente, viene, invece, mostrato un messaggio del tipo "`Esiste già un processo ''Product Planning''.`". Si ricorda che la un processo così creato ha "revisione" pari a 1.

11.1.2 Modifica processo

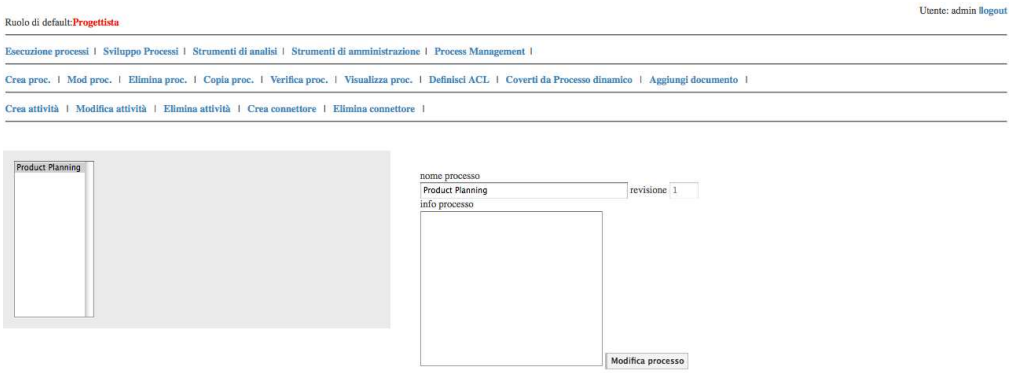

Figura 11.2. Modifica di un processo.

Accedendo alla pagina `Mod proc.`, vedi figura 11.2, è possibile modificare le informazioni (nome e descrizione) di un processo precedentemente creato. L'utente deve selezionare il processo dalla lista sulla sinistra, ricompilare il form con le nuove informazioni (il sistema carica di default quelle precedentemente inserite) e premere il pulsante `Modifica processo`. Accedendo a questa pagina viene, inoltre, espanso il menu delle funzioni disponibili visualizzando tutte le operazioni possibili su un processo. Si ricorda, infatti, che, al momento della sua creazione, un nuovo processo è, di fatto, una scatola vuota con solamente l'attività *start* e quella *finish*. Nei paragrafi seguenti verranno illustrate le procedure da seguire per utilizzare ognuna delle funzioni di editing dei processi.

Crea attività

Mediante la pagina `Crea attività`, di cui è riportato un esempio in figura 11.3, è, appunto, possibile creare una nuova attività per un certo processo precedentemente creato. Per prima cosa, l'utente deve selezionare il processo a cui l'attività

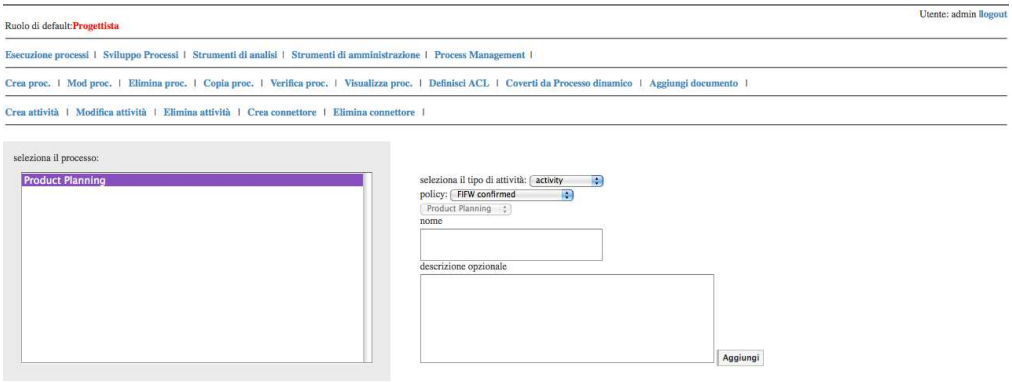

Figura 11.3. Creazione di una nuova attività.

fa riferimento. In seguito si devono indicare rispettivamente il tipo, il nome e la descrizione (opzionale) dell'attività. Nel caso si selezioni il tipo *Subprocess* si deve, inoltre, indicare anche il sottoprocesso da attivare selezionandolo tra quelli già presenti nella base di dati. Per terminare l'operazione premere il pulsante `Aggiungi`. Anche per le attività valgono le regole di unicità del nome all'interno però di un processo; il sistema comunica l'avvenuto inserimento o la presenza di errori di compilazione del form con modalità simili a quelle precedentemente descritte.

Modifica attività

Come avviene con i processi, anche le attività vengono create come delle box vuote. Per modificarle, si deve accedere alla pagina `Modifica attività`. L'esempio in figura 11.4 mostra la modifica di un'attività di tipo GAO; è stato volutamente scelto questo tipo di attività perchè la pagina richiede l'inserimento di un numero maggiore di informazioni.

Dopo aver selezionato il processo di riferimento e l'attività da modificare, nella parte sinistra della pagina viene visualizzato un form nel quale si possono modificare i valori di nome e descrizione precedentemente inseriti. Per le attività di tipo GAO come quella dell'esempio, viene, inoltre, visualizzata la lista delle attività per il processo scelto dalla quale si devono selezionare quelle che saranno da essa gestite (si ricorda che la scelta multipla è effettuabile attraverso il comando `ctrl+click`). Premendo il tasto `Modifica` le informazioni vengono aggiornate.

La parte destra della pagina, invece, viene utilizzata per inserire le informazioni riguardanti i documenti di input/output. In base alla tipologia di do-

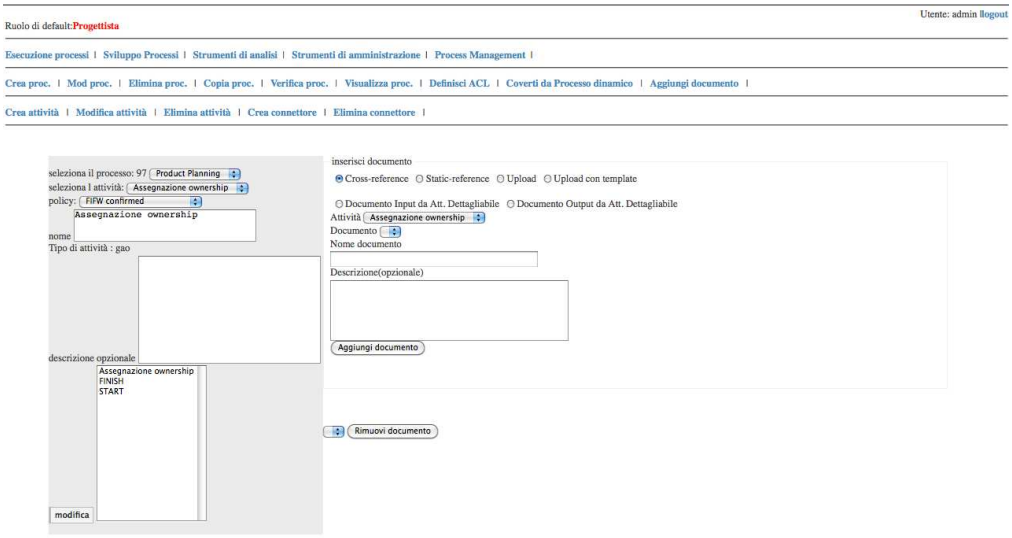

Figura 11.4. Modifica di un'attività.

cumento da collegare all'attività, viene richiesto l'inserimento di informazioni differenti:

- *Cross-reference.* L'utente deve indicare l'attività che produce il documento, selezionarlo tra le opzioni possibili ed indicare un nome ed una descrizione (facoltativa) che verranno, poi, visualizzati nella pagina per l'esecuzione dell'attività in modifica.

- *Static-reference.* L'utente deve indicare, come al punto precedente, nome e descrizione del documento e selezionare il riferimento tra i documenti disponibili nel menu a tendina. Il documento da collegare deve essere stato precedentemente inserito usando la funzione per l'aggiunta nel sistema di documenti/template in seguito illustrata.

- *Upload.* Viene richiesto di inserire il nome ed, opzionalmente, una descrizione del documento.

- *Upload con template.* Si richiede di inserire i dati relativi al documento (nome e descrizione facoltativa), nonchè il riferimento al template (precaricato) come nel caso delle *Static-reference.*

- *Documento Input da Att. Dettagliabile.* Questo tipo di documento è selezionabile nel caso di modifica di un'attività che si trova all'interno di un processo dinamico.

- *Documento Output da Att. Dettagliabile*. Questo output è selezionabile se l'attività in modifica fa parte di un processo dinamico.

Per concludere la procedura di inserimento di un documento si deve premere il pulsante `Aggiungi documento`. Nella parte inferiore della pagina viene riportata la lista dei documenti già inseriti per l'attività in modifica.

Si ricorda che non sono modificabili attività già eseguite appartenenti a processi dinamici.

Elimina attività

Per eliminare un'attività dal disegno di un processo, si deve accedere alla pagina `Elimina attività`. Selezionando il processo dall'elenco a sinistra, viene automaticamente aggiornata la lista delle attività precedentemente create per esso. È, quindi, possibile eliminare un'attività selezionandola e agendo sul pulsante `Delete activity`. Si ricorda che le attività "START" e "FINISH" non possono essere eliminate in quanto create in automatico dal sistema, perciò non compariranno nella lista.

Inoltre si ricorda che non è possibile eliminare attività già eseguite di un processo dinamico.

Crea connettore

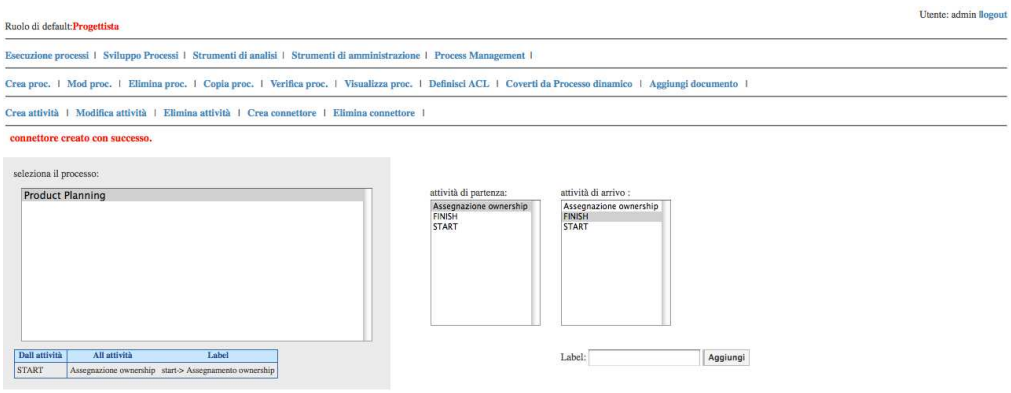

Figura 11.5. Pagina per la creazione dei connettori tra le attività di un processo.

Le connessioni tra le diverse attività di un processo possono essere definite accedendo alla pagina `Crea connettore`. Una volta selezionato il processo da modificare (elenco a sinistra), si devono indicare l'attività da cui parte il connettore e quella di arrivo dalle due liste popolate automaticamente sulla destra

(vedi figura 11.5). L'operazione si conclude con la pressione del pulsante `Crea connettore`. A creazione avvenuta viene visualizzato un messaggio di conferma del tipo "`Connettore creato con successo`". Mano a mano che i connettori vengono inseriti, viene aggiornata la tabella dei connettori già presenti visibile a fondo pagina. Si ricorda che i connettori uscenti da attività di scelta devono necessariamente essere etichettati. La label può essere inserita nell'apposita casella di testo. nel caso si tenti di creare un connettore uscente da una scelta privo di etichetta, il sistema lo segnalerà con il messaggio di errore "`Non hai inserito la label per un attività di tipo ''scelta''`". Si ricorda che non è possibile creare un connettore la cui attività di arrivo sia già stata eseguita.

Elimina connettore

Figura 11.6. Eliminazione di uno o più connettori.

La procedura per eliminare uno o più connettori è disponibile alla pagina `Elimina connettore` rappresentata in figura 11.6. Qui si deve, per prima cosa selezionare il processo da modificare; in questo modo il sistema genera, in automatico, la lista di tutti i connettori precedentemente creati. A questo punto è sufficiente selezionare il/i connettore/i da eliminare (selezionando le checkbox) e premere il pulsante `Delete`. Si ricorda che non è possibile eliminare connettori già percorsi, nel contesto di un processo dinamico.

11.1.3 Elimina processo

Mediante la pagina `Elimina processo` è, appunto, possibile eliminare processi che sono nello stato *Development*. La procedura consiste nel selezionare il processo da eliminare dalla lista visualizzata e premere il pulsante `Delete process`. Il buon esito dell'operazione viene segnalato con il messaggio "`Processo eliminato`".

Si ricorda che l'eliminazione di un processo comporta la conseguente cancellazione di attività e connettori ad esso affini.

11.1.4 Copia processo

Figura 11.7. Copia di un processo.

Se si desidera creare una revisione di processo esistente, è sufficiente accedere alla pagina `Copia proc.` (vedi figura 11.7). In essa vengono visualizzati tutti i processi presenti nella base di dati con status diverso da *Development* con revisione massima. L'utente deve solamente selezionare le checkbox relative ai processi che intende recuperare e premere il pulsante `Copia processi`. Il sistema copia attività e connettori dei singoli processi. Non vengono, invece, recuperati eventuali sottoprocessi, documenti *cross-reference* e le informazioni di ACL.

11.1.5 Verifica processo

Prima di concludere il disegno di un processo, è consigliabile effettuare una verifica di compatibilità con il formalismo per la descrizione dei processi. Questa funzione è disponibile alla pagina `Verifica proc.` (vedi figura 11.8) e coincide perfettamente con la funzione chiamata dal sistema al momento del commit di un processo. In seguito alla selezione da parte dell'utente del processo da verificare, vengono visualizzate tutte le eventuali incongruenze con il formalismo. I controlli effettuati sono:

- verifica che l'attività Finish sia raggiungibile;
- verifica che ogni attività Scelta abbia almeno due connettori in uscita;
- verifica che ogni attività Or abbia almeno due connettori in ingresso;

Figura 11.8. Verifica della presenza di errori di formalismo in un processo.

- verifica che ogni attività sia raggiungibile dallo Start;
- verifica che sia stato definito un owner per il processo;
- verifica che sia stato definito un owner per ogni attività.

Nel caso il processo sia dinamico, si verifica che tutti i documenti in input dell'attività dettagliabile associata vengano "utilizzati" almeno una volta e tutti i documenti da produrre (in output) vengano prodotti almeno una volta.

11.1.6 Visualizza processo

Nel disegnare i processi è, spesso, utile visualizzare il lavoro svolto. La pagina `Visualizza proc.`, vedi figura 11.9, permette di verificare quali informazioni sono state inserite e quali no. Selezionando un processo tra quelli disegnati, viene generata una tabella che mostra, per ogni attività, chi la precede, chi la segue, il tipo, l'eventuale descrizione ed i documenti ad essa riferiti.

11.1.7 Converti un processo dinamico

Sulla pagina vengono visualizzati tutti i processi che contengono almeno una attività dettagliabile per cui è stato associato almeno un processo dinamico. Cliccando sul processo desiderato si visualizzano le attività dettagliabili contenute, cliccando su una di esse si visualizzano i processi dinamici associati. Indicando il nome del processo che si vuole convertire e cliccando su "crea", un nuovo processo statico viene creato. A titolo di esempio si veda la figura 11.10.

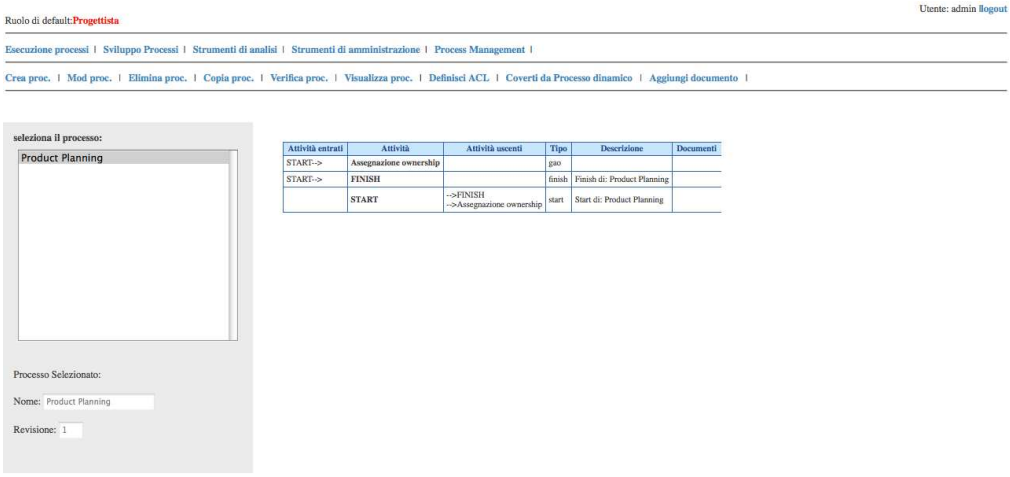

Figura 11.9. Visualizzazione di un processo.

Figura 11.10. Conversione di un processo dinamico.

11.2 Definizione Access Control List

Attraverso la pagina `Definisci ACL`, vedi figura 11.11, il Process Designer definisce i permessi su processi e attività per ognuno dei ruoli aziendali.

La prima tabella che viene costruita rappresenta l'Access Control List per i processi; in essa vi è una colonna per ogni ruolo. Qui l'utente seleziona il processo per cui saranno definiti i permessi (anche per le sue attività, come verrà spiegato a breve). La compilazione della tabella dei permessi sul processo richiede obbligatoriamente soltanto l'indicazione di almeno un ruolo come owner

del processo.

La seconda tabella rappresenta, invece, l'ACL per le singole attività del processo selezionato. Vi è una riga per ogni attività, mentre le colonne, come nella tabella di processo, corrispondono ai ruoli aziendali. Se l'ACL di un'attività rispecchia in toto quella del processo, è possibile selezionare la checkbox nella colonna `Eredita` e premere il pulsante `Aggiorna activity ACL`: tutte le righe con la checkbox selezionata rispecchieranno la struttura dei permessi del processo e l'ACL verrà salvata. Il pulsante `Reload` può essere utilizzato, invece, per annullare modifiche non ancora salvate e ricaricare l'ACL. Si ricorda che anche le attività devono avere almeno un ruolo con permesso owner. Il pulsante `Aggiorna activity ACL` deve essere premuto al termine della compilazione per rendere definitivi i cambiamenti. Essendo la compilazione della tabella un lavoro lungo e delicato, si consiglia di effettuare salvataggi periodici.

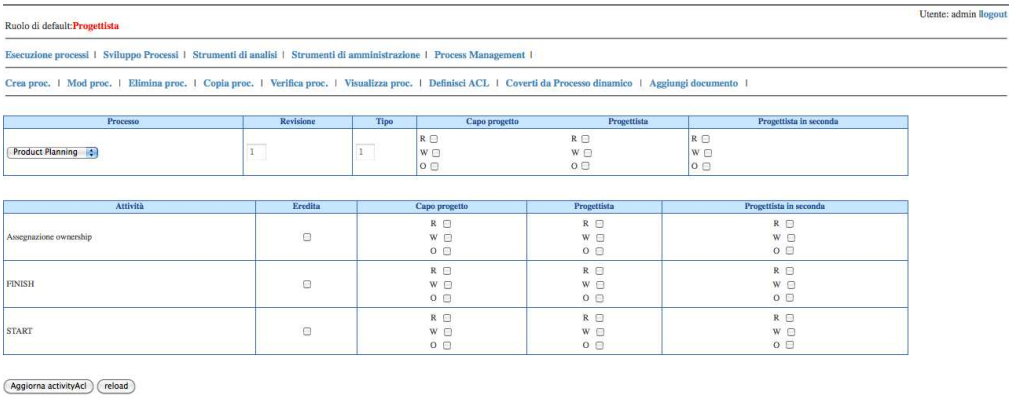

Figura 11.11. Access Control List: gestione dei permessi su processi e attività.

11.3 Aggiunta documenti e template

Il sistema mette a disposizione una procedura per l'inserimento dei documenti statici e dei template da utilizzare nell'esecuzione delle diverse attività. Questa è accessibile alla pagina `Aggiungi documento` rappresentata in figura 11.12. Dopo aver indicato se si intende inserire un documento o un template (vengono salvati in differenti tabelle della base di dati), l'utente deve indicarne il nome, l'URL da dove recuperarlo, la data di creazione ed, opzionalmente, il numero di revisione e una nota descrittiva. La procedura si conclude con la pressione del pulsante `Aggiungi`. Un inserimento avvenuto con successo viene segnalato con il messaggio "`Documento inserito`".

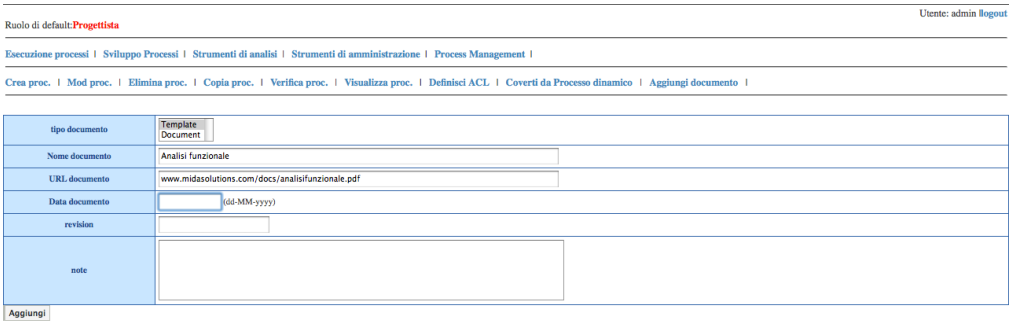

Figura 11.12. Aggiunta di un documento o template.

11.4 Gestione degli stati dei processi

Questo insieme di funzionalità ha la finalità di far passare di stato *in modo coerente* i processi statici definiti a sistema; sono accessibili alla pagina `Process management`.

11.4.1 Rendi Obsoleto o Available

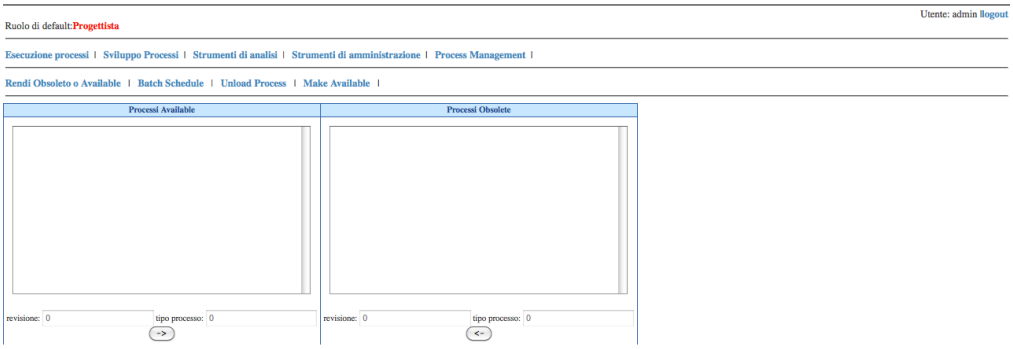

Figura 11.13. Passaggio da *Available* a *Obsolete* e viceversa.

Questa pagina (si veda la figura 11.13), tramite una maschera di interazione, ha il compito di far passare di stato un processo *Available* a *Obsolete* e viceversa. Si noti che il passaggio è istantaneo.

11.4.2 Unload Process

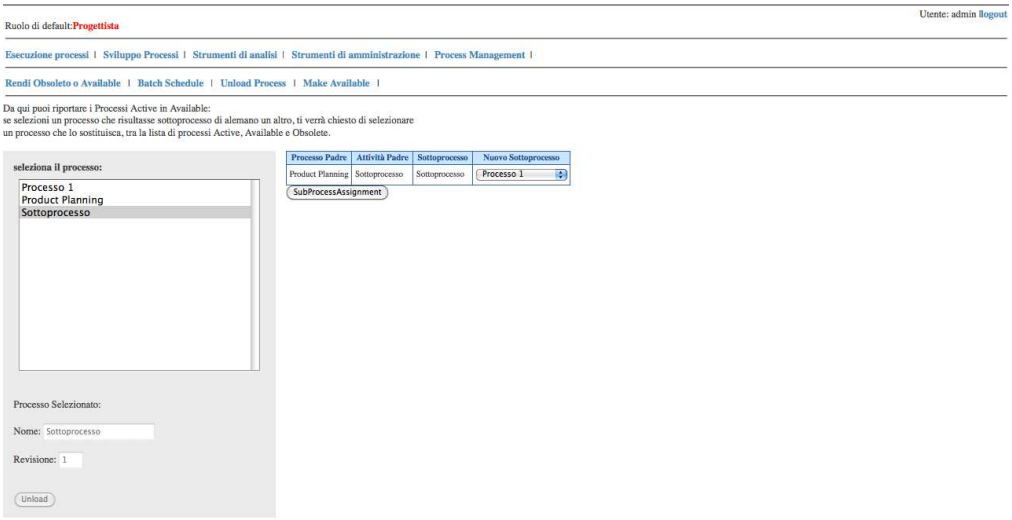

Figura 11.14. Unload di un processo *Active*.

Da questa pagina si possono "scaricare" i processi in *Active* riportandoli in *Available*. Come si nota dalla figura 11.14, quando si cerca di fare l'unloading di un processo SP utilizzato come sottoprocesso da un altro processo P, al momento dell'unloading si chiede al designer quale sottoprocesso vada a sostituirsi a SP.

11.4.3 Make Available

Un processo statico (in *Development*) che si ritenga pronto per passare in *Available*, può utilizzare questa pagina per il passaggio di stato. Si ricorda che prima che il processo selezionato passi effettivamente di stato, ne viene verificata la consistenza.

11.4.4 Batch Schedule

Da questa pagina (vedasi figura 11.15) si possono schedulare i processi. Si ricorda che il sistema ritiene un insieme di processi schedulati solo se si ha inserito una data valida e si è premuto il bottone "schedula". Se si volesse far permanere i

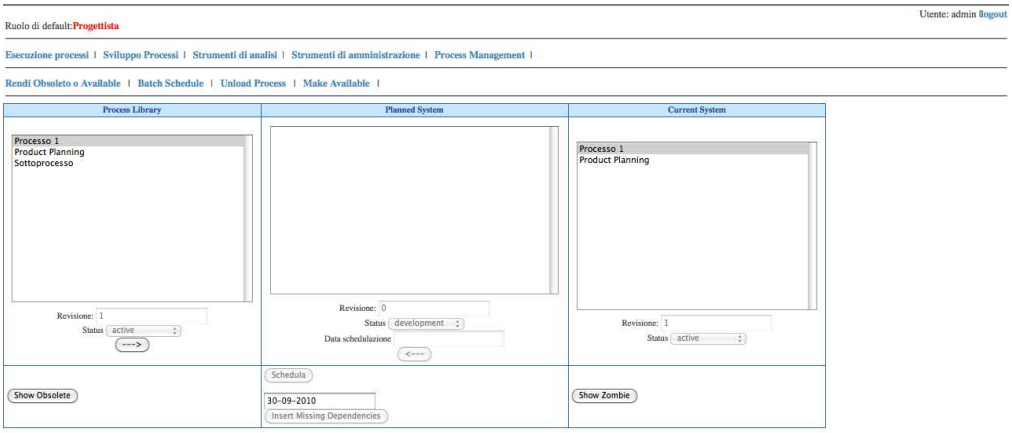

Figura 11.15. Schedulazione di processi.

processi *Active* nel loro stato, li si deve schedulare, altrimenti il sistema li scarica riportandoli in *Available*. Il bottone "show obsolete" ha significato ovvio, mentre "show zombie" visualizza nella pagina anche i processi che hanno istanze non finite, ma che si trovano in uno stato diverso da *Active*. Infine "insert missing dependencies" permette di caricare in "Planned System" eventuali sottoprocessi dei processi già pianificati.

Capitolo 12

Manuale utente

12.1 Gestione dei ruoli

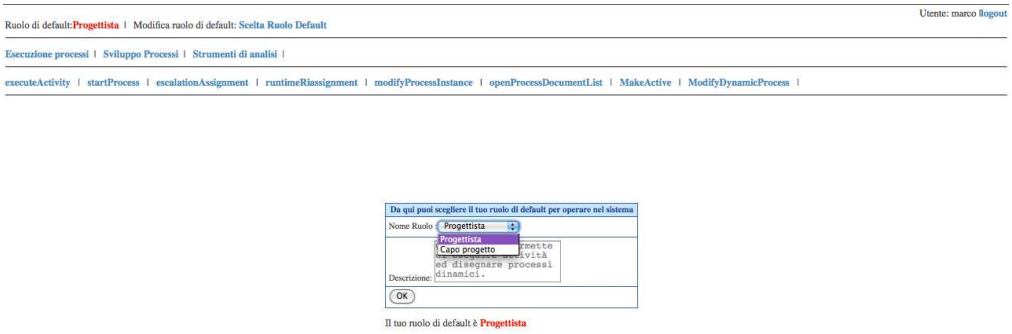

Figura 12.1. Scelta del ruolo aziendale.

Nel caso a un utente fosse stato assegnato più di un ruolo, egli può scegliere il ruolo di default per interagire col sistema. La figura 12.1 mostra la maschera con il quale è possibile scegliere con quale ruolo aziendale operare nel sistema. Si noti che un utente normale che abbia un solo ruolo aziendale associato, non potrà mai accedere a questa funzione.

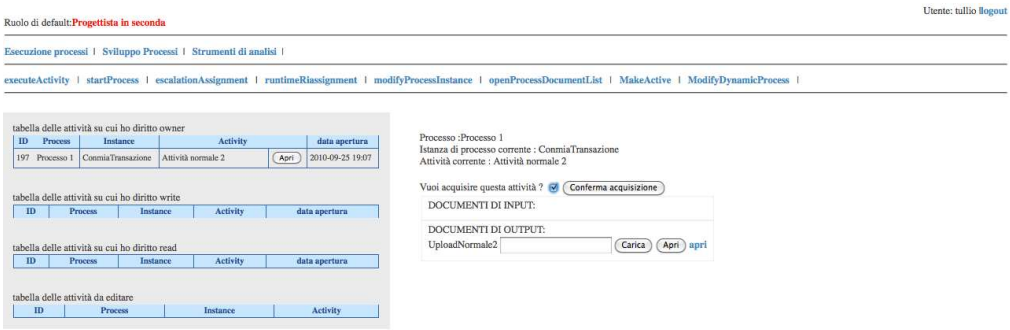

Figura 12.2. Accettazione di un'attività.

12.2 Esecuzione processi

12.2.1 Esecuzione di un'attività

Cliccando su `Execute activity` ogni utente vede, sulla sinistra, l'elenco delle attività da svolgere, suddivise in base al permesso ottenuto per ognuna di esse (owner, write e read). Cliccando sul pulsante `Apri` vicino al nome di un'attività, sulla parte destra della pagina compariranno i dettagli relativi all'attività selezionata. Per poter eseguire le istruzioni è, però, necessario confermare l'accettazione dell'attività, vedi figura 12.3, in caso questa sia stata definita con politica FIFW. Una volta acquisita, l'utente può svolgere il lavoro sull'attività e, in base ai permessi accordati, può leggere e/o caricare documenti. Una volta concluso il lavoro sull'attività (devono essere stati caricati tutti i documenti richiesti), l'utente con permesso owner potrà chiuderla semplicemente agendo sul pulsante `termina attività`, il quale viene visualizzato solamente non appena è realmente possibile effettuare la chiusura (vedi figura 12.3).

Un discorso a parte va fatto per la creazione di processi attraverso l'esecuzione di attività di tipo dettagliabile. La figura 12.4 appositamente riporta un'esempio in cui un'istanza di attività dettagliabile può essere "eseguita" creando un processo *ex novo* oppure copiando un processo associato a una istanza già eseguita della stessa attività. Prima di effettuare quest'ultima operazione, è offerta la possibilità all'utente di visualizzare il processo da copiare.

Si noti che se la creazione del processo ha avuto successo, esso risulterà disponibile per lo sviluppo al gruppo di funzioni accessibili cliccando su `Sviluppo Processi` e l'attività scomparirà dalla lista di attività da "editare".

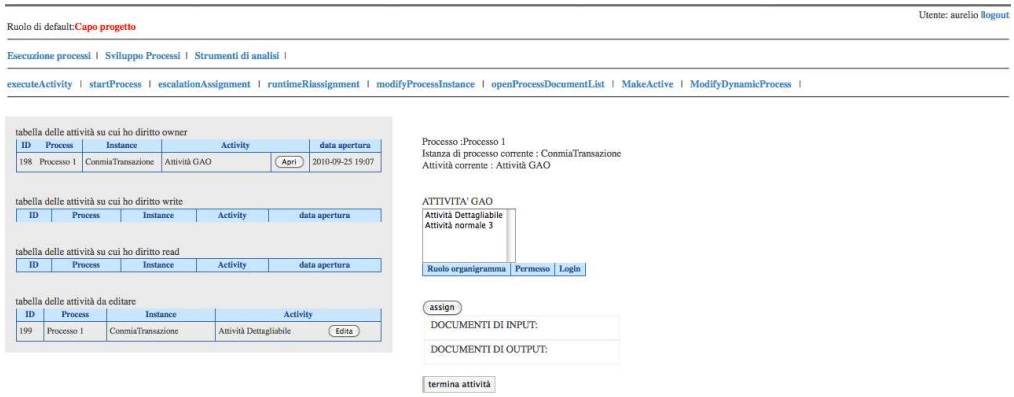

Figura 12.3. Upload documenti e chiusura attività.

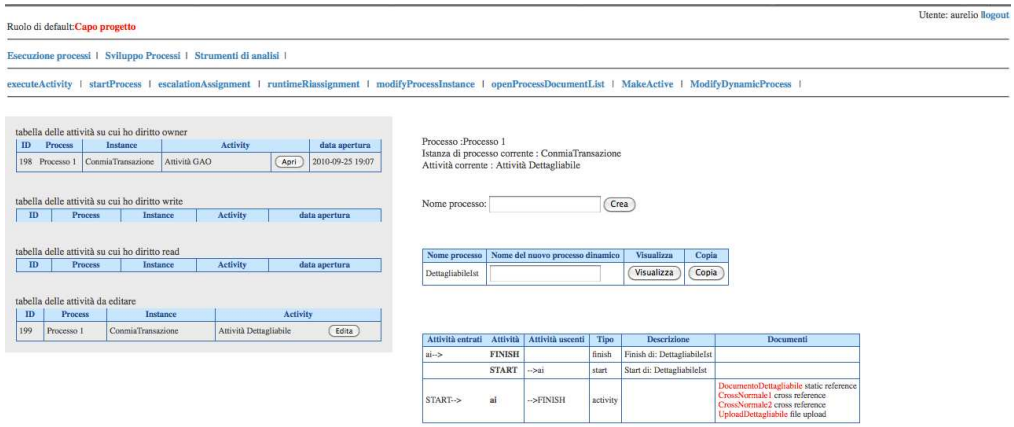

Figura 12.4. Esecuzione di una attività dettagliabile.

12.2.2 Apertura di una nuova istanza di processo

Nella pagina `Start Process`, un utente può avviare nuove istanza dei processi per i quali dispone del permesso owner sulla relativa attività Start. La struttura della pagina è visualizzata in figura 12.5. Sulla parte sinistra viene visualizzato l'elenco dei processi avviabili; una volta selezionato il processo per cui si vuole aprire una nuova istanza dal menu a tendina in alto a destra, viene richiesto di inserire il nome della nuova istanza. L'inserimento della descrizione è facoltativo. Per avviare il processo si deve agire sul pulsante `Start Process`. Non possono

essere presenti più istanze di processo con il medesimo nome, per questo, al momento dell'inserimento, viene effettuato un controllo di unicità del nome. Nel caso in cui il nome digitato sia già stato usato in precedenza, l'operazione di avvio del processo non va a buon fine ed il tutto viene segnalato con un avviso sulla pagina. Per evitare di ripetere l'errore, l'applicazione mette a disposizione una funzione di ricerca, avviabile tramite il pulsante `cerca`, che cerca nella base di dati tutti i nomi delle istanze del processo selezionato che iniziano con il testo inserito. Se, per esempio, è stato digitato il testo "Ist", verranno visualizzati risultati del tipo "Istanza 1", "ist-2", ecc.

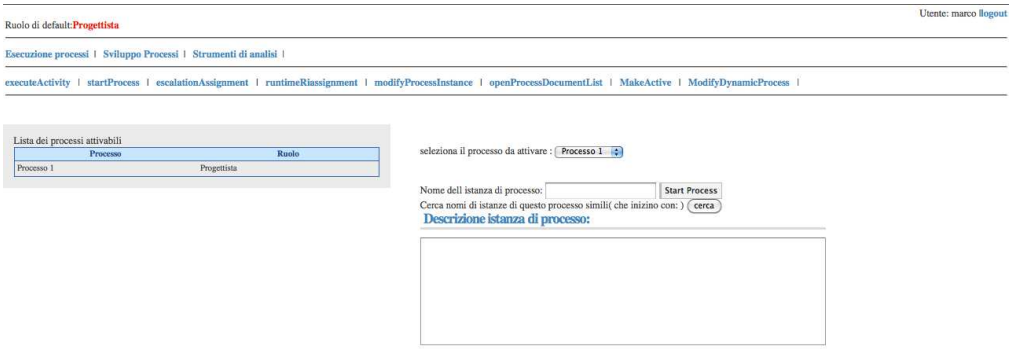

Figura 12.5. Apertura di una nuova istanza di processo.

12.2.3 Escalation per assignmenti

Quando il sistema apre un attività con politica di assegnazione Owner escalation, esso produce una segnalazione verso la login prestabilita (la persona indicata come assigner per il ruolo con permesso owner sull'attività in esame) richiedendo l'assegnazione delle risorse lavorative. Gli utenti trovano le richieste di assegnamento nella pagina `Escalation assignment` come rappresentato in figura 12.6. Selezionando dalla lista a sinistra l'attività per la quale si desidera effettuare l'assegnamento, compare sulla destra l'elenco dei ruoli che presentano un qualche permesso sull'attività e, per ciascuno di essi, l'elenco delle relative login. L'assegnatore deve selezionare le login a cui assegnare l'attività (la selezione multipla va effettuata con il comando `ctrl+click`. È obbligatorio selezionare almeno una login per il ruolo con permesso owner, altrimenti l'attività resterebbe bloccata.

Figura 12.6. Assegnamento risorse lavorative in seguito ad un'escalation.

12.2.4 Riassegnamenti runtime

L'assigner di un ruolo può, in ogni momento, modificare gli assegnamenti fatti in precedenza accedendo alla pagina `Runtime riassignment` visualizzata in figura 12.7. La struttura della pagina è del tutto simile a quella degli assenamenti per escalation descritta nella sezione precedente. Selezionando un'attività dalla lista viene visualizzata la tabella per l'assegnamento delle risorse. Viene data la possibilità di modificare i permessi anche per le attività aventi come policy una delle due FIFW (esclusiva e non), a patto che esse non siano ancora state acquisite da un utente. Pre-assegnare una di queste attività toglie la possibilità ad altri utenti di acquisirla. Questo è stato fatto in quanto si presuppone che un'assegnazione da parte dell'assigner abbia una priorità superiore rispetto alle scelte degli utenti.

12.2.5 Modifica di un'istanza di processo

Il sistema dà la possibilità agli utenti di modificare le istanze di processo da loro aperte per, ad esempio, rinominarle o cambiarne la descrizione. Questa operazione è disponibile aprendo la pagina `Modify process instance` rappresentata in figura 12.8. La pagina è praticamente identica a quella per la creazione di una nuova istanza. selezionando l'attività da modificare (anzichè il processo da aprire) vengono visualizzati i dati precedentemente inseriti con la possibilità di modificarli. Il controllo sull'unicità del nome avviene come con l'apertura di una nuova istanza.

L'utente owner del processo ha, inoltre, la possibilità di far abortire l'istanza di processo selezionata premendo il pulsante `Abort instance`. In questo caso,

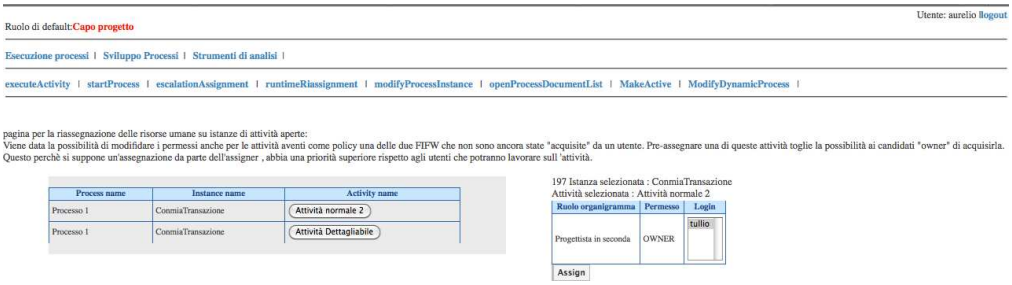

Figura 12.7. Rissegnamento risorse lavorative in runtime.

quindi, tutte le attività aperte relative all'istanza selezionata verranno, a loro volta, abortite.

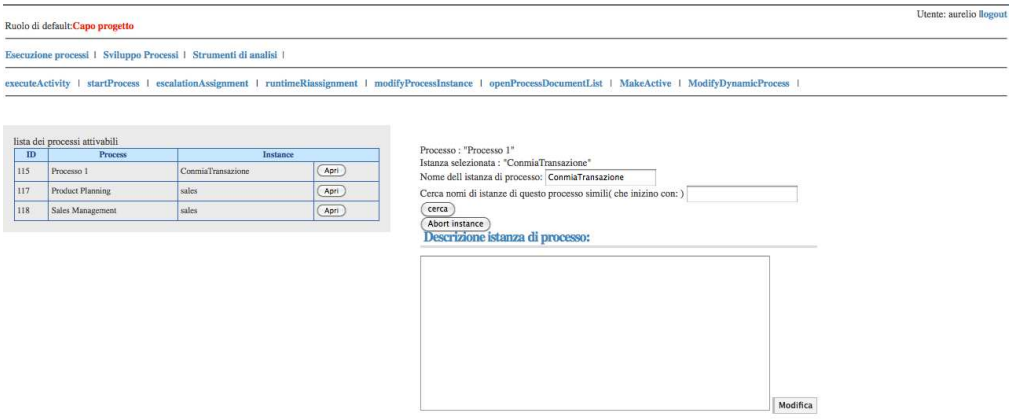

Figura 12.8. Modifica di un'istanza di processo.

12.2.6 Riepilogo documenti

La pagina `Open process document list` (vedi figura 12.9) permette agli utenti di visualizzare l'elenco dei documenti relativi ad un'istanza di processo attiva. La funzione può risultare molto utile nel caso in cui si debbano prendere alcune

decisioni sulla base di risultati prodotti da altre attività e che non sono direttamente consultabili svolgendo l'attività corrente. Nella pagina viene presentata la lista dele istanze di processo attive per le quali l'utente gode di qualche permesso. Selezionando una delle istanze, viene generata la lista dei documenti che afferiscono ad essa. Si tenga presente che se un documento si riferisce ad un'attività per la quale l'utente che sta effettuando la richiesta non presenta alcun permesso esso non sarà presente nella lista.

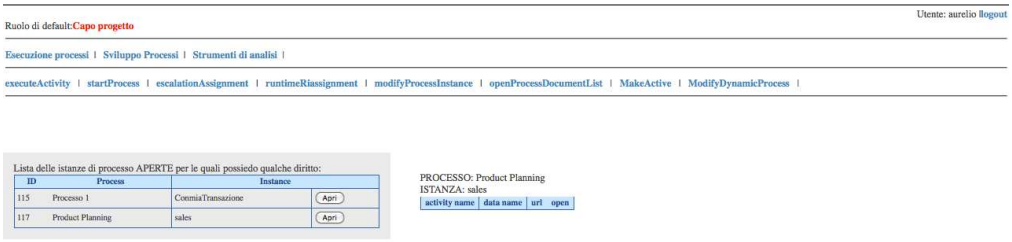

Figura 12.9. Visualizzazione dei documenti relativi ad una particolare istanza (attiva) di processo.

12.2.7 Modifica di un processo in esecuzione

Accedendo alla pagina `Modify Dynamic Process`, si possono riportare in disegno i processi in esecuzione. Tuttavia si rammenta che la modifica di attività appartenenti a un siffatto processo, possono essere limitate: possono essere modificate solo attività a valle di connettori non ancora percorsi.

Una volta confermata la modifica, il processo sarà modificabile accedendo al gruppo di funzioni `Sviluppo Processi`.

12.2.8 Attiva un processo

Dalla pagina `Make Active` un processo può essere attivato e le attività associate rese eseguibili. Prima che effettivamente il processo sia attivato, vengono fatte le verifiche per testarne la correttezza sintattica. Se il processo non supera test, non viene attivato e vengono visualizzate le cause della non-attivazione.

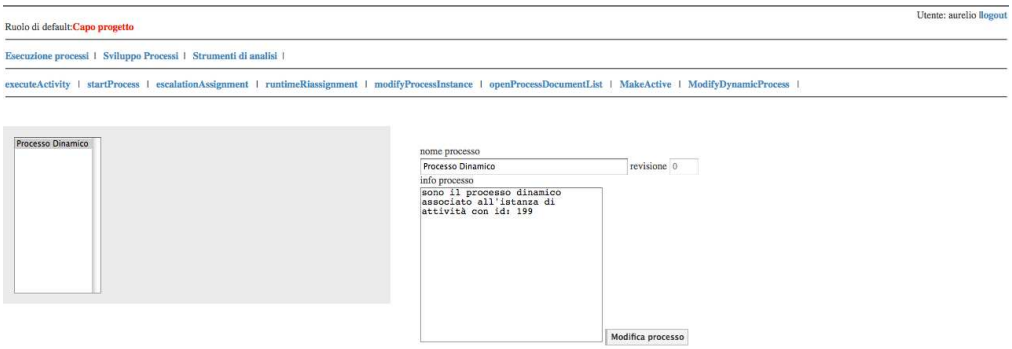

Figura 12.10. Modifica di un processo in esecuzione.

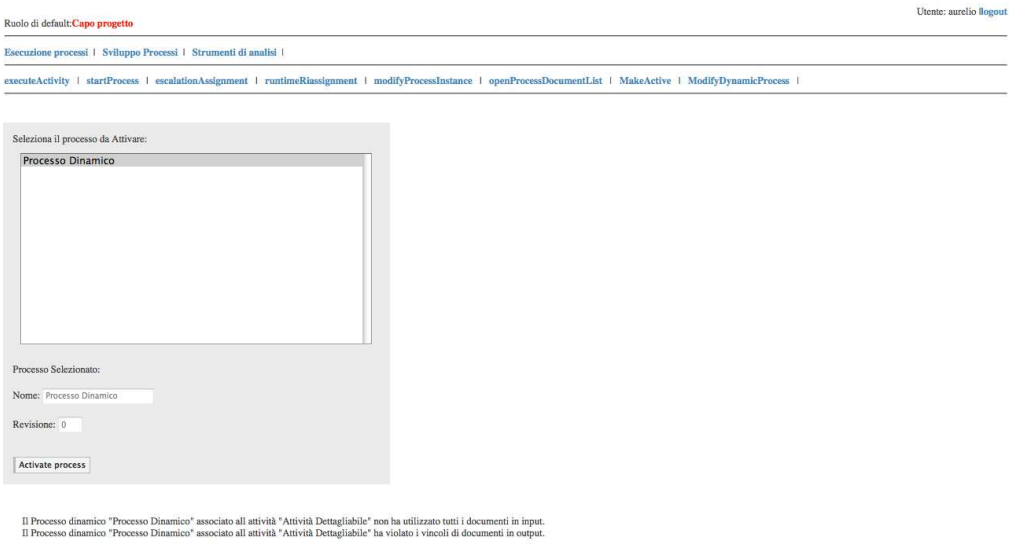

Figura 12.11. Attiva un processo.

12.3 Sviluppo processi

12.3.1 Modifica processo

Accedendo alla pagina `Mod proc.`, vedi figura 11.2, è possibile modificare le informazioni (nome e descrizione) di un processo precedentemente creato. L'utente deve selezionare il processo dalla lista sulla sinistra, ricompilare il form

con le nuove informazioni (il sistema carica di default quelle precedentemente inserite) e premere il pulsante `Modifica processo`. Accedendo a questa pagina viene, inoltre, espanso il menu delle funzioni disponibili visualizzando tutte le operazioni possibili su un processo. Si ricorda, infatti, che, al momento della sua creazione, dopo che è stata "eseguita" l'attività dettagliabile associata, un nuovo processo è, di fatto, una scatola vuota con solamente l'attività *start* e quella *finish*. Nei paragrafi seguenti verranno illustrate le procedure da seguire per utilizzare ognuna delle funzioni di editing dei processi.

Crea attività

Mediante la pagina `Crea attività`, di cui è riportato un esempio in figura 11.3, è, appunto, possibile creare una nuova attività per un certo processo precedentemente creato. Per prima cosa, l'utente deve selezionare il processo a cui l'attività fa riferimento. In seguito si devono indicare rispettivamente il tipo, il nome e la descrizione (opzionale) dell'attività. Nel caso si selezioni il tipo *Subprocess* si deve, inoltre, indicare anche il sottoprocesso da attivare selezionandolo tra quelli già presenti nella base di dati. Per terminare l'operazione premere il pulsante `Aggiungi`. Anche per le attività valgono le regole di unicità del nome all'interno però di un processo; il sistema comunica l'avvenuto inserimento o la presenza di errori di compilazione del form con modalità simili a quelle precedentemente descritte.

Modifica attività

Come avviene con i processi, anche le attività vengono create come delle box vuote. Per modificarle, si deve accedere alla pagina `Modifica attività`. L'esempio in figura 11.4 mostra la modifica di un'attività di tipo GAO; è stato volutamente scelto questo tipo di attività perché la pagina richiede l'inserimento di un numero maggiore di informazioni.

Dopo aver selezionato il processo di riferimento e l'attività da modificare, nella parte sinistra della pagina viene visualizzato un form nel quale si possono modificare i valori di nome e descrizione precedentemente inseriti. Per le attività di tipo GAO come quella dell'esempio, viene, inoltre, visualizzata la lista delle attività per il processo scelto dalla quale si devono selezionare quelle che saranno da essa gestite (si ricorda che la scelta multipla è effettuabile attraverso il comando `ctrl+click`). Premendo il tasto `Modifica` le informazioni vengono aggiornate.

La parte destra della pagina, invece, viene utilizzata per inserire le informazioni riguardanti i documenti di input/output. In base alla tipologia di documento da collegare all'attività, viene richiesto l'inserimento di informazioni differenti:

- *Cross-reference.* L'utente deve indicare l'attività che produce il documento, selezionarlo tra le opzioni possibili ed indicare un nome ed una descrizione

(facoltativa) che verranno, poi, visualizzati nella pagina per l'esecuzione dell'attività in modifica.

- *Static-reference*. L'utente deve indicare, come al punto precedente, nome e descrizione del documento e selezionare il riferimento tra i documenti disponibili nel menu a tendina. Il documento da collegare deve essere stato precedentemente inserito usando la funzione per l'aggiunta nel sistema di documenti/template in seguito illustrata.

- *Upload*. Viene richiesto di inserire il nome ed, opzionalmente, una descrizione del documento.

- *Upload con template*. Si richiede di inserire i dati relativi al documento (nome e descrizione facoltativa), nonchè il riferimento al template (precaricato) come nel caso delle *Static-reference*.

- *Documento Input da Att. Dettagliabile*. Questo tipo di documento è selezionabile se l'attività da cui il processo è stato creato lo richiede.

- *Documento Output da Att. Dettagliabile*. Questo tipo di documento è selezionabile se l'attività da cui il processo è stato creato lo richiede.

Per concludere la procedura di inserimento di un documento si deve premere il pulsante `Aggiungi documento`. Nella parte inferiore della pagina viene riportata la lista dei documenti già inseriti per l'attività in modifica.

Si aggiunge infine che nel caso l'attività fosse stata già eseguita, la modifica non è possibile.

Elimina attività

Per eliminare un'attività dal disegno di un processo, si deve accedere alla pagina `Elimina attività`. Selezionando il processo dall'elenco a sinistra, viene automaticamente aggiornata la lista delle attività precedentemente create per esso. È, quindi, possibile eliminare un'attività selezionandola e agendo sul pulsante `Delete activity`. Si ricorda che le attività "START" e "FINISH" non possono essere eliminate in quanto create in automatico dal sistema, perciò non compariranno nella lista.

Inoltre si ricorda che non è possibile eliminare attività già eseguite.

Crea connettore

Le connessioni tra le diverse attività di un processo possono essere definite accedendo alla pagina `Crea connettore`. Una volta selezionato il processo da modificare (elenco a sinistra), si devono indicare l'attività da cui parte il connettore e quella di arrivo dalle due liste popolate automaticamente sulla destra (vedi figura 11.5). L'operazione si conclude con la pressione del pulsante `Crea`

`connettore`. A creazione avvenuta viene visualizzato un messaggio di conferma del tipo "`Connettore creato con successo`". Mano a mano che i connettori vengono inseriti, viene aggiornata la tabella dei connettori già presenti visibile a fondo pagina. Si ricorda che i connettori uscenti da attività di scelta devono necessariamente essere etichettati. La label può essere inserita nell'apposita casella di testo. nel caso si tenti di creare un connettore uscente da una scelta privo di etichetta, il sistema lo segnalerà con il messaggio di errore "`Non hai inserito la label per un attività di tipo ‘‘scelta’’`". Si ricorda che non è possibile creare un connettore la cui attività di arrivo sia già stata eseguita.

Elimina connettore

La procedura per eliminare uno o più connettori è disponibile alla pagina `Elimina connettore` rappresentata in figura 11.6. Qui si deve, per prima cosa selezionare il processo da modificare; in questo modo il sistema genera, in automatico, la lista di tutti i connettori precedentemente creati. A questo punto è sufficiente selezionare il/i connettore/i da eliminare (selezionando le checkbox) e premere il pulsante `Delete`. Si ricorda che non è possibile eliminare connettori già percorsi.

12.3.2 Elimina processo

Mediante la pagina `Elimina processo` è, appunto, possibile eliminare processi che sono nello stato *Development*. La procedura consiste nel selezionare il processo da eliminare dalla lista visualizzata e premere il pulsante `Delete process`. Il buon esito dell'operazione viene segnalato con il messaggio "`Processo eliminato`". Si ricorda che l'eliminazione di un processo comporta la conseguente cancellazione di attività e connettori ad esso affini. Si ricorda che nel caso si eliminasse il processo, l'attività da cui è stato creato il processo deve essere "rieseguita".

12.3.3 Verifica processo

Prima di concludere il disegno di un processo, è consigliabile effettuare una verifica di compatibilità con il formalismo per la descrizione dei processi. Questa funzione è disponibile alla pagina `Verifica proc.` (vedi figura 11.8) e coincide perfettamente con la funzione chiamata dal sistema al momento del "make active" di processo. In seguito alla selezione da parte dell'utente del processo da verificare, vengono visualizzate tutte le eventuali incongruenze con il formalismo. I controlli effettuati sono:

- verifica che l'attività Finish sia raggiungibile;
- verifica che ogni attività Scelta abbia almeno due connettori in uscita;
- verifica che ogni attività Or abbia almeno due connettori in ingresso;
- verifica che ogni attività sia raggiungibile dallo Start;

- verifica che sia stato definito un owner per il processo;
- verifica che sia stato definito un owner per ogni attività.

Si verifica inoltre che tutti i documenti in input dell'attività associata da cui è stato creato il processo vengano "utilizzati" almeno una volta e tutti i documenti da produrre (in output) vengano prodotti almeno una volta.

12.3.4 Visualizza processo

Nel disegnare i processi è, spesso, utile visualizzare il lavoro svolto. La pagina `Visualizza proc.`, vedi figura 11.9, permette di verificare quali informazioni sono state inserite e quali no. Selezionando un processo tra quelli disegnati, viene generata una tabella che mostra, per ogni attività, chi la precede, chi la segue, il tipo, l'eventuale descrizione ed i documenti ad essa riferiti.

12.3.5 Definizione Access Control List

Attraverso la pagina `Definisci ACL`, vedi figura 11.11, il Process Designer definisce i permessi su processi e attività per ognuno dei ruoli aziendali.

La prima tabella che viene costruita rappresenta l'Access Control List per i processi; in essa vi è una colonna per ogni ruolo. Qui l'utente seleziona il processo per cui saranno definiti i permessi (anche per le sue attività, come verrà spiegato a breve). La compilazione della tabella dei permessi sul processo richiede obbligatoriamente soltanto l'indicazione di almeno un ruolo come owner del processo.

La seconda tabella rappresenta, invece, l'ACL per le singole attività del processo selezionato. Vi è una riga per ogni attività, mentre le colonne, come nella tabella di processo, corrispondono ai ruoli aziendali. Se l'ACL di un'attività rispecchia in toto quella del processo, è possibile selezionare la checkbox nella colonna `Eredita` e premere il pulsante `Aggiorna activity ACL`: tutte le righe con la checkbox selezionata rispecchieranno la struttura dei permessi del processo e l'ACL verrà salvata. Il pulsante `Reload` può essere utilizzato, invece, per annullare modifiche non ancora salvate e ricaricare l'ACL. Si ricorda che anche le attività devono avere almeno un ruolo con permesso owner. Il pulsante `Aggiorna activity ACL` deve essere premuto al termine della compilazione per rendere definitivi i cambiamenti. Essendo la compilazione della tabella un lavoro lungo e delicato, si consiglia di effettuare salvataggi periodici.

12.4 Strumenti di analisi

12.4.1 Creazione di filtri di ricerca da parte dell'amministratore

Gli utenti con ruolo di sistema ROLE_ADMIN hanno la possibilità di creare dei filtri di ricerca per effettuare l'analisi dei dati per qualsiasi altro utente come

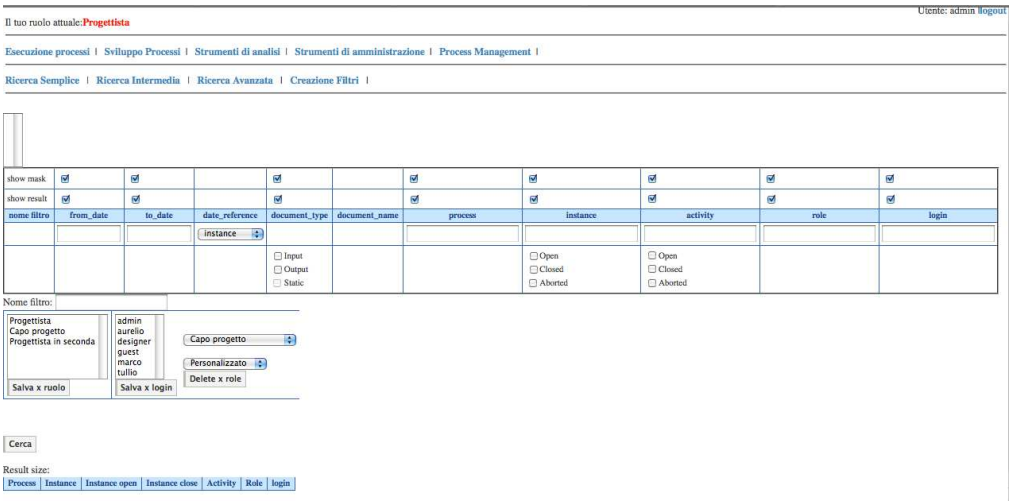

Figura 12.12. Creazione di filtri di ricerca da parte dell'amministratore.

visualizzato in figura 12.12. La pagina è accessibile al percorso `Analisi dati > Creazione filtri`. Dopo aver selezionato i campi con cui effettuare l'interrogazione sui dati (show mask)ed i campi su cui visualizzare i risultati (show result) ed aver, eventualmente, inserito i valori su cui effettuare la ricerca, è possibile salvare il filtro specificando chi lo visualizzarà e lo potrà, di conseguenza, usare. È possibile salvare un filtro per uno o più ruoli o per le singole login premendo, rispettivamente, sul pulsante `Salva per ruolo` o `Salva per login`. Al momento del salvataggio del filtro il sistema si occupa di memorizzare la login del creatore che sarà l'unica persona che lo potrà eliminare.

12.4.2 Ricerca semplice

Nella pagina `Ricerca semplice`, l'utente visualizza la lista dei filtri di ricerca da lui/per lui creati. Qui si deve selezionare un filtro dalla lista premere il pulsante `Cerca`. Vengono, quindi, immediatamente visualizzati i risultati della ricerca nello spazio inferiore della pagina. Nell'intestazione di ogni colonna è presente un bottone. Premendolo, i risultati vengono ordinati con ordine crescente rispetto al campo selezionato. La figura 12.13 mostra i risultati della ricerca effettuata dall'utente "mario" utilizzando il filtro di ricerca "Filtro personalizzato" creato per cercare tutte le attività già chiuse su cui l'utente possedeva un qualche permesso.

12.4.3 Ricerca intermedia

La pagina `Ricerca intermedia` consente all'utente di caricare filtri di ricerca esistenti e di modificare i valori dei campi di ricerca. Non è possibile aggiun-

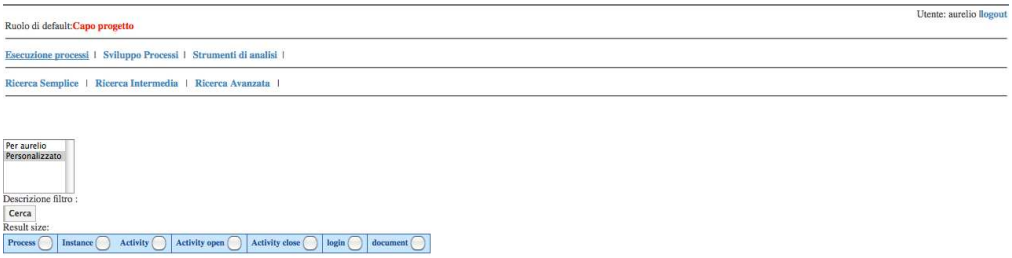

Figura 12.13. Ricerca semplice: utilizzo di un filtro pre-caricato.

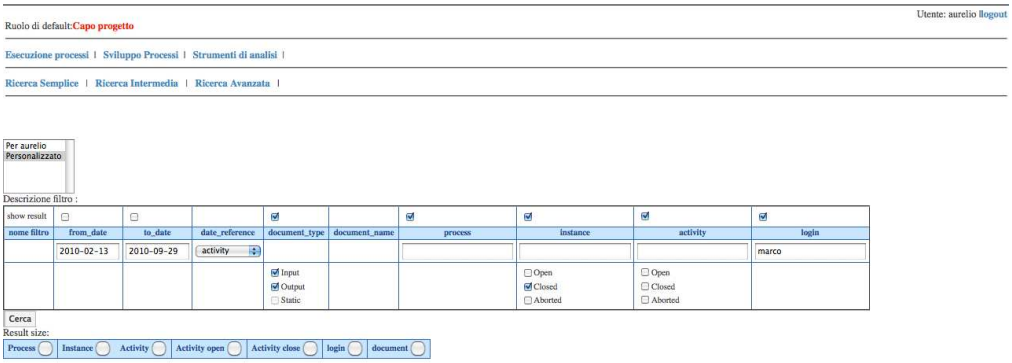

Figura 12.14. Ricerca intermedia: personalizzazione dei criteri di ricerca forniti da un filtro pre-caricato.

gere/eliminare campi di ricerca, mentre è possibile personalizzare i campi da visualizzare nei risultati. Nella figura 12.14 è rappresentata la pagina di ricerca intermedia in cui l'utente "mario" carica il filtro di ricerca "Filtro personalizzato" descritto nella sezione precedente e lo modifica per far si che le attività cercate siano solo quelle che si chiamano "start".

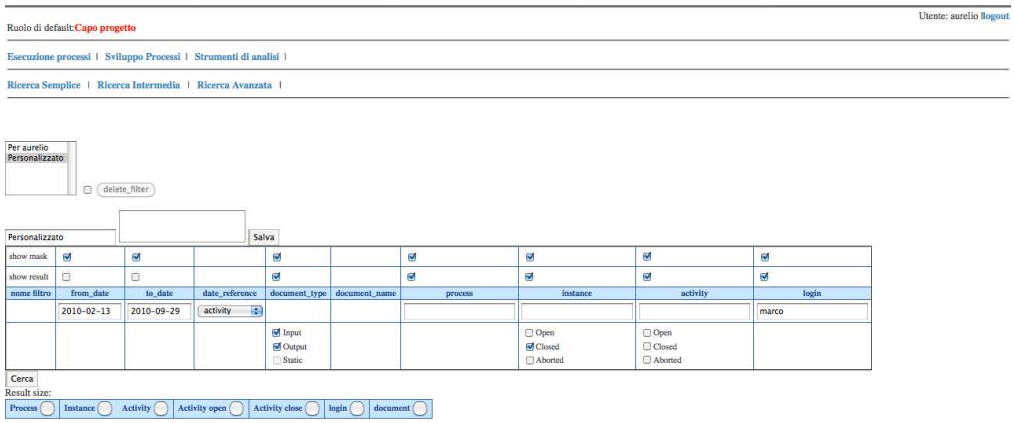

Figura 12.15. Ricerca avanzata: modifica di un filtro esistente e salvataggio di quello nuovo.

12.4.4 Ricerca avanzata

La pagina di `Ricerca avanzata` (vedi figura 12.15) permette il caricamento di filtri precedentemente creati, la loro modifica ed il salvataggio di nuovi filtri. È inoltre possibile eliminare i filtri creati da se stessi o quelli creati dall'amministratore appositamente per l'utente che si è autenticato. In particolare, la figura 12.15 mostra la modifica di "Fitro personalizzato" da parte dell'utente "mario" e il salvataggio dei nuovi criteri di ricerca come "Filtro personalizzato 2".

Appendice A

Sorgenti applicazione di prova

A.1 Albero delle directory

In questa sezione viene riportata la struttura delle directory dell'applicazione `security_project` (vedi figura A.1) spiegando il significato di ciascuna cartella ed il relativo contenuto.

.settings : contiene le impostazioni dell'IDE Eclipse.

build : contiene i files compilati organizzati in package. Nel caso specifico di `security_project` contiene le classi `WelcomeController.class` e `Permessi.class` nel package `myPack`.

src : contiene i files sorgenti organizzati in package. Nel caso specifico di `security_project` contiene le classi `WelcomeController.java` e `Permessi.java` nel package `myPack`.

WebContent :

- **META-INF**: è una cartella specifica delle applicazione *Java-based*.
- **WEB-INF**: è il cuore di un'applicazione web. Solo la directory WEB-INF risulta inaccessibile all'utente che usa il browser cercando di raggiungere una pagina direttamente tramite l'indirizzo URL; per questo motivo verranno posizionati qui tutti i contenuti che si vogliono nascondere a coloro che non passano attraverso l'autenticazione. La cartella contiene, inoltre, i files di configurazione (quelli con estensione `.xml`) che vengono utilizzati da Tomcat.
  - **jsp**: come facilmente intuibile dal nome, contiene le pagine JSP.
  - **lib**: contiene le librerie necessarie al corretto funzionamento dell'applicazione (per esempio quelle di Tomcat, quelle di Spring Security, ecc.).
  - **tags**: contiene i tag personalizzati.

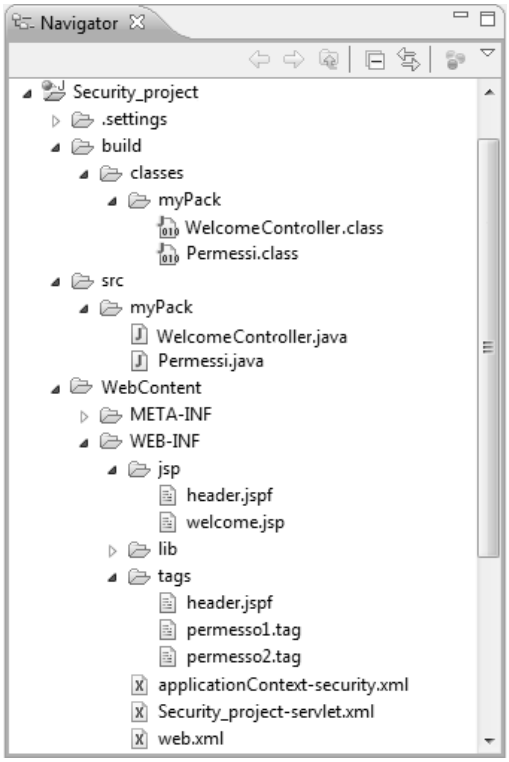

Figura A.1. Visione d'insieme delle directory dell'applicazione `security_project`.

A.2 Codice sorgente

A.2.1 `web.xml`

Il file è contenuto nella cartella `WEB-INF`.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance"
3      xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="
           http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
4      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
           http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5      id="WebApp_ID" version="2.5">
6      <display-name>Security_project</display-name>
7
8      <context-param>
9          <param-name>contextConfigLocation</param-name>
10         <param-value>
11             /WEB-INF/applicationContext-security.xml
12         </param-value>
```

```
13      </context-param>
14      <!--
15          - Loads the root application context of this web app
                at startup. - The application context is then
               available via WebApplicationContextUtils.
               getWebApplicationContext(servletContext). -->
16      <listener>
17          <listener-class>org.springframework.web.context.
               ContextLoaderListener</listener-class>
18      </listener>
19
20      <filter>
21          <filter-name>springSecurityFilterChain</filter-name>
22          <filter-class>org.springframework.web.filter.
               DelegatingFilterProxy</filter-class>
23      </filter>
24
25      <filter-mapping>
26          <filter-name>springSecurityFilterChain</filter-name>
27          <url-pattern>/*</url-pattern>
28      </filter-mapping>
29
30      <welcome-file-list>
31          <welcome-file>index.html</welcome-file>
32          <welcome-file>index.htm</welcome-file>
33          <welcome-file>index.jsp</welcome-file>
34          <welcome-file>default.html</welcome-file>
35          <welcome-file>default.htm</welcome-file>
36          <welcome-file>default.jsp</welcome-file>
37      </welcome-file-list>
38
39      <servlet>
40          <servlet-name>Security_project</servlet-name>
41          <servlet-class>org.springframework.web.servlet.
               DispatcherServlet</servlet-class>
42          <load-on-startup>1</load-on-startup>
43      </servlet>
44
45      <servlet-mapping>
46          <servlet-name>Security_project</servlet-name>
47          <url-pattern>*.htm</url-pattern>
48      </servlet-mapping>
49  </web-app>
```

A.2.2 applicationContext-security.xml

Il file è contenuto nella cartella WEB-INF.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <beans:beans xmlns="http://www.springframework.org/schema/
       security"
4  xmlns:beans="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/
       schema/beans http://www.springframework.org/schema/beans
       /spring-beans-3.0.xsd http://www.springframework.org/
       schema/security http://www.springframework.org/schema/
       security/spring-security-3.0.3.xsd">
5
6      <http auto-config='true'>
7          <intercept-url pattern="/login.jsp*" filters="none"
               />
8          <intercept-url pattern="/**"  access="ROLE_USER,
               ROLE_PROGETTISTA,ROLE_ADMIN,ROLE_UNIVERSO"
               requires-channel="http" />
9          <form-login default-target-url="/welcome.htm" always
               -use-default-target="true" />
10         <logout logout-success-url="/index.htm"/><logout />
11     </http>
12
13     <! -- ********** IN-MEMORY AUTHENTICATION ********** -->
14     <authentication-provider>
15         <user-service>
16             <user name="mario" password="rossi" authorities=
                   "ROLE_ADMIN" />
17             <user name="giuseppe" password="verdi"
                   authorities="ROLE_USER" />
18             <user name="eva" password="ca" authorities="
                   ROLE_PROGETTISTA" />
19             <user name="dio" password="omni" authorities="
                   ROLE_UNIVERSO" />
20             <user name="sadmin" password="sadmin"
                   authorities="ROLE_USER,ROLE_UNIVERSO,
                   ROLE_PROGETTISTA,ROLE_ADMIN" />
21         </user-service>
22     </authentication-provider>
23 </beans:beans>
```

A.2.3 Security_project-servlet.xml

Il file è contenuto nella cartella WEB-INF.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/
       context"
    xsi:schemaLocation="http://www.springframework.org/
       schema/beans
        http://www.springframework.org/schema/beans/spring-
           beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring
           -context-3.0.xsd">

    <context:component-scan base-package="myPack"/>

    <bean id="viewResolver" class="org.springframework.web.
       servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
</beans>
```

A.2.4 welcome.jsp

Il file è contenuto nella cartella jsp.

```
<%@ include file="header.jspf" %>
<%@ page import="org.springframework.security.context.
   SecurityContextHolder" %>
<%@ page import="org.springframework.security.Authentication
   " %>
<%@ page import="org.springframework.security.
   GrantedAuthority" %>
<%@ page import="org.springframework.security.adapters.
   AuthByAdapter" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//
   EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
           charset=ISO-8859-1">
        <title>Security_project welcome page</title>
    </head>
    <body>
        Benvenuto nel sistema!
```

```
15
16          <authz:authorize ifAnyGranted="ROLE_ADMIN,
               ROLE_UNIVERSO" >
17          <h3> scritta riservata a chi ha permesso universo o
               admin </h3>
18          </authz:authorize>
19          <%
20          Authentication auth = SecurityContextHolder.
               getContext().getAuthentication();
21          if (auth != null) { %>
22              <b><br>Sei autenticato con i seguenti ruoli:<br>
                   </b>
23              <%
24              GrantedAuthority[] granted = auth.getAuthorities
                   ();
25              for (int i = 0; i < granted.length; i++) { %>
26                  <%= granted[i].toString() %> <br>
27                  <%          }}%>
28          <b><br>permessi:<br></b>
29          permesso 1: <c:out value="${miopermesso.p1}" /><br>
30          permesso 2: <c:out value="${miopermesso.p2}" /><br>
31
32          <tags:permesso1 perm_value="${miopermesso.p1}"/><br>
33          <tags:permesso2 perm_value="${miopermesso.p2}"/><br>
34          <br>
35          <a href="<c:url value="/j_spring_security_logout" />
               ">logout</a>
36      </body>
37  </html>
```

A.2.5 WelcomeController.java

Il file è contenuto nella cartella `src/myPack`.

```
1  package myPack;
2
3  import org.springframework.security.core.Authentication;
4  import org.springframework.security.core.context.
       SecurityContextHolder;
5  import org.springframework.stereotype.Controller;
6  import org.springframework.web.bind.annotation.
       RequestMapping;
7  import org.springframework.web.bind.annotation.RequestMethod
       ;
8  import org.springframework.web.servlet.ModelAndView;
9
10 @Controller
11 @RequestMapping("/welcome.htm")
12 public class WelcomeController{
```

```
13      @RequestMapping(method=RequestMethod.GET)
14      public ModelAndView handleRequest(@ModelAttribute("
            miopermesso") Permessi perm){
15          Authentication auth = SecurityContextHolder.
                getContext().getAuthentication();
16          if (auth.getName().equals("mario"))
17          {
18              perm = new Permessi(2,1);
19          }
20          if (auth.getName().equals("giuseppe"))
21          {
22              perm = new Permessi(1,2);
23          }
24          if (auth.getName().equals("eva"))
25          {
26              perm = new Permessi(2,0);
27          }
28          if (auth.getName().equals("dio"))
29          {
30              perm = new Permessi(2,2);
31          }
32          return new ModelAndView("welcome","miopermesso",perm
                );
33      }
34      @ModelAttribute("miopermesso")
35      public Permessi reference()
36      {
37          return new Permessi();
38      }
39  }
```

A.2.6 Permessi.java

Il file è contenuto nella cartella src_myPack.

```
1  package myPack;
2
3  public class Permessi {
4
5      private int p1;
6      private int p2;
7
8      public Permessi() {
9          super();
10     }
11
12     public Permessi(int p1, int p2) {
13         super();
14         this.p1 = p1;
```

```
15          this.p2 = p2;
16      }
17
18      public int getP1() {
19          return p1;
20      }
21
22      public void setP1(int p1) {
23          this.p1 = p1;
24      }
25
26      public int getP2() {
27          return p2;
28      }
29
30      public void setP2(int p2) {
31          this.p2 = p2;
32      }
33  }
```

A.2.7 header.jspf

Il file è contenuto nella cartella jsp.

```
1  <%@ page session="false"%>
2  <%@ page contentType="text/html"%>
3  <%@ page pageEncoding="ISO-8859-1"%>
4  <%@ page language="java" contentType="text/html"%>
5
6  <%@ taglib prefix="tags" tagdir="/WEB-INF/tags" %>
7
8  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c
       " %>
9  <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="
       fmt" %>
10 <%@ taglib uri="http://java.sun.com/jsp/jstl/functions"
       prefix="fn" %>
11 <%@ taglib uri="http://www.springframework.org/tags" prefix=
       "spring" %>
12 <%@ taglib uri="http://www.springframework.org/tags/form"
       prefix="form" %>
13 <%@ taglib prefix="authz" uri="http://www.springframework.
       org/security/tags"%>
```

Bibliografia

[1] M. Bertocco, P. Callegaro, D. De Antoni Migliorati, *Ingegneria della qualità*. De Agostini Scuola, 1st ed ed., 2006. ISBN: 978-88-825-172942.

[2] A. Molinaroli, E. Righetto, "Analisi e implementazione di procedure per il workflow management in un'azienda certificata iso 9001," 2010. Università degli Studi di Padova.

[3] C. S. Horstmann, G. Cornell, *Core Java, Volume I- Fundamentals*. Prentice Hall, 8th edition ed., 2008. ISBN: 978-0-13-235476-9.

[4] C. S. Horstmann, G. Cornell, *Core Java, Volume II- Advanced Features*. Prentice Hall, 8th edition ed., 2008. ISBN: 978-0-13-235479-0.

[5] S. Ladd, D. Davison, S. Devijver, *Expert Spring MVC and Web Flow*. Apress, 1st edition ed., 2006. ISBN: 978-1-59059-584-8.

[6] J. Machacek, A. Vukotic, A. Chakraborty, J. Ditt, *Pro Spring 2.5*. Apress, 1st edition ed., 2008. ISBN: 978-1-4302-0506-7.

[7] T. Risberg, R. Evans, P. Tung, *Introduction to Spring MVC, developing a Springframework MVC application step-by-step*, 2008. http://static.springsource.org/docs/Spring-MVC-step-by-step/.

[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elementi per il riuso di software a oggetti*. Addison-Wesley, 1st italian ed., 2008. ISBN: 978-88-7192-150-1.

[9] V. Chopra, S. Li, J. Genender, *Professional Apache Tomcat 6*. Wiley Publishing, Inc., 1st ed., 2007. ISBN: 978-0-471-75361-2.

Elenco delle tabelle

Elenco delle figure

Ringraziamenti

Sono sempre stato convinto che in questo mondo per raggiungere risultati importanti, la forza del singolo non basti. Certo, l'essere perseveranti, inclini al sacrificio e quel briciolo di intelligenza a sufficienza possono aiutare, ma non sono garanzia di successo; per una piena riuscita, gli ingredienti devono andarsi a cercare soprattutto nelle persone che lo circondano e che lo appoggiato e sostengono. E penso che senza il supporto di quest'ultime, il singolo non andrebbe da nessuna parte.

Personalmente parlando, questa agognata laurea è il frutto di un percorso iniziato tanti anni addietro, a cui hanno contribuito non poche persone. Vorrei volgere i miei ringraziamenti alla seguente, sicuramente non esaustiva, lista di persone:

- ai miei genitori (*A bè* e *A ma*), che mi hanno sempre tacitamente sostenuto.
- a mio fratello, mia sorella e mia cognata che hanno sempre creduto in me.
- a *Sisi* che ha sempre voluto aiutarmi, anche quando non ne avevo bisogno.
- a *Ayi* e *Shushu* che sono stati dei secondi genitori per me.
- alla mia Prof.ssa delle medie Marilena e il suo compagno Fabio, che mi hanno sempre incoraggiato e aiutato.
- alla Prof.ssa Santilli, a cui devo la passione per i romanzi di Asimov.

In ultimo luogo vorrei ringraziare il mio Prof. delle superiori Sergio, a cui devo la visione diversa del mondo; Silvano, un mito delle calzature, che mi ha proferito tanti saggi consigli; Michele e Matteo, con cui ho condiviso tanti anni di Università a Roma.

Last but not least, un grazie di cuore al Prof. Bertocco e all'Ing. Franchin per avermi dato la possibilità di aver intrapreso (e finito) questo cammino non facile, ma ricco di soddisfazioni.

Ottobre 2010

Qifeng