Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di laurea

Sviluppo di un sistema documentale per i back office bancari utilizzando tecnologie open source

Candidato:
Lorenzo Caldera

Relatore:
Ch.mo Prof. Michele Moro

Anno Accademico 2011/2012

Ai miei genitori,
a Serena.

Indice

Sommario

Questa tesi espone in sintesi il lavoro di ricerca e sviluppo che ho svolto durante lo stage effettuato presso l'azienda e-Project srl di Padova. Tale attività si colloca all'interno di un progetto per la realizzazione di un sistema documentale per istituti bancari, finalizzato alla dematerializzazione e alla gestione di documenti bancari. Il sistema si compone di un'applicazione client per l'acquisizione e l'invio dei documenti ad un server appositamente configurato per gestirli e di un'altra applicazione client che permette l'accesso e l'elaborazione di questi documenti secondo i processi dell'istituto. La parte del progetto descritta in questa tesi riguarda la componente server, ovvero la sua installazione, configurazione ed estensione con servizi personalizzati per soddisfare i requisiti di progetto. Il progetto utilizza largamente tecnologie open source, in particolare il software ECM Alfresco che si appoggia al DBMS PostgreSQL ed è eseguito tramite il server di applicazioni Apache Tomcat. I servizi esposti dal server sono tutti servizi web che seguono lo stile architetturale REST e sono sviluppati tramite un linguaggio di scripting (javascript).

Capitolo 1

Introduzione

Esaminando le diverse gestioni delle aziende di medio-grandi dimensioni, un denominatore comune che quasi sempre si riscontra è l'approccio per processi. In maniera informale si può descrivere un processo come *un insieme di attività di lavoro, organizzate e gestite per raggiungere un risultato prestabilito* [6]. Si tratta quindi di componenti chiave per qualsiasi attività lavorativa e definirli e gestirli correttamente risulta un compito praticamente obbligatorio per tutte le aziende che hanno raggiunto un certo livello di complessità, e non solo loro.

Applicare un'adeguata gestione per processi significa che i processi devono essere progettati, documentati e gestiti, ovvero devono essere misurati e controllati. È facile quindi intuire che per supportare una tale infrastruttura sia necessaria una mole non indifferente di documenti e in generale di contenuti che devono essere appositamente gestiti e controllati. Il sistema di gestione, quindi, deve essere tale da non soffocare l'attività principale dell'organizzazione, ma da affiancarla in maniera intelligente, riducendo al minimo *l'overhead*[1] necessario per la documentazione.

In questo contesto si inserisce il progetto oggetto di questa tesi, in particolare si colloca all'interno del settore bancario, dove gran parte dei processi cor-

[1]Termine che indica le risorse accessorie richieste in aggiunta a quelle strettamente necessarie per lo svolgimento di un metodo o di un processo.

rispondono ad una movimentazione di documenti fra diversi soggetti o gruppi di lavoro, che possono essere dislocati in diverse filiali o uffici centrali.

1.1 Front Office e Back Office

Osservando gli istituti di credito, infatti, emerge la tendenza di questi a svilupparsi in due tipologie di uffici: i **Front Office**, che si possono identificare nelle filiali e sono degli sportelli orientati a gestire i rapporti con la clientela, e i **Back Office**, ovvero degli uffici centralizzati specializzati nell'esecuzione delle operazioni bancarie, il *core business* [2] dell'istituto.

1.2 Obiettivi

L'obiettivo principale del progetto è quello di fornire gli istituti di credito di un sistema documentale particolarmente flessibile, in grado di supportare tutte le funzionalità richieste dall'istituto, a partire dall'archiviazione e classificazione dei documenti, la gestione dei processi a loro associati e la condivisione degli stessi tra i vari utenti e i gruppi di lavoro. Non si vuole però costringere la banca a riorganizzare il suo sistema di gestione, bensì si cerca di adattare il sistema ai processi già esistenti, con l'intento più generale di rivoluzionare gli strumenti piuttosto che i metodi e le abitudini.

La strada per perseguire questo scopo comincia con l'implementazione delle funzionalità più semplici, come il passaggio di documenti tra front-office e back-office e coinvolge la digitalizzazione dei documenti: introdurre un sistema semplice ed immediato per informatizzare i contenuti comporta molteplici vantaggi: in primis, la semplificazione di tutta l'infrastruttura per l'immagazzinamento e il recupero dei documenti. Un beneficio, questo, che implica un sensibile incremento dell'efficienza di gestione, grazie agli automatismi per classificare e rintracciare i contenuti. Poi bisogna contare la notevole riduzione

[2] La principale attività aziendale, di tipo operativo, che ne determina il compito fondamentale preposto ai fini di creare un fatturato ed un conseguente guadagno.

dell'uso della carta, un tema che rientra nell'ambito dello sviluppo sostenibile e che è sempre più tenuto in considerazione in un momento in cui le grandi aziende, impegnate negli obiettivi di qualità e miglioramento continuo, guardano al profitto non solo in termini economici, ma anche ambientali e sociali. A questi si aggiungono poi le potenzialità informatiche del sistema, come la possibilità di automatizzare interamente (o in parte) la la movimentazione dei documenti o di applicare tecniche di *Business Intelligence*. [3]

Ma prima di passare all'esposizione nel dettaglio del lavoro svolto voglio presentare brevemente l'azienda promotrice del progetto, presso la quale è stata svolta l'attività di stage.

1.3 L'azienda E-project srl di Padova

E-project, fondata nel 2001 da professionisti provenienti dal mondo della consulenza aziendale ed informatica e dal mondo del *document managment*, realizza, gestisce e sviluppa applicazioni e progetti di tipo organizzativo ed informatico nei settori:

- Servizi;
- Assicurazioni;
- Banche;
- Pubblica Amministrazione.

Nel settore delle assicurazioni ha realizzato soluzioni utilizzando prodotti di *document managment e BPM*[4]; nel settore bancario possiede competenze nell'area della gestione documentale e l'organizzazione dell'area *operations* e dei

[3]Insime delle metodologie, dei processi e delle tecnologie che trasformano dati grezzi in informazioni significative e utili per la formulazione di strategie e decisioni aziendali.

[4]Business Process Management: l'insieme di attività necessarie per definire e ottimizzare i processi aziendali, al fine di rendere efficiente ed efficace il business dell'azienda.

back-office centralizzati; nel settore dei servizi ha contribuito alla realizzazione di soluzioni legate alla conservazione sostitutiva.

Le competenze che si sono sviluppate all'interno dell'azienda nell'ambito dell'Information Technology rientrano quindi nelle aree di:

- Business Process Reengineering;
- Program Management;
- Business Process Management e sistemi documentali;
- Business Intelligence;
- Competenze di virtualizzazione e sistemistiche in ambiente Linux;
- Realizzazione di Portali e applicazioni web per la Pubblica Amministrazione;
- Stress Test sulle performace di sistemi informativi.

E-project vanta 14 operatori nel settore informatico con il 92% di laureati e un network di aziende partner che permettono di rispondere tempestivamente alle esigenze progettuali

Figura 1.1: Logo dell'azienda E-project s.r.l.

Capitolo 2

Il Progetto

Come già introdotto, il progetto consiste nello sviluppo di un sistema documentale per supportare le attività dei back-office bancari e l'interazione con i front-office. Parte di esso consiste quindi nella dematerializzazione dei documenti nei processi più comuni, in particolare dove la presenza di caratteristiche standard nel tipo di documentazione permette un'agevole gestione informatica, consentendo di snellire tutta la struttura necessaria alla creazione, alla gestione e al mantenimento di tali processi. Il progetto pilota, qui presentato, è stato incentrato sulla gestione dei bonifici bancari, con l'obiettivo più ambizionso però di essere poi esteso a tutte le funzioni più comuni, come la gestione degli assegni, le domiciliazioni, i pagamenti, etc.

2.1 Architettura del sistema

Non essendo particolarmente elevata la mole di informazioni da gestire e non essendovi particolari esigenze di scalabilità, l'architettura scelta è quella di tipo centralizzato basata sul modello client-server. Come si può notare dalla figura 2.1 l'accesso al server, dove sono di fatto memorizzati i documenti, avviene da due tipologie di ufficio: dalla parte del front-office per caricare gli ordini di bonifico e dalla parte del back-office per consultare gli ordini da eseguire. Il sistema in generale, quindi, segue lo stile architetturale a livelli [3] stratificato nel seguente modo:

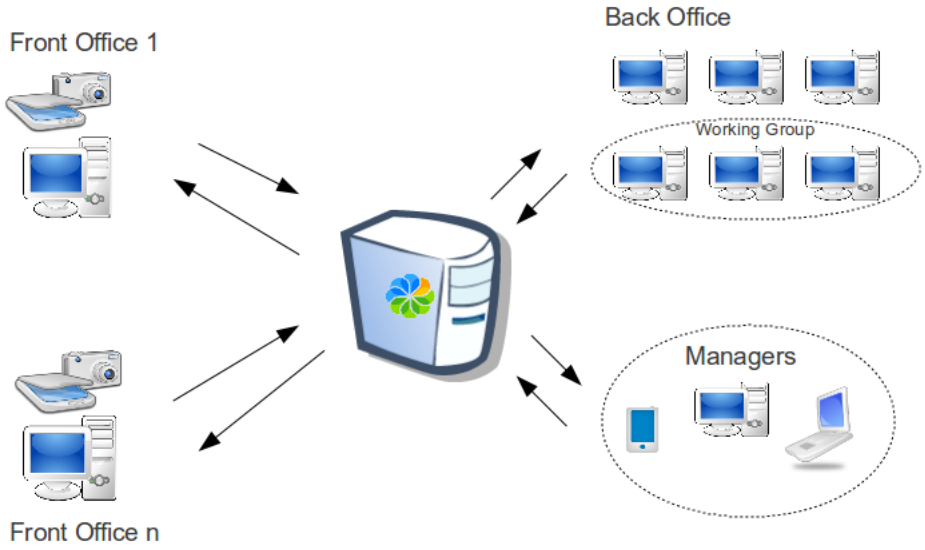

Figura 2.1: Componenti d'interazione del sistema

- il livello dell'interfaccia utente;
- il livello applicativo;
- il livello dei dati.

L'organizzazione di questi livelli è fortemente sbilanciata sulla parte del server. Questo significa che è compito del server memorizzare i contenuti, gestire le azioni e i flussi di lavoro a loro associati e mantenere perfino i metadati riguardanti l'interfaccia, lasciando alle applicazioni client i meri compiti di acquisizione e presentazione. La motivazione che sta alla base di questa scelta è quella di cercare di mantenere le configurazioni lato server, in modo che risulti il più semplice possibile introdurre modifiche e aggiungere funzionalità al sistema esistente.

Da una prospettiva di alto livello, i componenti chiave che intervengono sono:

- l'applicazione *front-end* [1] di interfacciamento con l'utente dal lato del front-office dedicata all'acquisizione dei documenti;

- l'applicazione server *back-end* [2] che fornisce i servizi per l'immagazzinamento dei documenti, la gestione dei processi ad essi associati e la loro rintracciabilità;

- un'altra applicazione *front-end* di interfacciamento con l'utente, questa volta dal lato del back-office.

Per chiarire meglio il funzionamento globale del sistema, le diverse fasi che si susseguono in un generico ciclo di utilizzo possono essere:

- Collegamento: dalla postazione di una filiale l'addetto lancia l'applicazione client per eseguire l'ordine di una distinta di bonifici. Il client, quindi, si collega al server scaricando la descrizione dei metadati da associare al documento e presentando un'interfaccia con i campi da inserire;

- Acquisizione: l'addetto lancia la scansione dei documenti da inserire, effettuando il controllo visivo del corretto inserimento dei campi essenziali;

- Inserimento: terminata la scansione il client richiede al server il servizio di inserimento di un documento passando il file acquisito ed i dati ad esso associati;

- Avvio del processo: in questo caso il tipo di documento prevede anche l'avvio di un processo di gestione, che comincia con la notifica dell'ordine al gruppo di lavoro dedicato alla gestione dei bonifici, passa per una fase di approvazione e termina con la notifica dell'esecuzione dell'ordine al committente;

Il lavoro svolto da me all'interno del progetto ha riguardato soprattutto la parte di back-end, ovvero l'installazione della piattaforma server e lo sviluppo

[1] front-end: è la parte di un sistema software che gestisce l'interazione con l'utente o con sistemi esterni che producono dati di ingresso.

[2] back-end: è la parte che elabora i dati generati dal front end.

dei servizi richiesti dal nostro sistema documentale, in particolare quello di acquisizione del documento e di gestione del processo associato. Sono stato poi occupato nello sviluppo dell'applicazione client di accesso da parte del back-office.

2.2 Le tecnologie adottate

Nella scelta delle applicazioni di terze parti l'attenzione si è sempre spostata sulle tecnologie *Open Source*[3]. Sicuramente il lato economico figura tra i motivi di questa scelta, ma non è il solo: la filosofia open source incoraggia il libero studio dei software e la condivisione del codice. Questo, in genere, comporta la formazione di importanti comunità di utenti che collaborano al progetto e costituiscono anche una considerevole risorsa di supporto, sia per la generazione della documentazione che per la risoluzione dei problemi.

Studiare progetti open source, quindi, è un'ottima esperienza formativa, soprattutto perché essendo fondati sulla condivisione, fanno degli standard il loro cavallo di battaglia. Vantaggio questo che permette di evitare, sia allo sviluppatore che all'azienda, costosi effetti di *lock-in*. [4]

Il cuore del sistema è certamente il software **Alfresco** che rientra nella categoria degli Enterprise Content Management (ECM) e sarà introdotto nel prossimo capitolo. Una caratteristica di questo software che però voglio citare subito è quella di essere indipendente dall'ambiente. Questo significa che l'utilizzatore può scegliere il sistema operativo, il database su cui Alfresco si appoggia per la gestione dei contenuti e il server di applicazioni che lo incorpora. Si tratta di una caratteristica importante, che vuole sottolineare il fatto

[3]Software che si contraddistinguono per la libera disponibilità del codice sorgente. I sorgenti sono forniti sotto apposite licenze open-source che garantiscono agli utenti il diritto di studiare, cambiare, migliorare e redistribuire il software.

[4]Termine economico che denota una situazione in cui, scelta una tecnologia, si è vincolati all'utilizzo della stessa per via delle barriere economiche dovute ai costi di transizione (non solo economici).

che ECM sia un termine che indica più i servizi offerti da un'applicazione piuttosto che l'applicazione in sé.

Rimanendo fedeli alla filosofia open source del progetto, la nostra implementazione della piattaforma server sarà quindi costituita da un sistema operativo **Linux**, sul quale sarà configurato il server applicativo **Apache Tomcat** necessario per l'esecuzione di **Alfresco** come applicazione web. A sua volta Alfresco si appoggerà su **postgreSQL** per i servizi riguardanti la creazione e il mantenimento della base di dati. Alfresco sarà poi configurato per implementare i servizi personalizzati richiesti dalla nostra applicazione. Essi saranno sviluppati in linguaggio javascript ed esposti alle applicazioni front-end tramite servizi web.

Riguardo alle tecnologie usate, vorrei anche citare il fatto che il sistema è stato totalmente installato e configurato all'interno di una macchina virtuale, utilizzando la tecnologia di virtualizzazione **Virtualbox**. Questa pratica di sviluppo è molto utile, in quanto consente di isolare l'ambiente di lavoro e permette di effettuare rapidi backup di tutto il sistema. La virtualizzazione consente inoltre una rapida migrazione del server, dal momento che separa di fatto il software dall'hardware sottostante.

2.3 Metodologia Agile

Il termine racchiude un insieme di metodi di sviluppo del software, basati su uno sviluppo di tipo iterativo ed incrementale, dove i requisiti e le soluzioni sono in continua evoluzione e si sviluppano attraverso l'auto-organizzazione e la composizione di gruppi di lavoro formati da persone con competenze interfunzionali. Questa metodologia cerca di promuovere lo sviluppo del software in maniera progressiva attraverso piccole finestre temporali, chiamate iterazioni. Ogni iterazione, della durata di qualche settimana, è un progetto a se stante che ha l'obiettivo di rilasciare un piccolo incremento nelle funzionalità del software. Per ridurre i rischi di fallimento, quindi, si incoraggia l'utilizzo di una

progettazione adattiva per dare una rapida e flessibile risposta ai cambiamenti.

I principi fondamentali che hanno ispirato questa metodologia e sono indicati nel manifesto sono solo quattro:

- le persone e le interazioni sono più importanti dei processi e degli strumenti, ossia, promuovere la collaborazione e la comunicazione tra gli attori di un progetto produce la miglior risposta ai requisiti richiesti;
- il software funzionante è una priorità rispetto alla documentazione, per cui è necessario mantenere il codice semplice ed avanzato tecnicamente, riducendo la documentazione al minimo indispensabile;
- la collaborazione con i clienti deve andare al di là del contratto, vale a dire, la collaborazione diretta con il cliente è più fruttuosa del puro rapporto contrattuale;
- è più utile rispondere ai cambiamenti piuttosto che aderire al progetto, quindi il team di sviluppo dovrebbe essere autorizzato a suggerire modifiche al progetto in qualsiasi momento.

Nonostante il progetto fosse solo allo stato embrionale, con solo due stagisti a disposizione, nello svolgimento si è cercato di adottare una metodologia Agile, cercando di avanzare attraverso piccoli cicli di sviluppo intervallati da proficue riunioni necessarie alla discussione degli obiettivi e delle proposte derivanti dai problemi e dai risultati del lavoro svolto.

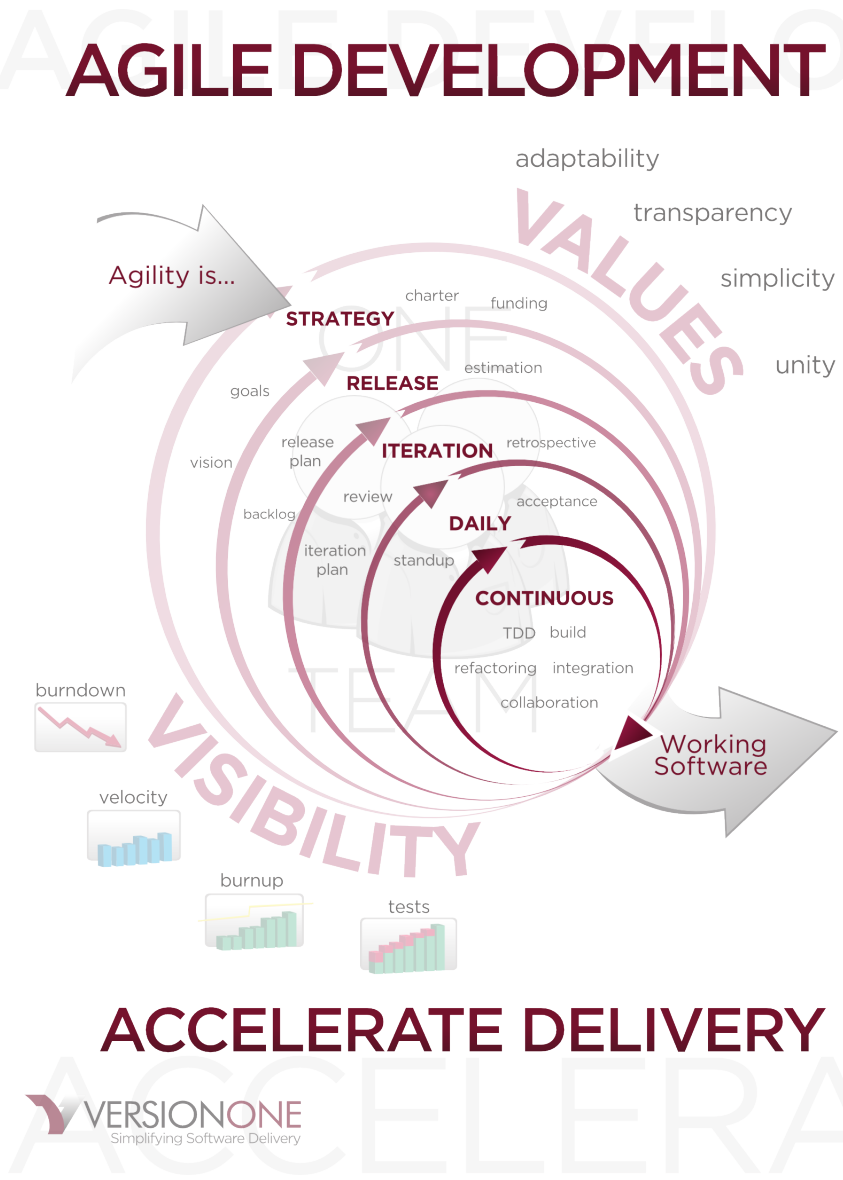

Figura 2.2: Agile Software Development

Capitolo 3

ECM e Alfresco

3.1 Enterprise Content Management

Secondo l'AIIM[1], la comunità globale dei professionisti dell'informazione, il termine Enterprise Content Management (ECM) indica :

> "L'insieme delle strategie, dei metodi e degli strumenti utilizzati per catturare, gestire, memorizzare, preservare e rendere disponibili i contenuti e i documenti collegati ai processi organizzativi. Essi coprono l'intero campo di azione dell'azienda e permettono la gestione dell'informazione non strutturata[2], ovunque essa sia presente."[1]

I sistemi ECM, in genere, sono composti sempre da un *repository*, ovvero un componente necessario alla classificazione e alla memorizzazione dei contenuti. Esso è poi accompagnato da un certo numero di applicazioni che ne

[1]Association for Information and Image Management: comunità globale di professionisti dell'informazione la cui missione è quella di far comprendere le sfide correnti e future della gestione dei beni dell'informazione in un era di tecnologie sociali, mobili, cloud e di grandi moli di dati.

[2]Si riferisce all'informazione computerizzata che non dispone di un modello dati. Il termine distingue questo tipo di informazione dall'informazione memorizzata in campi strutturati, come nelle basi di dati.

permettono l'accesso, il controllo e la presentazione. I contenuti possono essere un qualsiasi tipo di informazione non strutturata, come documenti, pagine web, video, registrazioni o semplici file.

Più nel dettaglio, un sistema ECM deve essere in grado di gestire il formato binario del contenuto digitalizzato, i metadati che descrivono il suo contesto, le associazioni con altri contenuti e gli indici per accedervi rapidamente. Inoltre, deve poter gestire i processi associati al contenuto durante tutto il suo ciclo di vita e deve assicurare che l'informazione sia corretta.

Attualmente si possono identificare cinque macro-aree di applicazione ECM:

- gestione dei documenti: per la cattura, la modifica e la distribuzione di documenti di ufficio e file;
- gestione dei contenuti web: per la gestione dei siti web e delle pubblicazioni aziendali;
- gestione delle registrazioni: per l'archiviazione a lungo termine di importanti documenti e registrazioni;
- gestione delle immagini;
- gestione di beni digitali.

3.2 Alfresco

Alfresco è un software ECM open source particolarmente interessante perché costruito secondo una moderna architettura che permette un elevato grado di modularità e di prestazioni scalabili. Esso fa largo uso di componenti open source di successo, come la piattaforma **Spring**, che permette la modularità delle funzioni, **Lucene**, celebre libreria per la ricerca testi, **Hibernate**, che fornisce il servizio ORM [3], **jBPM** e **Activiti**, motori per l'esecuzione dei dia-

[3]Object Relational Mapping: consiste nella rappresentazione e mantenimento, su un database relazionale, di un sistema di oggetti java.

grammi di flusso associati ai contenuti e **FreeMarker**, un motore java per la creazione di modelli di presentazione web che si focalizza sull'architettura MVC [4].

3.2.1 Architettura

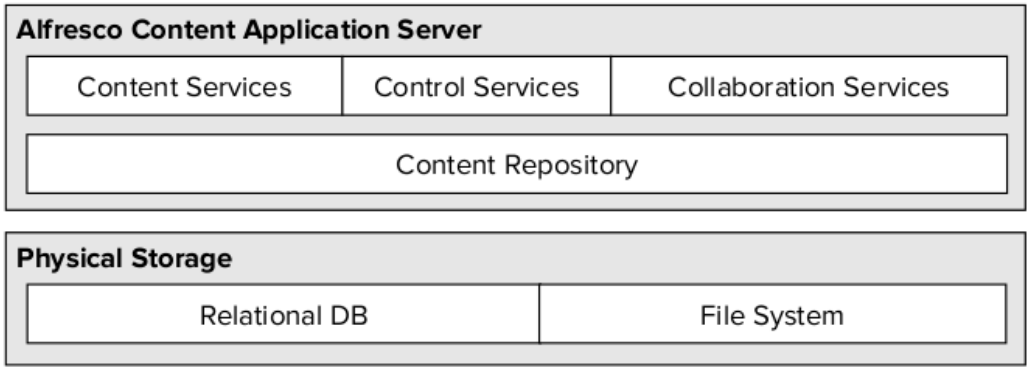

Figura 3.1: Architettura base di Alfresco

Come tutti i sistemi ECM, Alfresco dispone di un componente che svolge la funzione di *repository*, chiamato *Content Repository*, il quale si interfaccia con un altro componente di base, il *Content Application Server*, ovvero un server responsabile della logica di controllo che offre i servizi per l'accesso, la gestione, l'aggiornamento e il mantenimento del *Content Repository*. Quest'ultimo ha il compito di immagazzinare i contenuti, compresivi dei loro metadati e delle associazioni. Esso è paragonabile ad una base di dati con l'estensione necessaria al mantenimento di archivi binari e di indici di testo e per questo utilizza una combinazione di servizi di RDBMS [5] e di file system. Infatti, in Alfresco i binari dei contenuti sono mantenuti in file memorizzati utilizzando il file system del sistema operativo, secondo un metodo di organizzazione interno. Essi non sono accessibili in maniera trasparente dal disco fisso, ma sono

[4]Model View Controller: consiste in una modalità di interfacciamento che separa la rappresentazione dell'informazione dall'interazione con l'utente

[5]Relational Database Management System

gestiti ed organizzati esclusivamente dal *Content Repository*, che mantiene la loro organizzazione logica memorizzata in una base di dati.

3.2.2 Lo standard CMIS

Vista la grande somiglianza delle soluzioni ECM nelle funzioni supportate, il componente che adempie alle funzioni di *Content Repository* e stato definito più precisamente nei seguenti standard:

- CMIS (Content Management Interoperability Services)
- JCR (Java Content Repository / JSR-170/286)

Questi standards forniscono una specifica per la definizione e memorizzazione dei contenuti, per il loro recupero, per la modalità di versionamento e per la gestione dei permessi di accesso. CMIS, in particolare, è uno standard aperto, promosso da OASIS[6] che definisce un livello di astrazione per il controllo dei *repositories* aderenti allo standard utilizzando protocolli web. Esso definisce un modello dati che a sua volta definisce le entità gestite nel *repository* e specifica un insieme di servizi base che un'applicazione può usare per accedere e manipolare queste entità. Naturalmente questo modello dati non copre tutti i concetti che un *repository ECM* tipicamente supporta. Si tratta invece di un modello dati comune che specifica, per i file tipizzati e le cartelle, delle proprietà generiche che possono essere lette o impostate. Alcuni dei servizi descritti riguardano:

- il recupero dei documenti;
- il controllo degli accessi;
- il controllo del versionamento;
- la specifica di relazioni generiche.

[6]OASIS: Organization for the Advancement of Structured Information Standards, è un'associazione che promuove standards per il web.

Sono definiti poi due protocolli di comunicazione: uno che utilizza i principi dello stile architetturale REST[7] e uno che segue invece le specifiche di protocollo SOAP[8].

3.2.3 Alfresco come piattaforma di sviluppo

Il *Content Application Server* dispone di una serie di interfacce di programmazione che supportano molteplici linguaggi e protocolli di comunicazione. Alfresco, infatti, è conforme agli standards sopra citati e fa libero uso di linguaggi di scripting che possono essere utilizzati per supportare nuove funzionalità non previste dall'applicazione principale. Questa parte dell'architettura è nota come *Web Scripts* e può essere usata sia nei servizi riguardanti la modellazione e gestione dei dati che in quelli riguardanti la loro presentazione, ovvero dove non è richiesta una particolare complessità di programmazione. Per esempio, l'implementazione dello standard CMIS in Alfresco è stata supportata utilizzando i web scripts.

Le applicazioni client possono quindi comunicare con il *Content Application Server* e i suoi servizi attraverso numerosi protocolli supportati, come ad esempio l'accesso a livello di programmazione tramite servizi HTTP REST e SOAP, o l'accesso a livello applicativo come CIFS, FTP, WebDav, IMAP e Microsoft SharePoint.

Un'architettura di questo tipo consente di utilizzare Alfresco come una vera e propria piattaforma di sviluppo per applicazioni *content centric* e di impostare un modello di programmazione estremamente veloce, che consende di rispondere in maniera rapida ed efficace alle esigenze del cliente. Una caratteristica, quindi, in linea con la filosofia Agile adottata nel progetto.

---

[7] REpresentational State Transfer: modello di progettazione di servizi web nell'ambito dei sistemi distribuiti, tuttora predominante nel World Wide We.b

[8] Simple Object Access Protocol: specifiche di protocollo per lo scambio di informazione strutturata nell'implementazione di servizi web.

Capitolo 4

Installazione e configurazione del server

In questo capitolo si mostrano i diversi passaggi necessari all'installazione e alla configurazione di un sistema Alfresco. Sebbene sia disponibile una procedura automatica, un'installazione passo passo consente anzitutto di mantenere un miglior controllo sul sistema e inoltre permette di capire quali siano i servizi necessari al funzionamento del suddetto software, nonché di avere una migliore comprensione della sua architettura.

Come è già stato accennato, Alfresco viene eseguito come applicazione web e per questo necessita di un *Application Server*[1] che espone i servizi delle tecnologie JEE [2]. In realtà, per i nostri scopi, non è necessario l'utilizzo di un Application Server interamente compatibile con la specifica JEE e per questo è stato utilizzato **Apache Tomcat**, che ne implementa solamente una parte ed in compenso risulta molto più leggero.
Oltre al contenitore di applicazioni, Alfresco si appoggia anche ad un RDBMS

[1]Application Server: strato software che costituisce una piattaforma per l'esecuzione di una determinata tipologia di applicazioni, fornendo servizi relativi alla sicurezza, alla manipolazione dei dati, al bilanciamento del carico e molto altro.

[2]Java Enterprise Edition

[3], necessario per svolgere le funzioni esposte dal *Contet Repository*. Per tale sistema, la soluzione utilizzata è **PostgreSQL**.

Ci sono poi altri software che Alfresco utilizza per svolgere le sue funzioni e che sarebbe bene fossero presenti al momento dell'installazione. Essi sono:

- ImageMagick: utilizzato per manipolare le immagini per le anteprime;
- LibreOffice: utilizzato per convertire i documenti da un formato all'altro (per esempio da file di testo a pdf);
- FlashPlayer: utilizzato per caricare file multipli nel client web predefinito[4] e per vedere le anteprime Flash;
- SWF Tools: Alfresco utilizza l'utilità pdf2swf per le anteprime dei file PDF.

In seguito saranno presentati anche i comandi che permettono di ultimare i passaggi chiave per l'intallazione. Tali comandi fanno riferimento al sistema operativo utlizzato, ovvero la distribuzione linux **Ubuntu 12.04**, che si suppone sia dotata della piattaforma java SE Development Kit 6 o superiore.

4.1 PostgreSql

Questo software si compone di un processo server che gestisce le basi di dati, accetta connessioni da parte di applicazioni client e porta a termine le azioni sui dati per loro conto. Come per ogni *demone server*[5] accessibile dal mondo esterno, è consigliabile eseguire postgreSQL in un account utente separato. Questo utente deve solo essere proprietario dei dati che sono gestiti dal server,

[3]Relational Database Management System.

[4]Alfresco Web Client: l'applicazione web fornita con il server Alfresco che permette di accedere a quest'ultimo via browser.

[5]Programma eseguito in backgraund che mette a disposizione un insieme di servizi (e.g.: ssh, dhcp, postgres, etc.).

i quali non devono essere condivisi con altri demoni.

L'installazione attraverso i repository ufficiali è abbastanza diretta utilizzando l'utilità **apt-get** messa a disposizione da Ubuntu. La procedura si occupa anche della creazione dell'utente `'postgres'` nel sistema, rendendolo proprietario delle cartelle apposite che conterranno le basi di dati. Quello che rimane da fare è impostare una password per tale utente:

```
1  lorenzo@localhost:~\$sudo apt-get install
       postgresql-9.1 libpq-dev
2  lorenzo@localhost:~\$sudo passwd postgres
```

Il passo successivo è quello di creare una base di dati che possa essere utilizzata da Alfresco, ed un account utente per permettervi l'accesso. Per evitare confusione bisogna notare che gli utenti di postgreSQL e quelli del sistema operativo sono differenti, anche se in genere si cerca di fare in modo che essi coincidano. Conviene quindi creare prima l'utente Unix, successivamente creare l'utente `'alfresco'` in postgreSQL, tramite il comando `'create user'` e infine creare il database e assegnarlo all'utente, assieme ai privilegi per accedervi. Per effettuare queste ultime operazioni è possibile utilizzare l'applicazione front-end per postgreSQL chiamata **psql**, da cui è possibile eseguire dei comandi SQL:

Codice 4.1: PostgreSQL: creazione utente e DB

```
1  lorenzo@localhost:~\$ sudo useradd alfresco
2  lorenzo@localhost:~\$ sudo passwd alfresco
3  lorenzo@localhost:~\$ su postgres
4  postgres@localhost:/home/lorenzo\$ createuser
       alfresco
5  postgres@localhost:/home/lorenzo\$ psql
6  postgres=# CREATE DATABASE alfresco OWNER alfresco;
7  postgres=# GRANT ALL PRIVILEGES ON DATABASE
       alfresco TO alfresco;
8  postgres=# ALTER USER alfresco WITH PASSWORD '
       alfresco';
```

A questo punto è necessario permettere l'autenticazione del client (Alfresco) tramite l'utente `'alfresco'`. Questo tipo di permessi sono registrati in un file di configurazione solitamente presente all'indirizzo:
`'/etc/postgresql/9.1/main/pg_hba.conf'`
In particolare sarà necessario aggiungere a questo file la seguente riga:

```
1 # Database administrative login by Unix domain socket
2 # TYPE DATABASE USER ADDRESS METHOD
3 local   alfresco alfresco        ident
```

In pratica si permette all'utente `'alfresco'` di connettersi in locale al database `'alfresco'`. Questo ovviamente funziona quando il software Alfresco e postgreSQL sono installati sulla stessa macchina server.

4.2 Apache Tomcat

Per quando riguarda l'installazione di Apache Tomcat l'applicativo è anch'esso già presente nei repositories ufficiali di Ubuntu ed è quindi possibile procedere ad un'installazione immediata:

```
1 lorenzo@localhost:~\$ sudo apt-get install tomcat7
```

Da notare che Tomcat richiede che sia impostata la variabile d'ambiente `'JAVA_HOME'`, che deve puntare alla cartella d'installazione della piattaforma Java SE Development Kit. Tale variabile, può essere impostata all'interno del file:
`'/etc/default/tomcat6'`
dove è possibile anche calibrare i parametri della Java Virtual Machine, dato che la configurazione di default non è adatta per l'esecuzione di Alfresco che richiede un elevato quantitativo di memoria:

```
1 JAVA_HOME=/usr/lib/jvm/java-6-openjdk-i386
2 JAVA_OPTS="${JAVA_OPTS} -XX:+UseConcMarkSweepGC -XX:+
      CMSIncrementalMode -XX:MaxPermSize=512m -Xms128m -
      Xmx768m -Dalfresco.home=/opt/alfresco -Dcom.sun.
      management.jmxremote"
```

4.3 Alfresco

La versione community di Alfresco è open source e quindi liberamente scaricabile dal sito ufficiale. Una volta estratto il file zip bisogna spostare l'applicativo all'interno delle cartelle dedicate di tomcat, ricordandosi di caricare anche il driver `'jdbc'` che permette l'interazione con postgreSQL:

```
1 wget http://dl.alfresco.com/release/community/build-4003/
    alfresco-community-4.0.d.zip
2 unzip alfresco-community-4.0.d.zip
3 sudo cp -r web-server/shared /var/lib/tomcat6
4 sudo cp -r web-server/webapps /var/lib/tomcat6
5 sudo cp -r web-server/lib /var/lib/tomcat6/shared/lib
6 sudo cp -r bin /var/lib/tomcat6/bin
7 sudo cp -r licenses /var/lib/tomcat6/licenses
8 sudo cp -r README.txt /var/lib/tomcat6/README.txt
9 sudo cp postgresql-9.1-902.jdbc4.jar /var/lib/tomcat6/
    shared/lib/
```

È necessario poi configurare Tomcat in modo che sappia dove si trovano i pacchetti e le classi necessarie al funzionamento. Per fare questo bisogna modificare il file:
`'/var/lib/tomcat6/conf/catalina.properties'`
e impostare la seguente proprietà:

```
1 shared.loader=${catalina.home}/shared/classes,\${catalina.
    home}/shared/*.jar,/var/lib/tomcat6/shared/classes,/var/
    lib/tomcat6/shared/*.jar,/var/lib/tomcat6/shared/lib/*.
    jar
```

A questo punto è possibile passare alla configurazione delle proprietà di Alfresco: si parte da un file di configurazione di esempio presente nel pacchetto iniziale, che si trova in:
`'/var/lib/tomcat6/shared/classes/alfresco-global.properties.sample'`
che va rinominato per essere reso attivo. Allo stesso modo si può attivare la configurazione per l'applicazione client web fornita di default che si chiama **Share**[6], rinominando l'apposito file:

[6]Share ed Alfresco Web Client sono due applicazioni client per Alfresco, fornite di default e accessibili via browser. La prima è semplicemente più moderna.

```
1 mv /var/lib/tomcat6/shared/classes/alfresco-global.
    properties.sample /var/lib/tomcat6/shared/classes/
    alfresco-global.properties
2 mv /var/lib/tomcat6/shared/classes/alfresco/web-extension/
    share-config-custom.xml.sample /var/lib/tomcat6/shared/
    classes/alfresco/web-extension/share-config-custom.xml
```

Prima di procedere all'impostazione delle configurazioni nel suddetto file è necessario creare una cartella dove Alfresco potrà memorizzare tutti i contenuti e i suoi file necessari al funzionamento e assicurarsi che l'utente che esegue il server Tomcat (di default è l'utente `'tomcat6'`) abbia i privilegi necessari per l'accesso e la scrittura:

```
1 mkdir -p /srv/www/alfresco/alf_data
2 sudo chown -R tomcat6:tomcat6 /srv/www/alfresco/alf_data /
    var/lib/tomcat6
```

Ora è possibile impostare il file di configurazione:
`'/var/lib/tomcat6/shared/classes/alfresco-global.properties'`
con i parametri necessari per l'accesso al database postgreSQL e al server Alfresco:

Codice 4.2: Alfresco: configurazione proprietà globali

```
1  ### database connection properties ###
2  db.name=alfresco
3  db.username=alfresco
4  db.password=alfresco
5  db.host=localhost
6  db.port=5432
7  db.driver=org.postgresql.Driver
8  db.url=jdbc:postgresql://${db.host}:${db.port}/${db.name}
9
10 ### Common Alfresco Properties ###
11 dir.root=/srv/www/alfresco/alf_data
12 alfresco.context=alfresco
13 alfresco.host=127.0.0.1
14 alfresco.port=8080
15 alfresco.protocol=http
16 share.context=share
17 share.host=127.0.0.1
18 share.port=8080
19 share.protocol=http
20
21 ### Other softwares ###
22 ooo.exe=/usr/lib/libreoffice/program/soffice
23 ooo.enabled=true
24 jodconverter.officeHome=/usr/lib/libreoffice
25 jodconverter.portNumbers=8101
26 jodconverter.enabled=true
27 img.root=/usr/lib/ImageMagick-6.6.9
28 img.exe=/usr/bin/convert
29 swf.exe=/usr/bin/pdf2swf
```

A questo punto Alfresco è pronto per essere avviato, basta dare il comando per far partire Tomcat, che provvederà a sua volta all'avvio di Alfresco:

```
1 sudo service tomcat6 start
```

La prima cosa che si nota nel sistema operativo è l'incremento dell'utilizzo della memoria ram, la cui quantità dev'essere adeguata. Alfresco impiega diversi minuti per completare la procedura di avvio ed è quindi bene aspettare qualche momento prima di accedervi per controllarne il funzionamento. Per effettuare quest'ultima operazione è possibile accedere ad Alfresco tramite il client web fornito con il software: per farlo bisogna aprire il browser e digitare l'indirizzo:
`http://localhost:8080/alfresco`
se tutto è andato a buon fine comparirà una schermata di login da cui è possibile accedere all'interfaccia di gestione web tramite nomeutente e password di default (`admin, admin`).

In caso di errori è possibile controllare cosa è andato storto nei file di logs di Apache Tomcat presenti all'indirizzo:
`/var/lib/tomcat6/logs`
In particolare il file `'catalina.out'` contiene il log di tutte le operazioni svolte dal server sia in fase di avvio che di esecuzione, oltre che la stampa degli errori e delle eccezioni java lanciate dal software in esecuzione. Questo file tornerà motlo utile in seguito sia per comprendere eventuali errori di modellazione che per eseguire attività di debug.

Capitolo 5

Modellazione dei contenuti

Lo scopo della modellazione dei contenuti è quello di specificare come le entità (nodi) sono memorizzate e vincolate nel *Contet Repository*. In Alfresco, infatti, tutta la modellazione ruota attorno al concetto di nodo: i nodi possono rappresentare qualsiasi cosa memorizzata nel *repository*, come ad esempio cartelle, documenti, frammenti xml, scripts, etc.

Per memorizzare le entità e i loro metadati, il motore del *repository* utilizza una struttura dati che consiste in un albero di nodi, ognuno dei quali racchiude un'insieme di proprietà che possono essere un qualsiasi tipo di dato, a valore singolo o multiplo. Ogni nodo, quindi, ha un padre (eccetto la radice) e può a sua volta contenere uno o più figli. Questa struttura dati deve soddisfare i seguenti vincoli:

- Ogni nodo deve essere di un determinato genere e trasportare un insieme numerato di proprietà;
- Ogni proprietà deve essere di un tipo di dato stabilito;
- Il valore di una proprietà deve rientrare nell'insieme definito dei valori;
- Ogni nodo è sempre collegato ad altri nodi.

Questi vincoli permettono la modellazione delle entità all'interno del sistema ECM. Come esempio si può pensare ai concetti di cartella e documento:

fisicamente essi sono memorizzati in una struttura di dati (in particolare nodi che costituiscono un albero) e tramite la modellazione è appunto possibile aggiungere il significato semantico che conosciamo.

5.1 Livelli di astrazione

Nella modellazione dei metadati esistono diversi livelli di astrazione: ad esempio è possibile definire un metamodello, ossia un modello che descrive a sua volta un modello di dati. Per fare chiarezza e contestualizzare l'ambito di modellazione, Alfresco identifica quattro livelli (fig. 5.1):

- M0: è il primo livello, descrive i nodi, le proprietà e le relazioni mantenuti nel *Content Repository*;
- M1: a questo livello i modelli descrivono le proprietà e le relazioni che vincolano i nodi nel livello M0; all'interno del *repository* possono essere registrati molti modelli di contenuto, che permettono di strutturare i contenuti in maniera personalizzata;
- M2: è il livello del meta-modello che descrive i modelli di contenuto. Esistono anche modelli standard a questo livello di astrazione, come il modello dati CMIS;
- M3: si tratta del livello più astratto, che non sarà trattato in questa sede, e supporta la conversione dei modelli di contenuto espressi in un meta-modello in un altro meta-modello.

In pratica è necessario imparare la struttura del meta-modello (M2) per poter definire uno o più modelli di contenuto (M1) che vincolano la struttura dei nodi (M0) memorizzati nel *repository*. Prima però è necessario introdurre il linguaggio XML, ovvero il linguaggio che permette la definizione dei modelli di contenuto.

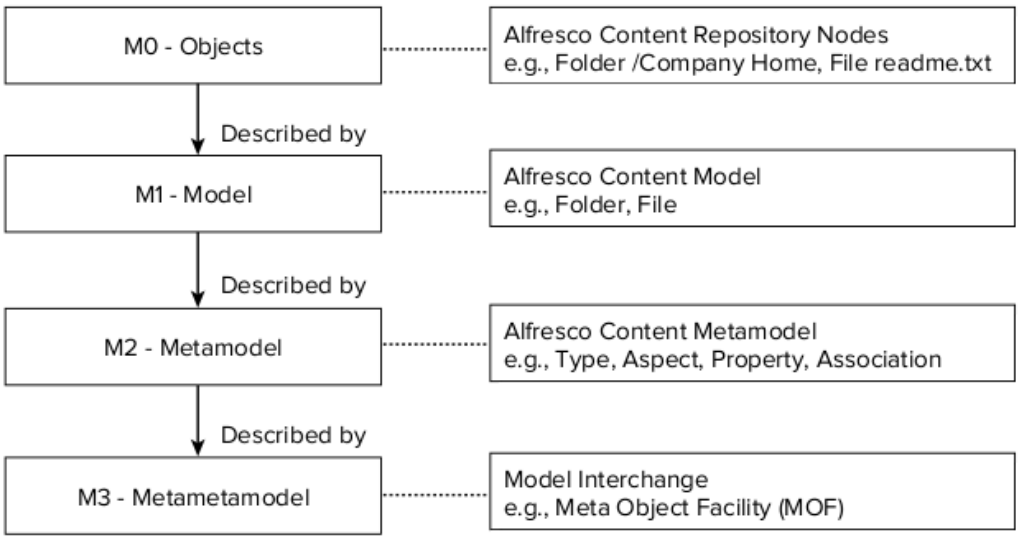

Figura 5.1: Modellazione dei contenuti: livelli di astrazione

5.2 Il linguaggio XML

L'XML (eXtensible Markup Language) è un linguaggio marcatore, all'apparenza simile al linguaggio HTML, ma con la fondamentale differenza che, invece di descrivere la formattazione di un documento, cerca di descrivere la semantica del testo. Si tratta infatti di un insieme di regole sintattiche che hanno lo scopo di modellare la struttura di dati e documenti dal punto di vista semantico utilizzando dei marcatori chiamati tag. Questi tag non rientrano in un'insieme ben definito, come nell'HTML, ma possono essere definiti a seconda del linguaggio che si vuole descrivere (eXtensible). L'XML, infatti, viene presentato come un metalinguaggio con cui è possibile definire nuovi linguaggi.

Un documento XML è intrinsecamente caratterizzato da una struttura gerarchica: esso è descritto, a livello logico, da componenti denominati elementi. Ciascuno di essi ha il compito di trasportare un certo significato semantico e può contenere altri elementi (sottoelementi) o del testo. Gli elementi possono avere associate altre informazioni, che ne descrivono le proprietà, chiamate attributi. L'organizzazione degli elementi segue un ordine gerarchico che prevede un elemento principale, chiamato *root element* o semplicemente radice,

la quale contiene l'insieme degli altri elementi del documento. È possibile rappresentare graficamente la struttura del documento tramite un albero, noto come *document tree*.

Fisicamente, un documento XML è costituito da un file di testo composto da marcatori (i tag), usati per rappresentare gli elementi e gli attributi. Un documento XML che non contiene errori è detto ben formato. Se un documento è ben formato e in più rispetta i requisiti logici della grammatica che lo definisce (la struttura logica) è detto valido. Per essere ben formato un documento XML deve quindi rispettare le seguenti regole:

- deve contenere un unico elemento di massimo livello (radice) che contenga tutti gli altri elementi del documento. Le sole parti di XML che possono stare all'esterno di questo elemento sono i commenti e le direttive di elaborazione;
- ogni elemento deve avere un tag di chiusura o, se si si tratta di un elemento vuoto, è possibile usare la forma abbreviata (`<nometag />`);
- gli elementi devono essere opportunamente nidificati, cioè i tag di chiusura devono seguire l'ordine inverso dei rispettivi tag di apertura;
- XML fa distinzione tra maiuscole e minuscole, per cui i nomi dei tag e degli attributi devono coincidere nei tag di apertura e chiusura anche in relazione a questo aspetto;
- i valori degli attributi devono sempre essere racchiusi tra singoli o doppi apici.

Attraverso la definizione del DTD (Document Type Definition), o meglio ancora dell'XML Schema (che è a sua volta un XML), è possibile creare la grammatica che definisce il nuovo linguaggio, ovvero la specifica della struttura logica del documento XML (che dà il significato semantico ai tag), e permette di verificarne la correttezza. In particolare, tali linguaggi schema definiscono l'insieme degli elementi del documento XML, le relazioni gerarchiche

tra gli elementi, l'ordine di apparizione nel documento XML e quali elementi e quali attributi sono opzionali o meno.

Un altro concetto importante che riguarda la definizione della struttura di un XML è quello di namespace: in sostanza, un namespace associato ad un documento XML rappresenta l'insieme dei nomi degli elementi e degli attributi del documento. Al namespace viene associato un identificatore che lo rappresenta, che solitamente consiste in una stringa espressa in forma di URI per garantirne l'univocità. L'identificatore viene utilizzato poi in altri documenti XML per stabilire l'ambito di provenienza di un qualsiasi tag, ovvero a quale insieme dei nomi ci si riferisce quando si specifica un tag. Non è infrequente, infatti, avere a che fare con documenti XML ibridi, ovvero che utilizzano più di una grammatica. In questo caso ci potrebbero essere dei tag e degli attributi che, pur proveniendo da due grammatiche diverse, hanno lo stesso nome. Un namespace associato alle due grammatiche permette di identificare la provenienza di ogni tag semplicemente anteponendo il prefisso del namespace (una forma abbreviata per identificarlo) ai nomi dei tag e quindi distinguendoli da eventuali omonimie. Un XML Schema definisce implicitamente un namespace degli elementi e degli attributi che possono essere usati in un documento XML.

5.3 Il meta modello di Alfresco

Il meta modello di Alfresco, usato per la definizione di modelli di contenuto è formalmente descritto nell' XML Schema presente all'indirizzo:
`http://svn.alfresco.com/repos/alfresco-open-mirror/alfresco/HEAD/root/projects/repository/config/alfresco/model/modelSchema.xsd`

Il meta-modello supporta due costrutti primari: il `content type` e il `content aspect`. Il `content type` è sostanzialmente lo stesso concetto di classe nel paradigma di programmazione ad oggetti. Esso fornisce l'abilità di descrivere una specifica struttura di un contenuto, tramite la definizione di un'insieme di proprietà (i metadati), dei vincoli che esse devono rispettare e delle relazioni con gli altri contenuti. Ogni proprietà deve essere di un ben definito tipo di

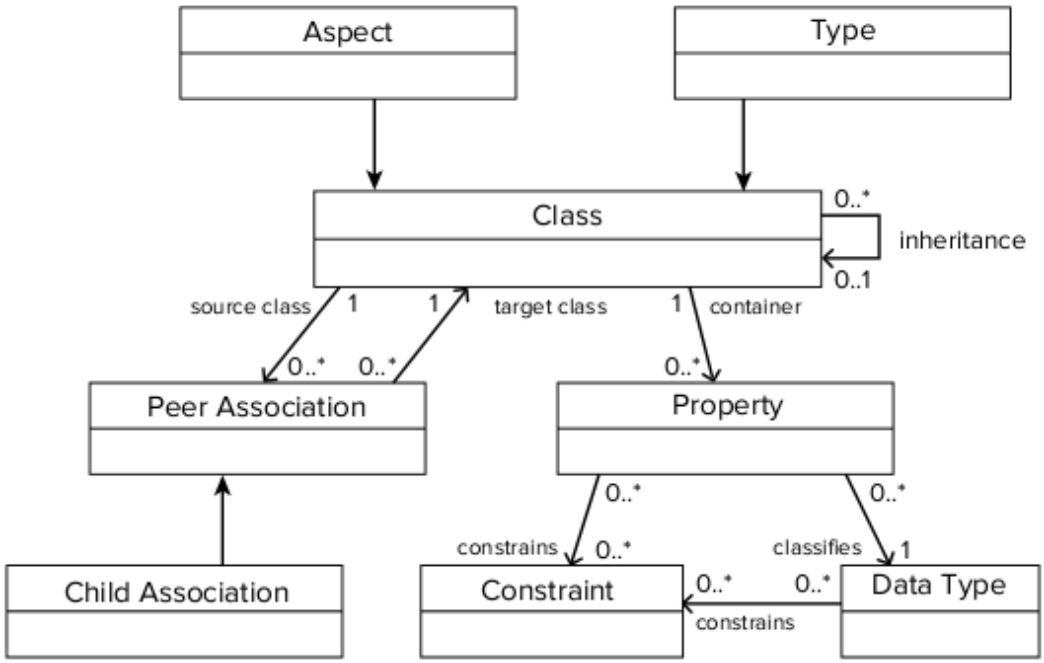

Figura 5.2: Il meta-modello Alfresco per la definizione di modelli di contenuto

dato, mentre i vincoli (*constraints*) servono per restringere il campo di valori che le proprietà possono assumere. Il costrutto permette inoltre l'ereditarietà, ovvero permette di ereditare la definizione di un `content type` genitore, ovvero di tutte le sue proprietà e relazioni, estendendole poi con le proprie.

Il concetto di `aspect` è molto simile a quello di `type`: esso incorpora praticamente le stesse funzionalità, come le proprietà, le relazioni e l'ereditarietà delle definizioni di `aspects` genitori. Tuttavia si distingue per un'importante caratteristica: esso permette di definire delle proprietà comuni che si applicano a contenuti di diverso tipo. Infatti, un nodo del *repository* può essere di un solo `content type`, mentre può essere collegato a più `content aspect` e nodi di tipo diverso possono essere collegati ad uno stesso costrutto `content aspect` che definisce una struttura di dati comune che si applica a loro.

Il *Content Repository* comprende diversi modelli preconfezionati per la specifica di `content types` chiave che ci si aspetta da un sistema ECM. Il modello

base su cui sono costruiti gli altri modelli è il *Data Dictionary Model*, che fornisce le definizioni per i tipi di dato fondamentali, come `d:text` e `d:boolean`. Il suo namespace è:
`www.alfresco.org/model/dictionary/1.0`
ed ha prefisso `'d'`.
Esiste poi un modello per il dominio ECM che fornisce le definizioni per i tipi definiti negli standard CMIS e JCR, come `cm:folder`, `cm:content` e `cm:versionable`. Il suo namespace è:
`www.alfresco.org/model/content/1.0`
ed ha prefisso `'cm'`.

5.4 Modellazione del documento Bonifico

Una volta compreso il metamodello è possibile creare dei modelli di contenuto personalizzati che permettono di definire la struttura dei metadati associata ai contenuti che si intende memorizzare nel sistema. Un modello di contenuto è costituito da una collezione di types, di aspects, di relazioni e di vincoli che descrivono una realtà, ed è descritto da un XML identificato dal namespace associato. Il namespace è composto da un URI e da un prefisso (un nome abbreviato).

Vedremo ora come è stata modellata la struttura di un semplice documento costituito da una distinta di bonifici (documento che contiene una o più disposizioni di bonifico effettuate da un cliente), ovvero come è stato costruito il modello di questo contenuto.

La definizione di un nuovo modello comincia con l'impostazione dell'intestazione, dove si indica il nome del modello e alcuni dati opzionali per descriverlo:

Codice 5.1: Modellazione del documento “distinta di bonifici”

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Definition of new Model -->
3 <model name="mbon:mbonifico" xmlns="http://www.alfresco.org
    /model/dictionary/1.0">
4   <!-- Optional meta-data about the model -->
5   <description>Modello di una distinta di bonifici</
      description>
6   <author>Lorenzo Caldera</author>
7   <version>1.1</version>
8   ...
```

Si devono poi importare due modelli preconfezionati, definiti da Alfresco, che permettono la strutturazione del nuovo contenuto, e si deve associare al modello stesso un namespace:

```
1  <!-- gli imports permettono di riferirsi alle definizioni
       presenti in altri modelli -->
2  <imports>
3    <import uri="http://www.alfresco.org/model/dictionary/1.0
       " prefix="d" />
4    <import uri="http://www.alfresco.org/model/content/1.0"
       prefix="cm" />
5  </imports>
6
7  <!-- introduzione del nuovo namespace definito dal modello
     -->
8  <namespaces>
9    <namespace uri="http://www.e-projectsrl.it/lnz/mbonifico
       /1.1" prefix="mbon" />
10 </namespaces>
11 ...
```

Si possono poi specificare a parte dei vincoli, che permettono di restringere il campo dei valori di una certa proprietà:

```
1  <constraints>
2    <constraint name="mbon:statiAccettati" type="LIST">
3      <parameter name="allowedValues">
4            <list>
5              <value>ordinato</value>
6                  <value>accettato</value>
7                  <value>rifiutato</value>
8                  <value>eseguito</value>
9            </list>
10         </parameter>
11   </constraint>
12 </constraints>
13 ...
```

Ora è possibile definire i `types` del modello, in questo caso un singolo oggetto che rappresenta il documento "distinta di bonifici":

```
1  <!-- distinta -->
2  <types>
3  <type name="mbon:bonifico">
4    <title>Distinta di bonifici</title>
5    <parent>cm:content</parent>
6    <properties>
7      <property name="mbon:codice_banca">
8        <type>d:text</type>
9        <mandatory>true</mandatory>
10     </property>
11     <property name="mbon:codice_filiale">
12       <type>d:text</type>
13       <mandatory>true</mandatory>
14     </property>
15     <property name="mbon:id_postazione">
16       <type>d:int</type>
17       <mandatory>true</mandatory>
18     </property>
19     <property name="mbon:userid">
20       <type>d:text</type>
21       <mandatory>true</mandatory>
22     </property>
23     <property name="mbon:ndg_cliente">
24       <type>d:text</type>
25     </property>
26 ...
```

```
1     <property name="mbon:nome">
2       <type>d:text</type>
3     </property>
4     <property name="mbon:cognome">
5       <type>d:text</type>
6     </property>
7     <property name="mbon:codice_fiscale">
8       <type>d:text</type>
9     </property>
10    <property name="mbon:stato">
11      <type>d:text</type>
12      <constraints>
13        <constraint ref="mbon:statiAccettati" />
14      </constraints>
15    </property>
16    <property name="mbon:data_invio">
17      <type>d:date</type>
18      <mandatory>true</mandatory>
19    </property>
20    <property name="mbon:numero_pagine">
21      <type>d:int</type>
22    </property>
23    <property name="mbon:numero_disposizioni">
24      <type>d:int</type>
25      <mandatory>true</mandatory>
26    </property>
27  </properties>
28 <associations>
29 </associations>
30 </type>
31 </types>
32 </model>
```

5.5 Caricamento del modello in Alfresco

Per registrare un modello all'interno del *Content Repository* ci sono due strade: è possibile effettuare la registrazione tramite i file di configurazione di Alfresco in modo che il modello venga caricato all'avvio del sistema, oppure utilizzare **Alfresco Web Client**. Sarà presentata solo la seconda modalità, in quanto più flessibile perché permette di caricare dei modelli anche a sistema avviato, senza la necessità di riavviare il server ogni volta (la fase di avvio è molto lenta). Bisogna quindi entrare in Alfresco tramite il browser all'indirizzo:
`http://localhost:8080/alfresco`

Si deve poi effettuare il login con l'utente di amministratore e navigare nella seguente sezione:
`Company Home > Data Dictionary > Models`

A questo punto si preme il pulsante `Add Content` e si carica il modello costruito, facendo attenzione a selezionare la casella `Model Active`, che permette di rendere attivo il modello all'interno di Alfresco (ovvero fare il *deploy del modello*).

Se non si nota la presenza di errori significa che il modello è corretto ed è stato caricato nel sistema. Per verificarlo è possibile accedere alla console di amministrazione all'indirizzo:
`http://localhost:8080/alfresco/faces/jsp/admin/repoadmin-console.jsp`
e dare il comando:
`show models`
Si dovrebbe notare un output del seguente tipo:

```
1 IsLoaded: Y , RepoVersion: null , RepoName: mbonifico.xml ,
    ModelQName: {http://www.e-projectsrl.it/lnz/mbonifico
    /1.1}mbonifico , Description: Modello di un documento
    bonifico , Author: Lorenzo Caldera , Published: null ,
    Version: 1.1
```

La modellazione del documento è quindi completata. Ora serve un meccanismo che permetta di inserire i documenti di questo tipo nel *Content Repository*, assieme ai metadati appena descritti.

Capitolo 6

Web scripts

In questa sezione verrà mostrato come sia possibile utilizzare Alfresco come piattaforma per l'esposizione di servizi web. In particolare verrà introdotta la sua funzionalità nota come *Web Scripts*, che permette di implementare in maniera rapida ed efficiente dei servizi web personalizzati di tipo REST.
Si procede quindi ad introdurre le caratteristiche di questo stile architetturale, per poi mostrare il meccanismo di funzionamento dei web scripts e come sia stato implementato il servizio di caricamento, nel *Content Repository*, dei documenti del tipo "distinta di bonifici", la cui modellazione è stata presentata nel capitolo precedente.

6.1 I Servizi Web

Il W3C [1], definisce un servizio web come:

> "un sistema software progettato per supportare l'interazione tra macchine sopra una rete".[10]

I servizi web nascono dalla visione del web come un grande sistema di elaborazione distribuito. In maniera più concreta è possibile identificare un servizio web come l'interfaccia, per un determinato servizio, messa a disposizione da un programma utilizzando le tecnologie web (internet). In sostanza, lo scopo

[1] World Wide Web Consortium

dei servizi web è quello di consentire l'interazione trasparente tra applicazioni sviluppate con linguaggi di programmazione diversi e che sono eseguite su sistemi operativi eterogenei. Sempre il W3C identifica due principali classi di servizi web:

> "i servizi *REST-compliant*, in cui lo scopo primario è quello di manipolare le rappresentazioni XML delle risorse web utilizzando un insieme di operazioni senza stato, e i servizi arbitrari, in cui il servizio può essere esposto da un insieme arbitrario di operazioni." [10]

Come introdotto, nel seguito l'attenzione sarà posta sui servizi di tipo REST.

6.2 Lo stile architetturale REST

REST, acronimo di REpresentational State Transfert, è nella sua definizione più generale uno stile archietturale per sistemi distribuiti [5]. Si tratta di un insieme di principi che forniscono delle linee guida per la progettazione di tali sistemi, e quindi non sono strettamente legati al web, ma si applicano anche al web per l'implementazione di servizi altamente efficienti e scalabili. Un sistema pienamente conforme a tali principi è detto **RESTful**. L'architettura REST prevede le applicazioni clients e servers. Tipicamente, un client effettua la richiesta al server, che la processa ed invia l'appropriata risposta. Una caratteristica importante è che le richieste e le risposte sono caratterizzate dal trasferimento, non delle risorse stesse, ma da una loro rappresentazione, che può assumere diversi formati, come HTML, XML, JSON, etc.
I principi da seguire nella progettazione del sistema sono:

- identificazione delle risorse;
- collegamenti tra risorse;
- utilizzo di metodi standard;
- risorse con rappresentazioni multiple;

- comunicazione senza stato.

Il concetto di risorsa è il fondamento dei servizi REST. Con tale termine si vuole indicare un qualsiasi elemento che può essere oggetto di elaborazione, come potrebbe essere un post all'interno di un blog, un archivio binario, oppure l'ordine su un sito e-commerce, un servizio, o ancora un documento. Il primo principio indica che ciascuna risorsa debba essere identificata univocamente, per cui devono essere identificate tutte le risorse che l'applicazione deve fornire. Nel web, le risorse sono individuate tramite lo standard URI[2].

Il principio del collegamento tra risorse indica che le risorse dovrebbero essere messe in relazione tramite link ipertestuali. Questo principio è anche chiamato HATEOAS, dall'acronimo di *Hypermedia As The Engine Of Application State*: si basa sul concetto di link e vuole porre l'attenzione sulle modalità di gestione dello stato dell'applicazione. In sostanza, tutto quello che un client deve sapere su una risorsa e sulle risorse ad essa correlate deve essere contenuto nella sua rappresentazione o deve essere accessibile tramite collegamenti ipertestuali. In questo modo, il server, fornendo un insieme di collegamenti al client, lo abilita ad effettuare un cambiamento di stato nell'applicazione, che avviene semplicemente seguendo questi collegamenti.

Il principio di utilizzo di metodi standard indica che devono essere uniformate le operazioni sulle risorse. Il vantaggio che ne deriva è che, dato l'identificatore di una risorsa, un client sa già quali sono le interfacce supportate per accedervi. Nel caso del web vengono utilizzati i metodi HTTP: GET, POST, PUT, DELETE e OPTIONS. Quello che il principio stabilisce, quindi, è una mappatura tra le tipiche operazioni CRUD (creazione, lettura, modifica ed eliminazione di una risorsa) e i metodi HTTP. Da notare, però, che non basta l'utilizzo di tali metodi: REST indica che debbano essere usati con il significato semantico per cui sono stati pensati, per cui, GET dovrebbe essere utilizzato per accedere alla rappresentazione di una risorsa, PUT per l'aggior-

[2]Uniform Resource Identifier.

namento, POST per la creazione e DELETE per l'eliminazione.

Come abbiamo detto le risorse sono concettualmente separate dalle loro rappresentazioni, che sono restituite ai clients o inviate ai servers. I principi REST non pongono nessun vincolo sulle modalità di rappresentazione. Questo perché non è sempre possibile avere a disposizione dei formati standard per rappresentare le risorse, anche se certamente sarebbe doveroso utilizzarli qualora essi fossero disponibili. Si possono tuttavia immaginare i vantaggi nel creare dei formati standard anche solo nell'ambito di un piccolo ecosistema, come può essere quello di un azienda e dei suoi collaboratori.
L'avere poi a disposizione diversi tipi di rappresentazione per una risorsa comporta un indubbio beneficio: per esempio, la presenza sia di un formato XML che HTML per una risorsa permette di consumarla non solo dall'applicazione client costruita per interagire col server, ma anche da qualsiasi browser web. Un'altro modo di sfruttare questa caratteristica è la possibilità di aggiungere facilmente, ad un servizio web fruibile attraverso un interfaccia utente, una web API, ovvero rendere consumabile il servizio anche da un'applicazione client. L'idea che sta alla base è quella che qualsiasi servizio utilizzabile tramite UI sia utilizzabile anche tramite una API.

Nel web, il tipo di rappresentazione inviata viene specificato direttamente nell'intestazione della risposta tramite i tipi MIME[3], così come avviene nella classica comunicazione tra web server e browser. Un client a sua volta ha la possibilità di richiedere una risorsa in uno specifico formato indicandolo nell'intestazione della richiesta.

Il principio della comunicazione *stateless* è ben noto a chi lavora con il web. Questa è infatti una delle caratteristiche principali del protocollo HTTP, cioè ciascuna richiesta non ha alcuna relazione con le richieste precedenti e successive. Lo stesso principio si applica ad un servizio web RESTful, cioè le interazioni tra client e server devono essere senza stato.

[3]Multipurpose Internet Mail Extensions.

È importante sottolineare che sebbene REST preveda la comunicazione senza stato, non vuol dire che un'applicazione non debba avere stato. La responsabilità della gestione dello stato dell'applicazione non deve essere conferita al server, ma rientra nei compiti del client. La principale ragione di questa scelta è la scalabilità: mantenere lo stato di una sessione ha un costo in termini di risorse sul server e all'aumentare del numero di client tale costo può diventare insostenibile.

6.3 Alfresco Web Scripts

Con i *web scripts*, Alfresco cerca di catturare la logica vincente del web: tutti i contenuti del *repository*, infatti, possono essere visti come una collezione di documenti e risorse collegati, ed è quindi possibile applicare le tecnologie del web, come HTTP, URI e HTML, per gestirli e manipolarli. I *web scripts* permettono di implementare delle RESTful API personalizzate in maniera semplice e rapida, senza la necessità di utilizzare alcun tipo di compilatore, né di effettuare complesse installazioni o riavvii del server. Si utilizza solo un editor di testo e il web client di Alfresco: ne deriva una notevole facilità di sviluppo nella progettazione di API personalizzate per la gestione dei contenuti, accessibili tramite HTTP e identificate tramite un URI.

L'accesso di tipo RESTful ai contenuti, quindi, consente di implementare un controllo su di essi e, al tempo stesso, fornisce un accesso uniforme da poter sfruttare in diverse maniere, come l'utilizzo tramite applicazioni personalizzate allo scopo di manipolare i dati, oppure tramite un browser web per la loro presentazione. Queste caratteristiche rendono i *web scripts* la prima scelta per l'integrazione di un server Alfresco con dei client personalizzati per la gestione dei contenuti.

6.4 Il pattern MVC

Il framework per i Web Scripts segue il modello MVC (Model View Controller) [9], con il quale si indica un paradigma di progettazione che separa completamente la logica di controllo dalla presentazione dell'informazione. Per fare questo distingue i tre componenti:

- Model: rappresenta la conoscenza ed in genere si tratta di un oggetto, o una struttura di oggetti che fa da collegamento tra il Controller e il View, che sono strettamente associati al Model.
- View: si occupa di fornire una rappresentazione visuale del Model. In genere questa rappresentazione comprende alcuni attributi del Model, mentre ne nasconde degli altri secondo le necessità. Opera quindi come un filtro di presentazione. Possono esserci molti View e di formati diversi.
- Controller: questo componente contiene tutta la logica di controllo ed è il collegamento tra l'utente ed il sistema. In genere riceve i dati da un View, esegue l'elaborazione ed eventualmente popola il Model e restituisce delle informazioni ad un altro View.

Uno dei benefici del modello MVC è che si può avere una logica di controllo associata a tutti i View che si vuole perché la logica di controllo e la logica di visualizzazione sono separate.

Un webscript di Alfresco comprende quindi i seguenti componenti (6.1):

- Un documento descrittore che descrive l'URI e il metodo HTTP utilizzati per chiamare il web script, assieme ad altri parametri opzionali, come il tipo di autenticazione e le proprietà di transazione;

- Uno script di controllo che può effettuare un dialogo con il *Content Repository* al fine di creare un insieme di oggetti (il Model) necessari a costruire la risposta (per i metodi PUT, POST e DELETE si può avere un aggiornamento del *repository*). Tale script ha accesso alla stringa URI di chiamata, ai servizi di Alfresco e ovviamente al *Content Repository*;

- Uno o più *templates* [4] di risposta **FreeMarker**[5] (Views). Questi costruiscono l'output nel formato specificato (HTML, Atom, XML, RSS, JSON). La risposta HTTP è generata via uno di questi template, che viene scelto in base alla richiesta HTTP. Il *template* ha accesso alla stringa di chiamata URI, al *Content Repository* e agli oggetti costruiti dallo script di controllo;

[4]template: un documento che descrive una struttura di base per la presentazione dei dati, che devono essere poi inseriti da un motore di templates.

[5]FreeMarker: motore di templates scritto in java che si focalizza sull'architettura MVC. Il compito di FreeMarker è quello elaborare un template inserendovi i dati e produrre in output un documento web.

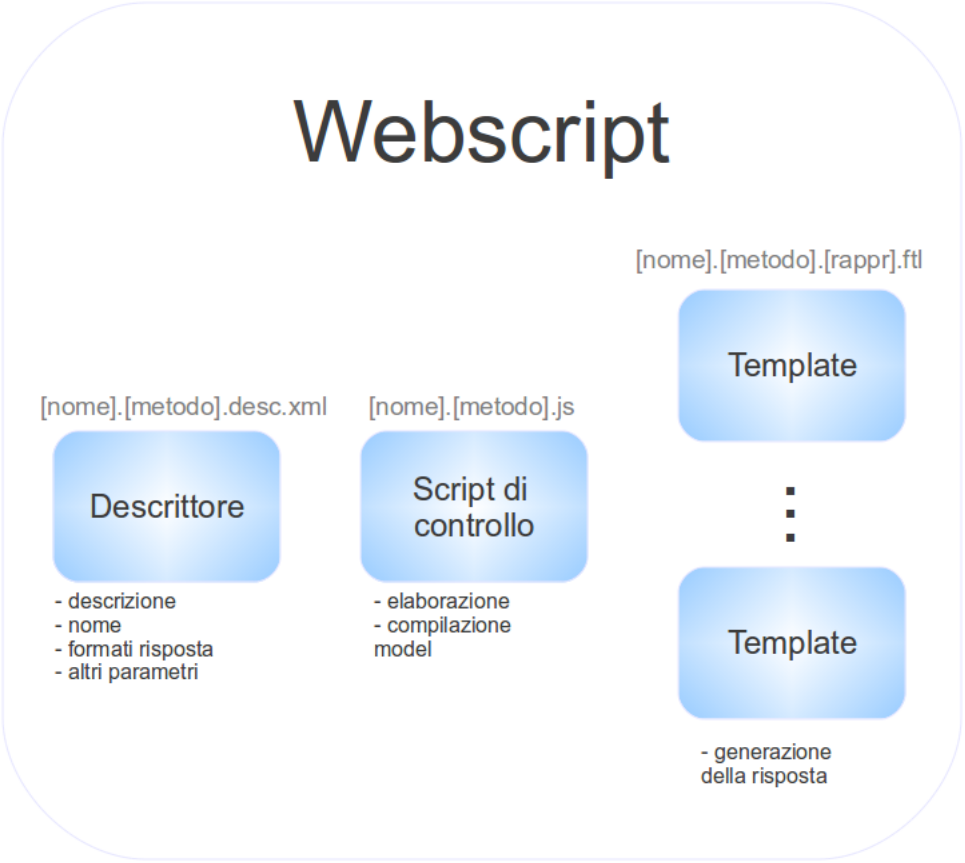

Figura 6.1: Webscript: componenti

6.5 Il web script di acquisizione

Per implementare un web script, quindi, sono necessari almeno tre file: un descrittore, uno script e un template di risposta. Il nome di questi file deve seguire uno schema ben definito, in modo da permettere ad Alfresco di riconoscere la loro funzione. Vedremo ora come è stato costruito il web script di acquisizione dei documenti del tipo "distinta di bonifici".

Il descrittore è costituito da un file XML dove viene specificato il nome del webscript, l'indirizzo per consumarlo e il formato predefinito per la ri-

sposta, più alcuni parametri di configurazione. In questo caso il nome è `'up-bonifico-tiff'`, volendo specificare che si tratta di un'operazione di upload di un documento di bonifici in formato tiff. Il nome del file descrittore invece è `'up-bonifico-tiff.post.desc.xml'`, dove il suffisso `'post.desc.xml'` indica che questo webscript si applica al metodo POST e che questo file costituisce il suo descrittore.

Codice 6.1: Webscript: descrittore

```
1  <webscript>
2    <shortname>up-bonifico-tiff</shortname>
3    <description> <![CDATA[
4      acquisisce una distinta di bonifici in formato tiff
           multipagina
5      ]]>
6    </description>
7    <url>/up-bonifico-tiff</url>
8    <format default="json"/>
9    <authentication>user</authentication>
10   <transaction>required</transaction>
11   <lifecycle>draft_public_api</lifecycle>
12   <family>myWS</family>
13 </webscript>
```

Questo descrittore è accompagnato da uno script che gestisce la logica di controllo e da due template per la generazione delle risposte. Questi ultimi due sono il file:

`up-bonifico-tiff.post.json.ftl`

ed il file:

`up-bonifico-tiff.post.json.400.ftl`

Il primo viene utilizzato per indicare il buon esito dell'inserimento: esso genera una risposta in formato JSON [6] in cui fornisce un riassunto dei parametri d'in-

[6]JSON: acronimo di JavaScript Object Notation, è un formato adatto per lo scambio dei dati in applicazioni client-server.

serimento, quali il riferimento del nodo appena creato nel *Content Repository* (il documento) e i valori dei metadati associati ad esso. Il secondo, invece, viene utilizzato per comunicare una condizione di errore.

Codice 6.2: Webscript: template di risposta in caso di successo

```
1  {
2  "name" : "${jsonUtils.encodeJSONString(node.name)}",
3  "node-uuid" : "${jsonUtils.encodeJSONString(node.properties
       ["{http://www.alfresco.org/model/system/1.0}node-uuid"])
       }",
4  "nodeRef" : "${jsonUtils.encodeJSONString(node.nodeRef)}",
5  "displayPath" : "${jsonUtils.encodeJSONString(node.
       displayPath)}",
6  "downloadUrl" : "${jsonUtils.encodeJSONString(node.
       downloadUrl)}",
7  "codice_banca" : "${jsonUtils.encodeJSONString(node.
       properties["{http://www.e-projectsrl.it/lnz/mbonifico
       /1.1}codice_banca"])}",
8  "codice_filiale" : "${jsonUtils.encodeJSONString(node.
       properties["{http://www.e-projectsrl.it/lnz/mbonifico
       /1.1}codice_filiale"])}",
9  "id_postazione" : "${jsonUtils.encodeJSONString(node.
       properties["{http://www.e-projectsrl.it/lnz/mbonifico
       /1.1}id_postazione"])}",
10 "data_invio" : "${xmldate(node.properties["{http://www.e-
       projectsrl.it/lnz/mbonifico/1.1}data_invio"])}",
11 "missingPro" : "${missing}",
12 "uploadedPro" : "${uploaded}",
13 "filename" : "${filename}"
14 }
```

Codice 6.3: Webscript: template di risposta in caso di errore

```
1 {
2 "code": "${status.code}",
3 "codeName":"${status.code},
4 "codeDescription:"${status.codeDescription},
5 "message:"${status.message}
6 }
```

La logica di controllo è invece gestita da uno script scritto in linguaggio javascript, che si chiama:
`up-bonifico-tiff.post.js`
Il suo compito è quello di verificare che la richiesta di inserimento di un documento del tipo "distinta di bonifici" sia effettivamente accompagnata da un file con estensione `'.tiff'`, che siano presenti i valori dei metadati obbligatori e che tali valori siano del formato corretto. Se tutte queste verifiche vanno a buon fine lo script procede con la creazione di un nodo nel *repository* utilizzando le API javascript previste da Alfresco. Il nodo creato diventa quindi il punto di accesso per accedere al contenuto memorizzato e ai suoi metadati. Per il codice del Controller e la sua descrizione si rimanda all'appendice A.

6.6 Caricamento del web script in Alfresco

Una volta creati tutti i file necessari per implementare il servizio web, bisogna procedere alla loro registrazione in Alfresco. Per fare questo si utilizza come al solito il web client Alfresco: si effettua il login con un utente amministratore e si naviga fino alla sezione:
`Company Home > Data Dictionary > Web Script Extension`

Conviene creare un nuovo *space* (cartella) per il nuovo web script da inserire (pulsante `Create Space`). A questo punto si possono caricare all'interno dello *space* il descrittore, lo script di controllo e i vari templates necessari all'implementazione del servizio. Come ultimo passo si deve aggiornare il componente di Alfresco che gestisce i web scripts. Si apre quindi nuovamente il browser e

si naviga alla pagina:
`http://localhost:8080/alfresco/service/index`
denominata *Web Script Home*, che contiene l'elenco dei web scripts attualmente disponibili suddivisi per famiglia, e che permette di effettuare l'aggiornamento di questo elenco tramite il pulsante:
`Refresh Web Scripts`.

6.7 Il collaudo con curl

L'azione che permette di utilizzare un servizio web implementato tramite web script in Alfresco, ovvero l'evento che scatena l'esecuzione dello script (in gergo si dice "consumare il web script"), è l'invio di una richiesta HTTP all'indirizzo:
`http://server-alfresco/alfresco/service/nome-webscript`

Nel caso specifico appena descritto, quindi, è possibile consumare il web script `'up-bonifico-tiff'` tramite una richiesta di tipo POST all'indirizzo:
`http://localhost:8080/alfresco/service/up-bonifico-tiff`

Per effettuare il collaudo di funzionamento è necessario dotarsi di un client HTTP. A tale scopo è stato utilizzato il programma **curl**, facilmente scaricabile dal gestore dei pacchetti in Ubuntu. Si tratta di un client HTTP utilizzabile da terminale. Il comando che permette di effettuare una richiesta POST all'indirizzo sopra citato, che trasporti dei campi di un form (ovvero una richiesta HTTP con header `'content-type'` impostato a `'multipart/form-data'`), compreso di un file che costituisce il documento da inserire è:

```
curl -X POST -H "Content-Type: multipart/form-data" --basic
    --user admin:admin http://127.0.0.1:8080/alfresco/
    service/up-bonifico-tiff -F "file=@./nomedocumento.tiff"
    -F "codice_banca=A123456" -F "data_invio=`date "+%Y-%m
    -%dT%H:%M:%S.$(( $(date +%N) / 1000000 ))Z"`" -F "
    id_postazione=1003" -F "codice_filiale=F45633" -F "
    userid=U7" -F "numero_pagine=2" -F "numero_disposizioni
    =1" -F "stato=ordinato" -F "ndg_cliente=4432341"
```

Nel comando si nota l'utilizzo del meccanismo di autenticazione base previsto da HTTP (è previsto in futuro l'utilizzo del protocollo HTTPS) e la specifica di valori di esempio per i campi previsti dal modello di documento. Nel caso della data è stato utilizzato un comando linux che stampa la data corrente nel formato appropriato per l'inserimento:
`'[YYYY]-[MM]-[DD]T[hh]:[mm]:[ss].[fff]Z'`
Il buon esito dell'operazione viene confermato da una risposta HTTP in formato JSON (il template del web script, compilato con i dati specifici dell'oggetto in questione), che contiene il riepilogo delle proprietà del nodo creato nel server Alfresco, assieme ad alcune informazioni previste dallo script:

Codice 6.4: Webscript: esempio di risposta

```
{
 "name" : "A123456_F45633_20120703T102341628Z_1003.tiff",
 "node-uuid" : "4a822768-c4bd-4d3d-b4a6-333fadb49586",
 "nodeRef" : "workspace://SpacesStore/4a822768-c4bd-4d3d-
    b4a6-333fadb49586",
 "displayPath" : "\/Company Home\/BONIFICO_HOME",
 "downloadUrl" : "\/d\/a\/workspace\/SpacesStore\/4a822768-
    c4bd-4d3d-b4a6-333fadb49586\/
    A123456_F45633_20120703T102341628Z_1003.tiff",
 "codice_banca" : "A123456",
 "codice_filiale" : "F45633",
 "id_postazione" : "1,003",
 "data_invio" : "2012-07-03T12:23:41.628+02:00",
 "missingPro" : "cognome,codice_fiscale,nome",
 "uploadedPro" : "stato,ndg_cliente",
 "filename" : "nomedocumento.tiff"
}
```

6.8 Osservazioni

In questo capitolo si è visto come sia possibile utilizzare un web script per l'inserimento di un documento nel server. Accanto ad esso, nel sistema documentale, è presente un altro servizio (che non sarà esemplificato per non appesantire l'esposizione), che mette a disposizione un file XML all'applicazione front-end (lato front-office). Questo XML mappa il modello di documento "distinta di bonifici" in un form per l'inserimento dei campi. In questo modo la configurazione dei campi strutturati da associare ai documenti è sempre mantenuta lato server, dando la possibilità di modificare agevolmente i modelli, senza dover mettere mano al sistema nel suo complesso.
La prima fase (quella d'inserimento nel *repository*) è quindi conclusa. Quello che serve ora è una un meccanismo che permetta di effettuare delle azioni

sul documento, ovvero consenta di implementare il processo aziendale che lo circonda.

Capitolo 7

Workflows con Alfresco

In questo capitolo saranno introdotti gli aspetti di Alfresco legati alla gestione dei processi collegati ai documenti. In particolare sarà introdotto il concetto di *workflow*, il linguaggio **BPMN 2.0** che ne permette la definizione e **Activiti**, il motore di workflow incorporato in Alfresco. Infine, si passerà a presentare come sia stata utilizzata questa funzionalità all'interno del progetto, ovvero presentando la definizione e l'impiego del workflow associato al documento "distinta di bonifici".

7.1 I Workflows

Il Business Process Management è un'approccio alla gestione di un'azienda che si focalizza sui processi aziendali, in particolare quelli di tipo operativo[1]. Con questo termine si indica l'insieme delle attività necessarie per definire, ottimizzare, monitorare e integrare i processi aziendali, al fine di creare un sistema orientato a rendere efficiente ed efficace il business dell'azienda. Attuare le tecniche di BPM significa comprendere appieno l'attività dell'azienda, quello che produce (elementi di output), come questi risultati vengono raggiunti

[1]Sono quei processi che interessano soprattutto variabili quantititative e si ripetono frequentemente. A differenza dei processi strategico-decisionali si prestano molto bene all'automazione.

(processi) e in base a quali risorse (elementi di input).

In questo contesto i workflows si rivelano molto utili perché permettono di modellare i processi, dandone una rappresentazione costituita da una sequenza di operazioni, che possono essere il lavoro di una persona, di un gruppo di persone oppure una procedura meccanica. Un workflow, quindi, descrive un flusso di attività, che possono riferirsi alle fasi di lavorazione di un prodotto o alla trasformazione di un documento passo dopo passo.

In Alfresco è disponibile il servizio di gestione dei workflow, che permette di modellare e automatizzare i processi che ruotano attorno ai contenuti. È infatti possibile specificare delle azioni automatiche, quali la movimentazione di documenti fra diversi utenti, o gruppi di lavoro, e l'esecuzione di scripts, con cui è possibile effettuare operazioni come la conversione di un documento da un formato ad un altro o l'invio di una comunicazione tramite posta elettronica, allo scopo di notificare, ad esempio, le diverse fasi di un ordine.

7.2 Activiti

Per implementare tale servizio è necessario disporre di un motore di workflows, ossia un componente software responsabile dell'esecuzione dei workflows, di gestire la loro definizione e le attività in essa specificate. Esso deve fornire l'ambiente in cui i processi possano essere eseguiti, assieme a tutti i servizi necessari che permettono al processo di svolgere il suo lavoro.
Ogni processo, o "istanza di workflow", che viene eseguito all'interno del motore, segue un cammino descritto dalla "definizione del workflow". Un processo può essere legato ad una ed una sola definizione, che descrive le fasi del processo nei vari aspetti (quando avvengono, cosa avviene durante ogni fase, etc.).

Activiti è il motore di workflow incorporato in Alfresco. Esso è a sua volta incorporato in un altro componente, detto *Workflow Service*, che fornisce i servizi necessari per la gestione dei workflows, nascondendo all'utente la

sottostante implementazione. Le sue API sono disponibili sia in java, che in versione javascript e permettono di accedere alle definizioni dei workflows, alle loro istanze, ai task e alle transizioni che ne fanno parte. Ad esempio, data una definizione di workflow, è possibile avviare una nuova istanza, cancellare delle istanze esistenti, oppure inviare dei segnali per l'avanzamento del cammino di un determinato workflow, fermo in attesa per una determinata operazione manuale.
Il punto di accesso, tramite javascript, a queste API è un oggetto che è esposto a livello globale in Alfresco, che si chiama *Workflow Manager* ed è possibile richiamarlo nel codice degli script attraverso il nome `workflow`. Nel webscript `up-bonifico-tiff`, il cui codice è descritto nell'appendice A, è possibile vedere un esempio di avvio di una nuova istanza di workflow effettuato utilizzando proprio quest'oggetto.

7.3 Introduzione a BPMN 2.0

BPMN 2.0, acronimo di Business Process Model and Notation, è il linguaggio per la definizione dei workflows supportato da Activiti. In realtà si tratta di uno standard per la definizione dei processi di business in generale, sviluppato da OMG[2] con lo scopo di fornire una notazione che sia facilmente comprensibile a tutti gli utenti business. Esso permette di modellare dei concetti come *attività umana, script eseguibile, decisione automatica, etc.* Questi vengono visualizzati tramite degli schemi in una maniera standardizzata. La seconda versione, quella attuale, ha introdotto delle semantiche di esecuzione[3] e un comune formato di scambio. Questo ha permesso non solo lo scambio delle modellazioni tra visualizzatori grafici, ma anche di poter eseguire quei modelli nei motori di workflows (detti anche sistemi BPM) compatibili con questo linguaggio, quale è Activiti.

[2]Object Management Group.

[3]Nei linguaggi di programmazione, le semantiche di esecuzione definiscono come e quando i vari costrutti di un linguaggio dovrebbero produrre un comportamento nel programma.

7.4 Definizione di workflow

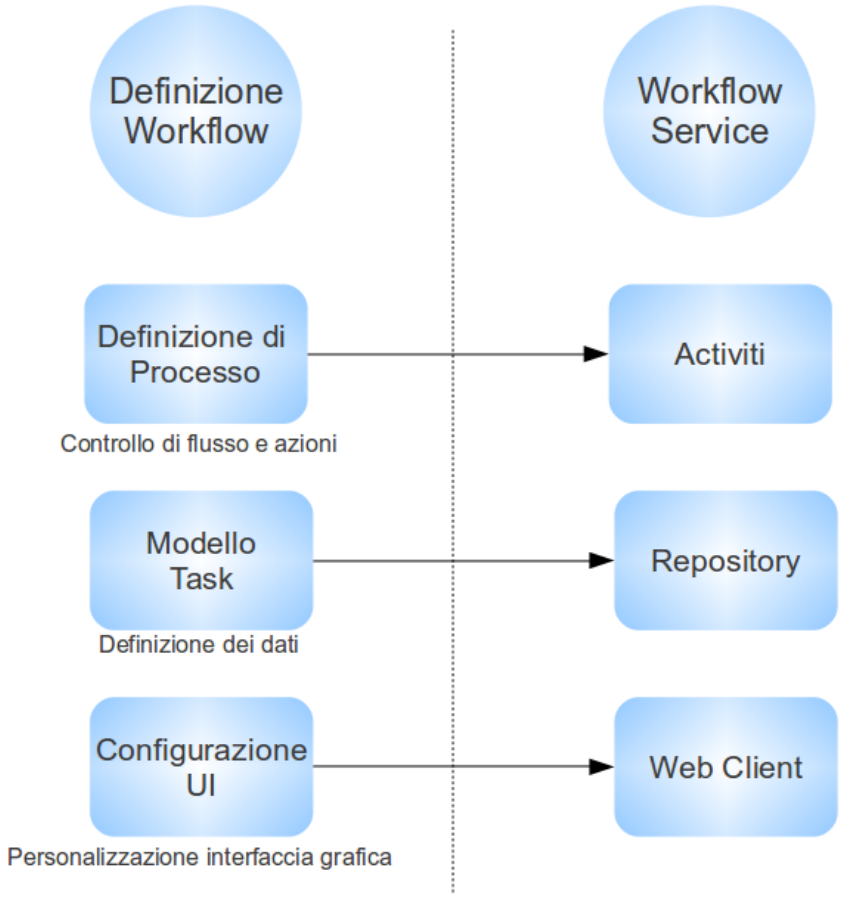

Figura 7.1: Definizione di Workflow

Per aggiungere un nuovo workflow in Alfresco è quindi necessario procedere alla scrittura della sua definizione, che si compone di tre elementi chiave:

1. la definizione del processo: descrive gli eventi, le attività (*tasks*) e le scelte (*gateways*). È costituita da un file XML scritto con il linguaggio BPMN 2.0;

2. il *Task Model*: fornisce una descrizione dei task di tipo utente, ovvero le attività che devono essere svolte da un utente assegnato. La descrizione avviene in termini di dati associati alle attività: in pratica si modellano i metadati che descrivono le informazioni a loro associate. Il linguaggio per la modellazione dei contenuti si applica anche alla definizione dei *Task Model*, quindi ogni task è descritto tramite le definizioni di *types, properties e associations*. Questa descrizione viene poi collegata ad un'interfaccia grafica, che permette di gestire e visualizzare il task;

3. la configurazione dell' UI (*User Interface)*: per l'interfaccia grafica con cui maneggiare i task è possibile utilizzare direttamente il web client di Alfresco (o l'alternativa applicazione web **Share**), definendo i controlli con cui le proprietà sono visualizzate: si descrive una maschera per l'inserimento e la modifica dei dati.

La figura 7.1 indica i vari componenti della definizione di workflow e la loro interazione con Alfresco.

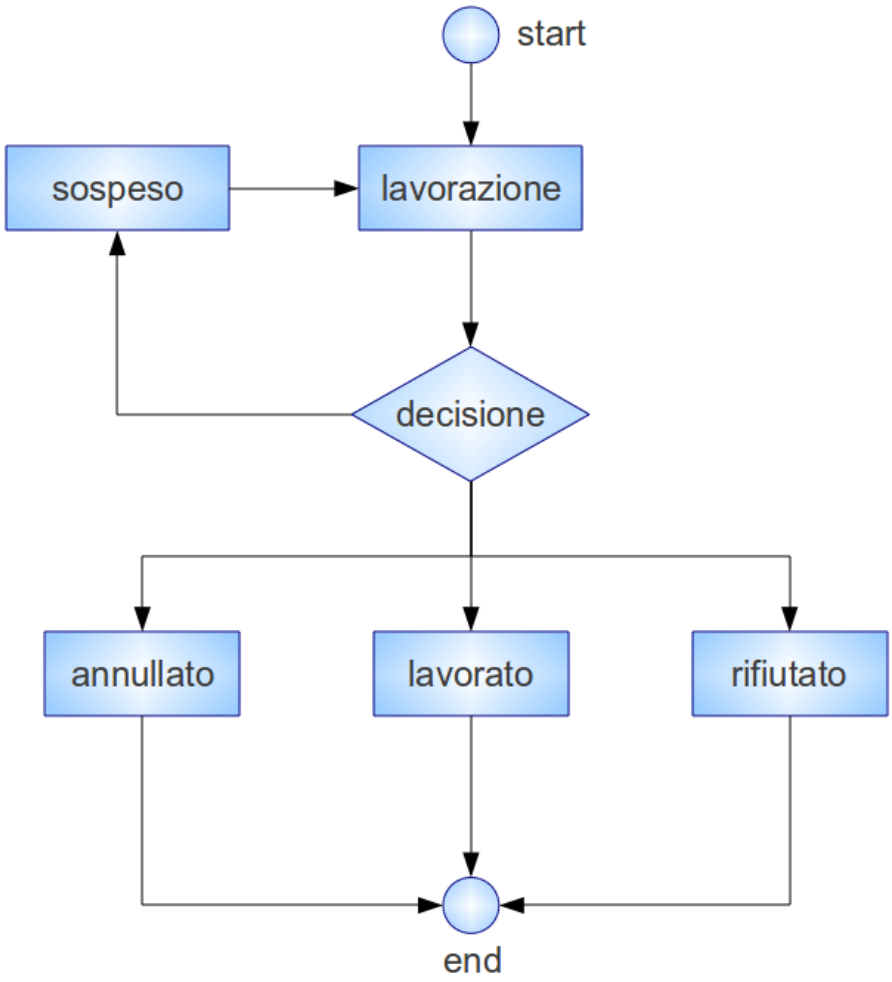

Figura 7.2: Workflow: processo da rappresentare

7.4.1 Definizione del processo

Il processo che vogliamo rappresentare è rappresentato in figura 7.2. In pratica, dopo l'inserimento di un ordine di bonifico, rappresentato dal documento "distinta di bonifici", viene avviato un processo che comincia con la fase di lavorazione dell'ordine, che è di competenza di un determinato gruppo di lavoro. In base all'esito di questa fase l'ordine può passare in uno dei seguenti stati: LAVORATO, RIFIUTATO, ANNULLATO, SOSPESO. L'XML che descrive questo processo ha nome:
`wfbonifico.bpmn20.xml`
e comincia con la seguente intestazione:

Codice 7.1: Workflow bonifico: definizione di processo - intestazione

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2
3  <definitions
4    xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
5    xmlns:activiti="http://activiti.org/bpmn"
6    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7    targetNamespace="http://www.e-projectsrl.it/lnz/wfbonifico
         /2.0">
8
9    <process id="bonifico" name="ordine di bonifico">
10   ...
```

Da notare che, oltre al namespace che identifica i tag del linguaggio BPMN, è presente un namespace con prefisso `'activiti'`. Questo introduce dei tag di estensione al linguaggio BPMN specifici di Activiti, chiamati *Activiti BPMN Extensions*. Queste estensioni, i cui tag sono facilmente identificabili grazie al prefisso, consistono sia di nuovi costrutti che di modi per semplificare i costrutti presenti nello standard, considerati da Activiti a volte troppo ingombranti. Questa caratteristica potrebbe minare gli scopi dello standard, tuttavia le estensioni vengono fornite con il prerequisito che esista sempre una trasformazione che permetta di tornare ai metodi di definizione standard. Il loro scopo è anche quello di proporre dei miglioramenti per le definizioni future di BPMN.

Dopo l'intestazione, si passa alla definizione dei tasks. È necessario cominciare con l'evento di start che identifica l'inizio del cammino del workflow:

Codice 7.2: Workflow bonifico: definizione di processo - tasks

```
1  ...
2  <startEvent id="start"
3     activiti:formKey="wmbon:invioBonificoTask" />
4
5  <userTask id='lavorazione' name='elaborazione bonifico'
6     activiti:formKey="wmbon:lavorazioneTask" >
7     <documentation>
8        Lavorazione ordine di bonifici
9     </documentation>
10    <extensionElements>
11       <activiti:taskListener event="complete" class="org.
             alfresco.repo.workflow.activiti.tasklistener.
             ScriptTaskListener">
12             <activiti:field name="script">
13                <activiti:string>
14                   execution.setVariable('wmbon_esitoLavorazione
                        ', task.getVariable('
                        wmbon_esitoLavorazione'));
15                </activiti:string>
16             </activiti:field>
17       </activiti:taskListener>
18    </extensionElements>
19    <potentialOwner>
20       <resourceAssignmentExpression>
21          <formalExpression>group(team bonifici)</
                formalExpression>
22       </resourceAssignmentExpression>
23    </potentialOwner>
24 </userTask>
25
26 <exclusiveGateway  id='decisione' name='esito lavorazione'
       />
27 ...
```

In questo frammento, l'attributo `activiti:formKey` nell'evento di start indica ad Alfresco quale form visualizzare quando l'utente seleziona il workflow. Allo stesso modo viene utilizzato per gli `userTask`, ossia fornisce il collegamento tra l'attività e i dati che la descrivono, i quali sono modellati nel *Task Model* tramite un `type` con nome corrispondente.
Un elemento di estensione che si può notare nel task di tipo utente di nome `'lavorazione'` è il costrutto `activiti:taskListener`, che permette di eseguire delle azioni al verificarsi di certi eventi. In questo caso viene utilizzato per eseguire uno script che imposta una proprietà del task, ovvero l'esito della lavorazione. Il tag `potentialOwner` serve per assegnare l'attività ad un gruppo di utenti (che ovviamente deve essere stato definito nella gestione degli utenti di Alfresco). L'ultimo costrutto, introdotto dal tag `exclusiveGateway`, serve per rappresentare il punto di decisione.

Similmente la definizione del processo prosegue con le altre definizioni: il task di nome `'sospeso'` è uno script al momento vuoto, ma chiarisce dove sia possibile inserire le azioni automatiche che si renderanno necessarie:

```
1 <scriptTask id="sospeso" name="Execute script sospeso"
      scriptFormat="groovy">
2  <script>
3  </script>
4 </scriptTask>
```

Segue il task `'annullato'`, dove è presente un esempio per inviare una notifica email in maniera automatica (richiede la configurazione di Alfresco con un server di posta):

```
1  <userTask id="annullato" name="ordine annullato"
2    activiti:formKey="wmbon:annullatoTask">
3    <documentation>
4      l'ordine di bonifico è stato annullato
5    </documentation>
6    <extensionElements>
7      <activiti:taskListener event="create" class="org.
           alfresco.repo.workflow.activiti.tasklistener.
           ScriptTaskListener">
8      <activiti:field name="script">
9      <activiti:string>
10       var mail = actions.create("mail");
11       mail.parameters.to="manager@gmail.com";
12       mail.parameters.subject="ordine annullato " +
             bpm_workflowDescription;
13       mail.parameters.from="bonificiteam@gmail.com";
14       mail.parameters.text="L'ordine è stato annullato";
15       mail.execute(bpm_package.children[0]);
16     </activiti:string>
17     </activiti:field>
18     </activiti:taskListener>
19   </extensionElements>
20   <humanPerformer>
21     <resourceAssignmentExpression>
22       <formalExpression>${initiator.properties.userName}</
             formalExpression>
23     </resourceAssignmentExpression>
24   </humanPerformer>
25 </userTask>
```

Similmente proseguono i task `'rifiutato'` e `'lavorato'`, seguiti dall'evento `'end'`:

```
1  <userTask id="rifiutato" name="ordine rifiutato"
2    activiti:formKey="wmbon:rifiutatoTask">
3    <documentation>
4      l'ordine di bonifico è stato rifiutato
5    </documentation>
6      <extensionElements>
7        <activiti:taskListener event="create" class="org.
             alfresco.repo.workflow.activiti.tasklistener.
             ScriptTaskListener">
8        <activiti:field name="script">
9        <activiti:string>
10       var mail = actions.create("mail");
11       mail.parameters.to="manager@gmail.com";
12       mail.parameters.subject="ordine rifiutato " +
             bpm_workflowDescription;
13       mail.parameters.from="teambonifici@gmail.com";
14       mail.parameters.text="L'ordine è stato rifiutato";
15       mail.execute(bpm_package.children[0]);
16       </activiti:string>
17       </activiti:field>
18       </activiti:taskListener>
19     </extensionElements>
20   <humanPerformer>
21     <resourceAssignmentExpression>
22       <formalExpression>${initiator.properties.userName}</
             formalExpression>
23     </resourceAssignmentExpression>
24   </humanPerformer>
25 </userTask>
```

```
1  <userTask id="lavorato" name="ordine lavorato"
2    activiti:formKey="wmbon:lavoratoTask">
3    <documentation>
4      il documento è stato visionato e lavorato
5    </documentation>
6    <extensionElements>
7      <activiti:taskListener event="create" class="org.
           alfresco.repo.workflow.activiti.tasklistener.
           ScriptTaskListener">
8      <activiti:field name="script">
9      <activiti:string>
10     var mail = actions.create("mail");
11     mail.parameters.to="manager@gmail.com";
12     mail.parameters.subject="ordine lavorato " +
           bpm_workflowDescription;
13     mail.parameters.from="teambonifici@gmail.com";
14     mail.parameters.text="L'ordine è stato lavorato";
15     mail.execute(bpm_package.children[0]);
16     </activiti:string>
17     </activiti:field>
18     </activiti:taskListener>
19   </extensionElements>
20   <humanPerformer>
21     <resourceAssignmentExpression>
22       <formalExpression>${initiator.properties.userName}</
             formalExpression>
23     </resourceAssignmentExpression>
24   </humanPerformer>
25 </userTask>
26 <endEvent id="end" />
27 ...
```

Da notare che i task `'annullato'`, `'rifiutato'` e `'lavorato'` vengono assegnati all'utente che aveva istanziato il processo (tramite l'oggetto `initiator`). Tale utente potrebbe rappresentare la filiale che aveva inoltrato l'ordine, mentre le attività rappresentano le azioni necessarie da svolgere in seguito alla risposta dell'elaborazione dell'ordine da parte del team bonifici.

Definite le attività che intervengono nel processo non resta che inserire i collegamenti, dove vengono specificate anche le condizioni per effettuare le transizioni nel punto di scelta (`exclusiveGateway`):

Codice 7.3: Workflow bonifico: definizione di processo - collegamenti

```
1  <sequenceFlow id='flow1' sourceRef='start' targetRef='
       lavorazione'/>
2  <sequenceFlow id='flow2' sourceRef='lavorazione'
       targetRef='decisione'/>
3  <sequenceFlow id='flow3' sourceRef='decisione' targetRef=
       'sospeso'>
4    <conditionExpression xsi:type="tFormalExpression">${
         wmbon_esitoLavorazione == 'sospeso'}</
         conditionExpression>
5  </sequenceFlow>
6  <sequenceFlow id='flow4' sourceRef='sospeso' targetRef='
       lavorazione'/>
7  <sequenceFlow id='flow5' sourceRef='decisione' targetRef=
       'annullato'>
8    <conditionExpression xsi:type="tFormalExpression">${
         wmbon_esitoLavorazione == 'annullato'}</
         conditionExpression>
9  </sequenceFlow>
10 <sequenceFlow id='flow6' sourceRef='annullato' targetRef=
       'end'/>
```

Si può notare come il passaggio condizionato tra due fasi del processo (ad esempio dalla fase `'lavorazione'` alla fase `'sospeso'`) venga effettuato tra-

mite una variabile di processo (`wmbon_esitoLavorazione`), che dovrà essere opportunamente modellata nel *Task Model*. Seguono poi gli altri collegamenti e la chiusura dei tag iniziali:

```
1   <sequenceFlow id='flow7' sourceRef='decisione' targetRef=
        'rifiutato'>
2     <conditionExpression xsi:type="tFormalExpression">${
          wmbon_esitoLavorazione == 'rifiutato'}</
          conditionExpression>
3   </sequenceFlow>
4   <sequenceFlow id='flow8' sourceRef='rifiutato' targetRef=
        'end'/>
5   <sequenceFlow id='flow9' sourceRef='decisione' targetRef=
        'lavorato'>
6     <conditionExpression xsi:type="tFormalExpression">${
          wmbon_esitoLavorazione == 'lavorato'}</
          conditionExpression>
7   </sequenceFlow>
8   <sequenceFlow id='flow10' sourceRef='lavorato' targetRef=
        'end' />
9 </process>
10 </definitions>
```

7.4.2 Il Task Model

Una volta definito il processo del workflow bisogna descrivere i task che ne fanno parte nel *Task Model*, in termini di proprietà ed associazioni. La sua definizione, quindi, avviene proprio come un modello di contenuto. Si comincia con l'intestazione:

Codice 7.4: Workflow bonifico: task model

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <model name="wmbon:wfmodelbonifico" xmlns="http://www.
       alfresco.org/model/dictionary/1.0">
4    <author>Lorenzo Caldera</author>
5    <version>2.0</version>
6
7    <imports>
8      <import uri="http://www.alfresco.org/model/dictionary
         /1.0" prefix="d" />
9          <import uri="http://www.alfresco.org/model/bpm/1.0"
             prefix="bpm" />
10   </imports>
11
12   <namespaces>
13         <namespace uri="http://www.e-projectsrl.it/lnz/
             wfmodelbonifico/2.0" prefix="wmbon" />
14   </namespaces>
15    ...
```

Poi si passa a descrivere i tasks, in particolare quelli dove si è specificato l'attributo `activiti:formKey`:

```
<types>
 <type name="wmbon:invioBonificoTask">
  <parent>bpm:startTask</parent>
    <mandatory-aspects>
         <aspect>bpm:groupAssignee</aspect>
        </mandatory-aspects>
 </type>
 <type name="wmbon:lavorazioneTask">
  <parent>bpm:workflowTask</parent>
  <properties>
   <property name="wmbon:esitoLavorazione">
    <type>d:text</type>
    <default>annullato</default>
        <constraints>
     <constraint type="LIST">
      <parameter  name="allowedValues">
          <list>
           <value>annullato</value>
           <value>rifiutato</value>
           <value>lavorato</value>
           <value>sospeso</value>
          </list>
          </parameter>
     </constraint>
        </constraints>
   </property>
  </properties>
  <overrides>
   <property name="bpm:packageItemActionGroup">
        <default>edit_package_item_actions</default>
   </property>
  </overrides>
 </type>
```

Nel type con nome `wmbon:lavorazioneTask` (`wmbon` è il prefisso del namespace cui appartiene il tag) è possibile vedere il costrutto `<constraints>`, che permette di restringere il campo di valori che può assumere la proprietà `wmbon:esitoLavorazione`. Il costrutto racchiuso fra i tags `<overraides>`, invece, serve per consentire la modifica delle proprietà del task. Molti degli attributi definiti in un workflow, infatti, sono collegati ad un *package*, che consiste in uno spazio memorizzatto all'interno del *Content Repository* dedicato alla memorizzazione delle informazioni associate all'istanza di un workflow. Non solo il task ha un riferimento al *package*, ma dispone anche di alcune proprietà che definiscono i diritti di accesso ad esso. Queste sono `bpm:packageActionGroup` e `bpm:packageItemActionGroup`, ereditate dalla definizione di task presente nel modello con prefisso `'bpm'`, e che vanno sovrascritte per permettere la modifica delle variabili memorizzate nel *package*.

Nella definizione di processo è possibile accedere agli attributi specificati nel *Task Model* attraverso la notazione `prefisso_attributo`. Ad esempio, nella definizione del processo presentata nella sezione precedente si accede alla variabile `wmbon_esitoLavorazione` per poter stabilire quale percorso intraprendere nel processo (ovvero se l'ordine è stato accettato, annullato, sospeso o rifiutato).

Il *Task Model* comprende poi le descrizioni degli altri tasks, al momento lasciate vuote in attesa di avere maggiori informazioni sulle attività svolte nel processo:

```
1  ...
2 <type name="wmbon:sospesoTask">
3  <parent>bpm:workflowTask</parent>
4  <properties>
5  </properties>
6 </type>
7
8 <type name="wmbon:annullatoTask">
9  <parent>bpm:workflowTask</parent>
10  <properties>
11  </properties>
12 </type>
13
14 <type name="wmbon:rifiutatoTask">
15  <parent>bpm:workflowTask</parent>
16  <properties>
17  </properties>
18 </type>
19
20 <type name="wmbon:lavoratoTask">
21  <parent>bpm:workflowTask</parent>
22  <properties>
23  </properties>
24 </type>
25 </types>
26 </model>
```

7.4.3 Configurazione UI

La possibilità di interagire con le diverse fasi del workflow, a partire dall'avvio, alla specifica delle proprietà nei diversi task, è resa disponibile attraverso il web client Alfresco. Per fare questo è necessario introdurre una configurazione, che specifica i form associati ai task da svolgere. Ancora una volta si tratta di un file xml di nome `web-client-config-custom.xml`:

Codice 7.5: Workflow bonifico: configurazione UI

```
1  <alfresco-config>
2  <!-- workflow bonifico -->
3   <config evaluator="node-type" condition="
        wmbon:invioBonificoTask" replace="true">
4    <property-sheet>
5     <separator name="sep1" display-label-id="general"
          component-generator="HeaderSeparatorGenerator" />
6     <show-property name="bpm:workflowDescription" component-
          generator="TextAreaGenerator" />
7     <show-property name="bpm:workflowPriority" />
8     <show-property name="bpm:workflowDueDate" />
9     <separator name="sep2" display-label-id="users_and_roles
          " component-generator="HeaderSeparatorGenerator" />
10    <show-association name="bpm:groupAssignee" />
11    </property-sheet>
12  </config>
```

In questo frammento è possibile notare la composizione del form di avvio del workflow: sono presenti alcune proprietà generali, come la descrizione e la data di scadenza, assieme al componente `bpm:groupAssignee` che permette di assegnare il workflow ad un gruppo.
Devono poi essere descitti anche i form degli altri tasks, in questo caso solo quello relativo al task `'lavorazione'`, che deve permettere di assegnare l'esito alla variabile `'wmbon:esitoLavorazione'`:

```
1  <config evaluator="node-type" condition="
       wmbon:lavorazioneTask" replace="true">
2   <property-sheet>
3    <separator name="sep1" display-label-id="general"
         component-generator="HeaderSeparatorGenerator" />
4    <show-property name="bpm:taskId" />
5    <show-property name="bpm:description" component-
         generator="TextAreaGenerator" read-only="true"/>
6    <show-property name="bpm:dueDate" read-only="true" />
7    <show-property name="bpm:priority" read-only="true" />
8    <show-property name="wmbon:esitoLavorazione" />
9   </property-sheet>
10 </config>
11 </alfresco-config>
```

7.5 Caricamento del workflow in Alfresco

Terminata la fase di definizione del workflow è possibile effettuare la registrazione all'interno del server Alfresco, ovvero fare il *deploying del workflow*. Tale operazione è effettuabile sempre attravreso il web client Alfresco e comincia con il caricamento della definizione del processo, che in questo caso consiste del file `wfbonifico.bpmn20.xml` descritto in precedenza. Bisogna quindi accedere al web client nella sezione:
`Company Home > Data Dictionary > Workflow Definitions`
poi bisogna premere sul pulsante `Add Content` e caricare la definizione del processo, facendo attenzione a selezionare la casella `Workflow Deployed` e a specificare il motore di workflow `'activiti'` al posto di `'jbpm'`[4]. A questo punto il sistema procederà alla validazione dell'XML che, se non presenta er-

[4]Si tratta di un altro motore di workflow supportato da Alfresco, attualmente però non compatibile con il linguaggio standard BPMN 2.0, ma con l'altrettanto famoso JPDL di JBOSS.

rori, sarà caricato con successo. Per verificarlo è disponibile la console per i workflows, all'indirizzo:
`http://localhost:8080/alfresco/faces/jsp/admin/workflow-console.jsp`
Da questa console è possibile controllare quali sono le definizioni di workflow caricate, effettuare il *deploying* e *undeploying* delle definizioni, verificare le istanze di workflow attive, effettuare delle operazioni su di esse, etc. Per controllare se la nostra definizione è stata caricata digitare il comando:
`show definitions all`
tra le stringhe di output deve comparirne una del tipo:

Codice 7.6: Workflow bonifico: output console

```
1 id: activiti$bonifico:1:3203 , name: activiti$bonifico ,
    title: ordine di bonifico , version: 1
```

Per il caricamento del *Task Model* del workflow, essendo questo un modello di contenuto, si può procedere nello stesso modo descritto per il modello "distinta di bonifici" descritto nel capitolo 5.

Per quanto concerne la configurazione del web client, si carica nella sezione:
`Company Home > Data Dictionary > Web Client Extension`
tuttavia, perché questa sia effettivamente riconosciuta da Alfresco, nelle prove effettuate si è reso necessario riavviare il server (Tomcat).

A questo punto è finalmente possibile creare delle istanze di processo per il workflow definito, sia in maniera manuale che automatica. Per fare una prova, sempre dal web client, si seleziona il documento interessato dal workflow e si avvia tramite il pulsante:
`Start Advanced Workflow`

L'interazione degli utenti con il sistema, ovvero lo svolgimento virtuale dei task previsti dal workflow avviene a questo punto attraverso il web client Alfresco. L'utente cui è stato assegnato un certo lavoro vede all'interno della propria *home* l'attività che deve svolgere e tramite i form descritti nella configurazio-

ne dell'UI può accedere alle variabili del processo e impostare le informazioni relative al lavoro svolto.

Capitolo 8

Applicazione web per il back office

Nonostante sia possibile configurare il web client Alfresco per poter svolgere tutte le operazioni di gestione dei contenuti, un'interfaccia dedicata risulta più intuitiva e semplice da utilizzare, senza contare il fatto che rende l'applicazione più professionale. Per tali motivi il progetto prevede anche lo sviluppo di un'applicazione client da utilizzare dal lato del back office bancario, che permette agli operatori di accedere agli ordini caricati sul server ed interagire con le fasi dei workflows.
Tale applicazione è ancora in fase di sviluppo, tuttavia si vuole qui presentare la strada intrapresa per implementarla.

8.1 Java Server Faces

Il client consisterà in un'applicazione web, sviluppata tramite la tecnologia Java Server Faces (JSF). Si tratta di una tecnologia parte della piattaforma Java Enterprise Edition e consiste in un framework basato sull'architettura MVC, utile per lo sviluppo di interfacce utente lato server basate sul web. JSF viene infatti presentato come un *UI component based Java Web application framework*. Esso utilizza dei file XML chiamati *facelets* (Views) composti da tag che corrispondono a dei componenti UI. Questi permettono di descrivere

interamente delle interfacce web senza utilizzare in alcun modo linguaggi come HTML o javascript, visto che questi vengono generati automaticamente a partire dai tag dal *FacesServlet*, che è il controllore centrale di un'applicazione JSF, a cui pervengono tutte le richieste.

Un'applicazione JSF viene eseguita all'interno di un contenitore servlet (Apache Tomcat) e si compone quindi di alcune pagine web descritte tramite i componenti UI e di alcuni file di configurazione xml. In particolare, tramite questi ultimi è possibile definire la navigazione tra le pagine, assieme a dei *Managed Beans*, che sono delle classi java che possono essere utilizzate dall'applicazione.

8.2 Comunicazione con il server

Grazie a queste caratteristiche sarà quindi possibile integrare l'applicazione con delle librerie java. Nello specifico, si potranno sfruttare le librerie **openCMIS** del progetto Apache Chemistry, che permettono di effettuare delle richieste conformi allo standard CMIS e quindi di sfruttare i servizi che esso mette a disposizione.

Per quanto riguarda la gestione dei workflow, invece, lo standard CMIS non prevede ancora alcuna funzionalità. Tuttavia, a tale scopo sono messi a disposizione da parte di Alfresco dei web scripts appositi (si veda [2]).

La comunicazione tra il client front-end e il back-end Alfresco, quindi, avverrà tramite chiamate a servizi web di tipo REST: utilizzando lo standard CMIS per le operazioni di base, come l'autenticazione e il recupero dei contenuti, e i web scripts proprietari di Alfresco per la gestione dei workflows. Se si rendessero neccessarie delle operazioni ancora più avanzate sarà sempre possibile implementare dei servizi appositi tramite dei web scripts personalizzati, come presentato nel capitolo dedicato.

Conclusioni

In questa breve esposizione è stato mostrato come sia possibile costruire un sistema documentale attorno al software Alfresco, un ECM professionale. Personalmente considero lo studio di questa categoria di applicativi molto utile, perché si tratta di soluzioni sempre più utilizzate nel mondo del lavoro, soprattutto nelle grandi aziende, dove si cercano costantemente metodi per snellire la complessità di gestione.
Anche se qualcuno potrebbe mostrarsi dubbioso riguardo all'accostamento di tecnologie open source a prodotti professionali, io considero questo tipo di approccio vantaggioso perché queste tecnologie hanno il pregio di focalizzarsi sempre sugli standard aperti. Questo permette di arricchire in maniera utile il proprio bagaglio di conoscenze, che diventano riutilizzabili poi anche in altri ambiti e nei sistemi proprietari.

Tornando al progetto, il sistema descritto è tutt'altro che concluso, ma è possibile intravedere la strada da perseguire per raggiungere gli obiettivi finali: il moderno approccio modulare proposto, che consiste nell'implementare ogni servizio separatamente, si mostrerà una soluzione flessibile e vincente perché permette di arricchire le funzionalità del sistema senza appesantirlo. Sarà quindi possibile integrare tutte le funzioni bancarie richieste che si appoggiano a dei documenti (gestione degli assegni, pagamenti, domiciliazioni, etc.) e aggiornare o sostituire in futuro quelle esistenti, il tutto lavorando in maniera isolata, senza dover compromettere il funzionamento degli altri servizi.

Gli sviluppi futuri saranno concentrati sicuramente sull'implementazione dell'applicazione client descritta nell'ultimo capitolo, ma anche nell'introdu-

zione di componenti di schedulazione per l'assegnazione dei task dei workflow, per supportare le politiche di assegnamento della gestione bancaria ben più complesse di quelle descritte in questa sede.

Voglio infine concludere valutando positivamente l'attività di stage. Essa infatti consente allo studente, che già conosce molto bene l'ambiente accademico, di dare uno sguardo anche al mondo del lavoro, rendendosi conto delle tecnologie che vengono applicate e delle capacità che possono essere richieste dalle aziende.

Appendice A

In questa sezione viene descritto il codice responsabile della logica di controllo del web script `up-bonifico-tiff` descritto nel capitolo 6.
Il *controller* è costituito da uno script scritto in linguaggio javascript e contenuto nel file:
`up-bonifico-tiff.post.js`
Lo script comincia con l'inizializzazione di alcune variabili globali:

Codice 8.1: Webscript: logica di controllo

```
1 //variabili globali
2 var tipoNodo="mbon:bonifico";
3 var estensione="tiff";
4 var homeSpace="BONIFICO_HOME";
5 var codifica="utf-8";
6 var tipoMime="image/tiff";
7 var prefisso="mbon";
```

Sono in sostanza dei parametri che caratterizzano il servizio, utili per la gestione della richiesta. In particolare indicano il tipo di estensione accettata per il file, il modello di contenuto cui il documento ricevuto fa riferimento e lo *space*[1] in cui dovrà essere memorizzato.

[1]Nodo di Alfresco che rappresenta una cartella.

Seguono poi degli array inizializzati con i metadati del modello associato al documento. In particolare, l'array `mandPro` contiene le proprietà che devono essere obbligatoriamente presenti per poter completare l'inserimento:

```
1  //proprietà obbligatorie, che devono essere presenti per l'
       inserimento
2  var mandPro = new Array();
3  populateMandPro(mandPro);
4  function populateMandPro(mandPro) {
5    mandPro["codice_banca"]="text";
6    mandPro["codice_filiale"]="text";
7    mandPro["id_postazione"]="int";
8    mandPro["userid"]="text";
9    mandPro["data_invio"]="date";
10   mandPro["numero_pagine"]="int";
11   mandPro["numero_disposizioni"]="int";
12 }
```

Mentre l'array `otherPro` indica le proprietà opzionali, quelle che ai fini della gestione della richiesta possono essere presenti o meno:

```
1  //il formato accettato per la data è
2  //2011-11-12T10:39:55.369Z
3  var otherPro = new Array();
4  populateOtherPro(otherPro);
5  function populateOtherPro(otherPro) {
6    otherPro["ndg_cliente"]="text";
7    otherPro["nome"]="text";
8    otherPro["cognome"]="text";
9    otherPro["codice_fiscale"]="text";
10   otherPro["stato"]="text";
11 }
```

Nell'array `campiNomeNodo`, invece, si indicano i campi con cui costruire il nome del documento e, per garantire che non vi siano collisioni, sono stati scelti dei campi che permettano di identificare univocamente il documento, ovvero la cui combinazione costituisce una chiave:

```
1 //memorizzo in questo array i campi con cui costruire il
      nome del nodo da memorizzare nel server
2 var campiNomeNodo = new Array();
3 populateCampiNomeNodo(campiNomeNodo);
4 function populateCampiNomeNodo(campiNomeNodo) {
5   campiNomeNodo[0]="codice_banca";
6   campiNomeNodo[1]="codice_filiale";
7   campiNomeNodo[2]="data_invio";
8   campiNomeNodo[3]="id_postazione";
9 }
```

Tutte queste informazioni potrebbero tuttavia essere recuperate dal modello di contenuto già registrato nel *repository* sotto forma di file XML, ma vista la mancanza di API apposite per il loro agevole recupero è stato necessario inserirle manualmente. Vi sono poi altre variabili, utili per verificare la correttezza dei vincoli per la proprietà "stato" e per tener traccia delle proprietà inserite e di quelle mancanti:

```
1  //variabili per il controllo della proprietà stato
2  var nomeProStato="stato";
3  var statusVal = new Array();
4  acceptedStatus(statusVal);
5  function acceptedStatus(statusVal){
6    statusVal["ordinato"]="";
7    statusVal["accettato"]="";
8    statusVal["rifiutato"]="";
9    statusVal["eseguito"]="";
10 }
11
12 //variabili per tener traccia delle proprietà inserite
13 var missingPro = new Array();
14 var insertedPro = new Array();
15
16 //indica una condizione di errore
17 var error = false;
```

Sono poi definite due funzioni principali:

- `ValidateMandProp()`
- `insertPro()`

Alla prima è demandato il compito di controllare la presenza e la correttezza dei metadati obbligatori associati al documento, mentre alla seconda è richiesto il compito di memorizzare tutte le proprietà nel nuovo nodo che si sta creando all'interno del *Content Repository*, comprese quelle non obbligatorie (essendo il genere del nodo del `type` modellato nel capitolo 5, la sua creazione richiederà la presenza dei valori per tutte le proprietà obbligatorie).

Codice 8.2: Webscript: funzione validateMandProp()

```
1  function validateMandProp(){
2    var mproVal;
3    for (mproName in mandPro) {
4    try    {
5      mproVal=argsM[mproName][0];
6    } catch(err)  {
7      status.code=400;
8      status.message="Missing mandatory property:"+mproName;
9      return false;
10   }
11
12   switch (mandPro[mproName]){
13     case "int":
14      if (isInt(mproVal)==false){
15       status.code=400;
16       status.message="Wrong property type: integer is
            expected for  " + mproName;
17       return false;
18      }
19      break;
20     case "date":
21      if(isDate(mproVal)==false){
22       status.code=400;
23       status.message="Wrong date format: expected format is
             [YYYY]-[MM]-[DD]T[hh]:[mm]:[ss].[fff]Z";
24       return false;
25      }
26      break;
27     case "text":
28      break;
29     default:
30      status.code=400;
31      status.message="type not supported by this script and
          can't be checked. Please update the script";
```

```
32          return false;
33        }
34      }
35      return true;
36  }
```

Codice 8.3: Webscript: funzione insertPro()

```
1  function insertPro(){
2  var mandValue, proValue;
3   //first insert mandatory properties that are already
       checked
4  for (mandName in mandPro) {
5    mandValue = argsM[mandName][0];
6    if (mandPro[mandName]=="int"){
7          upload.properties[prefisso+":"+mandName] = parseInt
               (mandValue);
8    } else {
9          upload.properties[prefisso+":"+mandName] =
               mandValue;
10   }
11 }
12
13 // if present, insert other properties
14 for (proName in otherPro) {
15   if (argsM[proName]!=null && argsM[proName]!=undefined){
16     try {
17      proValue = argsM[proName][0];
18           switch(otherPro[proName]) {
19      case "int":
20       if (isInt(proValue)==false){
21        status.code=400;
22        status.message="wrong property type: int is expected
               for: "+proName;
23        return false;
24       }
25       upload.properties[prefisso+":"+proName] = parseInt(
             proValue);
26       break;
27      case "text":
28       if(proName==nomeProStato) {
29       if(statusVal[proValue.toLowerCase()]==undefined){
```

```
30          status.code=400;
31          status.message="wrong "+nomeProStato+" value";
32          return false;
33         }
34         proValue = proValue.toLowerCase();
35         }
36         upload.properties[prefisso+":"+proName] = proValue;
37         break;
38        default:
39         status.code=400;
40         status.message="cannot check the property value";
41         return false;
42         break;
43        }
44        insertedPro.push(proName);
45           } catch (err) {
46         status.code=400;
47         status.message="something goes wrong in inserting not
               mandatory properties, please contact the
              developer";
48         return false;
49           }
50    } else {
51      missingPro.push(proName);
52    }
53 }
54 return true;
55 }
```

Seguono poi delle funzioni ausiliarie: la funzione `isInt(numero)` viene utilizzata per verificare la correttezza dei tipi di dato "intero":

```
1 function isInt(numero) {
2   if (isNaN(numero) == true) {
3     return false;
4   } else if (parseInt(numero)!=numero) {
5     return false;
6   }
7   return true;
8 }
```

La funzione `isDate(str)` viene utilizzata per controllare il corretto formato dei campi che rappresentano una data, secondo le specifiche stabilite:

```
1  function isDate(str) {
2   //esempio di formato valido 2011-11-12T10:39:55.369Z
3   var expr = /^[0-9]{4}[\-][0-9]{2}[\-][0-9]{2}[T
       ][0-2][0-9][\:][0-5][0-9][\:][0-5][0-9][\.][0-9]{1,3}[Z
       ]$/;
4   if (!expr.test(str))    {
5    return false;
6   } else {
7    y = parseInt(str.substr(0,4),10);
8    m = parseInt(str.substr(5,2),10);
9    d = parseInt(str.substr(8,2),10);
10   hour = parseInt(str.substr(11,2),10);
11   mins = parseInt(str.substr(14,2),10);
12   secs = parseInt(str.substr(17,2),10);
13   mils= parseInt(str.substring(20,str.lastIndexOf("Z")),10)
        ;
14   var date=new Date(y, (m-1), d, hour, mins, secs, mils);
15   if(date.getFullYear()==y && (date.getMonth()+1)==m &&
        date.getDate()==d && date.getHours()==hour && date.
        getMinutes()==mins && date.getSeconds()==secs && date.
        getMilliseconds()==mils ){
16     return true;
17   }else{
18     return false;
19   }
20  }
21 }
```

La funzione `costruisciNomeNodo()`, invece, viene utilizzata per costruire il nome che avrà il documento inserito in Alfresco, mentre la funzione `getFileExt(filename)` viene utilizzata per recuperare e poi controllare l'estensione del file allegato alla richiesta:

```
1  function costruisciNomeNodo() {
2    var i,valoreData,reformattedData,nome="";
3    for (i in campiNomeNodo) {
4      if (mandPro[campiNomeNodo[i]]=="date") {
5            valoreData=argsM[campiNomeNodo[i]][0]+'_';
6            reformattedData=valoreData.replace(/:/g, "").
                 replace(/-/g, "").replace(/\./g, "");
7            nome=nome + reformattedData;
8          }else {
9            nome=nome + argsM[campiNomeNodo[i]][0]+"_";
10         }
11   }
12   nome = nome.slice(0,-1);
13   nome = nome+"."+estensione;
14   return nome;
15 }
16
17 function getFileExt(filename) {
18   return filename.substring(filename.lastIndexOf(".")+1).
       toLowerCase();
19 }
```

L'elaborazione principale, che utilizza queste funzioni, viene eseguita nella funzione `main()`. Tale metodo comincia analizzando la correttezza della richiesta, ovvero controllando che l'HTTP header `'content-type'` abbia come valore `'multipart/form-data'` e che esista un file con estensione `.tiff` allegato alla richiesta:

```
1  main();
2  function main() {
3    var filefield=null;
4    var filename="";
5    var nomeNodo="";
6
7    //check HTTP headers
8    try{
9      var regex = /^multipart\/form\-data/i;
10     if (regex.test(headersM["content-type"][0])==false) {
11       status.code=400;
12       status.message="bad request, content-type must be
             ___multipart/form-data___ value is=___"+headersM["
             content-type"][0]+"___";
13       status.redirect=true;
14       return false;
15     }
16   }
17   catch (err) {
18     status.code=400;
19     status.message="bad request, content-type must be
          multipart/form-data";
20     status.redirect=true;
21     return false;
22   }
23
24   //extract filemultipart/form-data
25   for each (field in formdata.fields) {
26     if (field.name == "file" && field.isFile) {
27       filefield = field;
28           filename = field.filename;
29     }
30   }
```

Successivamente si eseguono una serie di verifiche che utilizzano le funzioni descritte in precedenza, mirate ad individuare delle condizioni di errore:

```
1  ...
2  // ensure file has been uploaded
3    if (filename == "") {
4      error=true
5      status.code = 400;
6      status.message = "Uploaded file cannot be located";
7      status.redirect = true;
8      return false;
9    } else if (getFileExt(filename) != estensione) { //check
       extension
10     error=true;
11     status.code = 400;
12     status.message = "Uploaded file is not a "+estensione+"
           image";
13         status.redirect = true;
14         return false;
15   } else if (validateMandProp()!=true){ //check mandatory
       properties
16         error=true;
17         status.redirect=true;
18         return false;
19   } else {
20   ...
```

Se queste verifiche vengono superate si procede alla creazione di un nuovo nodo tramite le API javascript di Alfresco, e all'associazione ad esso del contenuto da inserire (il file ricevuto):

```
1  //check if the homeSpace folder exists
2   ctHomeNode = companyhome.childByNamePath(homeSpace);
3   if (ctHomeNode==null){
4     try {
5      ctHomeNode = userhome.createFolder(homeSpace);
6     } catch (err) {
7       error=true;
8       status.code=500;
9       status.message="cannot create the "+homeSpace+" folder
            ";
10      status.redirect=true;
11      return false;
12    }
13  } //create the node. From here on, in case of error,
       delete the node manually
14  try {
15    nomeNodo=costruisciNomeNodo();
16    upload = ctHomeNode.createNode(nomeNodo, tipoNodo);
17  } catch (err){
18    error=true;
19    status.code=500;
20    status.message="Cannot create a content with this
         mandatory properties. Probably a content with the
         same values already exists :"+nomeNodo;
21    status.redirect=true;
22    return false;
23  }
24  if (insertPro()== false) {
25    error=true;
26    status.redirect=true;
27    upload.remove(); //deleting the node
28    return false;
29  } else { //all properties are correct, uploading the
      content
```

Da notare che vengono poi impostate delle proprietà dell'oggetto `model`. Quest'oggetto rientra nella definizione del *Model*, nel contesto del pattern MVC e permette di passare dei valori al template utilizzato per la risposta:

```
1   upload.properties.content.write(filefield.content);
2   upload.properties.content.encoding = codifica;
3   upload.properties.content.mimetype = tipoMime;
4   upload.save();
5   //setup model for response template
6   model.node = upload;
7   model.filename=filename;
8   model.uploaded=insertedPro.toString();
9   model.missing=missingPro.toString();
10  //starting the workflow
11  var workflow = actions.create("start-workflow");
12  workflow.parameters.workflowName = "activiti$bonifico";
13  workflow.parameters["bpm:workflowDescription"] = upload.
        name;
14  workflow.parameters["bpm:assignee"] = people.getPerson("
        barney");
15  var futureDate = new Date();
16  futureDate.setDate(futureDate.getDate() + 7);
17  workflow.parameters["bpm:workflowDueDate"] = futureDate;
18  workflow.execute(upload);
19 }
20 }
21 }
```

In caso di errore si può notare come vengano impostati dei differenti parametri, poiché al posto del template usato per comunicare il buon esito, viene utilizzato un altro template con lo scopo appunto di comunicare un errore. Le ultime istruzioni riguardano l'avvio di un workflow associato al documento, funzionalità che è descritta nel capitolo 7.

Bibliografia

[1] AIIM: *Association For Information and Image Management*. http://www.aiim.org/What-is-ECM-Enterprise-Content-Management.

[2] Alfresco: *Alfresco Workflow REST API*. http://wiki.alfresco.com/wiki/Workflow_REST_API.

[3] Andrew S. Tanenbaum, Maarten Van Steen: *Sistemi Distribuiti*. Pearson Education, 2 edizione, 2007.

[4] David Caruana, John Newton, Michael Farman Michael G. Uzquiano Kevin Roast: *Professional Alfresco: Pratical Solutions for Enterprise Content Management*. Wiley Publishing, Inc., 2010.

[5] Fielding, Roy Thomas: *Architectural Styles and the Design of Network-based Software Architectures*. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[6] Matteo Bertocco, Paolo Callegaro, Daniele De Antoni Migliorati: *Ingegneria della Qualità*. CittàStudi, 1 edizione, 2006.

[7] OASIS: *Content Management Interoperability Services*. http://docs.oasis-open.org/cmis/CMIS/v1.0/cmis-spec-v1.0.html.

[8] Pots, Jeff: *Alfresco Developer Guide*. Packt Publishing, 2008.

[9] Reenskaug, Trygve: *Model View Controller*. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html.

[10] W3C: *Web Services Architecture*. http://www.w3.org/TR/ws-arch/.

Elenco delle figure

Elenco dei codici

Ringraziamenti

Vorrei ringraziare il professor Moro per la professionalità dimostrata come relatore e per avermi fornito l'opportunità di effettuare questo stage.

Desidero poi ringraziare l'azienda e-project, in particolare Massimo Businaro e Fabio Valeri, per avermi inserito nel progetto e per avermi sempre indicato la strada da seguire.

Infine ringrazio davvero di cuore i miei genitori, che mi hanno costantemente sostenuto, e Serena, che mi è stata sempre vicina.