UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Corso di Laurea in Informatica

Progetto e Sviluppo di un Modulo AMP e di un Importatore di Documenti Tramite il Software Documentale Alfresco

Alessio Masola

Tesi di Laurea

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2015/2016

RINGRAZIAMENTI

Il primo ringraziamento va all'Ing. Riccardo Martoglia, il quale assieme ad Enrico Diena, mi ha assistito nel tirocinio, nella ricerca di informazioni e nella stesura della tesi; sono veramente grato della disponibilità presentata nel corso di questi mesi. Ringrazio di cuore i miei genitori che hanno sempre creduto in me e appoggiato ogni mia scelta, offrendomi la possibilità di laurearmi in informatica, la disciplina che mi ha da sempre affascinato. Un altro speciale ringraziamento va a tutti i miei amici e colleghi, sia quelli di una vita che quelli conosciuti nell'ambito universitario, che mi sono sempre stati accanto. L'ultimo ma non meno importante ringraziamento lo rivolgo alla mia ragazza, Maria, la quale in questi ultimi quattro anni di studio mi ha supportato e sopportato, aiutandomi a non arrendermi davvero davanti a niente. Grazie a tutti per la persona che mi avete fatto diventare e per avermi accompagnato in questo percorso di crescita.

Indice

Parte I – Il Caso di studio

1 Obiettivo del progetto e Software Documentale 2

1.1 Introduzione a un software documentale 3
1.2 Governo e sicurezza 3
1.3 Creazione e cattura 4
1.4 Ricerca e gestione 4
1.5 Organizzazione e distribuzione 4
1.6 Il Software documentale Alfresco 5
1.7 Versioni di Alfresco 5
1.8 Composizione di Alfresco 5
1.9 Content Repository 6
1.10 Content Application Server (alfresco.war) 6
1.11 Client 7
1.12 Alfresco e i Nodi 7
1.13 Tipi e Aspetti 8
1.13.1 Proprietà 8
1.14 Permessi dei nodi 9
1.15 Regole e Behaviours 10
1.16 Spazi di archiviazione 10
1.17 Ciclo di vita dei contenuti 10
1.17.1 Fasi di creazione di un Nodo: 11
1.17.2 Fasi di cancellazione di un Nodo: 11
1.18 WebScript e JavaScriptAPI 11
1.19 Repository Extension Point 12
1.20 CMIS 12

Parte II – Progettazione e Implementazione

2 Modellazione dei Documenti 14

2.1 Metodologie di Lettura di Metadati 15
2.2 Modellazione 15
2.3 Repository interni e localizzazione dei file 17
2.4 Codici del modello 19
2.5 Definizione Xml e Import 19
2.6 Constraint 20
2.7 Tipi di documenti 21
2.8 Definizione degli aspetti 23
2.9 Creare un collegamento tra Repo e Share 28
2.10 Codice Share 29
2.11 Sezione DocumentLibrary 29
2.12 Sezione Form per le ricerche avanzate (AdvancedSearch) 30
2.13 Sezione Creazione dei nodi 31
2.14 Sezione di ricerca avanzata e Form 32
2.15 Constraint e Label Localizzate 33
2.16 Le Behaviour 36
2.17 Problema affrontato 36
2.18 Policies dei nodi 37
2.19 La classe Java implementata 38
2.20 Configurare lo Spring Bean 40

3 Progetto All in One (AIO) 41

3.1 Implementazione delle specifiche per il cliente 42
3.1.1 Implementare le specifiche per la parte Repo 42
3.2 DictionaryBootstrap AMP dell'All In One 44
3.3 Implementare le modifiche sulla parte Share 44
3.3.1 Creazione dell'interfaccia di visualizzazione, modifica e ricerca 45
3.4 Moduli di estensione Surf 47
3.5 Creazione dei moduli d'estensione 47
3.6 Aggiunta dei moduli d'estensione tramite Share 49

4 Caricamento Massivo di Documenti 51

4.1 Importatore 51
4.1.1 Funzionamento generale 51
4.1.2 Funzionamento logico con workflow 53
4.2 Creazione e composizione interna dell'importatore 54
4.2.1 I file Config.properties 55
4.2.2 I file templates.ftl 56
4.3 Classi Java dell'importatore 57
4.3.1 Classe **ImportatoreDocumenti.java** 57
4.3.2 Classe **ConfigurazioneImporter.java** 57
4.3.3 Classe **ReadDocAndInsertTask.java** 57
4.3.4 Classe **GenericFileNameFilter.java** 58
4.3.5 Classe **ImporterAlfrescoUtils.java** 58
4.3.6 Classe **ManageDocument.java** 58
4.3.7 Classe **ImportableDocument.java** 58

Appendici

5 Appendice A 61

5.1 Installazione di Alfresco (su Windows 7/8/10) 61
5.2 Software Maven. 61
5.2.1 I componenti principali di Maven: 62
5.3 Impostare le variabili d'ambiente per Maven 62
5.4 Creazione Primo Progetto con Maven 63
5.4.1 L' archetipo 63
5.5 Configurazione Progetti in Eclipse per Repo e Share 64
5.5.1 Run Configuration 64

6 Appendice B 68

6.1 Indici, Ricerche e token in Alfresco 68
6.2 Creazione del progetto AIO 71
6.3 Importazione del progetto AIO in Eclipse 71
6.4 Creazione del Run (AIO) 72
6.5 Installazione di Springloaded 73
6.6 Installazione di un modulo AMP tramite linking al progetto AIO 74
6.6.1 Aggiungere la dipendenza (repo) 74
6.6.2 Aggiungere l'overlay (repo) 75
6.6.3 Aggiungere la dipendenza (share) 75
6.6.4 Aggiungere l'overlay (share) 75

7 Bibliografia 76
8 Collegamenti online per i file 77

Parte I

Il caso di studio

Capitolo 1

1 Obiettivo del progetto e Software Documentale

Il cliente PA richiede di avere un proprio Software gestionale per la sua azienda. Dal cliente vengono date specifiche in base ai documenti che deve archiviare e delle specifiche di ogni documento. La realtà aziendale dei documenti del nostro cliente si suddivideva in:

- Fatture Passive: tutte quelle fatture che hanno un costo fisso all'interno dell'azienda (come bolletta della luce, servizi di internet …).
- Fatture Attive: è una fattura emessa dall'azienda per qualche cliente.
- Documenti Di Trasporto Attivi: viene emesso questo tipo di documento per giustificare e tenere tracciato il trasferimento di un materiale da cedente a cessionario attraverso il trasporto dello stesso, sia che lo stesso venga effettuato in conto proprio dal mittente, sia che lo stesso venga affidato ad un trasportatore.
- Documenti Di Trasporto Passivi: stessa cosa dei documenti di trasporto attivi, ma è fatto dal destinatario al mittente.
- Documenti di ordine di Acquisto
- Documenti di ordine di Vendita

Successivamente si parlerà di quali proprietà dei documenti il cliente ha richiesto di acquisire per ciascun documento. Il cliente PA ha inoltre la necessità di dover caricare documenti massivamente, quindi richiede un importatore automatizzato che prelevi i file salvati in una specifica cartella e li carichi nel suo software gestionale Alfresco.

1.1 Introduzione a un software documentale

In questo capitolo verrà presentato il funzionamento di un software documentale generico con i relativi componenti base e verrà poi presentato il software Alfresco utilizzato per lo sviluppo del progetto.

Figura 1.1

Un Software documentale (come in Figura 1.1) si può dividere in quattro sezioni:

- Governo e Sicurezza
- Creazione e Cattura
- Ricerca e Gestione
- Organizzazione e Distribuzione

Oltre a queste quattro sezioni c'è il cuore centrale di un documentale, ossia il documento più i suoi metadati aggiunti (proprietà con valori aggiuntivi).

1.2 Governo e sicurezza

Con l'introduzione delle recenti normative sulla **conservazione sostitutiva**, il documento informatico acquista valore probatorio ai fini fiscali e legali. Infatti, con l'introduzione del Decreto del 23 gennaio 2004 del Ministero dell'Economia e delle Finanze e con la Deliberazione del Centro Nazionale per l'Informatica nella Pubblica Amministrazione nº 11 del 19 febbraio 2004, oggi è possibile archiviare e conservare su supporti ottici i documenti cartacei e, utilizzando la firma elettronica qualificata e la marcatura temporale

(permette di associare data e ora certe e legalmente valide ad un documento informatico), si può anche decidere di eliminarli. Negli ultimi tempi l'interesse verso le soluzioni di Document Management è aumentato in maniera significativa in Italia dovuto anche alla recente legge che autorizza, secondo certi criteri ben definiti, l'utilizzo di metodologie di archiviazione ottica sostitutiva e trattamento dei documenti anche di quelli considerati a valore fiscale (es. fatture, libro IVA, ecc.). Si tratta della cosiddetta legge "Bassanini" seguita dalle varie norme emesse dal CNIPA che garantiscono ai documenti informatici piena validità ai fini probatori, legali e fiscali e, dunque, possono pienamente sostituire i tradizionali metodi di conservazione delle registrazioni aziendali. Questo provvedimento apre innumerevoli vantaggi per le aziende e non solo: basti pensare alla riduzione degli spazi di conservazione dei documenti, all'incredibile risparmio cartaceo, all'abbattimento dei costi "nascosti" di gestione, alla velocità di ricerca dei documenti a distanza e molto altro. Utilizzando la conservazione elettronica, viene utilizzato un file (file di log), che permette di tener traccia di tutte le azioni che vengono svolte sul database, l'account delle persone che hanno svolto queste azioni e tutti gli eventi che sono accaduti a livello di database.

1.3 Creazione e cattura

Per poter utilizzare un documento nel nostro software documentale, dobbiamo avere una modalità che sia capace di caricare, creare e catturare i nostri file. Per la creazione e la cattura di questi documenti, vengono utilizzati dei modelli che, come delle maschere, vengono sovrapposte ad essi e riescono a carpire le informazioni necessarie per completare i metadati, troviamo anche la scannerizzazione manuale, l'importazione di cartelle, tools , allegati di e-mail certificate, ecc.

1.4 Ricerca e gestione

La parte di ricerca e gestione è una parte molto utilizzata. Questa sezione ci dà la possibilità di effettuare ricerche tramite dei tag, ossia di avere una suddivisione interna in base al tag che ha un documento, di effettuare dei report (relazioni) sui nostri dati (come avere un bilancio interno delle spese effettuate), la possibilità di revisionare documenti datati e di fare ricerche tramite indici tokenizzati o meno (più avanti si riparlerà nello specifico di questi indici).

1.5 Organizzazione e distribuzione

Parte delle notifiche è gestita in questa zona di documentale. L'utilizzo di notifiche per aiutare chi ha una grande mole di documenti è molto utile per evitare di incappare in informalità legali. Questa parte gestisce l'invio di mail tramite l'SMTP (Simple Mail Transfer Protocol), messaggi, relazioni finali e note. Sempre al suo interno vi è la suddivisione logica tra i documenti di vario tipo, utilizzando una metodologia a fascicoli, come una gerarchia di cartelle che tiene suddivisi i documenti di vario genere.

1.6 Il Software documentale Alfresco

Alfresco è un software documentale utilizzato per archiviare, modellare e cercare dei documenti. L'archiviazione viene svolta in formato digitale, tenendo conto di regole e logiche che possono essere automatizzate. La modellazione permette di conservare e di aggiungere Metadati specifici che riguardano i documenti e infine permette di cercarli in modo preciso attraverso il loro utilizzo.

1.7 Versioni di Alfresco

Alfresco è un Enterprise Content Management (ECM) disponibile in due versioni:

- Alfresco Community Edition: software libero, pubblicato sotto la licenza Open Source GNU/GPL che permette ai programmatori di creare nuove piattaforme per i clienti, nuovi moduli per recuperare i metadati ecc.
- Alfresco Enterprise Edition: la versione commerciale a pagamento che fornisce servizi aggiuntivi all’utente o programmatore che sceglie questa versione. Ad esempio in questa versione ci sono svariati moduli capaci di ricavare direttamente i dati dai documenti.

1.8 Composizione di Alfresco

Tomcat è un software che va a comporre Alfresco. La sua funzionalità di spicco è quella del Web Application Server, ovvero un server capace di gestire e supportare le pagine JSP (JavaServer Pages) e fornire altri tipi di supporto. Alfresco quindi, è composto da Tomcat con all'interno 3 web app che si suddividono in:

1. alfresco.war
2. share.war
3. solr.war

In aggiunta a queste 3 web app, c'è un DataBase e una cartella su FileSystem sulla quale salvare i contenuti binari(alf_data). L'utilizzo del FileSystem permette, oltre a non appesantire il Database tramite una enorme mole di dati, anche l'aggiunta e l'utilizzo dei metadati.

Si suddivide in vari macro livelli architetturali:

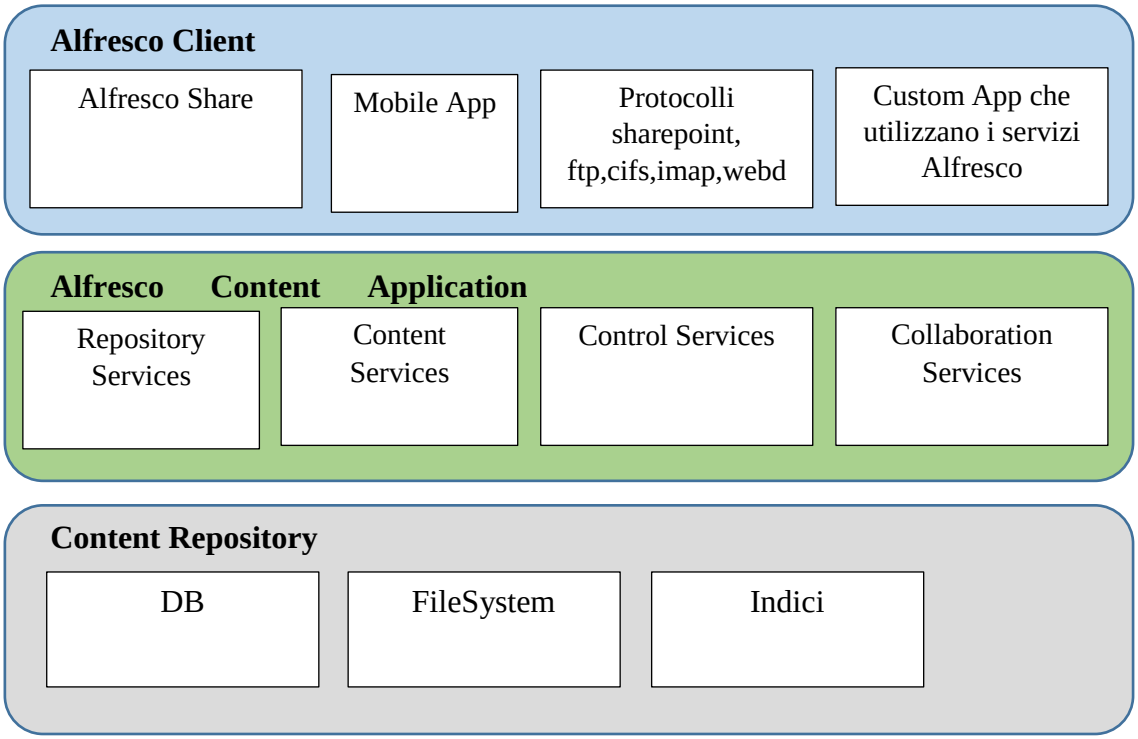


Figura 1.2

1.9 Content Repository

Il Content Repository è l'archivio vero e proprio di Alfresco, è composto da:

- un **DataBase** relazionale utilizzato per il salvataggio dei Metadati, ma sfruttato anche per componenti applicative come l'ACL, processi WorkFlow ecc.
- **ACL**: (Access Control List) è un meccanismo che viene usato per esprimere regole (anche complesse) che permettono o negano l'accesso ad alcune risorse del sistema da parte dei suoi utenti.
- il **FileSystem** (alf_data) su cui salvare i File binari dei documenti archiviati
- un **indice** (solr.war) che consente le ricerche rapide tra i metadati

1.10 Content Application Server (alfresco.war)

Il Content Application Server è l'applicazione che contiene tutti i servizi per interagire con l'archivio (Content Repository). Tutti i servizi vengono poi categorizzati in 4 grandi blocchi di services:

- **Repository**: insieme dei servizi fondamentali per poter interagire con il Content Repository.
- **Content**: regole, azioni, trasformazioni, estrapolazione di metadati e servizi di ThumbNailing (impronta digitale).

- **Control**: WorkFlow
- **Collaboration**: Sito, inviti di servizio, attività ecc.

1.11 Client

Sono tutte quelle applicazioni che, utilizzando i servizi, forniscono all'utente la possibilità di interagire con i contenuti.

Tipo:

- **Alfresco Share** (share.war): è la web app che consente di utilizzare Alfresco.
- **Mobile App** (per Android e IOS)
- **Protocolli** come FTP, WebDAV, SharePoint possono esser utilizzati per poter navigare e caricare i documenti sul Repository.

1.12 Alfresco e i Nodi

Un concetto nuovo che Alfresco porta è proprio il concetto di Nodo. Per Alfresco tutto può essere visto attraverso un nodo, il nodo è quindi considerata come un'entità di base. Sono entità di base le cartelle, i documenti, gli utenti ed i gruppi.

Tra questi nodi ci possono esser 2 tipi di relazione (come in figura 1.3):

- **Peer - Peer**
- **Parent - Child**

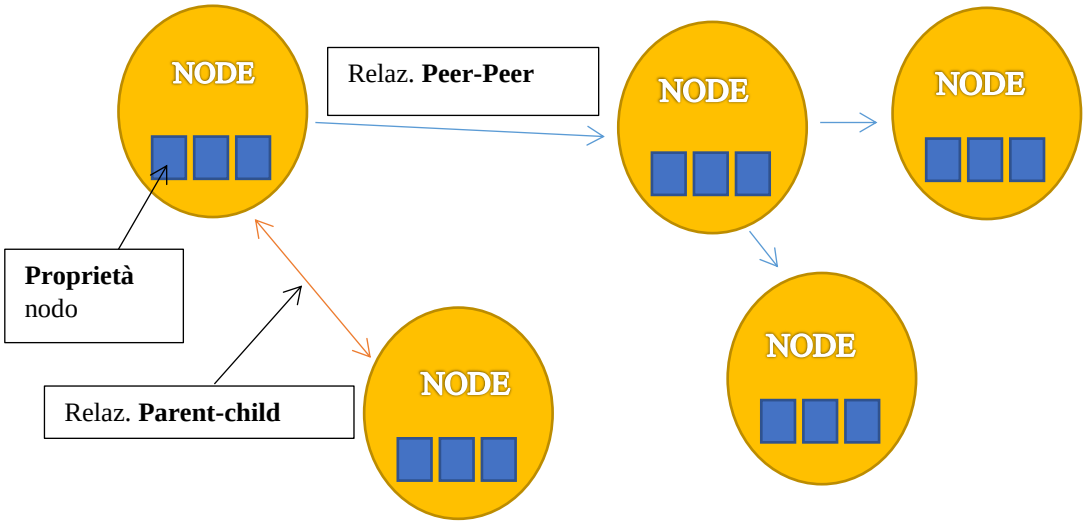


Figura 1.3

La relazione **Peer-Peer** è una relazione "unilaterale" che non permette la cancellazione a cascata.

Si parlerà di Relazione **Parent-Child** quando vi è una relazione "Bilaterale" Padre-Figlio in cui se il padre viene cancellato, automaticamente anche il Figlio verrà cancellato.

I nodi hanno 2 proprietà di base:

- **NodeID**: che indentifica in modo univoco il nodo all'interno del repository.
- **NodeRef**: è L'URI (Uniform Resource Identifier) che rappresenta il percorso assoluto all'interno del repository di quel Nodo.

Oltre a queste due proprietà, si aggiungono delle proprietà in base al Tipo o all'Aspetto del nodo. Non si possono assegnare delle proprietà se non sono presenti all'interno del Tipo o dell'Aspetto che ha un determinato nodo.

1.13 Tipi e Aspetti

I tipi e gli aspetti definiscono quali proprietà avrà il nodo a cui vengono applicati. Ogni nodo deve avere un Tipo associato (solo un tipo come quelli base, che sono cm:content e cm:folder).

Ogni nodo può avere 1 o più Aspetti applicati che possono anche essere rimossi in futuro, invece una volta assegnato un Tipo ad un nodo, esso non può cambiare, ma solo diventare un tipo "figlio" di quello attualmente associato.

1.13.1 Proprietà

Le proprietà hanno un nome univoco in modo da evitare conflitti e in più gli viene associato anche un NameSpace di riferimento (concetto simile al package di Java per risolvere i conflitti) e una tipologia (tipo: Stringa, Numero, Booleano...). A ogni proprietà posso associargli un Constraint, che obbliga l'immissione di soli valori validi, come ad esempio: la lunghezza massima di una stringa o un minimo di caratteri che questa stringa deve avere, quindi rispettare un determinato formato/pattern.

Se definissi il tipo di un nodo come "cm:content" (contenuto base), tutti i documenti che verranno inseriti saranno di default di questo tipo , ossia tra le sue proprietà avrà il "cm:content" che sarà un puntatore al file binario presente sul FileSystem.

Definire e creare tutti questi Tipi, Proprietà e relative constraint, associazioni ed aspetti viene definito come "**Modellare i documenti**".

Esempio sui Tipi (figura 1.4):

*type:**DocumentoFattura** extend type:**DocumentoGenerico***

*type:**DocumentoGenerico** extend cm:**content***

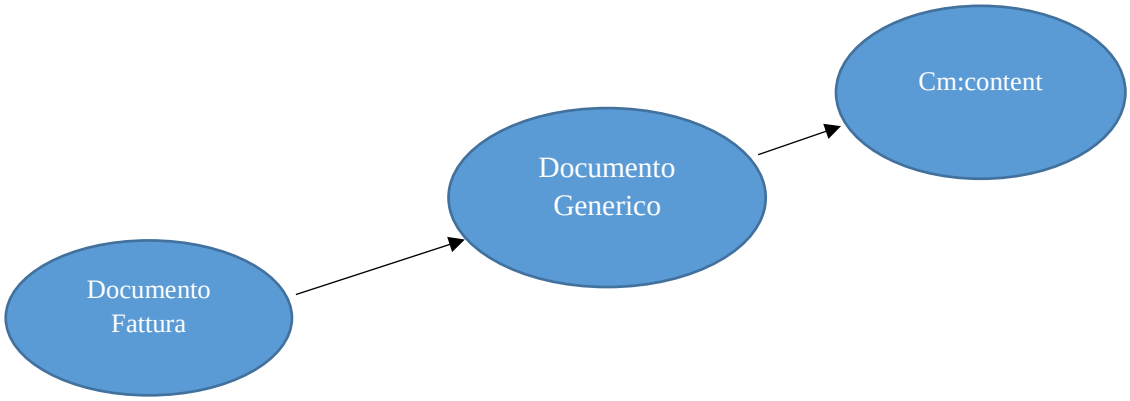


Figura 1.4

Nel caso in cui dovessi avere un nodo che è di tipo ***cm:content***, potrà diventare un **DocumentoGenerico** o **DocumentoFattura**, ma non potrà mai tornare a ridiventare un ***cm:content***. A volte può tornare utile aver definito degli aspetti che sono "senza proprietà" in modo di riuscire a "taggare" alcuni documenti. Infine si potranno eseguire delle Query andando a chiedere tutti i nodi che sono di un determinato tag.

1.14 Permessi dei nodi

Per mantenere la sicurezza sui vari Nodi, viene utilizzato un modello di Spring Security basato sulle ACL (Access Control List). Esso sostanzialmente si basa su Ruoli, Gruppi e Utenti, ed azioni che possono eseguire, come leggere un nodo, aggiornarlo, leggere le proprietà, modificarle ecc. Queste Security sono "Atomiche" sul nodo, ossia ogni proprietà subirà modifiche e aggiornamenti in base a chi ha il permesso. In modo semplificato: se si ha il permesso di poter modificare le proprietà di un nodo, il diritto sarà esteso a tutte le sue proprietà.

Una volta applicati i diritti a un nodo, essi finiranno anche a cascata sulle versioni precedenti di quel nodo (analogamente per le successive).

La lista dei ruoli è data da:

- **Consumer**: ha la possibilità di sola lettura.
- **Editor**: può solo modificare delle proprietà e non può aggiungerle.
- **Contributor**: non può modificare le proprietà, può aggiungere delle proprietà e modificarle.
- **Collaborator**: possono editare il contenuto dei documenti propri e non propri.
- **Coordinator**: può fare di tutto, tranne cambiare il Proprietario che ha creato il documento
- **Manager**: è un ruolo presente solo nei siti. Esso è colui che gestisce le Repository del sito di cui è manager, in più supporta la parte di collaborazione e condivisione di Link che riguardano parti che potrebbero servire ai collaboratori del progetto.

Questo ruolo è molto utile perché potrebbe guidare la fase di progettazione, rendendola molto più rapida e dando consigli sulla documentazione da guardare o sul come risolvere determinati errori.

- **Owner**: è colui che ha creato il documento, ossia può fare di tutto su quel determinato documento. NB: il Contributor potrebbe diventare Owner se carica un documento proprio.
- **Administrator**: colui che può fare di tutto.

Fortunatamente nel caso in cui abbiamo la necessità di nuovi ruoli c'è la possibilità di crearne di nuovi.

1.15 Regole e Behaviours

Su ogni nodo c'è la possibilità di definire dei Listener in modo da eseguire determinate azioni allo scattare di determinati eventi. Per esempio: la modifica di un contenuto potrebbe mandare una e-mail al personale notificando tale avvenimento. Avremo quindi:

- **Regole** (Action): sono definite tramite un'interfaccia grafica, possono esser create dinamicamente e possono esser applicate solo alle cartelle.
- **Behavior**: possono essere codificate in Java, sono più potenti, possono stare in ascolto di più eventi rispetto alle regole e sono applicabili su qualsiasi nodo.

1.16 Spazi di archiviazione

I contenuti vengono salvati principalmente in tre spazi differenti a seconda del tipo di nodo. Gli spazi quindi servono per mantenere una separazione logica fra i nodi.

Questi tre spazi sono:

- **Workspace**: è lo spazio dove risiedono tutti i contenuti utilizzati e in gergo detti ancora "vivi; tutte le ricerche agiscono solo su questo spazio.
- **Archive**: è lo spazio in cui vengono spostati i contenuti cancellati (spostati nel cestino)
- **Version**: è lo spazio dove sono tenute le versioni dei vari documenti

1.17 Ciclo di vita dei contenuti

Verrà spiegato brevemente cosa succede tecnicamente quando il contenuto viene caricato o cancellato su Alfresco.

1.17.1 Fasi di creazione di un Nodo:

1. I metadati vengono salvati su DB all'interno del Workspace.
2. Il file viene caricato e rinominato come .bin sul FileSystem (es di rinominazione: alf_data/contentstore/YYYY/MM/DD/hh/mm/url-id-of-the-file.bin).
3. Vengono inseriti i valori sull'indice.
4. Se il contenuto può presentare una Thumbnail, viene chiamato il servizio di trasformazione e una thumbnail viene creata e associata al nodo.

1.17.2 Fasi di cancellazione di un Nodo:

i. Il documento (o cartella) viene spostata nel cestino, il contenuto e tutti i suoi figli vengono spostati (solo a livello di DB) dallo spazio "Workspace" a quello "Archive", gli indici verranno spostati di conseguenza anche loro da "Workspace" ad "Archive".

ii. Il cestino viene svuotato. Il contenuto viene marcato come "Orfano". Sul Database il nodo acquista la proprietà orphan_time, ma non succede altro! Sia su Database che su FileSystem il documento non viene ancora modificato. Si passerà alla terza fase solo un periodo di tempo definito dalla proprietà system.content.orphanProtectDays (di default sono 14 giorni).

iii. Questo è l'ultimo passaggio che viene eseguito, ossia un job schedulato si occupa di verificare i nodi che sono marchiati come orfani e se superano il tempo specificato precedentemente, i nodi saranno eliminati dal Database, il file .bin verrà spostato nella cartella alf_data/contentstore.deleted e verranno rimossi i riferimenti dagli indici per tale nodo.

Alfresco con questo sistema di cancellazione dei nodi lascia sempre la possibilità di recuperare il file binario caricato, ma con una grande mole di dati bisogna stare attenti al crescere delle dimensioni della cartella alf_data/contentstore.deleted .

1.18 WebScript e JavaScriptAPI

La tecnologia WebScrpit è stata creata da Alfresco per consentire di realizzare degli script che rimangono in ascolto di determinate URL e che riescono a restituire un risultato in vari formati testuali come HTML, XML o JSON.

Di base si compone con:

- Un descrittore XML che definisce l'URL a cui sta in ascolto e quali variabili deve recuperare da esso.
- Un file.js (Javascript) che viene eseguito quando viene chiamato il webscript.
- Un file.ftl che rappresenta il Template da utilizzare per l'output del webscript.

Stando in ascolto di determinate URL questa tecnologia può richiamare degli script realizzati in Javascript, i quali possono interagire con Alfresco tramite un set molto ampio di API. Si va dalla creazione di nodi, al popolamento delle proprietà, alla possibilità di chiamare azioni su tali nodi, di eseguire ricerche, impostare permessi, e fornire come

risposta un output che può essere HTML o, più facilmente, JSON (così da poter consentire integrazioni con sistemi esterni).

1.19 Repository Extension Point

I punti di estensione più comuni del Repository (tralasciando quindi l'interfaccia utente) sono:

- **Content Models**: creazione di nuovi tipi o aspetti.
- **Content Behaviour**: creazione di listener custom sui tipi e gli aspetti gestiti.
- **Process Definition**: creazione di processi di Workflow.
- **Actions**: le azioni sono classi java che eseguono operazioni sui nodi, sono azioni l'invio di una mail con il nodo allegato, l'applicazione di un tag, la creazione di uno zip con tutti i contenuti di una folder o la lettura di un barcode all'interno del nodo.

1.20 CMIS

Nell'ottica di utilizzare "Alfresco come un Servizio" è importante prendere in considerazione l'uso di CMIS.

Semplificando, potremmo dire che CMIS è l'SQL dei Documentali, esso è quindi un insieme di API standard comune a tutti i Documentali che lo implementano, che consente di inserire, modificare, cancellare e cercare documenti all'interno del repository. Oltre allo standard CMIS, è possibile utilizzare una serie di webservice REST per coprire tutte quelle funzionalità specifiche di Alfresco che non vengono coperte dallo standard.

Parte II

Progettazione e Implementazione

Capitolo 2

2 Modellazione dei Documenti

Come accennato precedentemente nel capitolo 1, ciò che porta a modellare i documenti in modo elettronico è la legge di Archiviazione Sostitutiva che include un insieme di processi informatici regolamentati nella legislazione italiana, ai fini di **garantire nel tempo la validità legale di un documento elettronico.**

Come accennato all'inizio del capitolo 1 si sta modellando una realtà aziendale del cliente che ha bisogno di modellare questi tipi di documenti con le relative informazioni specificate:

Ciclo Passivo e Attivo delle Fatture:

- Fatture Passive (FP) con il numero della fattura, la data della fattura, la ragione sociale del fornitore, il codice del fornitore, l'IVA del fornitore, il codice fiscale del fornitore e il tipo di documento.

- Fatture Attive (FA) con il numero della fattura, la data fattura, la ragione sociale del cliente, il codice del cliente, l'IVA del cliente, il codice fiscale del cliente e il tipo di documento.

I Documenti di trasporto (chiamati anche DDT) sono anch'essi di due tipi:

- Attivi con il numero della fattura, la data della fattura, la ragione sociale del fornitore, il codice del fornitore, l'IVA del fornitore e il codice fiscale del fornitore.

- Passivi con il numero della fattura, la data fattura, la ragione sociale del cliente, il codice del cliente, l'IVA del cliente e il codice fiscale del cliente.

I Documenti di ordine di:

- Acquisto (ODA) con il numero della fattura, la data della fattura, la ragione sociale del fornitore, il codice del fornitore, l'IVA del fornitore e il codice fiscale del fornitore.

- Vendita (ODV) con il numero della fattura, la data fattura, la ragione sociale del cliente, il codice del cliente, l'IVA del cliente e il codice fiscale del cliente.

Nel modellare questi documenti nascono le difficoltà date dalle tipologie. Tenendo conto che le fatture passive sono interne all'azienda che le emette, si avrà la possibilità e la capacità di riuscire a standardizzare i documenti in modo da gestirli con determinate metodologie prefissate dall'inizio. La tipologia dei documenti passivi invece, sono più difficili da gestire perché il gestore potrebbe non avere degli standard per i suoi documenti (come ad esempio il salvataggio del documento attraverso un determinato formato: "NumeroFattura_DataFattrura_..."), quindi avere diverse aziende implica avere diversi standard.

2.1 Metodologie di Lettura di Metadati

Ci sono varie possibilità tra cui:

1. Si possono prendere i metadati di un documento anche attraverso il nome dell'allegato di una e-mail e successivamente utilizzarli.
2. Posso scannerizzare il documento, utilizzare un modulo implementato in Alfresco chiamato ERP (Enterprise Resource Programming: è un gestionale che genera dei documenti digitali), che consente di trasformare l'immagine in un documento con solo testo e dopo ho due tipi di soluzioni:
    - Una volta registrato, ho un ID e ho un BARCODE che andrò successivamente a riprendere per i metadati.
    - Utilizzare lo standard del documento fornito dell'azienda (viene utilizzato per aziende grandi come Telecom, Fastweb, Infostrada ecc.).

2.2 Modellazione

Dobbiamo tenere conto che avremo dei File ed associato ad esso un File Indice, ossia un file testuale, con più informazioni tramite una coppia di Chiave-Valore. All'interno avremo dei vincoli imposti che permetteranno l'inserimento solo di alcuni tipi di valori. Questi vincoli sulle proprietà vengono chiamati Constraint. Nella modellazione le proprietà di un documento assumeranno un determinato valore. Questo tipo di modellazione ricorda molto la modellazione Object-Oriented di java, con le classi che realizzano i tipi di documenti e le relative proprietà che forniscono le informazioni che

vogliamo mantenere di quel documento. Come si può notare ci sono sia Numero Fattura e la data di fattura in tutti i documenti, andando a modificarlo si può creare un Aspetto chiamato Documento IVA che avrà queste due proprietà (come in figura 2.1). Verrà aggiunta anche la proprietà Anno, che indicherà l'anno del documento e darà la possibilità, a chi utilizzerà il software, di poter effettuare ricerche dei documenti anche solo in base all'anno. In fase di progettazione è molto importante evitare di creare Tipi di documento specialmente quando potrebbero esser modellati in diverso modo con più Aspetti. Potremmo modellare con un unico aspetto sia i riferimenti del Cliente che quelli per il Fornitore, ma per scelte progettuali abbiamo deciso di tenere due aspetti divisi come in figura 2.1.

Pensiamo di modellare i nostri documenti attraverso l'UML in figura 2.1.

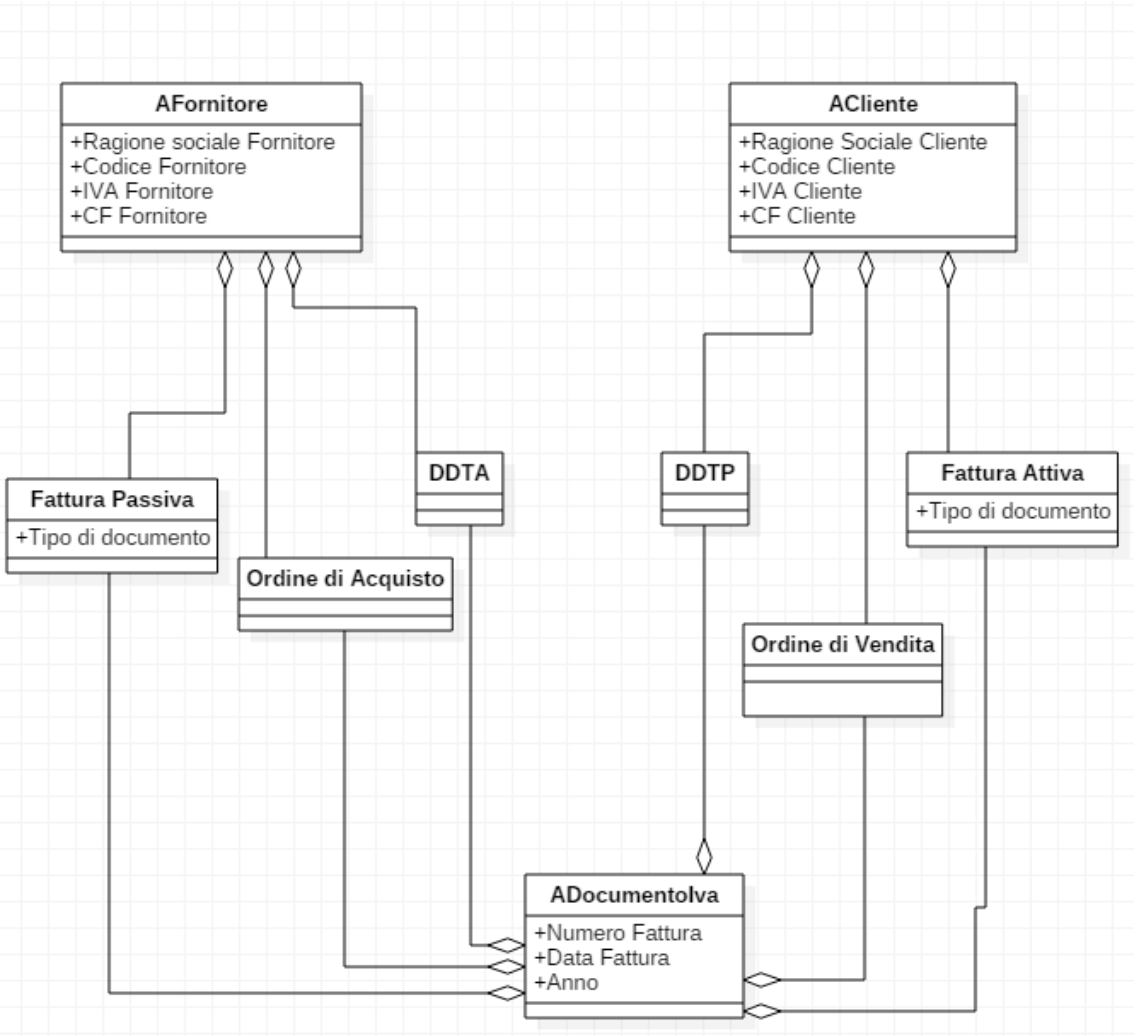


Figura 2.1 : Grafico UML di progettazione

2.3 Repository interni e localizzazione dei file

Prima di andare ad esporre i codici del modello è essenziale avere una mappa visiva di dove si trovino i file da modificare (figura 2.2 e 2.3).

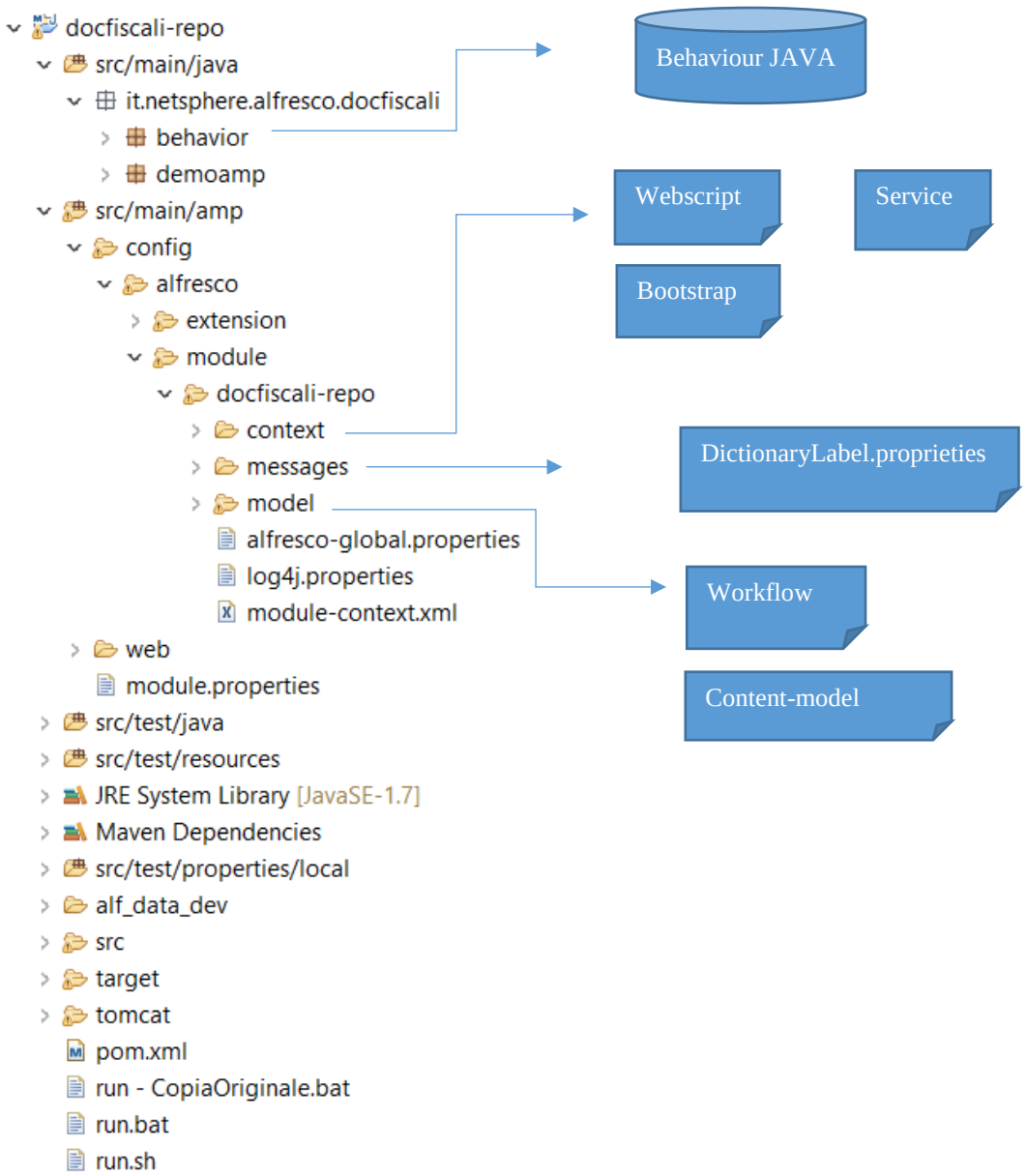


Figura 2.2 : Mappa dei file all'interno del progetto Repo

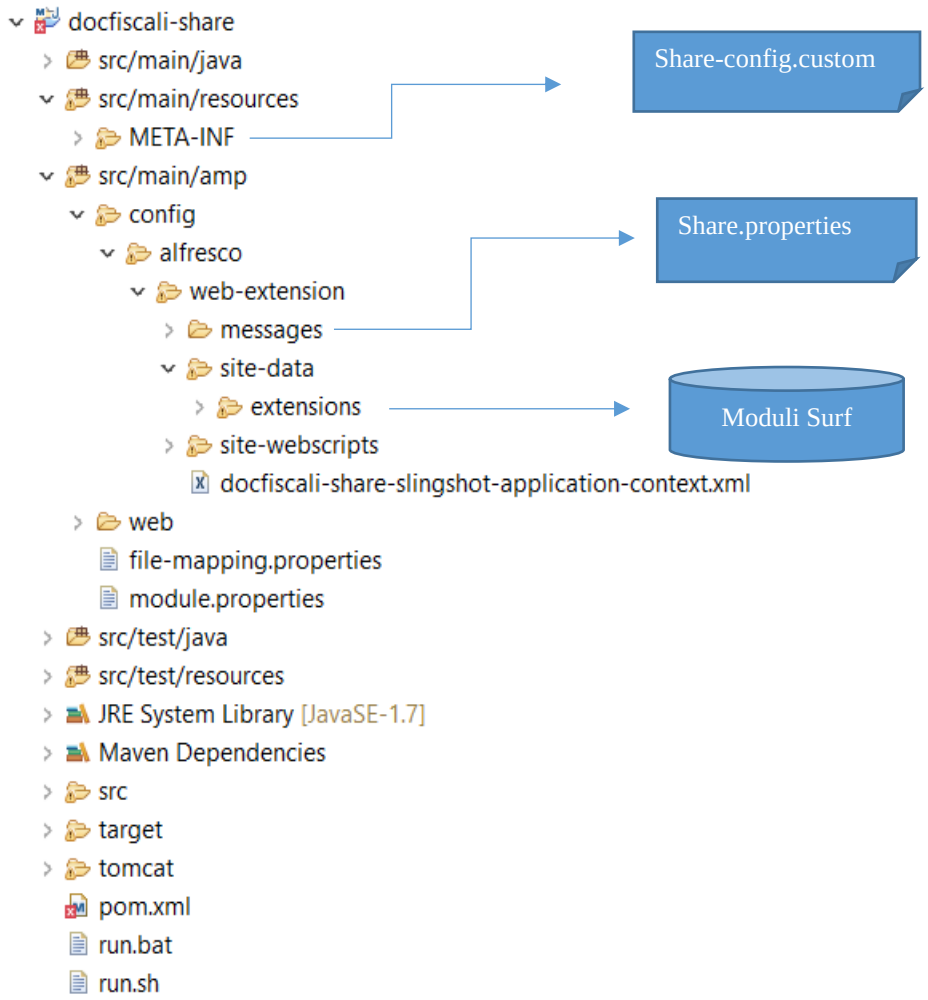


Figura 2.3 : Mappa dei file all'interno del progetto Share

2.4 Codici del modello

Per creare effettivamente il modello, si deve andare a modificare il file XML “content-model.xml” presente in: “C:\progetti\docfiscali-repo\src\main\amp \config\alfresco\module\docfiscali-repo\model” (visualizzazione in modalità progetto in figura 2.4).

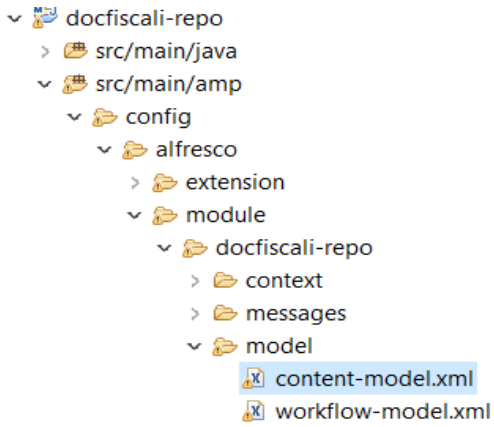


Figura 2.4 : posizionamento del file Content-model.xml

2.5 Definizione Xml e Import

Nelle successive prime righe di codice (figura 2.5), viene importato il dizionario, i modelli di base, il modello del sistema e viene impostato il nostro namespace con prefisso “dfm” che permetterà di evitare errori di conflitto. Il namespace ci aiuta a risolvere le disambiguità con altre variabili che potrebbero esser già presenti nel servizio Repo di Alfresco o nel caso in cui il modulo AMP verrà installato da qualche altro cliente.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <model name="dfm:contentModel" xmlns="http://www.alfresco.org/model/dictionary/1.0">
3
4   <!-- ********************************************************************
5   Optional meta-data about the model*****************************************************************************-->
6           <description>Document Model for Repo Extension Documenti Fiscali</description>
7           <author>Enrico Diena</author>
8           <version>1.0</version>
9
10          <imports>
11  <!-- ********************************************************************
14                  <import uri="http://www.alfresco.org/model/dictionary/1.0" prefix="d"/>
15
16  <!-- ********************************************************************
19                  <import uri="http://www.alfresco.org/model/content/1.0" prefix="cm"/>
20
21  <!-- ********************************************************************
24                  <import uri="http://www.alfresco.org/model/system/1.0" prefix="sys"/>
25          </imports>
26
27  <!-- ********************************************************************
30          <namespaces>
31                  <namespace uri="http://www.castgroup.it/model/docfiscali/1.0"  prefix="dfm" />
32          </namespaces>
```

Figura 2.5

2.6 Constraint

Vengono definiti 2 tipi di Constraint, uno per le fatture attive e l'altro per le fatture passive. In questi Constraint viene utilizzato il parametro allowedValues che obbligherà l'utente, al momento dell'inserimento, ad usare solo quella determinata lista di valori (figura 2.6).

```
        <constraints>

<!-- ********************************************************************
constraint per il tipo del documento di fattura passiva (usare codici poi file di properties)
*************************************************************************-->
                <constraint name="dfm:CTipoDocumentoFtPassiva" type="LIST">
                        <title>Tipo Documento Ft</title>
                        <parameter name="allowedValues">
                                <list>
                                        <value>FatPass</value>
                                        <value>NotaAcc</value>
                                        <value>BollaDog</value>
                                        <value>AvvisoPag</value>
                                        <value>Varie</value>
                                </list>
                        </parameter>
                        <parameter name="caseSensitive">
                                <value>false</value>
                        </parameter>
                </constraint>

<!-- ********************************************************************
Constraint per il tipo del documento di fattura attiva
*************************************************************************-->
                <constraint name="dfm:CTipoDocumentoFtAttiva" type="LIST">
                <title>Tipo Documento Ft Attiva</title>
                <parameter name="allowedValues">
                                <list>
                                        <value>FattAtt</value>
                                        <value>NotaAcc</value>
                                        <value>Varie</value>
                                </list>
                        </parameter>
                        <parameter name="caseSensitive">
                                <value>false</value>
                        </parameter>
                </constraint>
        </constraints>
```

Figura 2.6 : definizione dei Constraints

2.7 Tipi di documenti

Precedentemente non abbiamo parlato del documento generico base. Solitamente conviene avere un documento generico come in figura 2.7 per poter estendere gli altri tipi di documenti. Se nel futuro dovessimo aggiungere una proprietà che tutti i nostri tipi di documenti necessitano, essa potrà esser aggiunta semplicemente in questa sezione di codice senza andare ad intaccare il codice di tutti gli altri tipi di documento.

```
<types>

<!-- ********************************************************************
Tipo Documento Generico di base  Eredita da cm:Content
*********************************************************************-->
        <type name ="dfm:TDocGenericodfmBase">
                <title>Documento generico dfmBase</title>
                <description>Documento generico dfmBase</description>
                <parent>cm:content</parent>
        </type>
```

Figura 2.7 : Codice del Documento Generico

Di seguito (figura 2.8 , 2.9 e 2.10) ci sono le implementazioni di tutti gli altri tipi di documento e il codice relativo per aggiungere gli aspetti ad essi.

```
*******************************************************
attura Passiva      extends TDocGenericodfmBase
umentoFtPassiva :: text     indica tipo di documento di fattura passiva
ry Aspect:     ADocumentoIVAdfmBase    ;;   AFornitore
*******************************************************-->
<type name="dfm:TDocFatturaPassiva">
        <title>Fattura Passiva</title>
        <description>Fattura Passiva Modellazione Finanza</description>
        <parent>dfm:TDocGenericodfmBase</parent>
        <properties>
                <property name="dfm:tipoDocumentoFtPassiva">
                        <title>Tipo documento</title>
                        <description>Tipo documento</description>
                        <type>d:text</type>
                        <mandatory>false</mandatory>
                        <constraints>
                                <constraint name="dfm:tipoDocumentoFtConstrPassiva" ref="dfm:CTipoDocumentoFtPassiva" />
                        </constraints>
                </property>
        </properties>
        <mandatory-aspects>
                <aspect>dfm:ADocumentoIVAdfmBase</aspect>
                <aspect>dfm:AFornitore</aspect>
        </mandatory-aspects>
</type>
```

Figura 2.8 : Codice della fattura passiva

Come si può notare in figura 2.8 per il tipo documento fattura passiva, viene utilizzato il collegamento al “Constraint” che abbiamo dichiarato in precedenza.

```
i Trasporto Passivo      extends TDocGenericodfmBase
ry Aspect:     ADocumentoIVAdfmBase    ;;   AFornitore
*******************************************************-->
<type name="dfm:TDocDDTPassivo">
        <title>DDT Passivo</title>
        <description>DDT Passivo Modellazione Finanza</description>
        <parent>dfm:TDocGenericodfmBase</parent>
        <mandatory-aspects>
                <aspect>dfm:ADocumentoIVAdfmBase</aspect>
                <aspect>dfm:AFornitore</aspect>
        </mandatory-aspects>
</type>

*******************************************************
attura Attiva      extends TDocGenericodfmBase
umentoFtAttiva :: text     indica tipo di documento di fattura Attiva
ry Aspect:     ADocumentoIVAdfmBase    ;;   ACliente
*******************************************************-->
<type name="dfm:TDocFatturaAttiva">
        <title>Fattura Attiva</title>
        <description>Fattura Attiva Modellazione Finanza</description>
        <parent>dfm:TDocGenericodfmBase</parent>
        <properties>
                <property name="dfm:tipoDocumentoFtAttiva">
                        <title>Tipo documento </title>
                        <description>Tipo documento</description>
                        <type>d:text</type>
                        <mandatory>false</mandatory>
                        <constraints>
                                <constraint name="dfm:tipoDocumentoFtConstrAttiva" ref="dfm:CTipoDocumentoFtAttiva"/>
                        </constraints>
                </property>
        </properties>
        <mandatory-aspects>
                <aspect>dfm:ADocumentoIVAdfmBase</aspect>
                <aspect>dfm:ACliente</aspect>
        </mandatory-aspects>
</type>
```

Figura 2.9 :codice DDT Passivo e fattura attiva

```
<!-- ******************************************************************
Tipo Documento Di trasporto Attivo      extends TDocGenericodfmBase
        Mandatory Aspect:     ADocumentoIVAdfmBase    ;;   ACliente
*********************************************************************-->
                <type name="dfm:TDocDDTAttivo">
                        <title>DDT Attivo</title>
                        <description>DDT Attivo Modellazione Finanza</description>
                        <parent>dfm:TDocGenericodfmBase</parent>
                        <mandatory-aspects>
                                <aspect>dfm:ADocumentoIVAdfmBase</aspect>
                                <aspect>dfm:ACliente</aspect>
                        </mandatory-aspects>
                </type>

<!-- ******************************************************************
Tipo Ordine di Acquisto     extends TDocGenericodfmBase
        Mandatory Aspect:     ADocumentoIVAdfmBase    ;;   AFornitore
*********************************************************************-->
                <type name="dfm:TODA">
                        <title>Ordine di Acquisto</title>
                        <description>Ordine di Acquisto</description>
                        <parent>dfm:TDocGenericodfmBase</parent>
                        <mandatory-aspects>
                                <aspect>dfm:ADocumentoIVAdfmBase</aspect>
                                <aspect>dfm:AFornitore</aspect>
                        </mandatory-aspects>
                </type>

<!-- ******************************************************************
Tipo Ordine di Vendita     extends TDocGenericodfmBase
        Mandatory Aspect:     ADocumentoIVAdfmBase    ;;   Acliente
*********************************************************************-->
                <type name="dfm:TODV">
                        <title>Ordine di Vendita</title>
                        <description>Ordine di Vendita</description>
                        <parent>dfm:TDocGenericodfmBase</parent>
                        <mandatory-aspects>
                                <aspect>dfm:ADocumentoIVAdfmBase</aspect>
                                <aspect>dfm:ACliente</aspect>
                        </mandatory-aspects>
                </type>
        </types>
```

Figura 2.10 codice di DDT Attivo, ordine di acquisto e ordine di vendita

2.8 Definizione degli aspetti

La motivazione dell'utilizzo degli aspetti in Alfresco è data dalla necessità che quando a un documento viene data una relativa tipologia, non è più possibile cambiarla con un'altra. Utilizzare gli aspetti è conveniente per gestire anche piccole modifiche al codice senza andare a intaccare il tipo di documento, ove basterà modificare solamente il codice dell'aspetto. Ricordiamo inoltre che quando si da un tipo a un file, quel file può cambiare il tipo solo ed unicamente se il tipo è figlio del tipo precedentemente dato. Questo mostra come il tipo di documento limita molte modifiche, mentre per gli aspetti non abbiamo questi relativi problemi, perché si ha la possibilità cambiare le proprietà, ed anche di andare a modificare le proprietà di una proprietà di un aspetto già definito. Come vediamo il nostro Aspetto per il Documento IVA (figura 2.11) è caratterizzato dalle proprietà che abbiamo elaborato in precedenza, ed è stato aggiunto il "numeroDocumento" che è dichiarato come indice, ma questo indice non deve essere tokenizzato. La proprietà "<mandatory>true</mandatory>" obbliga, l'inserimento di quel campo al momento delle modifiche delle proprietà. Nel caso contrario non è obbligatorio l'inserimento del valore.

Il mandatory è utile specialmente per la parte Share del nostro progetto, proprio perché obbliga (in modo automatico) l'utente a inserire il valore. Per la proprietà "annoDocumento", dichiarato di tipo int, la parte Share non farà inserire valori che conterranno delle lettere o caratteri speciali, ma unicamente dei numeri.

```
>

*******************************************************
lla il documento IVA
ocumento :: text    indica il numero del documento
umento :: text      indica la data del documento
umento :: text      indica l'anno del documento ( gestito poi con behavior)
*******************************************************-->
<aspect name ="dfm:ADocumentoIVAdfmBase">
        <title>Documento IVA Modellazione Base</title>
        <description>Documento IVA Modellazione Finanza</description>
        <properties>
                <property name="dfm:numeroDocumento">
                        <title>Numero Documento</title>
                        <description>Numero Documento</description>
                        <type>d:text</type>
                        <mandatory>true</mandatory>
                        <index enabled="true">
                                <tokenised>false</tokenised>
                         </index>
                </property>
                <property name="dfm:dataDocumento">
                        <title>Data documento</title>
                        <description>Data documento</description>
                        <type>d:date</type>
                        <mandatory>true</mandatory>
                </property>
                <property name="dfm:annoDocumento">
                        <title>Anno documento</title>
                        <description>Anno documento</description>
                        <type>d:int</type>
                        <mandatory>false</mandatory>
                </property>
        </properties>
</aspect>
```

Figura 2.11 : Codice dell'aspetto Documento Iva

L'aspetto del Fornitore (figura 2.12), come abbiamo detto in precedenza, modellerà tutte le proprietà che ci servono per identificare un Fornitore. Notiamo che nella definizione di "CFFornitore" ci sono 2 campi per l'indice. Il campo "<atomic>" metterà il nostro valore come indice in background, mentre il campo "<stored>false</stored>" non permetterà il salvataggio del valore della proprietà nell'indice, in modo da non appesantirlo.

```
<aspect name="dfm:AFornitore">
        <title>Fornitore</title>
        <description>Dati del Fornitore</description>
        <properties>
                <property name="dfm:codiceFornitore">
                        <title>Codice Fornitore</title>
                        <description>Codice Fornitore</description>
                        <type>d:text</type>
                        <mandatory>true</mandatory>
                        <index enabled="true">
                                <tokenised>false</tokenised>
                         </index>
                </property>
                <property name="dfm:IVAFornitore">
                        <title>IVA Fornitore</title>
                        <description>IVA Fornitore</description>
                        <type>d:text</type>
                        <mandatory>false</mandatory>
                        <index enabled="true">
                                <tokenised>false</tokenised>
                        </index>
                </property>
                <property name="dfm:RagioneSocialeFornitore">
                        <title>Fornitore</title>
                        <description>Nome Fornitore</description>
                        <type>d:text</type>
                        <mandatory>false</mandatory>
                        <index enabled="true">
                                <tokenised>false</tokenised>
                         </index>
                </property>
                <property name="dfm:CFFornitore">
                        <title>CF Fornitore</title>
                        <description>CF Fornitore</description>
                        <type>d:text</type>
                        <mandatory>false</mandatory>
                        <index enabled="true">
                            <atomic>true</atomic>
                            <stored>false</stored>
                            <tokenised>false</tokenised>
                        </index>
                </property>
        </properties>
</aspect>
```

Figura 2.12 : Codice aspetto del fornitore

L'aspetto Cliente (figura 2.13) modella le proprietà che sono state richieste dal cliente per avere le relative informazioni sul cliente.

```
<aspect name="dfm:ACliente">
        <title>Cliente</title>
        <description>Dati del Cliente</description>
        <properties>
                <property name="dfm:codiceCliente">
                        <title>Codice Cliente</title>
                        <description>Codice Cliente</description>
                        <type>d:text</type>
                        <mandatory>true</mandatory>
                        <index enabled="true">
                                <tokenised>false</tokenised>
                         </index>
                </property>
                <property name="dfm:IVACliente">
                        <title>IVA Cliente</title>
                        <description>Iva del Cliente</description>
                        <type>d:text</type>
                        <mandatory>false</mandatory>
                        <index enabled="true">
                                <tokenised>false</tokenised>
                         </index>
                </property>
                <property name="dfm:RagioneSocialeCliente">
                        <title>Ragione Sociale</title>
                        <description>Ragione Sociale Del Cliente</description>
                        <type>d:text</type>
                        <mandatory>false</mandatory>
                        <index enabled="true">
                                <tokenised>false</tokenised>
                         </index>
                </property>
                <property name="dfm:CFCliente">
                        <title>CF Cliente</title>
                        <description>CF Cliente</description>
                        <type>d:text</type>
                        <mandatory>false</mandatory>
                        <index enabled="true">
                            <atomic>true</atomic>
                            <stored>false</stored>
                            <tokenised>false</tokenised>
                        </index>
                </property>
        </properties>
</aspect>
```

Figura 2.13 : Codice aspetto cliente

```
<aspect name="dfm:AProtocollo">
        <title>Dati Protocollo</title>
        <description>Dati del protocollo</description>
        <properties>
                <property name="dfm:numeroProtocollo">
                        <title>Numero Protocollo</title>
                        <description>Numero Protocollo</description>
                        <type>d:text</type>
                        <mandatory>false</mandatory>
                        <index enabled="true">
                                <tokenised>false</tokenised>
                         </index>                                  </property>
                <property name="dfm:dataProtocollo">
                        <title>Data Protocollo</title>
                        <description>Data Protocollo</description>
                        <type>d:date</type>
                        <mandatory>false</mandatory>
                </property>
        </properties>
</aspect>
```

Figura 2.14 : Codice aspetto protocollo

L'aspetto del Protocollo (figura 2.14), anche se non richiesto e non sarà utilizzato, è stato implementato per creare un modulo AMP che sia il più generale possibile, esso darà la possibilità di modellare i protocolli interni dell'azienda.

2.9 Creare un collegamento tra Repo e Share

Post modellazione per la nostra parte Repo, necessitiamo di far comprendere alla parte Share che c'è il nostro modello contenente gli aspetti e tipologie di file. Dobbiamo quindi andare a modificare il file "docfiscali-share.properties" presente in "C:\progetti\docfiscali-share\src\main\amp\config\alfresco\web-extension\messages" (come in figura 2.15).

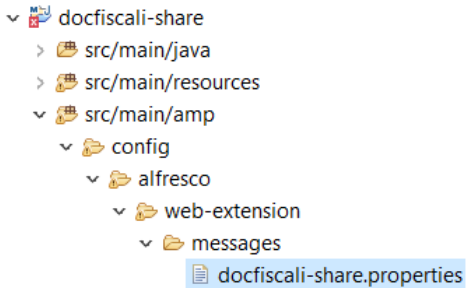


Figura 2.15 : posizionamento file docfiscali-share.properties

Il formato di questo file sarà dato da sezioni. Nel nostro caso avremo la sezione type e aspect.
Le righe di codice sono così formate:

[TIPOLOGIA DEL MODELLO].PREFISSO NAMESPACE_OGGETTO DEL MODELLO = LABEL IDENTIFICATIVA

In figura 2.16 è presente il codice che implementa questo collegamento tra Repo e Share.

```
type.dfm_TDoc=Documento Unico modello base
type.dfm_TDocFatturaPassiva=Fattura Passiva
type.dfm_TDocDDTPassivo=DDT Passivo
type.dfm_TODA=Ordine Acquisto
type.dfm_TODV=Ordine Vendita
type.dfm_TDocFatturaAttiva=Fattura Attiva
type.dfm_TDocDDTAttivo=DDT Attivo


aspect.dfm_AFornitore=Dati Fornitore
aspect.dfm_ADocumentoIVAdfmBase=Dati documento IVA
aspect.dfm_ACliente=Dati Cliente
aspect.dfm_AProtocollo=Dati Protocollo
```

Figura 2.16: Collegamento tra Repository e Share.

In questo punto stiamo dunque assegnando ai nomi degli oggetti del content model, una label che ci permetterà di rendere visibile e leggibile le nostre tipologie e aspetti nella parte Share.

2.10 Codice Share

Per creare il codice della parte Share che sarà la sezione utilizzata poi dall'utente, si dovrà andare a modificare il file presente in "C:\progetti\docfiscali-share\src\main\resources\META-INF" di nome "share-config-custom.xml" (figura 2.17).

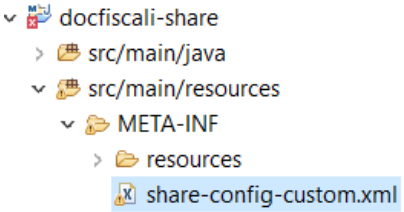


Figura 2.17

Questo file XML è composto da varie sezioni. Nella seguente sezione verranno analizzate una ad una.

2.11 Sezione DocumentLibrary

La prima sezione che dovremmo implementare si chiama DocumentLibrary, questa è la nostra libreria dei documenti. Il codice che inseriremo in questa sezione (figura 2.18) ci darà la possibilità di far vedere all'utente gli aspetti e i tipi che può utilizzare, in più possiamo creare una gerarchia di padre-figlio tra i tipi dei nostri documenti, così da poter permettere il cambio del tipo solo con tipologie figlie. Si può rendere disponibile all'utente anche la modalità di rimozione di alcuni aspetti.

```
<alfresco-config>

        <config evaluator="string-compare" condition="DocumentLibrary">
                <aspects>
                        <!-- Aspects that a user can see -->
                        <visible>
                                <aspect name="dfm:AFornitore" />
                                <aspect name="dfm:ADocumentoIVAdfmBase" />
                                <aspect name="dfm:ACliente" />
                                <aspect name="dfm:AProtocollo" />
                        </visible>

                        <!-- Aspects that a user can add. Same as "visible" if left empty -->
                        <addable>
                        </addable>

                        <!-- Aspects that a user can remove. Same as "visible" if left empty -->
                        <removeable>
                        </removeable>
                </aspects>

                <types>
                        <type name="cm:content">
                                <subtype name="dfm:TDocFatturaPassiva" />
                                <subtype name="dfm:TDocFatturaAttiva"/>
                                <subtype name="dfm:TDocDDTPassivo" />
                                <subtype name="dfm:TDocDDTAttivo"/>
                                <subtype name="dfm:TODA"/>
                                <subtype name="dfm:TODV"/>
                                </type>
                </types>
        </config>
```

Figura 2.18

2.12 Sezione Form per le ricerche avanzate (AdvancedSearch)

La sezione che dichiara i Form per le ricerche avanzate si chiama “AdvancedSearch”(figura 2.19). Questa sezione ci permette di dichiarare dei Form programmati per effettuare ricerche. Tramite un ID di riferimento riusciamo a collegarci all’implementazione successiva del Form. Questo ID deve essere univoco.

```
<config evaluator="string-compare" condition="AdvancedSearch">
        <advanced-search>
        <forms>
                <form label="Fattura Passiva" description="Ricerca per Fattura Passiva " id="searchDocFtPassiva">dfm:TDocFatturaPassiva</form>
                <form label="Fattura Attiva" description="Ricerca per Fattura Attiva" id="searchDocFtAttiva">dfm:TDocFatturaAttiva</form>
                <form label="DDT Passivo" description="Ricerca per DDT Passivo" id="searchDocDDTPassivo">dfm:TDocDDTPassivo</form>
                <form label="DDT Attivo" description="Ricerca per DDT Attivo" id="searchDocDDTAttivo">dfm:TDocDDTAttivo</form>
                <form label="Ordine Di Acquisto" description="Ricerca per Ordine d'acquisto" id="searchDocTODA">dfm:TODA</form>
                <form label="Ordine Di Vendita" description="Ricerca per Ordine di vendita" id="searchDocTODV">dfm:TODV</form>
        </forms>
        </advanced-search>
</config>
```

Figura 2.19

2.13 Sezione Creazione dei nodi

Occorre ricordare che in Alfresco tutto è un nodo, quindi quando si va a inserire un qualunque file o una cartella, è come se stessimo creando un nodo. Prendiamo in esempio il nodo per la Fattura Passiva (figura 2.20).

```
<config evaluator="node-type" condition="dfm:TDocFatturaPassiva">
        <forms>
        <form>
                <field-visibility>
                    <show id="cm:name" />

                    <!-- tags and categories -->
                    <show id="cm:taggable" for-mode="edit" force="true" />
                    <show id="cm:categories" />

                    <!-- dati del tipo -->
                    <show id="dfm:numeroDocumento"/>
                    <show id="dfm:dataDocumento" />
                    <show id="dfm:annoDocumento" />
                    <show id="dfm:tipoDocumentoFtPassiva" />
                    <show id="dfm:codiceFornitore" />
                    <show id="dfm:IVAFornitore" />
                    <show id="dfm:RagioneSocialeFornitore" />
                    <show id="dfm:CFFornitore" />
                </field-visibility>
            <appearance>
                    <set appearance="fieldset" id="docfiscaliDocFtPassivaTitle" label="Dati Fattura Passiva" />
                    <field id="dfm:dataDocumento" set="docfiscaliDocFtPassivaTitle"/>
                    <field id="dfm:annoDocumento" set="docfiscaliDocFtPassivaTitle"/>
                    <field id="dfm:numeroDocumento" set="docfiscaliDocFtPassivaTitle"/>
                    <field id="dfm:tipoDocumentoFtPassiva" set="docfiscaliDocFtPassivaTitle"/>
                    <field id="dfm:codiceFornitore" set="docfiscaliDocFtPassivaTitle"/>
                    <field id="dfm:IVAFornitore" set="docfiscaliDocFtPassivaTitle"/>
                    <field id="dfm:RagioneSocialeFornitore" set="docfiscaliDocFtPassivaTitle"/>
                    <field id="dfm:CFFornitore"  set="docfiscaliDocFtPassivaTitle"/>
            </appearance>
              </form>
              </forms>
        </config>
```

Figura 2.20

La creazione di un nodo si divide in un Form, con una sezione di “field-Visibility” che ci mostrerà tutti i campi che specificheremo quando verrà visualizzato il nodo (documento) con modalità visualizzazione. La seconda parte del nodo chiamata “appearance” è utilizzata per creare il Form nella modalità di modifica (rimando agli indici per i codici della creazione del nodo delle altre tipologie di documenti).

2.14 Sezione di ricerca avanzata e Form

Prendiamo sempre come caso d'esempio la nostra fattura passiva (figura 2.21). Vogliamo modificare i Form di ricerca avanzata che sono presenti in Alfresco per il nostro tipo di documento.

```
<config evaluator="model-type" condition="dfm:TDocFatturaPassiva">
        <forms>
        <!-- Search form -->
        <form id="searchDocFtPassiva">
                <field-visibility>
                <show id="cm:name" />

                <show id="dfm:numeroDocumento"/>
                <show id="dfm:dataDocumento" />
                <show id="dfm:annoDocumento" />
                <show id="dfm:codiceFornitore" />
                <show id="dfm:IVAFornitore" />
                <show id="dfm:RagioneSocialeFornitore" />
                <show id="dfm:CFFornitore" />
                </field-visibility>

                <appearance>
                        <set appearance="fieldset" id="docfiscaliDocFtPassivaTitle" label="Dati Fattura Passiva" />
                <field id="dfm:dataDocumento" set="docfiscaliDocFtPassivaTitle">
                        <control template="/org/alfresco/components/form/controls/daterange.ftl" />
                </field>
                <field id="dfm:annoDocumento" set="docfiscaliDocFtPassivaTitle" />
                <field id="dfm:numeroDocumento" set="docfiscaliDocFtPassivaTitle" />
                <field id="dfm:codiceFornitore" set="docfiscaliDocFtPassivaTitle" />
                <field id="dfm:IVAFornitore" set="docfiscaliDocFtPassivaTitle" />
                <field id="dfm:RagioneSocialeFornitore" set="docfiscaliDocFtPassivaTitle" />
                <field id="dfm:CFFornitore" set="docfiscaliDocFtPassivaTitle" />
                </appearance>
        </form>
        </forms>
</config>
```

Figura 2.21

Questa parte di ricerca avanzata è quasi identica a quella precedente. Una cosa che si nota è il campo field della "dfm:dataDocumento". In questo campo viene utilizzato un template di controllo che controllerà l'inserimento corretto della nostra data e nel caso segnalerà la non validità.

2.15 Constraint e Label Localizzate

Per poter modellare delle Constraint con dei codici e successivamente far vedere quei codici tradotti in lingua locale, bisogna andare a configurare 2 file (figura 2.22):

- **NomeDizionario.properties** (C:\progetti\docfiscali-repo\src\main\amp\config\alfresco\module\docfiscali-repo\messages)
- **Bootstrap-context.xml** (C:\progetti\docfiscali-repo\src\main\amp\config\alfresco\module\docfiscali-repo\context)

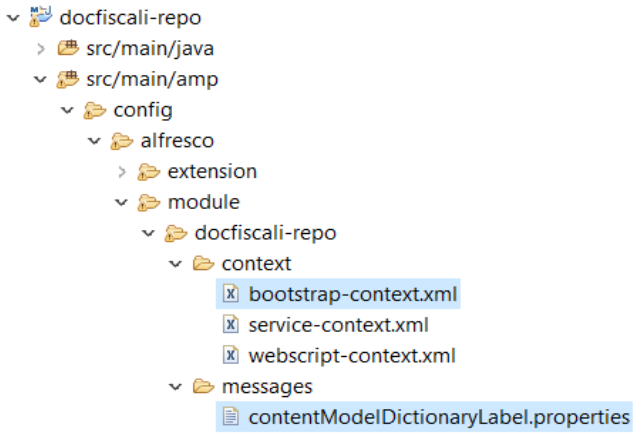


Figura 2.22

Utilizziamo per l'esempio due Constraint di nome "dfm:CTipoDocumentoFtPassiva"(figura 2.23) e di "dfm:CTipoDocumentoFtAttiva" (figura 2.24) situate all'interno del content-model.xml e vogliamo che la lista dei valori di questi Constraint abbiano una traduzione non contratta (nel caso in cui i valori fossero dei numeri si potrà sfruttare questo metodo anche per la manutenibilità del codice o per cambiamenti):

```
<constraint name="dfm:CTipoDocumentoFtPassiva" type="LIST">
        <title>Tipo Documento Ft</title>
        <parameter name="allowedValues">
                <list>
                        <value>FatPass</value>
                        <value>NotaAcc</value>
                        <value>BollaDog</value>
                        <value>AvvisoPag</value>
                        <value>Varie</value>
                </list>
        </parameter>
        <parameter name="caseSensitive">
                <value>false</value>
        </parameter>
</constraint>
```

Figura 2.23

```
        <constraint name="dfm:CTipoDocumentoFtAttiva" type="LIST">
        <title>Tipo Documento Ft Attiva</title>
        <parameter name="allowedValues">
                        <list>
                                <value>FattAtt</value>
                                <value>NotaAcc</value>
                                <value>Varie</value>
                        </list>
                </parameter>
                <parameter name="caseSensitive">
                        <value>false</value>
                </parameter>
        </constraint>
</constraints>

<types>
```

Figura 2.24

Bisognerà andare a modificare (nel caso in cui non esista bisognerà crearlo) il "NomeDizionario.properties", che per questo progetto chiameremo: "contentModelDictionaryLabel.properties". Nel file possiamo includere le traduzioni per il Constraint e/o anche per le proprietà di alcuni tipi di documento che vogliamo tradurre (figura 2.25).

```
1  # Display labels for Content Domain Model
2  #Label per Constraint CTipoDocumentoFtPassiva
3  listconstraint.dfm_CTipoDocumentoFtPassiva.FatPass=Fattura Passiva
4  listconstraint.dfm_CTipoDocumentoFtPassiva.NotaAcc=Nota Di Accredito
5  listconstraint.dfm_CTipoDocumentoFtPassiva.BollaDog=Bolla Doganale
6  listconstraint.dfm_CTipoDocumentoFtPassiva.AvvisoPag=Avviso Pagamento
7  listconstraint.dfm_CTipoDocumentoFtPassiva.Varie=Altre e Varie
8
9  listconstraint.dfm_CTipoDocumentoFtAttiva.FattAtt=Fattura Attiva
10 listconstraint.dfm_CTipoDocumentoFtAttiva.NotaAcc=Nota Di Accredito
11 listconstraint.dfm_CTipoDocumentoFtAttiva.Varie=Varie e Altre
```

Figura 2.25

Arrivati ad avere tutto il necessario per poter effettuare la traduzione, dobbiamo però aggiungere il file di traduzione alla compilazione del nostro modello per poter farla funzionare. Aggiungeremo, in gergo tecnico, una "property" nel nostro "Bean" sul quale stiamo lavorando. La modifica dovrà essere effettuata nel file Bootstrap-context.xml aggiungendo una property "labels" come in figura 2.26.

```
<!-- Registration of new models -->
<bean id="docfiscali-repo.dictionaryBootstrap" parent="dictionaryModelBootstrap"
    depends-on="dictionaryBootstrap">
    <property name="models">
        <list>
            <value>alfresco/module/${project.artifactId}/model/content-model.xml</value>
            <value>alfresco/module/${project.artifactId}/model/workflow-model.xml</value>
        </list>
    </property>
    <property name="labels">
        <value>alfresco/module/${project.artifactId}/messages/contentModelDictionaryLabel</value>
    </property>
</bean>

ans>
```

Figura 2.26

2.16 Le Behaviour

Precedentemente abbiamo discusso di come creare un proprio modello per modellare le nostre tipologie di file e gli aspetti che possiamo applicare ad essi, ma ciò non ci basta. Per poter incominciare ad automatizzare il nostro processo, necessitiamo che ci sia poco intervento dell'utente che utilizza il nostro programma e quindi più automatismo. Si può pensare a quando si desidera creare ed eseguire codice strettamente accoppiato ad un tipo di file e che venga eseguito ogni volta che succede un determinato evento come la modifica, cambi di versione, il caricamento e così via dicendo. Fortunatamente, Alfresco offre dei meccanismi che vengono chiamate Behaviours. Vi sono vari esempi di problemi per capire in cosa potrebbero esser utilizzate le Behaviour:

- Potrebbe essere necessario impostare alcuni metadati di default utilizzando la logica che non può esser espressa tramite il modello XML, inoltre vorremmo generare un identificatore univoco per un oggetto quando viene aggiunto al Repository.

Oppure:

- Supponiamo di avere alcuni metadati memorizzati su una cartella e che si vogliano salvare questi metadati anche sui file che inseriremo al suo interno.

2.17 Problema affrontato

Introduciamo il problema che affronteremo:

Vorremmo che tutti i file che abbiano l'aspetto DocumentoIVA, abbiano l'anno ("annoDocumento") che venga ricavato in modo automatico dalla data del documento ("dataDocumento"). Per fare ciò possiamo implementare la nostra Behaviour tramite una classe Java oppure scegliere un'implementazione tramite un JavaScript. Per l'implementazione di questo Behaviour sceglieremo una classe Java chiamata "GestioneAnnoDocFiscali.java".

La modalità per poter far attivare qualunque Behaviour è data delle politiche di comportamento a cui possiamo legare la nostra Behaviour. Se ricerchiamo nelle classi Java tutte quelle classi che terminano per "*Policies" troveremo alcune interfacce come:

- CheckOutCheckInServicePolicies
- ContentServicePolicies
- CopyServicePolicies
- LockServicePolicies
- NodeServicePolicies
- TransferServicePolicies
- VersionServicePolicies

In ognuna di queste interfacce sono presenti le interfacce a cui possiamo fare il Bind per la nostra Behaviour.

2.18 Policies dei nodi

Nella seguente tabella sono disponibili le Policies per il Binding di Behaviour:

Interfacce	***Metodo***
`org.alfresco.repo.content.ContentServicePolicies`	**onContentPropertyUpdate** **onContentRead** **onContentUpdate**
`org.alfresco.repo.copy.CopyServicePolicies`	**beforeCopy** **onCopyComplete** **onCopyNode**
`org.alfresco.repo.node.NodeServicePolicies`	**beforeAddAspect** **beforeArchiveNode** **beforeCreateNode** **beforeCreateStore** **beforeDeleteAssociation** **beforeDeleteChildAssociation** **beforeDeleteNode** **beforeMoveNode** **beforeRemoveAspect** **beforeSetNodeType** **beforeUpdateNode** **onAddAspect** **onCreateAssociation** **onCreateChildAssociation** **onCreateNode** **onCreateStore** **onDeleteAssociation** **onDeleteChildAssociation** **onDeleteNode** **onMoveNode** **onRemoveAspect** **onSetNodeType** **onUpdateNode** **onUpdateProperties**
`org.alfresco.repo.version.VersionServicePolicies`	**beforeCreateVersion** **afterCreateVersion** **onCreateVersion** **calculateVersionLabel**

In questa tabella notiamo un metodo che si chiama “onUpdateProperties” che è il metodo che viene chiamato da Alfresco quando si deve fare un aggiornamento delle proprietà di un nodo. Andiamo a creare la cartella behaviour (se non è già presente) in “C:\progetti\docfiscali-repo\src\main\java\it\netsphere\alfresco\docfiscali\”, poi creiamo il file Java “GestioneAnnoDocFiscali.java”.

2.19 La classe Java implementata

La classe già implementata si presenterà come in figura 2.27.

```
28 public class GestioneAnnoDocFiscali implements
29     NodeServicePolicies.OnUpdatePropertiesPolicy{
30
31     // logging (used only in debug mode)
32     public static Log log = LogFactory.getLog(GestioneAnnoDocFiscali.class);
33
34     //Dipendenze
35     private NodeService nodeService;
36     private PolicyComponent policyComponent;
37     private ServiceRegistry serviceRegistry;
38
39     //NameSpaces
40     private static final String NAMESPACE_NETSPHERE_DOCFISCALI = "http://www.castgroup.it/model/docfiscali/1.0";
41
42     //Proprieties
43     private static final String PROP_DataDocumento = "dataDocumento";
44     private static final String PROP_AnnoDocumento = "annoDocumento";
46    //Aspect
47    private static final String ASPECT_DOCUMENTOIVA = "ADocumentoIVAdfmBase";
48
49    //QName
50    private static final QName PROP_QNAME_DATA = QName.createQName(NAMESPACE_NETSPHERE_DOCFISCALI , PROP_DataDocumento);
51    private static final QName PROP_QNAME_ANNO = QName.createQName(NAMESPACE_NETSPHERE_DOCFISCALI, PROP_AnnoDocumento);
52    private static final QName QNAME_ONUPDATEPROPERTIES = QName.createQName(NamespaceService.ALFRESCO_URI, "onUpdateProperties");
53
54    //Behaviour
55    private Behaviour onUpdateProperties;
56
57
58    /**
59     * Funzione di inizializzazione per le behaviour e le BindClassBehaviour
60     */
61    public void init(){
62        //Creazione Behaviour
63        this.onUpdateProperties = new JavaBehaviour(this,  "onUpdateProperties" , NotificationFrequency.EVERY_EVENT);
64        //Bind behaviour alle policies del nodo
65        this.policyComponent.bindClassBehaviour(QNAME_ONUPDATEPROPERTIES, ContentModel.TYPE_CONTENT, this.onUpdateProperties);
66    }
```

Figura 2.27

Questa prima parte di codice (figura 2.27) mostra che la classe GestioneAnnoDocFiscali implementa l'interfaccia chiamata NodeServicePolicies.OnUpdatePropertiesPolicy presente nella tabella precedente. Per implementare questa interfaccia dobbiamo implementare il metodo OnUpdatePropertiesPolicy che ci servirà per fare le modifiche sulle proprietà prima che si aggiornino effettivamente sul nodo. Abbiamo le nostre dipendenze sul nodo per poter lavorare e prendere tutte le informazioni di cui necessitiamo (si vedrà in seguito come collegarle le informazioni a questa classe), il nostro Namespace, con lo scopo di evitare confusione ed equivoci nel caso ci siano molte entità con nomi simili e i QName che rappresentano una definizione di un identificatore per elementi e attributi. Il QName è utilizzato per creare una mappa tra il nostro URI (Uniform Resource Identifier) e il prefisso del Namespace, come nel nostro caso, è usato

per far rifermienti agli attributi e proprietà del nostro modello XML, in più è anche utilizzato per definire una mappatura tra URI del servizio e il metodo da utilizzare.

Nella funzione init, che sarà chiamata a tempo di esecuzione del servizio server Repo, viene effettuata la creazione e il Bind delle Behaviour.

```
@Override
public void onUpdateProperties(NodeRef nodeRef, Map<QName, Serializable> before, Map<QName, Serializable> after) {

    NodeService nodeService = serviceRegistry.getNodeService();
    QName ASPECT_QNAME_DOCUMENTOIVA = QName.createQName(DFM ,ASPECT_DOCUMENTOIVA );

    if( nodeService.hasAspect(nodeRef, ASPECT_QNAME_DOCUMENTOIVA ) ) // se contengo l
    {
            Date Data = (Date)nodeService.getProperty(nodeRef, PROP_QNAME_DATA);
            log.debug("*LA DATA è : "+Data.toString());
            int Anno = this.getYear(Data);
            nodeService.setProperty(nodeRef, PROP_QNAME_ANNO, (Serializable) Anno );
    }
}
```

Figura 2.28

Il metodo in figura 2.28 verrà chiamato quando ci sarà da fare un update delle proprietà del nodo, esso effettuerà un controllo per vedere se il documento contiene l'aspetto Documento IVA e successivamente ricaverà l'anno dalla data e andrà a modificarlo sulle proprietà del nodo.

```
/**
 * Funzione che prende una data e restituisce l'Anno Serializable
 * @param data :data da cui si vuole ricavare un Anno di tipo Serializable
 * @return Anno :l'anno della data Serializable
 */
private int getYear(Date data){
    Calendar calendar= Calendar.getInstance();
    calendar.setTime(data);
    int Anno = calendar.get(Calendar.YEAR);
    return Anno;
}
```

Figura 2.29

La Funzione getYear (figura 2.29) ritorna l'anno di una data. I getter e i setter della classe sono stati omessi.

2.20 Configurare lo Spring Bean

L'ultimo passaggio prima di testare la nostra Behaviour è configurare lo Spring Bean "service-context.xml" che si trova in "C:\progetti\docfiscali-repo\src\main\amp\config\alfresco\module\docfiscali-repo\context" (figura 2.30). Senza configurare lo Spring Bean la classe java per la Behaviour non sarà inizializzata e quindi non sarà presente nei nostri servizi attivi.

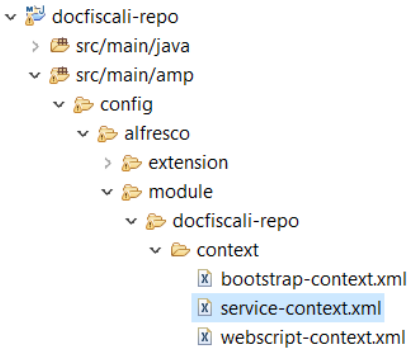


Figura 2.30

In figura 2.31 viene mostrato il codice per creare un Bean e utilizzare le nostre dipendenze per i nodi che sono nel file Java della Behaviour.

```
<bean id="GestioneAnnoDocFiscali" class="it.netsphere.alfresco.docfiscali.behavior.GestioneAnnoDocFiscali" init-method="init">
    <property name="nodeService">
        <ref bean="nodeService" />
    </property>
    <property name="policyComponent">
        <ref bean="policyComponent" />
    </property>
    <property name="serviceRegistry">
        <ref bean="ServiceRegistry" />
    </property>
</bean>
```

Figura 2.31

Capitolo 3

3 Progetto All in One (AIO)

L'utilizzo del progetto AIO è dato dal fatto che abbiamo precedentemente creato un AMP di base che modella una realtà aziendale generica da cui vorremmo implementare, per il nostro cliente, delle personificazioni specifiche in base a quello che ci richiederà. L'AIO viene utilizzato per essere il punto di partenza per altri AMP, quindi ci sarà il nostro AIO con tanti relativi moduli AMP che installeremo (come in figura 3.1).

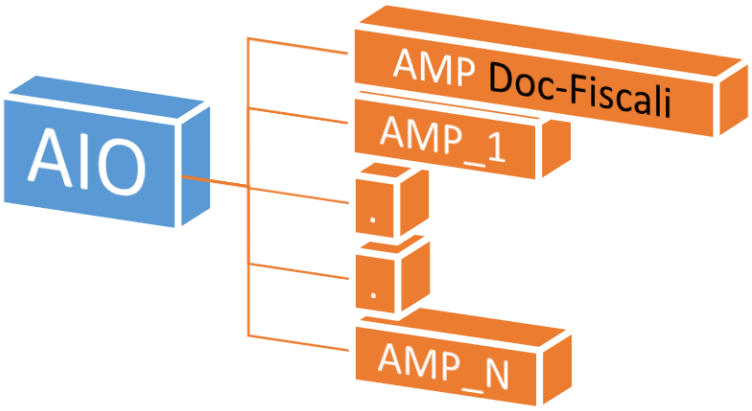


Figura 3.1

I benefici che si hanno ad utilizzare un AIO con diversi AMP al suo interno sono:

- Il facile utilizzo e sviluppo per i programmatori che possono programmare in parallelo su differenti moduli/funzionalità, invece di valorare in un unico grande AMP.
- La manutenzione viene svolta in modo molto più efficiente.
- Il rilascio di nuove specifiche o funzionalità sono più semplici fatte a tempo di necessità o in un secondo momento, ma mantenendo la funzionalità del progetto iniziale.
- Incoraggia il riutilizzo delle estensioni presenti nell'AIO (che non sarebbero presenti in un unico progetto AIO con lo sviluppo di un unico grande AMP).

Ci sono vari modi per aggiungere degli AMP creati in precedenza:

1. Creare degli AMP separati dal progetto AIO (approccio raccomandato) che da tutti i benefici elencati in precedenza.

2. Creare degli AMP come parte del progetto AIO, questo non darà tutti i benefici elencati prima. Tuttavia, è utile se si desidera dividere il vostro codice, strutturarlo un po' e se si ha un piccolo team di programmatori. L'utilizzo di questa metodologia facilità l'implementazione dell'AMP, ma sarà difficile decidere quando la funzionalità dovrebbe essere inclusa o meno nella build. In sostanza se la funzionalità esiste nell'AMP del progetto AIO, sarà incluso nella build, anche se essa non è ancora completa.

3.1 Implementazione delle specifiche per il cliente

Come abbiamo detto in precedenza, dopo aver modellato la base dei nostri documenti fiscali ed aver installato il nostro AMP nel progetto (specificato nella sezione indice) che poi daremo al nostro cliente, dobbiamo anche aggiungere delle specifiche aggiuntive che il cliente ci ha richiesto. La richiesta del cliente è stata di aggiungere a tutti i suoi documenti la possibilità di salvare il codice a barre. Possiamo mettere in pratica questa aggiunta di proprietà tramite l'utilizzo di un aspetto che verrà applicato a tutti i tipi di documenti. Chiameremo questo aspetto "AProtocollo" ed al suo interno avrà un campo di nome "CodiceABarre" di tipo text.

3.1.1 Implementare le specifiche per la parte Repo

Per poter andare a fare queste modifiche dobbiamo innanzitutto andare a modificare il nostro content-model.xml della parte Repo(AIO). Il nostro content-model si presenterà come in figura 3.2.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <model name="pa:contentModel" xmlns="http://www.alfresco.org/model/dictionary/1.0">
3
4      <!-- Optional meta-data about the model -->
5      <description>Document Model for Repo Extension X</description>
6      <author>Alessio Masola</author>
7      <version>1.0</version>
8
9      <imports>
10         <!-- Import Alfresco Dictionary Definitions -->
11         <import uri="http://www.alfresco.org/model/dictionary/1.0" prefix="d"/>
12         <!-- Import Alfresco Content Domain Model Definitions -->
13         <import uri="http://www.alfresco.org/model/content/1.0" prefix="cm"/>
14         <!-- Import Alfresco System Model Definitions -->
15         <import uri="http://www.alfresco.org/model/system/1.0" prefix="sys"/>
16
17         <!--**  import content model di docfiscali  **-->
18         <import uri="http://www.castgroup.it/model/docfiscali/1.0"  prefix="dfm" />
19     </imports>
20
21     <!-- namespace for your domain -->
22     <namespaces>
23         <namespace uri="http://www.castgroup.it/model/pa/1.0" prefix="pa"/>
24     </namespaces>
```

Figura 3.2

Nella zona degli import andiamo a caricare il nostro modello di docfiscali tramite l'import. Abbiamo infine modificato il namespace del nostro modello utilizzando come prefisso, l'abbreviazione del nostro cliente, ovvero "pa". Abbiamo ora la necessità di aggiungere tutti i tipi che avranno in più il nostro aspetto dichiarato precedentemente.

```
<types>
    <type name ="pa:TDocGenericodfmBase">
        <parent>dfm:TDocGenericodfmBase</parent>
        <mandatory-aspects>
            <aspect>pa:AProtocollo</aspect>
        </mandatory-aspects>
    </type>
    <type name ="pa:TDocFatturaPassiva">
        <parent>dfm:TDocFatturaPassiva</parent>
        <mandatory-aspects>
            <aspect>pa:AProtocollo</aspect>
        </mandatory-aspects>
    </type>
    <type name ="pa:TDocFatturaAttiva">
        <parent>dfm:TDocFatturaAttiva</parent>
        <mandatory-aspects>
            <aspect>pa:AProtocollo</aspect>
        </mandatory-aspects>
    </type>
    <type name ="pa:TDocDDTPassivo">
        <parent>dfm:TDocDDTPassivo</parent>
        <mandatory-aspects>
            <aspect>pa:AProtocollo</aspect>
        </mandatory-aspects>
    </type>
    <type name ="pa:TDocDDTAttivo">
        <parent>dfm:TDocDDTAttivo</parent>
        <mandatory-aspects>
            <aspect>pa:AProtocollo</aspect>
        </mandatory-aspects>
    </type>

    <type name ="pa:TODA">
        <parent>dfm:TODA</parent>
        <mandatory-aspects>
            <aspect>pa:AProtocollo</aspect>
        </mandatory-aspects>
    </type>
    <type name ="pa:TODV">
        <parent>dfm:TODV</parent>
        <mandatory-aspects>
            <aspect>pa:AProtocollo</aspect>
        </mandatory-aspects>
    </type>
</types>
```

Figura 3.3

Come si nota in figura 3.3 abbiamo chiamato i nostri documenti come nella modellazione generale dei documenti fiscali, abbiamo aggiunto come parent le tipologie di documenti generati nel precedente AMP, modificato il namespace e aggiunto il nostro aspetto obbligatorio.

Infine in figura 3.4 vi è la dichiarazione dell'aspetto che abbiamo aggiunto in precedenza su ogni tipologia di documento.

```
<aspects>
    <!-- ****Aspetto di Protocollo da aggiungere per richiesta del cliente -->
    <aspect name="pa:AProtocollo">
        <title>Aspetto Protocollo Codici</title>
        <description>aspetto che modella la specifica del Codice A Barre</description>
        <properties>
                <property name="pa:CodiceABarre">
                        <title>Codice A Barre</title>
                        <description>Variabile del codice a barre del documento</description>
                        <type>d:text</type>
                        <mandatory>true</mandatory>
                </property>
        </properties>
    </aspect>
```

Figura 3.4

3.2 DictionaryBootstrap AMP dell'All In One

L'ultimo aspetto che ci manca è di modificare il DictionaryBootstrap presente nel bootstrap-context.xml nella cartella "C:\progetti\pa-allinone\pa-allinone-repo-amp\src\main\amp\config\alfresco\module\pa-allinone-repo-amp\context". Dobbiamo registrare alla creazione del bean il nostro modello, ossia di non dover utilizzare più il dizionario presente nel nostro AMP dell'AIO, ma di utilizzare il dizionario della parte Repo dell'AMP installato. Andiamo a modificare la proprietà del bean chiamata "depends-on" mettendo l'ID bean del nostro AMP installato, ossia "*docfiscali-repo.dictionaryBootstrap*".

3.3 Implementare le modifiche sulla parte Share

Dobbiamo andare ad aggiungere le label per la parte Share, in modo che dallo Share sia più leggibile la scelta del tipo del documento e degli aspetti presenti. Come fatto in precedenza per creare l'AMP, dobbiamo andare a modificare il file "pa-allinone-share-amp.properties" presente in "C:\progetti\pa-allinone\pa-allinone-share-amp\src\main\amp\config\alfresco\web-extension\messages" (figura 3.5).

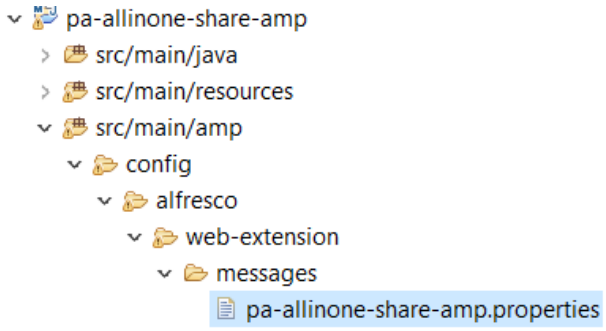


Figura 3.5

Il risultato dell'aggiunta dei tipi e degli aspetti sarà come in figura 3.6.

```
type.pa_TDoc=Documento Unico modello base(PA)
type.pa_TDocFatturaPassiva=Fattura Passiva(PA)
type.pa_TDocDDTPassivo=DDT Passivo(PA)
type.pa_TODA=Ordine Acquisto(PA)
type.pa_TODV=Ordine Vendita(PA)
type.pa_TDocFatturaAttiva=Fattura Attiva(PA)
type.pa_TDocDDTAttivo=DDT Attivo(PA)


aspect.pa AProtocollo= Aspetto Protocollo BarCode (PA)
```

Figura 3.6

3.3.1 Creazione dell'interfaccia di visualizzazione, modifica e ricerca

La nostra parte Share subito dopo l'overlay, ossia l'installazione del nostro AMP generale, sarà data dal "share-config-custom.xml" dell'AMP installato. Questa installazione non ci permette di vedere le nostre modifiche apportate alle tipologie di file, quindi dobbiamo andare a ricreare il nostro file di config-custom con una parte di riferimenti ai nostri nuovi dati e una parte riferita ai dati installati dall'AMP. Prendiamo in analisi solo alcuni pezzi di codice, perché il file intero è molto prolisso.

```
<config evaluator="string-compare" condition="DocumentLibrary">
        <aspects>
                <!-- Aspects that a user can see -->
                <visible>
                        <aspect name="pa:AProtocollo" />
                </visible>

                <!-- Aspects that a user can add. Same as "visible" if left empty -->
                <addable>
                </addable>

                <!-- Aspects that a user can remove. Same as "visible" if left empty -->
                <removeable>
                </removeable>
        </aspects>

        <types>
                <type name="cm:content">
                        <subtype name="pa:TDocFatturaPassiva" />
                        <subtype name="pa:TDocFatturaAttiva"/>
                        <subtype name="pa:TDocDDTPassivo" />
                        <subtype name="pa:TDocDDTAttivo"/>
                        <subtype name="pa:TODA"/>
                        <subtype name="pa:TODV"/>
                </type>
        </types>
</config>
```

Figura 3.7

Nel codice xml in figura 3.7 viene dichiarato che ci saranno gli aspetti e i tipi di documenti che abbiamo creato nel content-model.xml. Per l'utilizzo dei Form delle ricerche avanzate, cambierà rispetto all'AMP solamente il namespace, che da "dfm" diventerà "pa". Ora analizziamo il codice dello Share per la fattura passiva (figura 3.8).

```
<!-- Creazione nodo per Fattura Passiva -->
<config evaluator="node-type" condition="pa:TDocFatturaPassiva">
  <forms>
  <form>
        <field-visibility>
            <show id="cm:name" />

            <!-- tags and categories -->
            <show id="cm:taggable" for-mode="edit" force="true" />
            <show id="cm:categories" />

            <!-- dati del tipo -->
            <show id="dfm:numeroDocumento"/>
            <show id="dfm:dataDocumento" />
            <show id="dfm:annoDocumento" />
            <show id="dfm:tipoDocumentoFtPassiva" />
            <show id="dfm:codiceFornitore" />
            <show id="dfm:IVAFornitore" />
            <show id="dfm:RagioneSocialeFornitore" />
            <show id="dfm:CFFornitore" />
            <show id="pa:CodiceABarre"/>

        </field-visibility>
    <appearance>
            <set appearance="fieldset" id="docfiscaliDocFtPassivaTitle" label="Dati Fattura Passiva" />
            <field id="dfm:dataDocumento" set="docfiscaliDocFtPassivaTitle"/>
            <field id="dfm:annoDocumento" read-only="true"  set="docfiscaliDocFtPassivaTitle"/>
            <field id="dfm:numeroDocumento" set="docfiscaliDocFtPassivaTitle"/>
            <field id="dfm:tipoDocumentoFtPassiva" set="docfiscaliDocFtPassivaTitle"/>
            <field id="dfm:codiceFornitore" set="docfiscaliDocFtPassivaTitle"/>
            <field id="dfm:IVAFornitore" set="docfiscaliDocFtPassivaTitle"/>
            <field id="dfm:RagioneSocialeFornitore" set="docfiscaliDocFtPassivaTitle"/>
            <field id="dfm:CFFornitore"  set="docfiscaliDocFtPassivaTitle"/>
            <field id="pa:CodiceABarre"  set="docfiscaliDocFtPassivaTitle"/>
    </appearance>
  </form>
```

Figura 3.8

Da come notiamo prendiamo le variabili dell'AMP installato e in più mettiamo le variabili che abbiamo aggiunto tramite l'AIO (Stessa cosa per la parte di ricerca avanzata).

3.4 Moduli di estensione Surf

L'utilizzo dei moduli di estensione Surf è la miglior strada per personalizzare le caratteristiche dell'interfaccia grafica, rivolta agli utenti che utilizzeranno lo Share. Per personalizzare le nostre caratteristiche avremmo modificato il file "share-config-custom.xml", ma ciò non ci renderà possibile, tramite l'interfaccia grafica (UI), di disabilitare o abilitare le nostre configurazioni. Nel caso precedente, dopo aver installato l'AMP nell'AIO, non riusciamo a disattivare la parte di Share che fa riferimento ai docfiscali, visto che il cliente ha richiesto una specializzazione, la parte di docfiscali non ci interesserà più. Necessitiamo quindi di dover accedere al file System. L'utilizzo dei moduli Surf ci abilita il controllo dinamico da interfaccia grafica a tempo di esecuzione. Si potrà quindi aggiungere, aggiornare e rimuovere i nostri moduli versionati, senza aver bisogno di riavviare il Server per vedere le modifiche apportate. Le differenze tra il modificare lo "share-config-custom.xml" rispetto ai moduli d'estensione Surf si possono riassumere in una piccola tabella comparativa:

Opzioni	**share-config-custom.xml?**	**Surf Extension Module?**
Sovrascrivere le configurazioni di default in tomcat/webapps/share/WEB-INF/classes/*-config.xml files	✓	✓
Aggiungere, Aggiornare, e nascondere componenti nella Pagina Surf	✗	✓
Override Script Spring Surf Web	✗	✓
Caricare Risorse Web (CSS, JS)	✓	✓
Deploy e Un-Deploy delle configurazioni dall' UI	✗	✓
Configurazione Gruppi tramite "named packages"	✗	✓
Mantenere differenti versioni di un package di configurazione	✗	✓

3.5 Creazione dei moduli d'estensione

Per creare dei moduli d'estensione dobbiamo spostarci sulla parte Share dell'AMP che dobbiamo installare, quindi su "docfiscali-share". La cartella dove andremo a creare tanti file quanti tipi di documento che abbiamo in quel modulo AMP, sarà "C:\progetti\docfiscali-share\src\main\amp\config\alfresco\web-extension\site-data\extensions".

Creiamo questi file dandoci uno standard di nominazione. Lo standard utilizzato sarà dato da:

```
TYPE-[NAMESPACE]_[NOME DEL TIPO DI DOCUMENTO]-EXTENTION.XML
```

A fine della creazione di questi file, la cartella extensions si presenterà come in figura 3.9.

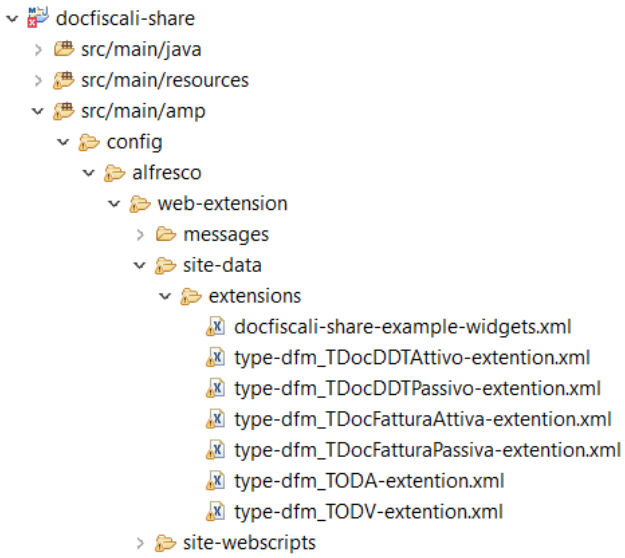


Figura 3.9

Nel file di esempio abbiamo il codice di come deve esser implementato il modulo di estensione. Partendo da questo esempio, implementiamo l'estensione del TODV. Le prime righe di codice implementeranno l'estensione e il modulo con le relative proprietà. Visioniamo le prime righe:

```
1 <extension>
2   <modules>
3     <module>
4       <id>type-dfm_TODV-extention</id>
5       <version>1.0</version>
6       <auto-deploy>false</auto-deploy>
```

Tramite il tag <id> viene definito il nostro id dell'estensione che verrà visualizzato nella parte di "Module Deployment Share", mentre definendo l'auto-deploy a false, eviteremo al momento dell'installazione del nostro AMP che venga anche direttamente già installata l'estensione di questo modulo.

```
<configurations>
  <config evaluator="string-compare" condition="DocumentLibrary" replace="false">
        <types>
           <type name="cm:content">
                   <subtype name="dfm:TODV" />
           </type>
        </types>
      </config>
      <!-- utilizzo di form per le ricerche avanzate-->
      <config evaluator="string-compare" condition="AdvancedSearch">
          <advanced-search>
              <forms>
                      <form label="Ordine Di Vendita" description="Ricerca per Ordine di vendita" id="searchDocTODV">
              </forms>
          </advanced-search>
      </config>
```

Figura 3.10

Nella sezione in figura 3.10 andiamo a definire il codice che deve aggiungere la nostra estensione quando viene aggiunta per l'utilizzo (ci sono le stesse identiche righe di codice del file "share-config-custom"). In aggiunta avremo il codice per la creazione del nodo dell'ordine di vendita e per la parte di ricerca avanzata. Dopo aver fatto tale procedimento per ogni nostro tipo di documento, e quindi finito di creare i moduli Surf, dobbiamo rifare un clean del nostro modulo AMP di "docfiscali-share", tramite il comando:

```
MVN CLEAN INSTALL -DSKIPTESTS
```

da lanciare tramite console nella cartella di "docfiscali-share".

3.6 Aggiunta dei moduli d'estensione tramite Share

L'aggiunta di questi moduli, come si è detto in precedenza, è possibile solo ed unicamente tramite interfaccia grafica Share. Dopo aver fatto partire il server, bisogna spostarsi all'URL http://localhost:8080/share/page/modules/deploy dove ci si presenterà un'interfaccia (figura 3.11) che ci permetterà di aggiungere i nostri moduli Surf creati precedentemente.

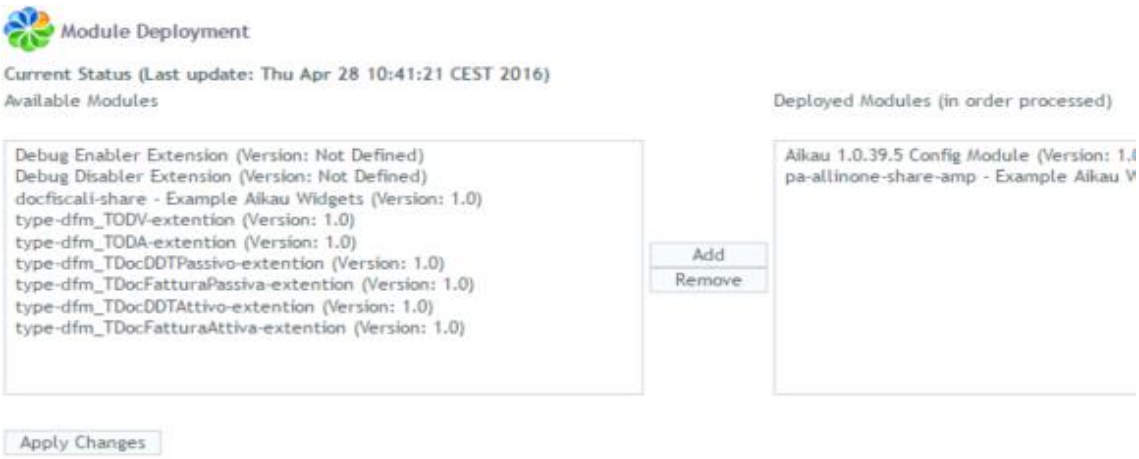


Figura 3.11

Mettiamoci nella situazione che il cliente PA ha richiesto delle specifiche in più come nel caso precedente, ma solo per l'ordine di vendita. Quindi vorremmo mantenere tutta la parte dell'AMP di docfiscali attiva, tranne quella per l'ordine di vendita che abbiamo già implementato in precedenza nelle specifiche dell'AIO (si tiene conto che nell'AIO sia implementata solo la parte relativa alle specifiche per l'ordine di vendita). Basterà semplicemente aggiungere tutti i moduli che non sono stati specificati nell'AIO tramite l'interfaccia precedente e poi confermare i cambiamenti.

Capitolo 4

4 Caricamento Massivo di Documenti

Potremmo pensare di caricare ogni documento a mano tramite l'interfaccia grafica, ma pensare di aggiungere tutti i documenti degli anni precedenti tramite l'interfaccia Share, dopo l'installazione del software dal cliente è impensabile. Per ovviare a questo problema ed effettuare un caricamento massivo e automatizzato di migliaia di documenti, dobbiamo andare a creare un importatore personalizzato per il nostro cliente.

4.1 Importatore

L'importatore ci andrà a facilitare il compito di importare in nostri documenti sulla piattaforma di Alfresco. Per fare ciò ci dovremmo appoggiare su un servizio chiamato CMIS. CMIS sta per Content Management Services Interoperability, esso è uno standard open che consente di utilizzare i sistemi di gestione dei contenuti (come Alfresco), di operare al suo interno manipolando i contenuti archiviati nel repository o la manipolazione attraverso internet, tramite l'utilizzo di protocolli web.

4.1.1 Funzionamento generale

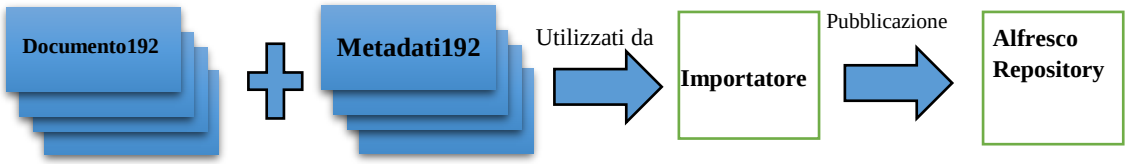


Figura 4.1 : Schema Rappresentativo del funzionamento generale

L'importatore andrà a prendere i documenti che risiedono in una determinata cartella del sistema, riconoscerà che al file vi sia associato il relativo file indice presente in un'altra cartella e caricherà il documento con i relativi metadati sul repository esatto di Alfresco (figura 4.1). Per fare ciò dobbiamo creare una gerarchia di cartelle sul nostro Filesystem come se fosse un archivio contenente le nostre informazioni. Ci spostiamo in "C:" ed andiamo a creare la suddetta gerarchia di cartelle come in figura 4.2.

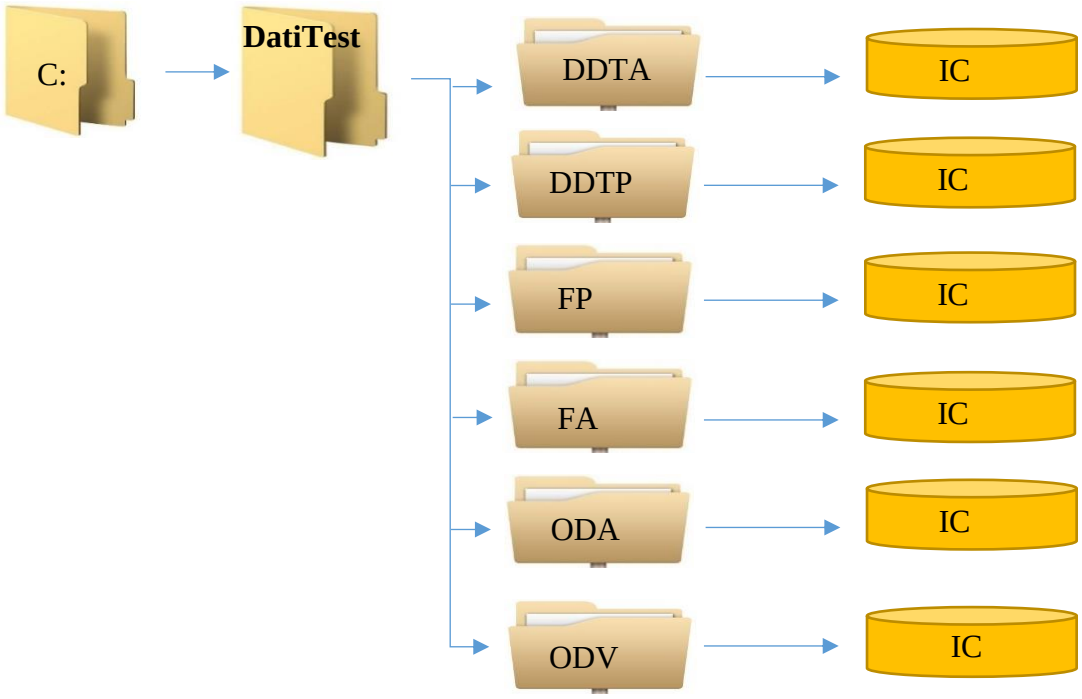


Figura 4.2 :gerarchia di cartelle, ove IC= Interno Cartella che è specificato in figura 4.3.

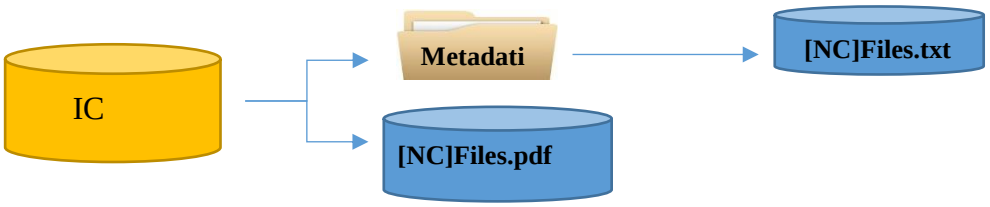


Figura 4.3 : gerarchia interna di cartelle di IC , ove NC= nome cartella

Avremo dentro ad ogni cartella tutti i documenti relativi alla tipologia della cartella e in più, ci sarà la cartella che contiene i file di indice, ove al suo interno, saranno contenuti i metadati per il relativo documento. Dobbiamo ricreare questa gerarchia anche in Alfresco. Quindi con il servizio di Alfresco in esecuzione sulla porta 8080 ci colleghiamo all'interfaccia Share, andiamo nella sezione "repository", creiamo la cartella "pa" e poi andiamo a creare tutte le cartelle per suddividere i documenti (come in figura 4.2), ma escluderemo l'IC.

4.1.2 Funzionamento logico con workflow

Il funzionamento logico dell'importatore si può riassumere tramite il workflow in figura 4.4.

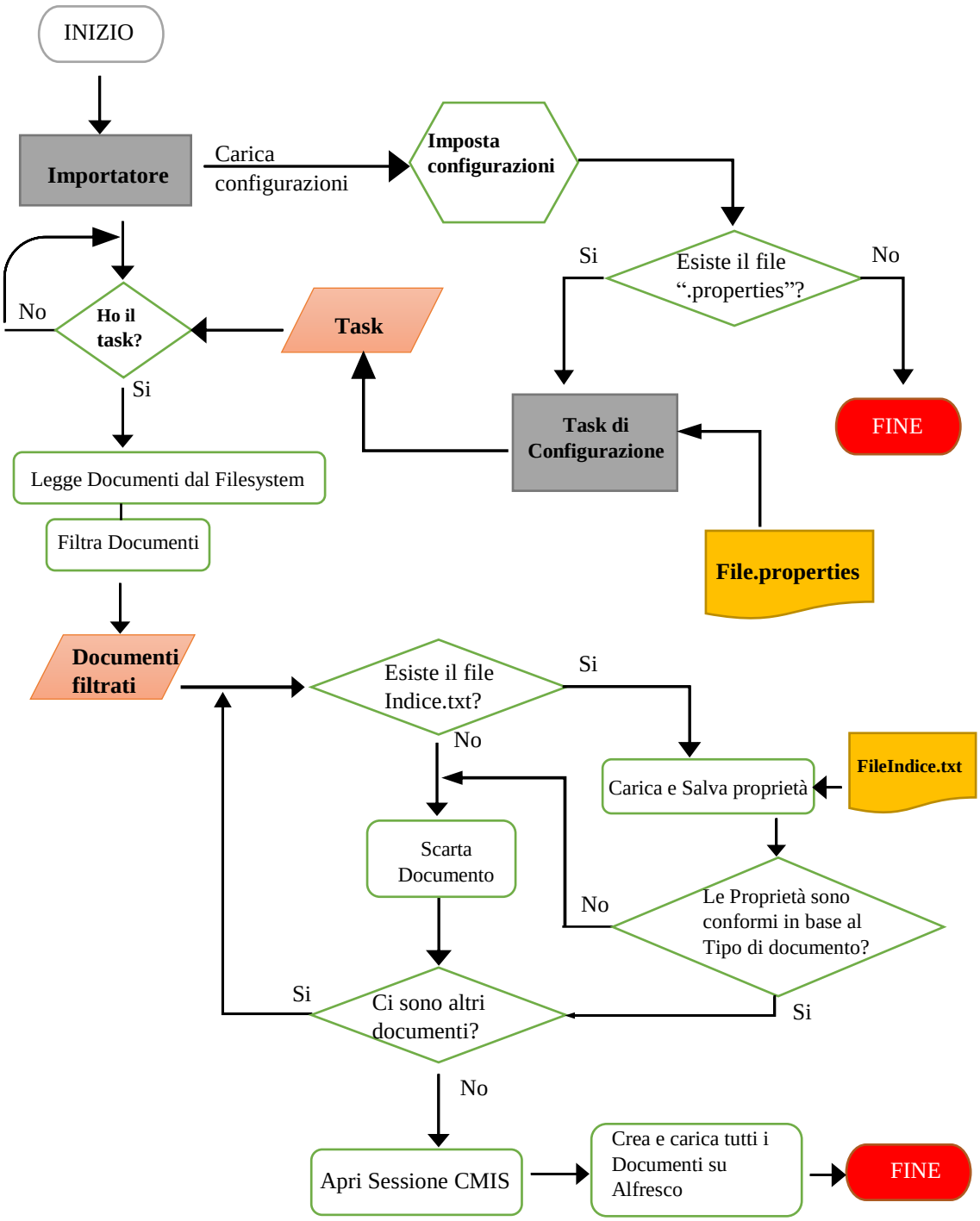


Figura 4.4 : workflow di logica del funzionamento

4.2 Creazione e composizione interna dell'importatore

Per incominciare creiamo il nostro progetto java tramite Eclipse. Una volta creato con il nome che vogliamo e con lo stesso JDK che abbiamo utilizzato precedentemente, dobbiamo andar a creare i nostri package e quindi suddividere le componenti all'interno del nostro importatore. Come namespace è stato deciso di utilizzare "it.netsphere.alfresco.importer". Ora andremo a creare tutte le cartelle come in figura 4.5.

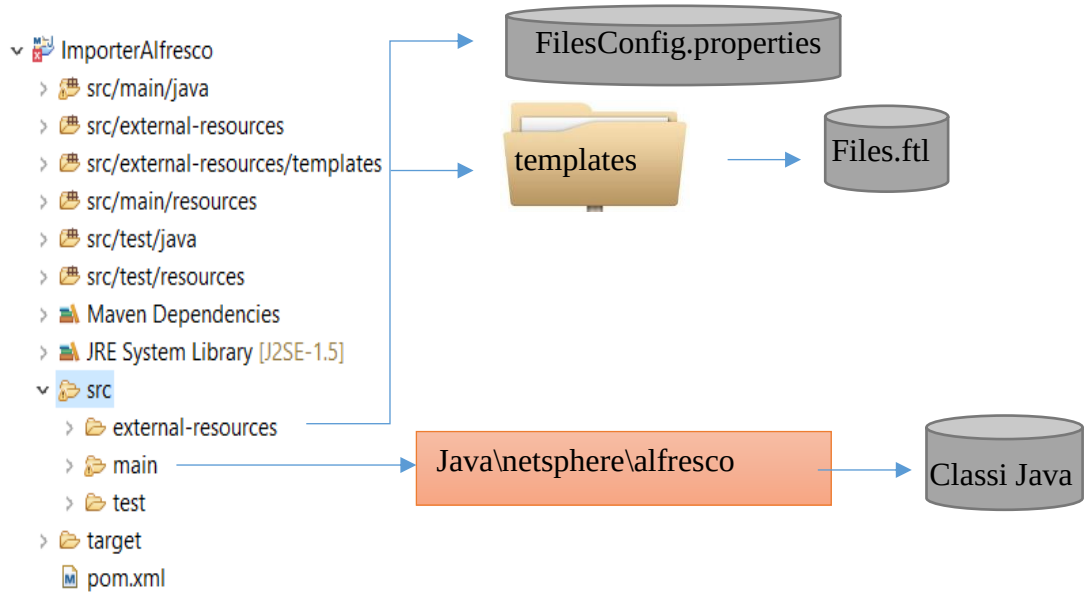


Figura 4.5

Le classi java saranno organizzate come in figura 4.6.

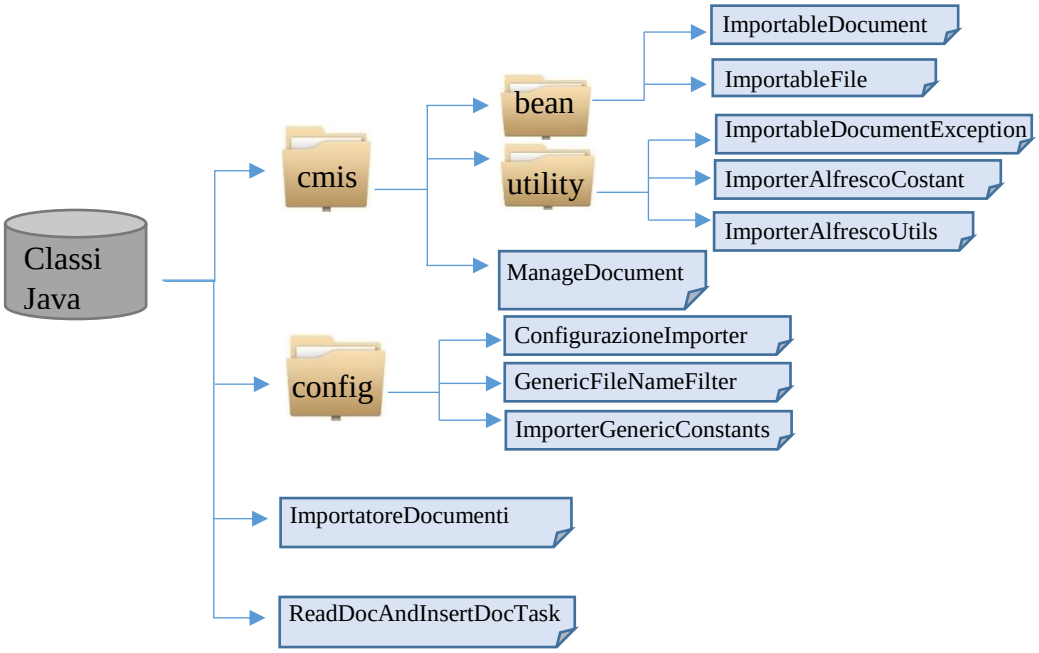


Figura 4.6

Nel susseguirsi del capitolo non farò vedere tutto il codice presente all'interno di ogni classe java, ma tutto l'importatore sarà reperibile dal link della sezione collegamenti file online. Andiamo ora ad analizzare i file presenti nel nostro importatore.

4.2.1 I file Config.properties

I file ".properties" sono quei determinati file, contenuti in external-resurce, che ci permettono di avere variabili di configurazioni. Queste variabili di configurazioni possono essere le configurazioni di accesso per utilizzare CMIS, le nostre logiche interne, la codifica dei file, le variabili di configurazione per le cartelle sul dove risiedono i documenti, le estensioni ecc. Ogni tipologia di file avrà il suo determinato file di properties con i relativi settaggi delle variabili. Analizziamo il ".properties" delle fatture passive che si chiama ConfigFP.properties (figura 4.7).

```
# Configurazioni accesso CMIS
cmis.repository.id =-default-
cmis.repository.url = http://localhost:8080/alfresco/api/-default-/public/cmis/versions/1.1/browser
cmis.repository.user = admin
cmis.repository.password = admin

# File da inserire
inbox.folder=C:/DatiTest/FP

#il namingFile è la variabile per dire inizia con:"FP" ,stessa cosa per per i metadati
inbox.namingFile=FP
inbox.namingFile_Metadati=FP

# il content_extension indica l'estensione che devono avere i documenti ,
# mentre index_extension indica l'estensione che dovranno avere gli indici
inbox.content_extension=.pdf
inbox.index_extension=.txt

# Variabile che indica dove sono i metadati di questo determinato settaggio
# e a seguire, quella dei template
inbox.metadati.folder=C:/DatiTest/FP/Metadati
template.folder=C:/progetti/ImporterAlfresco/src/external-resources/templates


# Pubblicazione documenti su Alfresco repository
destAlfrescoFolder=/pa/InboxFP

#logicaSovrascrittura=0
logicaSovrascrittura=1

#se a 1 cancella i file PDF e il file dei metadati, altrimenti li lascia anche a file system
inbox.delete.source.file = 0

index.file.encoding=UTF-8
```

Figura 4.7 :codice del file ConfigFP.properties

È importante ricordare che se si sposta l'importatore in un'altra cartella rispetto a quella indicata dalla variabile template.folder, bisognerà cambiare la suddetta variabile con il path esatto per arrivare alla cartella che contiene i template.

4.2.2 I file templates.ftl

Con il termine template, in un ECM, si intende un modello di documento, in questo tipo di file andremo a mettere uno scheletro del nostro tipo di documento, con tutte le variabili che vogliamo successivamente leggere dal nostro file di indice presente nella cartella dei metadati. Un template di esempio potrebbe essere quello della fattura passiva chiamato FP.ftl (figura 4.8).

```
<#assign dataDocumento = DATADOC?date("dd/MM/yyyy")>
type=pa:TDocFatturaPassiva
aspects=dfm:ADocumentoIVAdfmBase,dfm:AFornitore,pa:AProtocollo
cmis\:name=${inputFileName}
dfm\:numeroDocumento=${NUMDOC}
dfm\:dataDocumento=${dataDocumento?iso_local}
dfm\:annoDocumento=${ANNODOC}
dfm\:tipoDocumentoFtPassiva=${TIPOFATT}
dfm\:codiceFornitore=${CODFORN}
dfm\:IVAFornitore=${IVAFORN}
dfm\:RagioneSocialeFornitore=${RAGSOCFORN}
dfm\:CFFornitore=${CFFORN}
pa\:CodiceABarre=${BARCODE}
```

Figura 4.8

Come notiamo all'interno di questo template (figura 4.8) avremo il tipo che ha il documento, gli aspetti che sono applicati a questo tipo e tutte le variabili degli aspetti e tipi che andremo poi a recuperare. La struttura delle variabili è data da NomeVariabile←→${Riferimento}. Il riferimento è il nome della variabile che dovrà essere presente nel nostro file di indice (metadati). Quando il nostro importatore andrà a caricare questi file saprà esattamente come prelevare i metadati.

4.3 Classi Java dell'importatore

Occorre spiegare in modo generale come funzionano le classi java all'interno dell'importatore.

4.3.1 Classe ImportatoreDocumenti.java

Questa classe è la Main Class dell'importatore. Dal Main vengono fatti eseguire diversi metodi in base ai tipi di documenti che dobbiamo importare. Ogni metodo al suo interno carica le configurazioni per il determinato tipo di documento caricando il file **Config.properties**. Ogni variabile presente in esso si appoggerà alla classe **ConfigurazioneImporter.java** ,verrà poi creato il Task di configurazione tramite **ReadDocAndInsertTask.java** e infine farà partire il Task di importazione.

4.3.2 Classe ConfigurazioneImporter.java

All'interno della suddetta classe sono contenute le variabili globali che deve avere l'importatore come il "repositoryId", la "inboxFolder", la "inboxMetadatiFolder". Tramite il costruttore queste variabili di tipo stringa vengono configurate attraverso la classe **ImporterGenericCostants.java** che contiene gli effettivi nomi delle variabili che sono sul file di **Config.properties**. Ad esempio per "inboxFolder" andrà a prendere "inbox.folder" e salverà questa stringa nella variabile "inboxFolder". Alla fine delle configurazioni verrà ritornato il Task con all'interno le configurazioni.

4.3.3 Classe ReadDocAndInsertTask.java

ReadDocAndInsertTask.java va a creare 2 filtri tramite la classe **GenericFileNameFilter.java**, impostando l'estensione che devono avere i documenti, l'estensione che dovranno avere i file indice ed anche le relative cartelle in cui deve esser applicato questo filtro. Come secondo passo andrà a leggere tutti i documenti presenti nella cartella che abbiamo impostato e filtrerà questi file attraverso il filtro. Alla fine si avrà una lista di file che soddisfa il filtro. A questo punto per ogni elemento della lista, si andrà a controllare se ogni documento ha anche il relativo file indice associato (utilizzando anche il secondo filtro), e nel caso in cui questo non esista, il documento verrà scartato. Attraverso una sessione CMIS, creata tramite la classe **ImporterAlfrescoUtils.java**, per ogni documento vengono letti e convalidati i Metadati prima attraverso la convalidazione di CMIS, che tiene conto della correttezza delle proprietà scritte, successivamente tenendo conto del tipo e degli aspetti che gli sono stati applicati. Nel caso in cui un documento non è conforme viene scartato e non caricato, invece tutti i documenti che hanno passato questi test vengono caricati su Alfresco tramite la classe **ManageDocument.java**.

4.3.4 *Classe* **GenericFileNameFilter.java**

Questa classe implementa l'interfaccia **FilenameFilter.java** che contiene il metodo accept(). Questo metodo, implementato nella suddetta classe, verrà eseguito quando viene applicato ad una lista il filtro. Il metodo controllerà e ritornerà "true" se la stringa soddisfa i requisiti logici implementati, ossia che il path(URL) del file inizi in un certo modo e finisca come impostato alla creazione del filtro.

4.3.5 *Classe* **ImporterAlfrescoUtils.java**

La classe ImporterAlfrescoUtils andrà a creare, tramite il metodo openCmisSession(), una sessione con variabili necessarie per effettuare una connessione con Alfresco.

4.3.6 *Classe* **ManageDocument.java**

Questa classe crea il documento su Alfresco tramite il metodo addDocumentToAlfresco(), la sessione CMIS che gli viene passata e il documento importabile (**ImportableDocument.java**). A questo punto viene controllato se il documento importabile soddisfa i requisiti per Alfresco tramite la sessione CMIS.

4.3.7 *Classe* **ImportableDocument.java**

La classe ImportableDocument.java ha 4 metodi (principali):

- Metodo **validateMetadata**():
  Questo metodo recupera il tipo del documento tramite la sessione CMIS, le proprietà definite nel modello, carica tutte le proprietà del tipo del documento in base alla tipologia e agli aspetti e infine controlla che siano tutti conformi.
- Metodo **createDocument**():
  Esso carica le proprietà del tipo e degli aspetti e infine, tramite uno Stream, crea il file su Alfresco.
- Metodo **createOrUpdateDocument**():
  Il metodo viene utilizzato quando il file potrebbe già esistere su Alfresco. Nel caso in cui esista verrà fatto un update delle proprietà, nel caso in cui non esista, verrà creato.

- Metodo **updateProperties**():
  Questo metodo aggiorna le proprietà del documento passato come parametro.

Potrebbe essere necessario avere una visualizzazione dei package e dell'utilizzo delle classi tra loro come in figura 4.9.

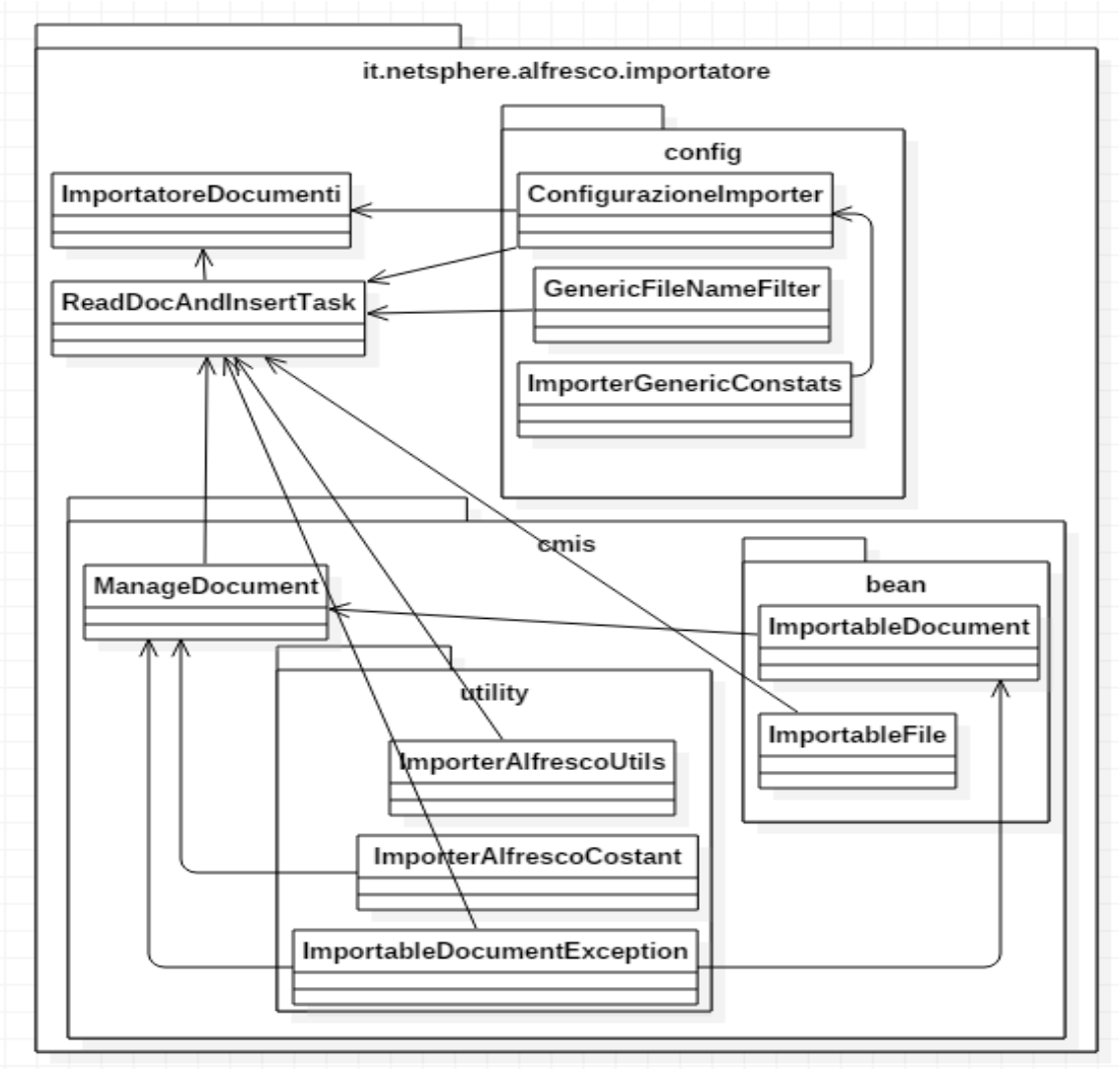


Figura 4.9: Package e utilizzo delle classi.

Appendici

Appendice A

5 Appendice A

Per raggiungere l'obiettivo posto dal cliente, abbiamo scelto di utilizzare il Software gestionale Alfresco.

5.1 Installazione di Alfresco (su Windows 7/8/10)

Per l'utilizzo di Alfresco verrà utilizzato:

- Alfresco Community Edition
- IDE Eclipse + Subclipse plug-in
- Tortoise SVN
- Maven
- SQLyog
- Java JDK 1.8.0_66

Per incominciare ad utilizzare Alfresco bisogna andare sul relativo sito e scaricare la Versione Community Edition (su https://www.alfresco.com/it/products/enterprise-content-management), creare una cartella in C di nome "alfresco-community", poi installare la versione di Alfresco in questa cartella appena creata. Come IDE verrà utilizzato Eclipse, quindi scaricare e installare Eclipse e creare una cartella in C chiamata eclipse-workspace, in modo che al primo avvio di Eclipse venga utilizzata quella cartella come spazio di lavoro.

Aperto Eclipse bisogna scaricare il plug-in "Subclipse" con questa procedura:

i. andare su HELP -> Install New Software -> selezionare dalla finestra "Subclipse" e spuntare tutti i plug-in da installare e infine continuare l'installazione.
ii. Scaricare e installare Tortoise SVN.
iii. Scaricare dal sito di Java l'SDK 1.8 ed installarlo nella cartella C:\jdk1.8.

5.2 Software Maven.

Maven è un progetto open source, sviluppato dalla Apache, che permette di organizzare in modo molto efficiente un progetto. I principali vantaggi che Maven fornisce sono:

- Standardizzazione della struttura di un progetto
- Test ed esportazione automatizzate

- Gestione e download automatico delle librerie necessarie al progetto
- Creazione automatica di un semplice sito di gestione del progetto contenente informazioni

5.2.1 I componenti principali di Maven:

- **Pom.xml** (POM, Project Object Model): questo è il file di configurazione; al suo interno sono contenute tutte le informazioni del progetto, come le dipendenze, test, documentazione, ecc.
- **Goal**: singola funzione che può essere eseguita sul progetto. I Goal possono essere sia specifici per il progetto dove sono inclusi, sia riusabili
- **Jelly script**: linguaggio XML con il quale vengono definiti i Goal
- **Plug-in**: goal riutilizzabili in tutti i progetti
- **Repository**: directory strutturata destinata alla gestione delle librerie. Un Repository può essere locale o remoto.

I principali plug-in disponibili sono i seguenti:

- **Clean**: che permette di cancellare i compilati dal progetto
- **Compiler**: che permette di compilare i file sorgenti
- **Deploy**: che permette di depositare il pacchetto generato nel Repository remoto
- **Install**: che permette di depositare il pacchetto generato nel Repository locale
- **Site**: che permette di generare la documentazione del progetto
- **Archetype**: che permette di generare la struttura di un progetto a partire da un template

I comandi Maven hanno la seguente struttura:

```
mvn <nome-plugin>:[<nome_goal>] [{<parametro>}]
```

Maven permette di creare un progetto a partire da un template che stabilisce la struttura base delle cartelle e dei file che devono essere creati (verrà utilizzato in seguito).

Per utilizzare Alfresco SDK si deve utilizzare Maven, quindi serve scaricarlo (dal sito: http://maven.apache.org/download.cgi) e installarlo. Dopo l'installazione di Maven bisogna impostare le variabili d'ambiente per permettere il corretto funzionamento.

5.3 Impostare le variabili d'ambiente per Maven

Per configurarlo bisogna andare in Pannello di controllo e cercare "Proprietà del sistema", poi cliccare su "Variabili d'ambiente…" e configurare le seguenti variabili:

- **PATH** aggiungendo il valore della cartella di Maven\bin (nel mio caso aggiungo: "C:\apache-maven-3.3.9\bin")
- **M2_HOME** con valore "C:\apache-maven-3.3.9"

- **MAVEN_OPTS** con valore: "-Xms256m -Xmx1G -noverify" (NB: Maven ha dei valori in meno se viene utilizzata la versione dell'SDK 1.8, se si utilizza JDK 1.7 aggiungere "-XX:PermSize=1024m")
- **JAVA_HOME** (nel caso in cui manchi) con valore:" C\jdk1.8"

Fatto ciò verifico che la mia installazione di Maven abbia avuto successo con un prompt dei comandi e digitando: "**mvn –version**".

5.4 Creazione Primo Progetto con Maven

Per creare il nostro primo progetto dobbiamo:

1. Creare una cartella in C (ad esempio: "progetti")
2. Ci si posiziona con un prompt dei comandi nella cartella appena creata e si immette:

"**mvn archetype:generate -Dfilter=org.alfresco**"

Utilizzando questo comando, maven si connetterà al server org.alfresco e scaricherà le tipologie di archetipi e di progetti impostati in modo standard da alfresco. A questo punto verrà richiesta la scelta dell'archetipo.

5.4.1 L' archetipo

L'utilizzo di un archetipo consente al programmatore di creare progetti base da cui far partire gli sviluppi, evitando le noiose fasi di configurazione, integrazione e i copia e incolla di sorgenti comuni. L'uso di un archetipo consente di definire dei progetti standard, la cui alberatura e le cui classi rispondono agli standard scelti, nel nostro caso gli standard sono scelti da Alfresco.

Ritornando alla creazione del progetto, si selezionerà la 2° scelta per la Repository e poi la 8° per la versione dell'archetipo. Per completare la creazione saranno chiesti altri parametri da inserire come:

- groupID: gruppo che rappresenterà il dominio (tipo "it.netsphere.alfresco")
- artifactID: che sarebbe la Repository (cartella) del nostro progetto (docfiscali-repo)

Esempio di output finale:

```
Define value for property 'groupId': : it.netsphere.alfresco
Define value for property 'artifactId': : docfiscali-repo
[INFO] Using property: version = 1.0-SNAPSHOT
Define value for property 'package': it.netsphere.alfresco: :
it.netsphere.alfresco.docfiscali
Confirm properties configuration:
groupId: it.netsphere.alfresco
artifactId: docfiscali-repo
version: 1.0-SNAPSHOT
package: it.netsphere.alfresco.docfiscali
```

Alla fine di questo output, verrà creata una struttura gerarchica di cartelle come la seguente:

```
    pom.xml
    src/main/amp
      |-> module.properties
      |-> file-mapping.properties (optional)
      |-> config/
      |-> web/
      |-> licenses/
    src/main/java             //Classi Java che saranno impacchettate nel mod
ulo AMP)
    src/main/resources        //Risorse che saranno impacchettate nel modulo
AMP)
    src/test/java             // Unità di tests
    src/test/resources        // Unità di test delle risorse
    src/test/properties/<env> // Ambiente per alfresco-global.properties.
                              // Di default<env>=local
```

3. Sempre dal prompt e nella cartella dei progetti si immette il comando:

”**mvn archetype:generate -Dfilter=org.alfresco**”

Verrà chiesto l’archetipo anche questa volta. Si utilizza la 3° e si reinserisco tutti i campi come prima. A questo punto sono stati creati 2 progetti di cui uno sarà il REPO e l’altro lo SHARE. A questo punto bisogna scaricare i file necessari e le librerie per far partire entrambi i progetti. Andiamo nella cartella di Repo e si fa partire il file “.bat” che in automatico scaricherà tutte le dipendenze, in modo analogo anche per la cartella Share.

5.5 Configurazione Progetti in Eclipse per Repo e Share

All’interno di Eclipse, nella zona dei progetti, premere tasto destro ->” Import” ->” import...” ->Maven-> Exiting Maven Project, poi vado in root directory browse e vado a prendere la cartella del progetto, ossia prima scelgo la cartella per Repo e poi vado ad aggiungere la cartella Share, spuntando per entrambe l’opzione “add projects to working set”.

5.5.1 Run Configuration

Per le configurazioni del run dei nostri due progetti in Eclipse, si deve andare a impostare le configurazioni. Bisogna posizionarsi nella barra degli strumenti in alto e “Run”->”Run Configuration”, mi si aprirà una finestra per la gestione delle configurazioni. Mi

posiziono sotto Maven e Facendo click sul pulsante “New launch configuration” (in alto a destra del riquadro) e configuro (come in figura 5.1).

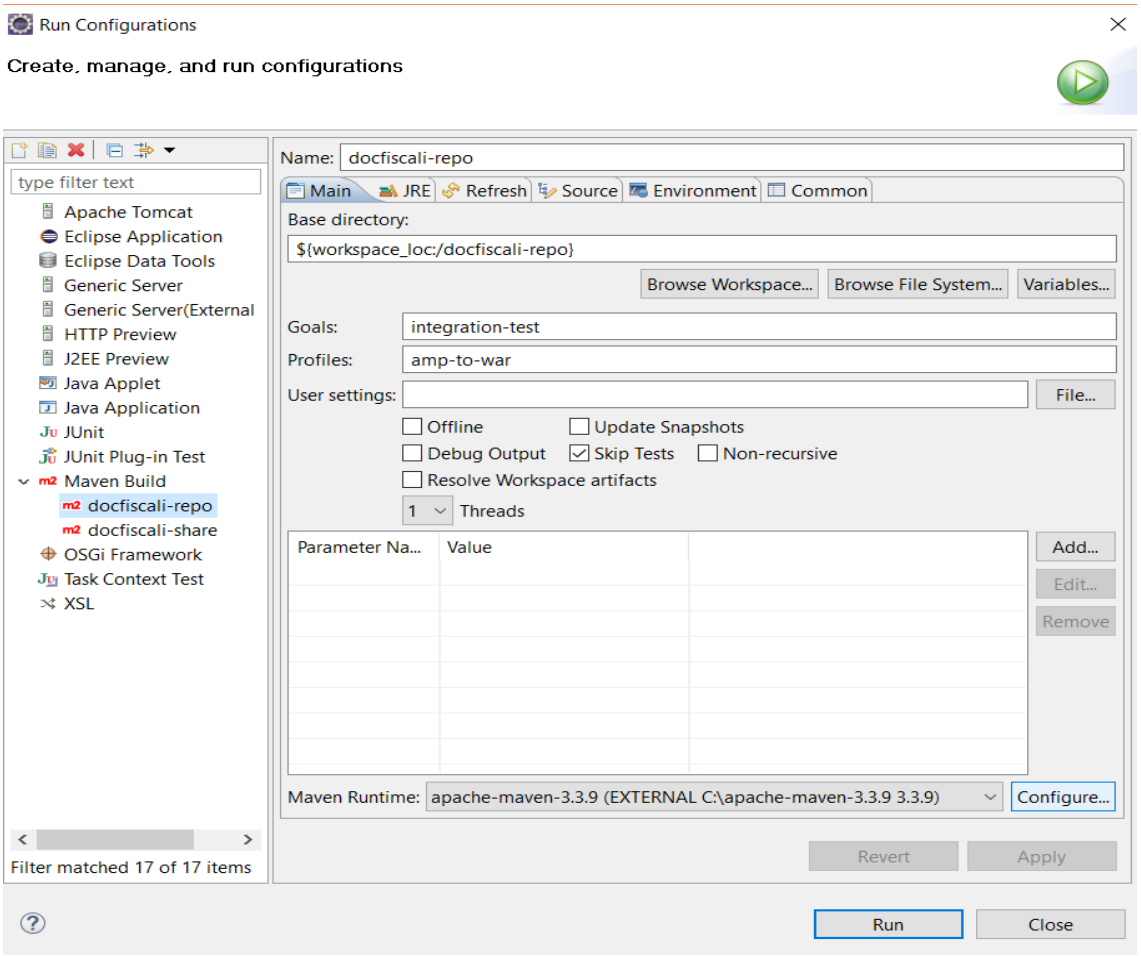


Figura 5.1

Imposto poi il Maven Runtime prendendo la cartella di apache Maven e mi posiziono sul JRE (figura 5.2).

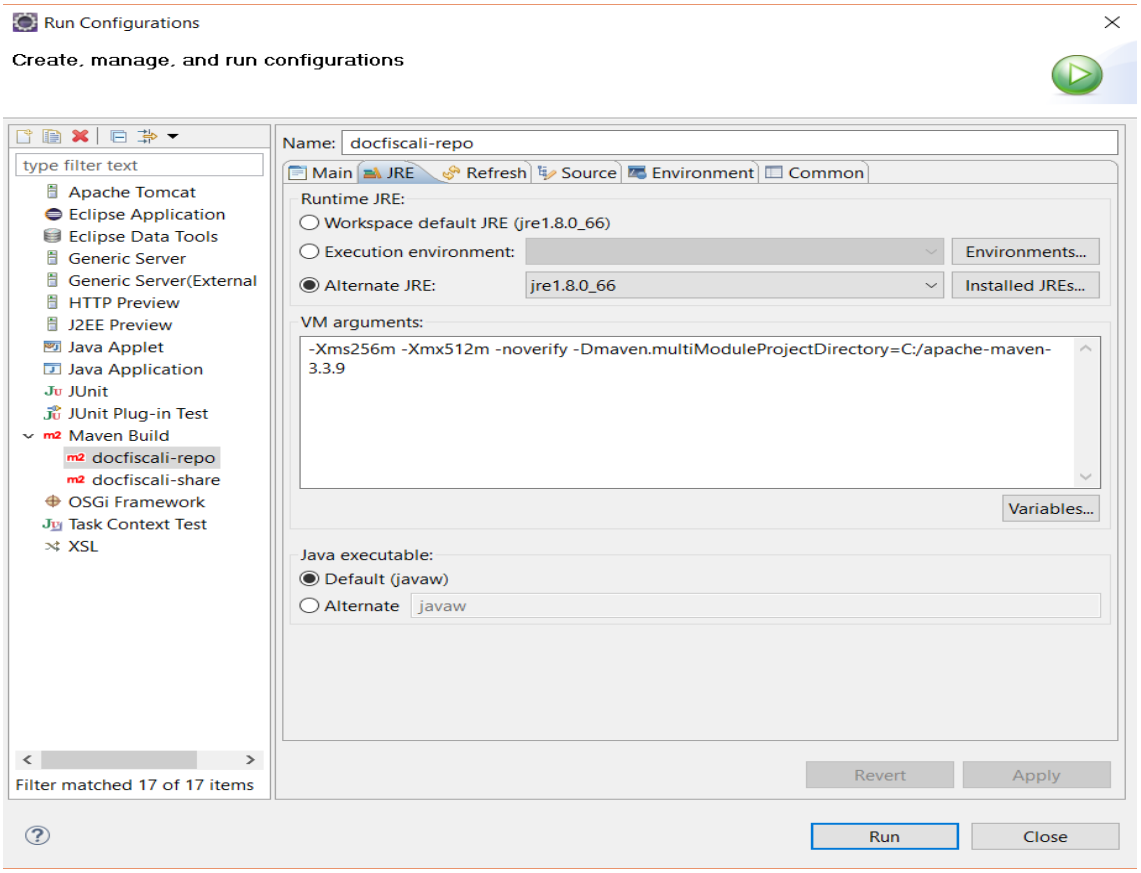
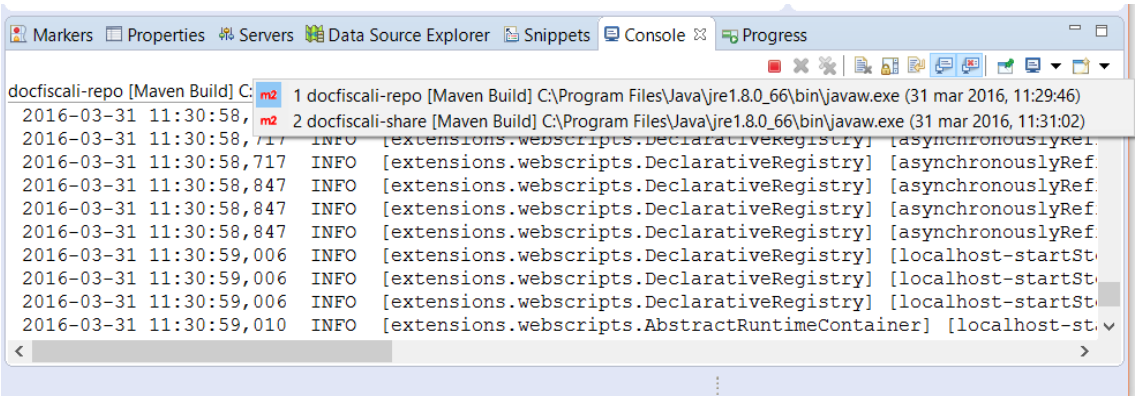


Figura 5.2

Applico tutti i cambiamenti e rifaccio lo stesso procedimento con lo Share facendo con il tasto destro su docfiscali-repo e “Duplica”. Infine cambio la base directory e metto quella della share.

Per far partire i servizi server per poi poter utilizzare la parte Repo e parte Share devo cliccare su Run Configuration -> mi posiziono su quello che voglio far partire e successivamente clicco “Run”. Bisogna ricordare che senza la parte di Repo in esecuzione, la parte Share non potrà mai funzionare. In basso a destra avrò questi due terminali che creeranno su questi due servizi:



Per poter accedere alla nostra interfaccia grafica, dopo aver lanciato i 2 servizi server, basterà accedere tramite un browser con URL:

- Per la parte grafica(Share) : http://localhost:8081/share/page/
- Per la parte di repository (Repo): http://localhost:8080/alfresco/

Appendice B

6 Appendice B

6.1 Indici, Ricerche e token in Alfresco

In informatica, un token è un blocco di testo categorizzato, normalmente costituito da caratteri indivisibili chiamati lessemi. Un analizzatore lessicale inizialmente legge i lessemi e li suddivide in categorie a seconda della loro funzione, dando loro un significato. Questa assegnazione di significato è chiamata **tokenizzazione** (o analisi lessicale). Un **token** può essere scritto in qualsiasi linguaggio, deve però avere una qualche utilità in un testo strutturato. Inoltre, bisogna tener presente, che nella programmazione, il codice sorgente si può suddividere in 5 classi di token (costanti, identificatori, operatori, parole riservate e separatori), in accordo con le regole del linguaggio di programmazione scelto.

I tipi di opzioni di ricerca che Alfresco supporta sono:

Opzioni di ricerca	**Descrizione**	**Cercabile**	**Tipi di dato Supportati**
None (nessuna)	La ricerca non è supportata. Viene utilizzato questo tipo di opzione se non si vuole usare la ricerca con quella proprietà.	✗	• int • long • float • double • date • datetime • boolean Dati di tipo testo: • text • mltext • content
Basic	La proprietà è ricercabile, ma i valori non saranno disponibili nella zona “ricerca tramite filtri”. (utilizzata solo per i tipi di dato non testo)	✓	• int • long • float • double • date • datetime • boolean

Enhanced search	Viene utilizzato questo tipo se si vuole sfruttare l'uso di quella proprietà per effettuare query di faceting, stats, ordinamento e di range. NB: questa opzione aumenta le performance delle query, ma richiede l'utilizzo di più spazio sul disco per la ricerca degli indici.	✓	• int • long • float • double • date • datetime
Free text	La proprietà è ricercabile, ma i valori non saranno disponibili nella zona "ricerca tramite filtri". (utilizzata solo per i tipi di dato testo)	✓	• text • mltext • content
List of values – whole match	Questa opzione di indicizzazione viene utilizzata per abilitare la possibilità di filtrare tramite questo campo nella zona "ricerca tramite filtri" quando cerchiamo per l'intero termine	✓	• text • mltext • content
List of values – Partial match	Questa opzione di indicizzazione consente di filtrare (su una proprietà) i risultati di una ricerca, utilizzando i caratteri Jolly come: % ,? , * ecc..	✓	• text • mltext • content
Pattern – Unique matches	Questa opzione abilita la possibilità di utilizzare identificatori univoci che vengon cercati sulla base dell'intero valore di della proprietà. (La proprietà non verrà mostrata come filtri nella zona "ricerca tramite filtri")	✓	• text • mltext • content

Pattern – Many matches	Questa opzione di indicizzazione abilita l'uso di identificatori che potrebbero esser cercati sulla base dell'intero valore del testo o tramite filtro. (La proprietà non verrà mostrata come filtri nella zona "ricerca tramite filtri")	✓	• text • mltext • content

Differenti valori possono avere diversi impatti sulle performance delle ricerche, richieste di memoria e spazio su disco richiesto per l'installazione. Per usi di ingenti dimensioni dei Repositories, i valori che assume un indice può essere di significante impatto, specialmente se le proprietà vengono utilizzate per ricerche frequenti.

6.2 Creazione del progetto AIO

Per poter creare un progetto All In One basta posizionarsi tramite un terminale, nella cartella dei progetti in "C:\progetti" e inserire il comando:

"`mvn archetype:generate -Dfilter=org.alfresco:`".

Si aprirà una creazione interattiva per il progetto:

1. Bisognerà impostare l'archetipo per l'All In One, dunque selezioniamo il numero 1.
2. Selezioniamo la versione dell'archetipo più recente (nel mio caso la 2.2.0, ossia la 8).
3. Dobbiamo ora inserire il GroupID, l'artifactId e il package. Per fare questo, visto che prima non esisteva uno standard, abbiamo impostato come:
    a. GroupID: it.netsphere.alfresco (per ogni progetto)
    b. Artifact ID: pa-allinone (NomeCliente-archetipo)
    c. Package: it.netsphere.alfresco.pa (GroupID.NomeCliente)

Fatto ciò, maven creerà tutte le dipendenze del nostro progetto. Ora siamo pronti per importare il progetto AIO appena creato in Eclipse.

6.3 Importazione del progetto AIO in Eclipse

Per poter lavorare con Eclipse dobbiamo importare il progetto su di esso. Basta posizionarsi sul lato sinistro del Project Explorer, tasto destro -> import -> Existing Maven Project -> tramite il browser selezionare la cartella del nostro progetto AIO ed importare. Al momento dell'import verranno importati: pa-allinone, pa-allinone-repo-amp, pa-allinone-share-amp, repo, share, runner, solr-config.

Al momento dell'import è possibile che ci siano degli errori di importazione da risolvere come il "Web Import error" sia per la parte "repo" che "share". Questo errore si presenta perché Maven si aspetta che ci sia il file web.xml nel progetto, proprio perché è una web application, tuttavia per le recenti web applications, il file è totalmente opzionale. Per risolvere questo errore, dobbiamo andare ad aggiungere un plugin nel pom.xml di entrambi sotto le dichiarazioni dei build, plugin.

Ecco il plugin aggiunto:

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.6</version>
        <configuration>
        <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
</plugin>
```

(NB: si devono andare a modificare i pom relativi solo alla la parte “repo” e “share”). Nel caso in cui gli errori siano ancora presenti: tasto destro-> “Quick Fix”.

6.4 Creazione del Run (AIO)

Per far eseguire per la prima volta il nostro servizio server e quindi tutto il nostro progetto, dobbiamo andare a configurare il nostro Run. Ci basterà andare nel “Run Configurations”, copiare un run di un altro progetto maven che già abbiamo preimpostato in precedenza (o crearne uno nuovo), cambiare la directory di base e impostare i valori come in figura 6.1.

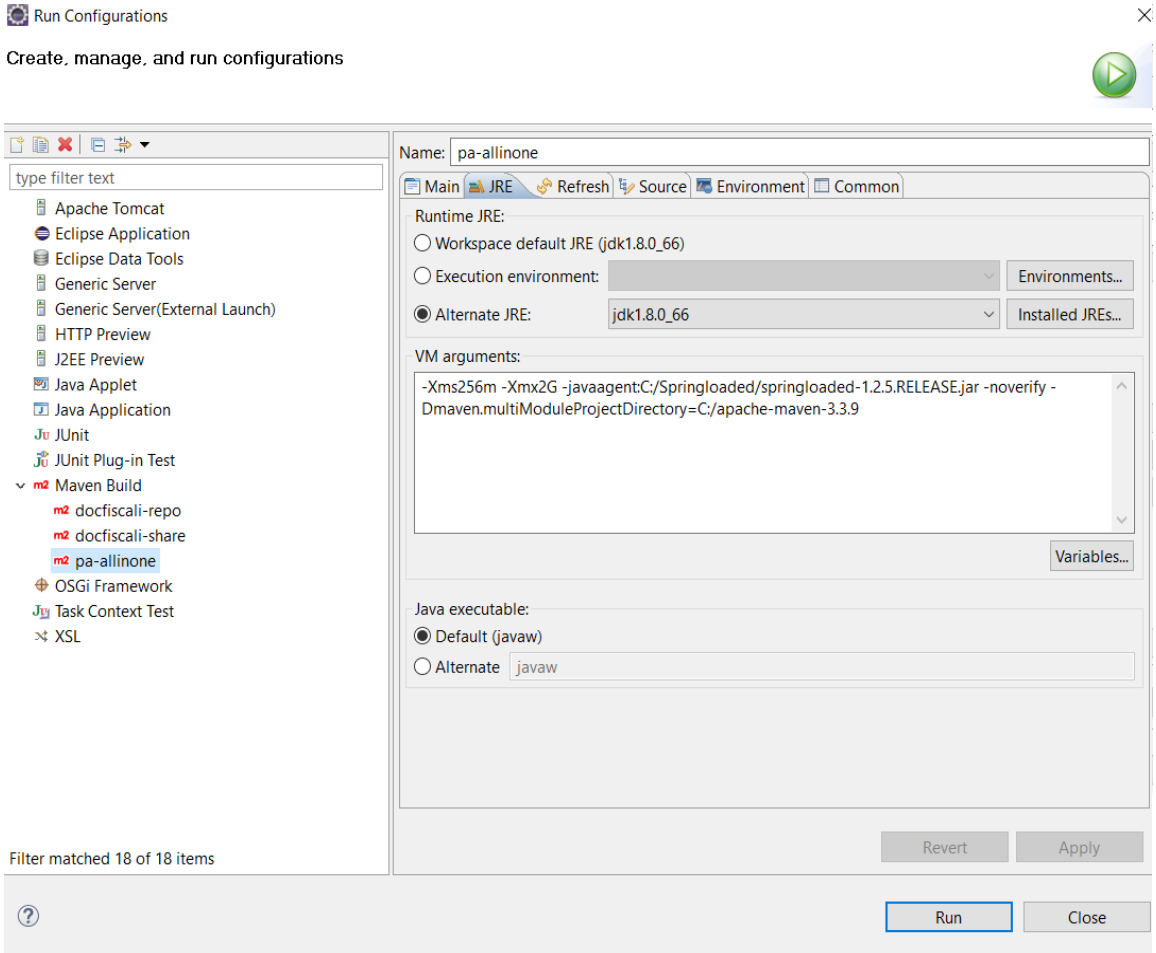


Figura 6.1

La prima volta che si avvierà tramite run il nostro processo, si dovrebbe impiegare una decina di minuti perché ci sono da scaricare tutte le dipendenze per poterlo far partire (nel caso in cui non si sia ancora installato lo springloaded bisogna andare al punto seguente o semplicemente omettere il -javaagent… -noverify).

I collegamenti per la visualizzazione dei servizi attivati sono:

- Per la parte Repo: http://localhost:8080/alfresco/
- Per la parte Share: http://localhost:8080/share/page/

6.5 Installazione di Springloaded

Springloaded è un agente Java che permette di ricaricare a "caldo" le classi in una JVM (Java Virtual Machine) in esecuzione. Esso permette di aggiungere, modificare, cancellare metodi, campi e costruttori. Le annotazioni sui tipi, metodi, campi e costruttori possono esser modificati ed è possibile anche aggiungere, rimuovere e modificare i valori in tipi enum. Lo Springloaded.jar viene utilizzato spesso nei progetti Alfresco All-in-one per evitare che ad ogni modifica di un modulo AMP bisogni riavviare tutti i servizi server. Nei progetti All-in-one ci sono sempre lo Share e Repo, ma si avrà un unico file "run.bat" per far partire il servizio server Share (sulla porta 8080) e Repo (sulla porta 8080).

Per utilizzare e quindi installare Springloaded con progetti Maven bisogna:

[1]. Scaricare la versione della libreria di Springloaded preferita (https://github.com/spring-projects/spring-loaded)

[2]. Creare una cartella e mettere il Springloaded.jar al suo interno (es: C\Springloaded)

[3]. Modificare il run.bat presente nella cartella del progetto (All-in-One) Maven creato in precedenza aggiungendo la riga che imposta la variabile d'ambiente MAVEN_OPTS (dopo la prima riga di codice):
SET MAVEN_OPTS=-Xms256m -Xmx2G -javaagent:C:/Springloaded/springloaded-1.2.5.RELEASE.jar -noverify

[4]. Per colpa di un bug in Alfresco bisogna andare a modificare il bootstrap-context.xml presente nella parte Repo del progetto (\repo\src\main\alfresco\extension\module\NomeProgettoAIO\context) aggiungendo un Bean "fittizio":

```
<!-- workaround for running SpringLoaded -->
   <bean id="unserializerValidatorBootstrap" abstract="true" />
```

Nel caso si voglia utilizzarlo in un progetto non All-in-one bisogna ad andare a modificare il bootstrap presente in "repo\src\main\amp\config\alfresco\module\docfiscali-repo\context" e ovviamente fare anche il punto 3 per il file run.bat sia della parte Repo che su quella della Share.

Nel caso si stia utilizzando Eclipse come IDE e si vuole utilizzare Springloaded per il run del progetto Maven bisogna:

1. Fare i passi 1 e 2 precedentemente elencati
2. Andare a modificare gli argomenti della VM (Virtual Machine). Per fare ciò si deve andare (con Eclipse già aperto) in: "Run -> Run Configurations…" e posizionarsi nel Run Maven da modificare e modificare i precedenti argomenti

con: -Xms256m -Xmx2G -javaagent:C:/Springloaded/springloaded-1.2.5.RELEASE.jar -noverify

3. Aggiungere il Bean “fittizio”:
   <!-- workaround for running SpringLoaded -->
   <bean id="unserializerValidatorBootstrap" abstract="true" />
   nel bootstrap-context.xml del progetto.

6.6 Installazione di un modulo AMP tramite linking al progetto AIO

Per poter installare il nostro modulo AMP base che abbiamo implementato precedentemente (doc-fiscali), abbiamo bisogno di modificare il pom.xml di “repo” e “share”. La parte di “repo” e “share” è come se fosse il nostro Alfresco. Qui possiamo aggiungere e modificare tutte le informazioni per poter far partire i servizi server con le dipendenze che ci servono ed effettuare l’utilizzo dei moduli tramite l’installazione degli amp.

6.6.1 Aggiungere la dipendenza (repo)

Aggiungere la dipendenza per il nostro AMP serve a caricare, in fase di “start”, tutte le dipendenze che sono presenti nella nostra parte Repo dell’AMP ed anche tutte le definizioni come il content-model. Ci posizioniamo ed andiamo a vedere il codice del file pom.xml presente in “repo” ed aggiungiamo la dipendenza in figura 6.2.

```
<!-- Amp Aggiunti -->
<dependency>
    <groupId>it.netsphere.alfresco</groupId>
    <artifactId>docfiscali-repo</artifactId>
    <version>1.0-SNAPSHOT</version>
    <type>amp</type>
</dependency>
```

Figura 6.2: Dipendenza della parte Repo dell’AMP

Per impostare la dipendenza del modulo da installare, abbiamo preso le informazioni dal file pom.xml della parte Repo dell’AMP che dobbiamo aggiungere. Nel nostro caso eravamo come in figura 6.3.

```
pa-allinone-share-am...   docfiscali-repo/pom....   share-config-custom.x...   content-model.xml   pa-allinone-repo-amp...   bootstrap-context.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5      <groupId>it.netsphere.alfresco</groupId>
6      <artifactId>docfiscali-repo</artifactId>
7      <version>1.0-SNAPSHOT</version>
8      <name>docfiscali-repo Repository AMP project</name>
9      <packaging>amp</packaging>
10     <description>Manages the lifecycle of the docfiscali-repo Repository AMP (Alfresco Module Package)</description>
11
```

Figura 6.3

6.6.2 Aggiungere l'overlay (repo)

Aggiungere l'overlay ci permette di applicare il nostro AMP come se fosse una patch dell'AMP già esistente del progetto AIO. Aggiungiamo nel file pom.xml dell'AIO sotto la dichiarazione delle overlay, ossia il codice di overlay presente in figura 6.4 (preso sempre dal pom.xml dell'AMP che dobbiamo installare).

```
<!-- overlay AMP aggiunto -->
<overlay>
    <groupId>it.netsphere.alfresco</groupId>
    <artifactId>docfiscali-repo</artifactId>
    <type>amp</type>
</overlay>
```

Figura 6.4: Overlay Amp

6.6.3 Aggiungere la dipendenza (share)

Andiamo a modificare con lo stesso procedimento di prima, il file pom.xml della parte Share dell'AMP dell'AIO, aggiungendo (il codice come in figura 6.5) le dipendenze dello Share AMP che dobbiamo caricare.

```
<dependency>
    <groupId>it.netsphere.alfresco</groupId>
    <artifactId>docfiscali-share</artifactId>
    <version>1.0-SNAPSHOT</version>
    <type>amp</type>
</dependency>
```

Figura 6.5: dipendenza della parte Share-Amp

6.6.4 Aggiungere l'overlay (share)

Ora sempre nel file pom.xml della parte Share dell'AMP dell'AIO aggiungiamo l'overlay presente in figura 6.6.

```
<!-- overlay aggiunto -->
<overlay>
    <groupId>it.netsphere.alfresco</groupId>
    <artifactId>docfiscali-share</artifactId>
    <type>amp</type>
</overlay>
```

Figura 6.6

Abbiamo così finito la nostra installazione dell'AMP sull'AIO. L'unica modalità per vedere se l'installazione è andata a buon fine, visto che nei file di log non è presente l'installazione dell'AMP, è di andare a testare a mano se l'installazione ha avuto consistenza e quindi andare a fare dei test con tipologie di file e modifiche.

7 Bibliografia

[1]. Installing Alfresco:
http://docs.alfresco.com/5.0/concepts/master-ch-install.html

[2]. Content Model:
http://docs.alfresco.com/5.1/references/dev-extension-points-content-model.html

[3]. Form:
https://community.alfresco.com/docs/DOC-5408-forms

[4]. Share extensions:
http://docs.alfresco.com/5.1/concepts/dev-extensions-share.html

[5]. Custom type:
http://ecmarchitect.com/alfresco-developer-series-tutorials/content/tutorial/tutorial.html

[6]. Custom action:
http://ecmarchitect.com/images/articles/alfresco-behavior/behavior-article.pdf

[7]. Creating a new property for custom types and aspects
http://docs.alfresco.com/community/tasks/admintools-ct-properties-create.html

[8]. Node Ref:
https://community.alfresco.com/docs/DOC-5359-noderef-cookbook#Reading_a_property_of_a_node

[9]. Creating All in one project:
http://docs.alfresco.com/5.0/tasks/alfresco-sdk-tutorials-all-in-one-archetype.html

[10]. Add Custom AMPs to an AIO project:
http://docs.alfresco.com/5.1/concepts/alfresco-sdk-advanced-add-custom-amps.html

[11]. Surf Extension Modules:
http://docs.alfresco.com/5.1/concepts/dev-extensions-share-surf-extension-modules.html

8 Collegamenti online per i file

- Importatore Java per documenti di Alfresco:
  https://drive.google.com/open?id=0B2ejMq9KnRy1RFFJOFZuSXYxZEk

- Codice per la parte di repository del modulo AMP (Repo):
  https://drive.google.com/open?id=0B2ejMq9KnRy1YkN4RlgweHl3czQ

- Codice per la parte Share del modulo AMP (Repo):
  https://drive.google.com/open?id=0B2ejMq9KnRy1cmdqa3VsN1ZoSjA

- Codice del progetto All In One:
  https://drive.google.com/open?id=0B2ejMq9KnRy1MWNmZVFwbXlqdWM