

POLITECNICO DI MILANO

Polo Regionale di Como

Facoltà di Ingegneria dell'Informazione

Corso di Studi in Ingegneria Informatica



Chansonnier: applicazione Web di ricerca multimediale su video musicali

Relatore: Prof. Piero Fraternali
Correlatore: Ing. Alessandro Bozzon

Elaborato finale di
Giorgio Sironi
Matr. 714673

A.A. 2009/2010

Abstract

La ricerca multimediale e' un campo in grande sviluppo che comprende l'indicizzazione di grandi quantita' di dati in formato video, audio e grafico e il loro accesso tramite richieste mirate come la ricerca di testo libero e l'applicazione di filtri.

Chansonnier e' un caso di studio che utilizza strumenti open source e servizi Web pubblici per costruire un indice di canzoni a partire da video musicale presenti sul Web.

Chansonnier permette di effettuare ricerche nel proprio indice su parametri quali lingua, artista ed emozione espressa, generando insiemi di canzoni che soddisfano le richieste e visualizzando i risultati in un'interfaccia Web dinamica. È disponibile anche un'interfaccia orientata all'interoperabilità con altre applicazioni, sotto forma di servizio Web.

Ringraziamenti

Desidero ringraziare l'Ingegnere Bozzon e il Professor Fraternali per il tempo dedicatomi durante lo sviluppo del caso di studio e la scrittura di questa tesi.

Un grazie va anche alla mia famiglia che mi ha permesso di continuare a studiare oltre la scuola superiore.

Indice

1	Introduzione.....	1
1.1	Problema.....	1
1.2	Obiettivi e contributo.....	1
1.3	Organizzazione del documento.....	1
2	Caso di studio.....	3
3	Background.....	4
3.1	Motori di ricerca multimediali.....	4
3.1.1	Analisi dei dati.....	4
3.1.2	Tipi di ricerca.....	6
3.2	Annotazione.....	7
3.2.1	Audio.....	7
3.2.2	Immagini.....	7
3.2.3	Video.....	7
3.2.4	Testo.....	8
3.3	Applicazioni AJAX e Web 2.0.....	8
3.3.1	Tecnologico.....	8
3.3.2	Sociale.....	10
3.3.3	Servizi Web.....	10
	REST.....	11
3.4	SMILA.....	13
3.4.1	Modello dei dati.....	14
	Blackboard.....	14
3.4.2	Architettura.....	16
	Connectivity.....	17
	JMS Queue.....	17
	Processing.....	17
	BPEL Pipelines.....	17
	Webservice.....	21
	Pipelet e ProcessingService.....	21
	Scelta architetturale.....	25
	Search.....	26
	Interfaccia Web.....	26
	Search Pipelines.....	27
	Storage.....	27
	XmlStorage.....	28
	BinStorage.....	28
	Service Specific Store (Lucene, Solr).....	29
3.4.3	Distribuzione.....	30
	Connectivity.....	30
	Processing.....	31
	Pipeline.....	31
	Search.....	32
	Storage.....	32
	Scalabilità.....	32
	Altri mezzi di distribuzione.....	33
	Cloud computing.....	34
	Limiti di utilizzo.....	35
4	Design.....	37

4.1 Modello dei dati.....	37
4.2 Annotatori.....	38
4.2.1 LyricWiki	38
4.2.2 Google Translate	39
4.2.3 Synesketch	40
4.2.4 Ffmpeg.....	41
4.3 Servizi OSGi.....	42
4.3.1 LinkGrabberAgent	42
4.3.2 LyricsProcessingService	44
4.3.3 EmotionProcessingService	46
4.3.4 LanguageProcessingService	47
4.3.5 FrameExtractionProcessingService.....	48
4.3.6 SolrPipelet.....	50
4.3.7 LastIndexedService.....	51
4.4 Indice.....	51
5 Implementazione.....	54
5.1 Metodologia.....	54
5.1.1 Source control.....	54
5.1.2 Test-Driven Development.....	54
End-to-end (aka acceptance).....	55
HttpUnit.....	55
Selenium.....	57
SolrJ.....	60
Functional.....	60
Integration.....	63
Unit	67
Unit test in OSGi (Plugin tests)	67
Unit test senza OSGi (plain old JUnit tests)	71
5.1.3 Automazione.....	72
Build.....	72
Test suite.....	74
Solr.....	74
Selenium.....	74
5.2 Pattern.....	75
5.2.1 Dependency Injection	75
Fra servizi OSGi.....	75
All'interno di un bundle.....	76
5.2.2 Bridge.....	76
5.3 Tecnologie.....	78
5.3.1 OSGi.....	78
5.3.2 BPEL.....	79
5.4 Interfaccia.....	81
5.4.1 Utente.....	81
Servlet.....	82
Ajax.....	83
5.4.2 Esempio di utilizzo.....	83
5.4.3 Api.....	88
6 Conclusione.....	91
Bibliografia.....	93

1 Introduzione

1.1 Problema

I motori di ricerca disponibili sul Web presentano una mancanza di capacità di ricerca di file multimediali come tracce audio e video, nonostante negli ultimi anni la banda di trasmissione e la potenza di calcolo comunemente disponibile abbiano reso possibile la trasmissione e l'elaborazione di risorse multimediali attraverso la rete e non più solo localmente.

In generale, un'applicazione per la ricerca sul Web deve eseguire differenti processi per fornire risultati consistenti ai propri utenti:

- analisi e indicizzazione: trasformazione dell'informazione in una forma adatta alla ricerca, ed estrazione di metadati.
- Query e presentazione di risultati: ricezione di richieste di diverso tipo da parte del motore di ricerca e produzione dei rispettivi risultati (insiemi di entità che soddisfano i criteri di ricerca).
- Interazione utente: produzione di query da parte dell'interfaccia utente e visualizzazione dei risultati in una forma orientata all'utente o alla macchina.

Mentre questi processi sono comunemente attuati su contenuto di tipo testuale o ad esso assimilabile, non esistono soluzioni standard per implementarli su contenuti multimediali. Il campo di ricerca che interessa i motori di ricerca multimediali è molto ampio.

1.2 Obiettivi e contributo

Il contributo di questa tesi riguarda l'utilizzo di tecnologie open source allo scopo di costruire un sistema di ricerca multimediale. Il caso di studio scelto è un sistema per l'indicizzazione e la ricerca di canzoni e, conseguentemente, degli artisti ad esse associati. Il nome scelto per l'applicazione che implementa questo caso di studio è Chansonnier.

1.3 Organizzazione del documento

Il testo di questo documento è organizzato come segue:

- Il capitolo **Caso di studio** introduce i requisiti dell'applicazione di ricerca multimediale che si vuole costruire.
- Il capitolo **Background** affronta le tematiche della ricerca multimediale e delle interfacce Web, offrendo una panoramica delle soluzioni esistenti.
- Il capitolo **Design** delinea l'architettura dell'applicazione e i vari componenti che ne fanno

parte, siano essi codice nativo, servizi Web o software open source integrato in essa.

- Il capitolo **Implementazione** contiene un rapporto sulla metodologia applicata per lo sviluppo di Chansonnier e documenta l'implementazione dei suoi vari componenti.
- Il capitolo **Conclusione** riassume i risultati ottenuti dall'implementazione del caso di studio.

2 Caso di studio

Il caso di studio, denominato qui e in seguito Chansonnier, è un'applicazione per l'indicizzazione di canzoni provenienti dal Web, e la loro successiva ricerca.

Chansonnier è totalmente guidato da un'interfaccia Web, sia per quanto riguarda la scoperta di nuove canzoni da indicizzare, sia per quanto riguarda la ricerca da parte di un utente.

Le funzionalità dell'applicazione del caso di studio sono espressi dalle seguenti user story:

- un utente può inserire un link di una pagina del portale video YouTube affinché Chansonnier possa indicizzare la canzone contenuta nella pagina
- un utente può ricercare alcune parole di un testo e risalire alle canzoni contenenti tali parole
- un utente può visualizzare, per ogni risultato di una ricerca, la parte di testo, autore o titolo che contiene la propria query
- un utente può visualizzare tutti gli attributi di indicizzazione nei risultati di una ricerca, così da poterla raffinare
- un utente può seguire il link alla pagina originale della canzone a partire dal risultato di una ricerca.
- un utente può visualizzare fotogrammi del video di una canzone nei risultati della ricerca allo scopo di identificare la canzone e l'artista in questione
- un utente può filtrare la ricerca richiedendo le canzoni associate a un determinato artista
- un utente può filtrare la ricerca richiedendo le canzoni che esprimono una determinata emozione
- un utente può filtrare la ricerca richiedendo le canzoni in una certa lingua
- un utente può combinare diversi attributi in una sola ricerca, come artista, lingua e emozione delle canzoni richieste.
- un servizio esterno può accedere a un'Api di ricerca orientata alle macchine anziché all'utente umano, allo scopo di utilizzare Chansonnier in ambito di ricerca multidominio.

3 Background

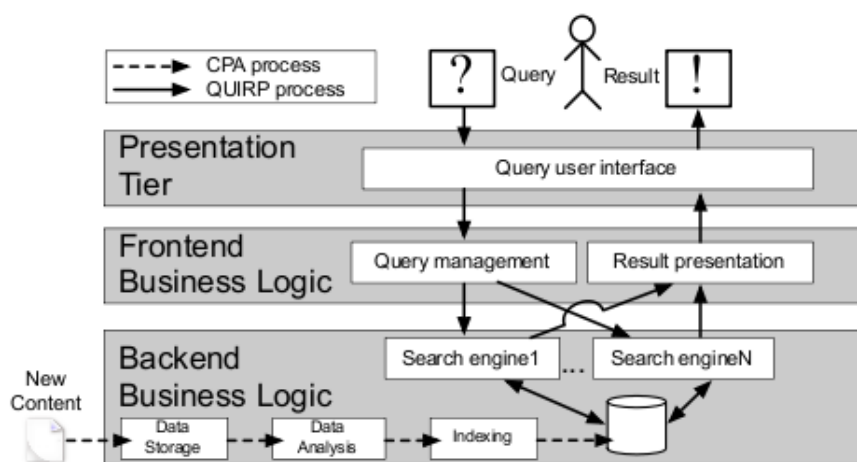
3.1 Motori di ricerca multimediali

La ricerca multimediale (Multimedia Information Retrieval) si occupa di dell'accesso a risorse di tipo multimediale quali video, audio, ed immagini. Mentre la ricerca di tipo testuale è un problema sostanzialmente risolto nelle soluzioni software moderne, la ricerca multimediale è un campo in grande sviluppo.

Un motore di ricerca multimediale immagazzina quindi contenuti non solamente testuali ma di tipo binario, sottoponendoli a un processo di annotazione affinché essi diventino accessibili in una successiva ricerca.

I processi di annotazione (Content Provision and Annotation, CAP da qui in poi) hanno lo scopo di introdurre nuovi dati nel sistema, e di produrre annotazioni su di essi, le quali rappresentano informazioni estratte dai contenuti testuali e binari indicizzati dall'applicazione. I processi di ricerca (Query and Result Presentation, QUIRP da qui in poi) comprendono invece le operazioni effettuate dal sistema per interpretare le richieste dell'utente e produrre un risultato in uscita.

La seguente figura, tratta da ¹, descrive l'architettura di un applicazione di ricerca multimediale che accomoda tali processi.



3.1.1 Analisi dei dati

In base al tipo di contenuto, vi sono diversi processi applicabili ad esso per estrarne delle informazioni adatte alla ricerca all'interno dei processi CAP. Questa analisi dei dati può riguardare sia contenuti direttamente inseriti nel sistema di ricerca stesso, sia ricorsivamente metadati estratti da tali contenuti da un precedente processo di analisi.

I contenuti di tipo testuale possono essere sottoposti a procedure come:

- riconoscimento lingua (language detection): associare ad un testo la lingua in cui è scritto, sia allo scopo di inserirlo in una categoria specifica, sia allo scopo di differenziare il successivo processo di analisi in base al risultato di questa attività.
- normalizzazione dei caratteri (char normalization): eliminazione di caratteri speciali come accenti o diresesi, allo scopo di consentire il matching tra differenti espressioni scritte con grafia diversa (come perché, perchè, o Strasse, straÙe)
- correzione ortografica (spell checking): sostituzione di forme linguistiche o ortografiche scorrette con un equivalente univoco.
- lemmatizzazione (lemmatization): riduzione di parole in forma flessa ad una radice comune (da praticità, pratica, pratiche a pratico)
- rimozione di stopwords (stopword removal): eliminazione di parole molto comuni o irrilevanti, come articoli, preposizioni o pronomi personali.



Illustrazione 1: Lemmatizzazione nel motore di ricerca Google

I contenuti di tipo multimediale seguono differenti procedure per la propria indicizzazione:

- transcoding: trasformazione del proprio formato in uno equivalente o “minore”, allo scopo di essere accessibili per diversi annotatori. L'estrazione di una traccia audio da un video musicale rientra in questo processo.
- classificazione: assegnazione di un contenuto multimediale ad una determinata categoria, come nel caso di canzoni al genere musicale.
- feature extraction: estrazione di altri contenuti multimediali da un'annotazione, come ad esempio i fotogrammi di un video.

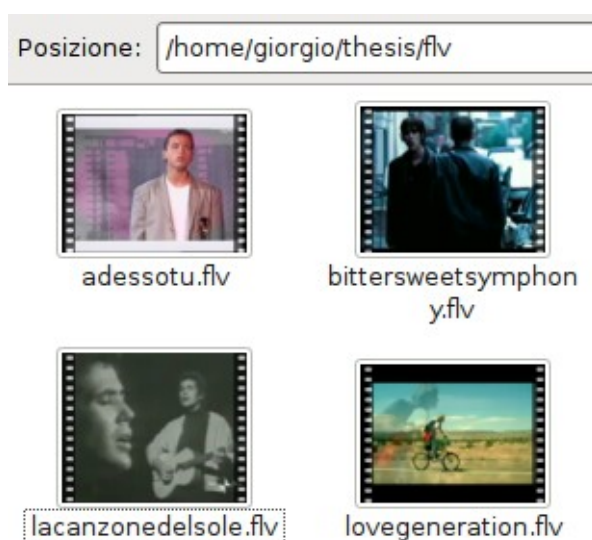


Illustrazione 2: Feature extraction effettuata da Ubuntu Linux

3.1.2 Tipi di ricerca

I motori di ricerca in genere offrono diverse soluzioni per la propria interfaccia di ricerca:

- testuale (full text): l'indice è organizzato come testo non strutturato e liberamente ricercabile. Le query sono non strutturate. I motori di ricerca disponibili sul Web sfruttano molto questo tipo di approccio.
- fielded: l'indice è suddiviso in campi, ed è possibile specificare query che riguardino i valori di particolari campi. La necessità di omogeneità tra i dati indicizzati rende questo approccio adatto per motori di ricerca che lavorano su un singolo dominio.

I motori di ricerca multimediali possono sperimentare nuovi paradigmi di ricerca, grazie al particolare tipo di contenuti:

- basata sul contenuto (content-based): i contenuti multimediali sono indicizzati tramite annotazioni di basso livello e non direttamente interpretabili da un umano. La ricerca viene quindi svolta fornendo contenuti di esempio al sistema, i quali vengono annotati a loro volta per estrarre dei contenuti simili dall'indice.

3.2 Annotazione

Per estrarre informazione dai contenuti da indicizzare e porla in un formato adatto alla ricerca, è necessario introdurre un processo di annotazione nel quale una catena di annotatori producano i metadati desiderati, basandosi su dati originali o su di annotazioni introdotte da annotatori precedentemente.

I prodotti di un processo di annotazione possono essere utili per diversi motivi.

- **Immagazzinamento:** i diversi valori di un'annotazione possono fare riferimento a una determinata categoria a cui il singolo contenuto appartiene. Contenuti appartenenti alla stessa categoria possono essere immagazzinati insieme per ragioni di efficienza e di semplicità di accesso.
- **Ricerca:** il motivo principale che spinge a mettere in atto un processo di annotazioni è la possibilità di indicizzarne i risultati ed effettuare ricerche su di essi, trasformando ad esempio una ricerca multimediale in una di tipo testuale.
- **Confronto:** annotazioni di modesto peso presentano generalmente miglior semantica per il confronto dei dati binari originali e possono servire all'associazione di contenuti simili fra loro.

A seconda del tipo di dato da processare, sia esso un contenuto in forma di audio, video, immagine o testo, è possibile l'applicazione di diverse tecniche.

3.2.1 Audio

- **segmentazione:** divisione di una traccia audio in differenti segmenti.
- **speech recognition:** riconoscimento delle frasi pronunciate in una traccia e produzione di una trascrizione.
- **speaker identification:** identificazione della persona o gruppo di persone che ha prodotto l'audio sotto analisi.
- **music classification:** identificazione del genere di una traccia musicale.

3.2.2 Immagini

- **optical character recognition:** riconoscimento di segmenti di testo rasterizzati all'interno di un'immagine.
- **face recognition:** riconoscimento di persone (o oggetti) all'interno di un'immagine.

3.2.3 Video

- **shot detection:** divisione di un video in scene realizzate in un'unica ripresa.

- frame extraction: estrazione di singoli fotogrammi da una traccia video
- video optical character recognition: riconoscimenti di testo ripreso all'interno di un video.

3.2.4 Testo

- language detection: insieme alle tecniche di normalizzazione descritte nella sezione Motori di ricerca multimediali.
- emotion detection: discovery dell'emozione principale espressa da un testo.

L'utilizzo di queste tecniche in associazione le une con le altre porta ad un ricco insieme di annotazioni in diversi formati che possono servire alla ricerca basata sul testo o sul contenuto.

Ad esempio l'identificazione di singole scene tramite shot detection permette di estrarre un fotogramma da ognuna di esse in modo da costruire un campione significativo dell'insieme di tutti i frame; successivamente si possono sottoporre ad OCR questi fotogrammi e analizzarne il testo. Questo è solo uno degli esempi di pipeline di annotatori che si può costruire collegando diverse tecnologie.

3.3 Applicazioni AJAX e Web 2.0

AJAX (Asynchronous JavaScript and XML) è un gruppo di tecnologie client-side per lo sviluppo di applicazioni Web interattive. A differenza di altre tecnologie raccolte sotto l'ombrello di Rich Internet Applications, AJAX è nativamente supportato, entro certi limiti, dai browser moderni.

L'insieme delle applicazioni AJAX e delle altre tecnologie ha dato origine al fenomeno che prende il nome di Web 2.0 e che comprende due aspetti principali.

3.3.1 Tecnologico

Nell'ambito tecnologico, le Rich Internet Applications fanno un uso fondamentalmente differente del client HTTP rispetto ai loro predecessori.

Nelle applicazioni e siti Web, l'interazione fra client e server era limitata a eventi ben precisi, corrispondenti alla “selezione” di link o alla spedizione di form. AJAX disaccoppia le richieste effettuate al server da eventi cablati nel browser, permettendo agli sviluppatori di un'applicazione di associare la “produzione” di richieste a diversi eventi generati dall'utente.

L'interazione con il server è asincrona, in quanto le richieste HTTP non sono bloccanti e l'utente può continuare ad usare l'applicazione senza accorgersi delle sottostanti richieste continuamente effettuate verso il server.

L'ultima parte dell'acronimo AJAX, XML, è il nome del formato prevalentemente utilizzato nelle prime iterazioni di applicazioni per lo scambio di dati fra client e server. Non vi è però una stretta limitazione sul formato da utilizzare, purché sia interpretabile dall'applicazione caricata precedentemente dal client.

Si possono estrarre quattro principi fondamentali da questo approccio che definiscono un'applicazione Web come realizzata secondo il paradigma AJAX²:

- il browser ospita un'applicazione, non del contenuto. Le pagine Web quindi rappresentano un'entità con del comportamento associato per far fronte alle azioni dell'utente, non semplici documenti come quelli che HTTP ha gestito fin dagli albori del protocollo.
- Il server “consegna” dati, non contenuto, come risposta alle richieste HTTP da parte dell'applicazione. I dati sono in un formato interpretabile dal codice dell'applicazione, e non quindi direttamente interpretabile da una persona umana.
- L'interazione con l'applicazione può essere fluida e continua. Non c'è mai la stretta necessità di dover effettuare un ricaricamento completo della pagina dell'applicazione, o di cambiare l'URL presente nella barra del browser. L'attesa dovuta alla latenza della rete (il tempo che intercorre fra una richiesta di dati e la sua risposta) può essere gestita dall'applicazione stessa.
- Il codice funzionale viene eseguito sul client, e una parte fondamentale dell'applicazione è quindi scritta in un linguaggio lato client come JavaScript.

I vantaggi di AJAX risiedono quindi nel trasformare un browser da una piattaforma di visualizzazione di documenti remoti in una adatta all'esecuzione di applicazioni.



Illustrazione 3: Yahoo nel 1997



Illustrazione 4: Yahoo nel 2010

3.3.2 Sociale

Il termine Web 2.0 comprende anche una componente sociale, che riguarda le aspettative e la percezione del Web da parte delle ultime generazioni di utenti. Gli utenti moderni sono oramai abituati ad utilizzare il Web come mezzo di comunicazione e socializzazione, come magazzino delle proprie esperienze, e ancora come enciclopedia in costante espansione.³

Alcuni esempi di tratti “riconoscitivi” del Web 2.0 nelle applicazioni Web moderne sono⁴:

- La transizione della pubblicazione amatoriale di contenuti da home page personali a piattaforme di blogging.
- L'utilizzo per il download di dati di sistemi distribuiti che coinvolgono le macchine degli utenti finali (BitTorrent) invece che di sistemi di mirroring (Akamai). Le maggiori distribuzioni di Linux offrono le proprie ISO sulla piattaforma BitTorrent come mezzo per alleggerire il carico sui propri server.
- Il passaggio dalla pubblicazione di contenuti online (come Britannica Online, la versione accessibile via Web dell'Enciclopedia Britannica) alla costruzione di contenuti da parte degli utenti del servizio stessa (wiki e Wikipedia).
- Il passaggio da forme di screen scraping (estrazione di dati da pagine HTML) a Web services, i quali rendono disponibili gli stessi dati in un formato intelleggibile da macchine adeguatamente programmate, e che rispecchia uno degli standard...
- La transizione da rigide forme di catalogazione gerarchica (taxonomy) a un insieme di etichette (tag) applicati ai singoli elementi di informazione (folksonomy), che li rendono accessibili tramite diverse gerarchie allo stesso tempo, o tramite ricerche basate su facet.

Categorie: [Fisici italiani del XVI secolo](#) | [Fisici italiani del XVII secolo](#) | [Filosofi italiani del XVI secolo](#) | [Nati il 15 febbraio](#) | [Morti l'8 gennaio](#) | [Matematici italiani del XVI secolo](#) | [Matematici italiani del XV](#) | [legate a Padova](#) | [Storia della scienza](#) | [Persone condannate per eresia](#) | [\[altre\]](#)

Illustrazione 5: Categorizzazione tramite tag della voce Galileo Galilei di Wikipedia

3.3.3 Servizi Web

Service Oriented Architecture è un paradigma per la realizzazione di sistemi distribuiti basato su tre concetti: servizi, bus di comunicazione e basso accoppiamento fra di essi.⁵

Un servizio può essere definito come un entità “self-contained” che rende disponibile una funzionalità.

Perchè differenti servizi possano comunicare tra loro, è necessario un canale di comunicazione che sia accessibile come elemento comune da essi.

Il concetto di basso accoppiamento, infine, riguarda la necessità di ridurre le dipendenze fra le

varie parti di un sistema distribuito, in modo che gli effetti di una modifica o guasto non si propaghino attraverso il sistema.

Una possibile implementazione di SOA è quella proposta dai servizi Web. Nel caso che una tale soluzione sia adottata, i servizi in questione hanno una facciata HTTP accessibile tramite un URL. Il bus di comunicazione diventa idealmente il Web stesso (come insieme di nodi che parlano HTTP) o un suo sottoinsieme. I servizi Web così costruiti possono essere consumati sia da utenti finali, che da altri servizi che si pongano fra l'utente e le sorgenti originali dei dati, allo scopo di integrare più funzionalità o connettere differenti domini.

Grazie alla rivoluzione portata dal paradigma AJAX, sempre più applicazioni espongono un'Api sottoforma di servizio Web, in quanto tale Api è comunque necessaria al loro funzionamento come Rich Internet Applications. Questo “trend” solleva la problematica di uno standard di comunicazione stabilito fra servizi Web e i loro consumer.

REST

Sebbene uno standard che permetta la comunicazione di due servizi Web non sia univocamente definito, diverse proposte sono state avanzate nel corso degli anni. Questi standard sono orientati alla trasmissione di documenti fra servizi e consumatori, dato che il Web e il corrispondente protocollo HTTP sono sistemi orientati alle risorse e non all'invocazione di procedure remote (Remote Procedure Call).⁶

Un primo esempio di protocollo costruito al di sopra di HTTP allo scopo di permettere la comunicazione fra servizi è Simple Object Access Protocol (SOAP). Questo protocollo, insieme alle sue varie specifiche WS-*, prevede di incapsulare dati in formato XML nell'ambito di un documento conforme alla specifica (SOAP envelope), che correda i dati di intestazioni necessarie alla gestione della comunicazione. Il documento così formato viene spedito come entità all'interno di richieste o risposte HTTP.

Uno stile architetturale di molto più semplice utilizzo e adozione è Representational State Transfer, che prescrive di utilizzare direttamente il vocabolario HTTP per la gestione della comunicazione. Le architetture RESTful, quindi, utilizzano header HTTP, URI, MIME type e altri metadati definiti dalle specifiche HTTP 1.0 e 1.1 per definire e interpretare i metadati relativi alle richieste e risposte HTTP, eliminando quindi un livello di astrazione sopra di esso.

Lo stile architetturale REST è definito da sei principi:⁷

- separazione client/server: esiste una separazione degli aspetti applicativi fra client e server, i quali svolgono differenti compiti come interfaccia utente e immagazzinamento dati.
- stateless: ogni richiesta verso il server deve contenere tutte le informazioni necessarie alla propria comprensione. Non si può quindi fare affidamento su nessuno stato (relativo a un particolare client) memorizzato sul server.
- cacheable: le risposte HTTP devono dichiarare la propria “cache”-ability per prevenire il mantenimento di “stale data” o l'effettuazione di richieste non necessarie.

- uniform interface: i comandi usati dal client per manipolare le varie risorse devono essere consistenti e uniformi per tutte le risorse coinvolte.
- layered system: un client non può distinguere fra un server o un intermediario come un proxy, essendo le sue richieste gestite trasparentemente.
- code on demand: i server possono trasferire parte della logica temporaneamente sul client, fornendogli script eseguibili.

Un'architettura si può quindi definire RESTful quando rispetta questi principi. Non tutti i servizi Web pubblici che si dichiarano REST sono in realtà RESTful, ma REST-like in quanto rispettano solo in parte i principi. I principali tratti dei servizi Web moderni rispecchianti l'architettura REST sono la rappresentazione di dati sottoforma di risorse, senza incapsulamento a la SOAP, e l'utilizzo dei verbi HTTP come GET e POST in modo semanticamente corretto.

3.4 SMILA

Il software open source integrato all'interno di Chansonnier è costituito da un framework per le applicazioni di ricerca, SMILA, e diversi annotatori utilizzati sugli artefatti. Gli annotatori utilizzano wrapper di binari locali o di servizi Web per produrre nuovi metadati sulle canzoni indicizzate.

Chansonnier è disponibile online come software open source, e costituisce un esempio pratico dell'utilizzo di SMILA.

SMILA è un framework per l'indicizzazione e la ricerca di informazioni provenienti da differenti sorgenti dati, principalmente costituite da informazione non strutturata come documenti, e-mail, e risorse accessibili tramite Web.

Lo scopo di SMILA è fornire un'infrastruttura standard per il processo di discovery e indicizzazione, così da lasciare gli sviluppatori liberi di produrre soluzioni specifiche al di sopra di esso, che siano interoperabili fra di loro. Ad esempio, un'applicazione sviluppata sopra SMILA può specificare particolari crawler che forniscano nuovi dati, o un sistema particolare per immagazzinarli; o ancora un indice per la ricerca personalizzato, o un sistema di ricerca più avanzato o orientato a un paradigma content-based. Ogni componente già presente nella distribuzione di SMILA può essere utilizzato o modificato per fare fronte ad esigenze comuni, come l'indicizzazione di un disco, o più personalizzate.

La presenza di un framework open source porta diversi vantaggi rispetto a soluzioni proprietarie, come ad esempio il riutilizzo del codice in diversi domini, la definizione di uno standard per i vari componenti di un'applicazione orientate alla ricerca, e l'abbassamento del rischio dovuto alla possibilità della sparizione dei vendor (causa fusioni, fallimenti, fine del supporto ufficiale per end of life) o di un lock-in con un particolare produttore.

SMILA è un'applicazione OSGi, costituita da alcune decine di bundle che sono eseguiti da un framework OSGi. Il framework distribuito di default è Equinox in quanto SMILA è un progetto intrapreso dalla Eclipse Foundation e si integra in modo preferenziale con le tecnologie della piattaforma Eclipse.

3.4.1 Modello dei dati

SMILA produce una struttura dati ad uso interno il cui elemento base è il Record. Ogni Record è la rappresentazione di un elemento di informazione, come un file di testo, un video o una pagina Web.

Un Record contiene perlomeno un Id, che lo identifica univocamente utilizzando **insieme** il nome della sorgente dati e la chiave dell'elemento in tale sorgente.

Il contenuto di un record è costituito da attributes (attributi) e attachments (allegati), e da annotations (annotazioni) riferite ad essi o al record stesso.

La differenza fra attributi e attachments sta nel tipo di dato elementare: gli attributes si riferiscono a metadati testuali e perlopiù di piccola portata, mentre gli attachment a dati binari. Un esempio di attributo è il MIMEType generato per un file; un esempio di attachment è il contenuto di un'immagine.

Le annotazioni possono essere presenti su di un record, o su di un suo attributo o attachment, in una relazione multipla. Mentre attributi e attachments sono generati durante il processo di discovery dei dati, le annotazioni sono prodotte da servizi precedentemente integrati nel workflow di SMILA, solitamente allo scopo di essere indicizzate ma anche per il loro utilizzo da altri annotatori, a più alta astrazione.

Ogni valore per attributi, attachment o annotazioni è costituito da un Literal, un wrapper per gli oggetti Java elementari come String, Boolean e simili. Come si vedrà nella sezione dedicata allo storage, la struttura dati di un Record può essere navigata in modo analogo a un documento XML conforme a un determinato schema¹.

Nel caso di studio presentato, ogni Record è la rappresentazione di una canzone. Un suo attributo è ad esempio il titolo della canzone, mentre una tipica annotazione è il suo testo ricavato da una ricerca sul Web. L'attachment principale è il contenuto binario del file multimediale originale (un video), ma non si esclude la presenza di altri attachment che contengano la conversione in un altro formato riconosciuto dagli annotatori.

Blackboard

Sebbene sia presente un modello ad oggetti che descrive il Record, è raro utilizzare un oggetto Record nella sua interezza. La maggior parte delle interazioni avviene attraverso una Blackboard che astrae dai particolare servizi di storage utilizzati.

Una Blackboard necessita di due particolari implementazioni di XmlStorage (utilizzato per la persistenza della struttura Xml di un record) e BinStorage (utilizzato per la persistenza dei suoi attachment).

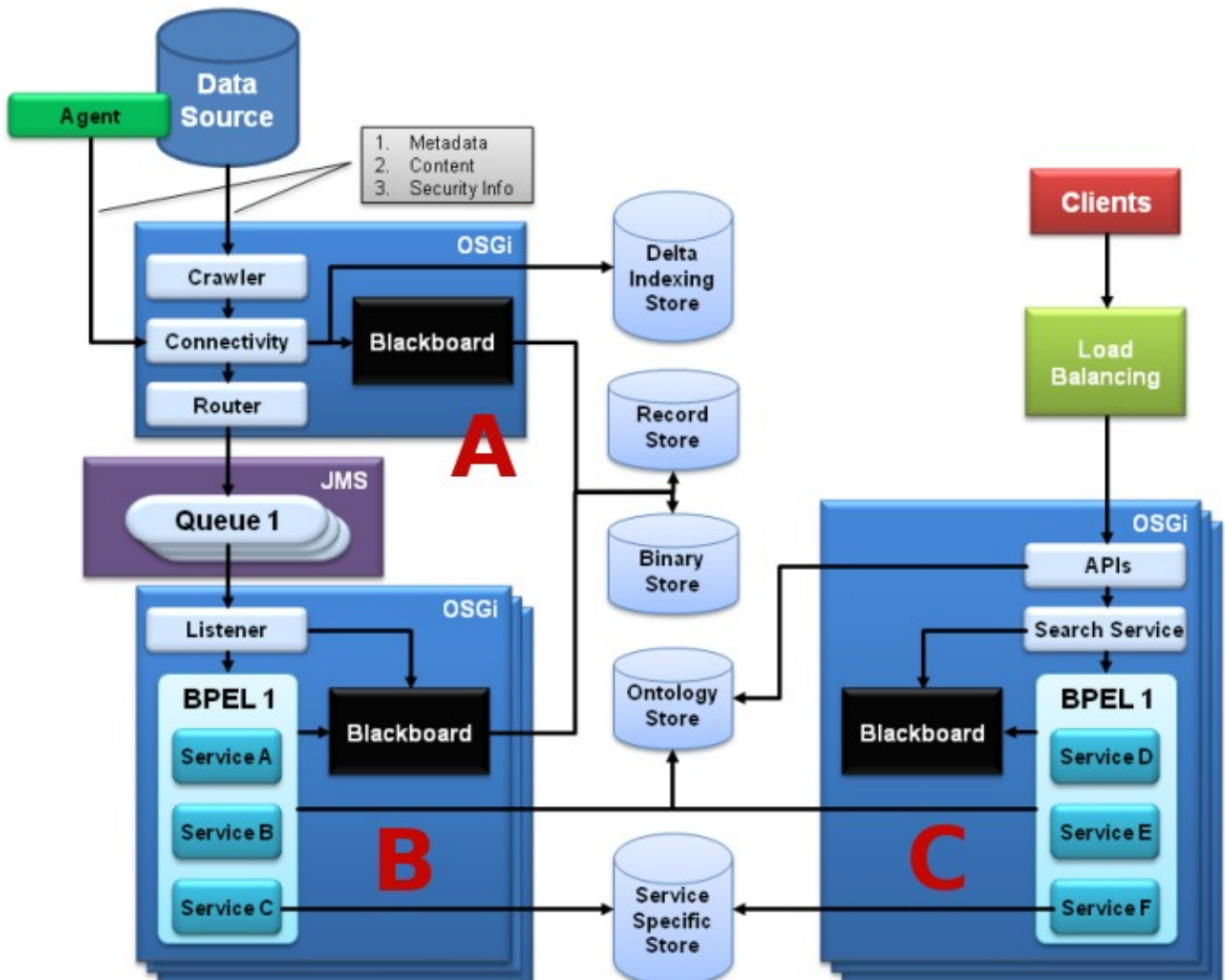
La Blackboard è un oggetto Java la cui Api permette di manipolare attributi, attachment e

¹ <http://dev.eclipse.org/svnroot/rt/org.eclipse.smila/tags/0.7/org.eclipse.smila.datamodel/xml/>

relative annotazioni utilizzando come parametro per le operazioni un Id. Questo oggetto è solo una Facade: più Blackboard possono essere presenti nello stesso momento, anche nel caso facciano riferimento allo stesso backend per adempiere alle proprie funzioni.

3.4.2 Architettura

SMILA è un framework costituito da bundles OSGi, e ne rispetta l'architettura modulare.



Possiamo suddividere i bundles forniti da SMILA in tre componenti principali:

- **Connectivity (A):** la sua responsabilità è fornire nuove informazioni sui record al componente Processing, siano esse l'aggiunta di un record ex-novo, l'aggiornamento di un record o la sua eliminazione dagli indici per via della sua eliminazione nella sorgente dati.
- **Processing (B):** organizza gli annotatori per produrre nuove annotazioni sui record
- **Search (C):** soddisfa le query da parte di Client utilizzando i metadati prodotti precedentemente.

Questi componenti fanno uso dell'infrastruttura configurabile di SMILA, che comprende la Blackboard e diversi sistemi di Storage, e di una o più Queue JMS che collegano Connectivity a Processing.

Connectivity

Il componente Connectivity utilizza due tipi di servizi per produrre nuovi record da indicizzare: crawler e agents.

Un Crawler, come si intenderebbe dal senso del termine nella ricerca Web, produce tutti i record estraibili da una determinata sorgente dati, seguendo tutte le connessioni possibili dai record inizialmente creati. In successive esecuzioni sulla medesima sorgente dati determina i record modificati che necessitano di essere riprocessati, e segnala la cancellazione di quelli non più esistenti al modulo Processing a valle.

Un Agent segue un modello diverso: rimane in esecuzione perennemente e monitora la sorgente dati per possibili cambiamenti. Successivamente notifica il componente Connectivity nel caso vengano introdotti nuovi Record.

Il componente Connectivity quindi, ottenuti nuovi Record, li salva negli Storage disponibili tramite la Blackboard, se essi non sono già presenti. Il Router si occupa quindi di spedire i Record ad una queue JMS che lo isola dal componente Processing, dopo averli eventualmente filtrati per spedire solo gli attributi necessari ed avendo rimosso gli attachment, la parte più pesante per la trasmissione.

Il caso di studio prevede la presenza di un'implementazione personalizzata di Agent, in grado di recuperare un video da Youtube seguendo il link della pagina in cui è contenuto. L'Agent rimarrà sempre in esecuzione, ispezionando una coda di link da esaminare che può essere rifornita di elementi da interfacce utente. Nel caso si dovesse introdurre anche un Crawler, nulla cambia al di fuori del componente Connectivity.

JMS Queue

Una o più Queue realizzate tramite JMS (di default ActiveMQ) ricevono messaggi dal Router (componente Connectivity) e li forniscono ad un Listener (componente Processing), fornendo un totale disaccoppiamento fra i due. I messaggi contengono i record da indicizzare o aggiornare, e per ognuno di essi un sottoinsieme di attributi e annotazioni configurabile (idealmente il più piccolo possibile).

Processing

Nel componente Processing, il Listener è il corrispettivo del Router nel componente Connectivity: esso riceve i messaggi contenenti Record dalla queue JMS e avvia diverse Pipeline in base ai loro metadati.

BPEL Pipelines

Le Pipeline sono dei business process eseguibili descritti tramite il linguaggio BPEL (in particolare WS-BPEL 2). Ogni Pipeline orchestra diversi servizi, locali o eventualmente remoti, che analizzano un Record per produrre nuove annotazioni in base ai suoi attributi, attachment o

annotazioni già esistenti.

Il potere espressivo di BPEL è pienamente utilizzabile, poiché gli annotatori sono orchestrati tramite tag `<extensionActivity>` che contengono a loro volta dei tag specifici per SMILA. Al di fuori di tali tag, si possono definire strutture tipiche di BPEL, come:

- sequenze di chiamate bloccanti o asincrone a Web Service: tramite il tag standard `<invoke>`.
- selezione di alternative: tramite il tag standard `<if>`. La condizione di selezione può essere specificata su gli attributi e annotazioni del singolo record estrapolate tramite espressioni XPath.
- parallelizzazione di attività: tramite il tag standard `<flow>` o il costrutto `<forEach>` nella sua versione parallela.

Si ricorda che solo i metadati che arrivano direttamente dai messaggi JMS sono accessibili all'interno di costrutti BPEL e da Web Service da essi richiamati. Questo esclude ad esempio gli attachment, e tutti gli attributi e annotazioni filtrati dalla Blackboard. Come si vedrà nel codice di esempio, la Blackboard utilizza un particolare filtro che specifica quali attributi e annotazioni inserire negli oggetti del workflow, e trasmette tutto il resto in quanto non necessario all'esecuzione del processo BPEL e recuperabile programmaticamente.

Gli annotatori utilizzati possono essere quindi Web Service generici, o particolari oggetti Java installati in un qualsiasi bundle OSGi. Questi oggetti sono chiamati Pipelet e ProcessingService e possono eventualmente essere wrapper per servizi remoti.

La Pipeline principale del caso di studio è la Pipeline di aggiunta, che riceve i nuovi Record generati dall'Agent e organizza il lavoro di Pipelet e ProcessingService per aggiornare uno o più indici di Lucene relativi a canzoni e artisti.

Segue l'esempio di una Pipeline elementare proveniente dalla test suite di SMILA e da me commentata.

```
<?xml version="1.0" encoding="utf-8" ?>
  <!-- i tag nel namespace <proc:> sono l'estensione comunemente utilizzata
        all'interno degli <extensionActivity>. -->
<process name="SimplePipeline"
targetNamespace="http://www.eclipse.org/smila/processor"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:proc="http://www.eclipse.org/smila/processor"
  xmlns:rec="http://www.eclipse.org/smila/record">

  <import location="processor.wsdl"
namespace="http://www.eclipse.org/smila/processor"
  importType="http://schemas.xmlsoap.org/wsdl/" />

  <partnerLinks>
    <partnerLink name="Pipeline"
partnerLinkType="proc:ProcessorPartnerLinkType" myRole="service" />
  </partnerLinks>

  <extensions>
```

```

        <extension namespace="http://www.eclipse.org/smila/processor"
mustUnderstand="no" />
    </extensions>

    <!-- l'input e l'output di una Pipeline sono un ProcessorMessage,
    essenzialmente un insieme di Id e di alcuni attributi e annotazioni.
-->
    <variables>
        <variable name="request" messageType="proc:ProcessorMessage" />
        <variable name="result" messageType="proc:ProcessorMessage" />
    </variables>

    <sequence>
        <receive name="start" partnerLink="Pipeline"
portType="proc:ProcessorPortType" operation="process" variable="request"
        createInstance="yes" />

        <!-- questa Pipeline definisce una sola invocazione -->
        <extensionActivity name="invokeSimpleTestPipelet">
            <!-- si tratta di una Pipelet, la cui classe è richiamata
            con il suo Fully Qualified Name -->
            <proc:invokePipelet>
                <proc:pipelet
class="org.eclipse.smila.processing.bpel.pipelet.SimpleTestPipelet" />
                <proc:variables input="request" output="result" />
                <!-- un oggetto Pipelet verrà istanziato solo per questa
Pipeline, quindi è possibile aggiungere una configurazione particolare. -->
                <proc:PipeletConfiguration>
                    <proc:Property name="FirstIsString">
                        <proc:Value>FirstValue</proc:Value>
                    </proc:Property>
                    <proc:Property name="SecondIsDate"
type="java.util.Date">
                        <proc:Value>2008-06-11 16:08:00</proc:Value>
                    </proc:Property>
                </proc:PipeletConfiguration>
                <!-- così come è possibile aggiungere annotazioni sul Record
per guidare la Pipelet nelle sue operazioni -->
                <proc:setAnnotations>
                    <rec:An n="test-annotation-1">
                        <rec:V>anonvalue1</rec:V>
                        <rec:V>anonvalue2</rec:V>
                        <rec:V n="name1">value1</rec:V>
                        <rec:V n="name2">value2</rec:V>
                    </rec:An>
                    <rec:An n="test-annotation-2">
                        <rec:V>anonvalue3</rec:V>
                        <rec:V>anonvalue4</rec:V>
                        <rec:V n="name3">value3</rec:V>
                        <rec:V n="name4">value4</rec:V>
                    </rec:An>
                </proc:setAnnotations>
            </proc:invokePipelet>
        </extensionActivity>

        <reply name="end" partnerLink="Pipeline"
portType="proc:ProcessorPortType" operation="process" variable="result" />
        <exit />
    </sequence>
</process>

```

Un esempio più esteso dell'orchestrazione tramite BPEL è la seguente Pipeline condizionale, sempre estratta dalla test suite:

```
<?xml version="1.0" encoding="utf-8" ?>
<process name="ConditionalPipeline"
targetNamespace="http://www.eclipse.org/smila/processor"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:proc="http://www.eclipse.org/smila/processor"
  xmlns:rec="http://www.eclipse.org/smila/record">

  <import location="processor.wsdl"
namespace="http://www.eclipse.org/smila/processor"
  importType="http://schemas.xmlsoap.org/wsdl/" />

  <partnerLinks>
    <partnerLink name="Pipeline"
partnerLinkType="proc:ProcessorPartnerLinkType" myRole="service" />
  </partnerLinks>

  <extensions>
    <extension namespace="http://www.eclipse.org/smila/processor"
mustUnderstand="no" />
  </extensions>

  <variables>
    <variable name="request" messageType="proc:ProcessorMessage" />
  </variables>

  <sequence>
    <receive name="start" partnerLink="Pipeline"
portType="proc:ProcessorPortType" operation="process" variable="request"
      createInstance="yes" />

    <if name="conditionSplitOnAttribute">
      <!-- tipicamente la request conterrà un solo Record.
Record multipli sono prodotti ad esempio da annotatori
che esaminano archivi zip|rar|tgz.
L'estensione activity verrà eseguita se l'attributo
'workflow-attribute' presente sul record contiene il valore
"split". Le condizioni sono espresse come espressioni XPath
e gli attributi e annotazioni utilizzati devono essere
espressamente resi disponibili al workflow BPEL
tramite configurazione (di org.eclipse.smila.blackboard).
-->
      <condition>$request.records/rec:Record[1]/rec:A[@n="workflow-
attribute"]/rec:L/rec:V = "split"</condition>
      <extensionActivity name="invokeSplitterService">
        <proc:invokeService>
          <proc:service name="SplitterService" />
          <proc:variables input="request"
output="request" />
        </proc:invokeService>
      </extensionActivity>
    </if>

    <if name="conditionSplitOnAnnotation">
      <condition>$request.records/rec:Record[1]/rec:An[@n="workflow-
annotation"]/rec:V = "split"</condition>
```

```

        <extensionActivity name="invokeSplitterPipelet">
            <!-- identico al precedente ma invocando una Pipelet,
                 e utilizzando un'annotazione anziché un attributo -->
            <proc:invokePipelet>
                <proc:pipelet
class="org.eclipse.smila.processing.bpel.pipelet.SplitterPipelet" />
                <proc:variables input="request"
output="request" />
            </proc:invokePipelet>
        </extensionActivity>
    </if>

    <reply name="end" partnerLink="Pipeline"
portType="proc:ProcessorPortType" operation="process" variable="request" />
    <exit />
</sequence>
</process>

```

Perché queste Pipeline possano funzionare, la Blackboard deve essere configurata per copiare gli attributi e annotazioni utilizzati all'interno del workflow. Ad esempio per la seconda:

```

<!-- configuration/org.eclipse.smila.blackboard/RecordFilters.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<RecordFilters
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=" ../org.eclipse.smila.datamodel.tools/schemas/RecordFilters.xsd"
>
  <Filter name="workflow-object">
    <Attribute name="workflow-attribute" keepAnnotations="false"/>
    <Annotation name="workflow-annotation" />
  </Filter>
</RecordFilters>

```

Webservice

Webservice esterni sono accessibili tramite il tag standard `<invoke>`. Gli attributi e le annotazioni sono disponibile per essere passati come parametri nelle richieste previa opportuna configurazione, mentre gli attachment non sono accessibili. Questa è una scelta architetturale di SMILA: l'accesso agli attachment non è quindi trasparente; si veda la sezione *Distribuzione > Limiti di utilizzo*.

Pipelet e ProcessingService

Pipelet e ProcessingService sono i principali mattoni utilizzati per costruire Pipeline più o meno complicate. Dal punto di vista dell'interfaccia i due tipi di oggetti sono simili: espongono un metodo `process()` che accetta un insieme di Id di record e la Blackboard. Questo metodo sarà chiamato in corrispondenza di un tag `<extensionActivity>` opportunamente configurato. Si noti che il metodo è definito come rientrante e deve essere quindi implementato in modo thread-safe: esso può essere chiamato da differenti processi agenti sui Record o addirittura da diverse Pipeline, nel caso di un ProcessingService. In generale, una Pipeline è creata insieme alle Pipelet che richiede (le cui istanze vengono quindi configurate specificatamente tramite BPEL per tale Pipeline) e tramite i riferimenti ai ProcessingService che verranno portati in esecuzione dal framework OSGi.

Il possesso di un riferimento a un oggetto Blackboard da parte di Pipelet e ProcessingService permette loro l'accesso ad ogni tipo di dato e metadato, sia esso attributo, annotazione o attachment.

Nonostante le similarità di interfaccia, Pipelet e ProcessingService presentano alcune importanti differenze:

- ciclo di vita: le Pipelet non sono condivise fra differenti Pipeline, mentre i ProcessingService sono a tutti gli effetti OSGi Declarative Services, di cui una sola istanza è mantenuta attiva dal framework OSGi. I ProcessingService sono quindi la scelta migliore per implementare servizi a grande footprint, la cui creazione comporti l'allocazione di risorse non indifferenti. Le Pipelet sono uno strumento più leggero, adatte per l'inclusione di puri algoritmi o di attività che differiscono solo per configurazione.
- configurazione: le Pipelet ricevono la propria configurazione tramite XML incapsulato nella definizione della Pipeline. I ProcessingService non hanno limitazioni riguardo alla provenienza della propria configurazione, che del resto è quindi una maggiore responsabilità da assolvere.
- accesso a servizi OSGi: è possibile solo da parte dei ProcessingService, tramite il ComponentContext iniettato durante l'attivazione o metodi di binding propri dei Declarative Service. Una Pipelet non ha di norma riferimenti esterni, a meno che non sia una Facade per un particolare modulo.

Nonostante queste differenze costruttive, quando si fa riferimento a Pipelet o ProcessingService in termini di interfaccia i due tipi di componenti sono intercambiabili. Ciò che viene asserito per uno in questo documento riguardo all'utilizzo all'interno di Pipeline è valido automaticamente anche per l'altro.

SimplePipelet: `Id[] process(Blackboard blackboard, Id[] recordIds)` ²

ProcessingService: `Id[] process(Blackboard blackboard, Id[] recordIds)` ³

Seguono esempi di Pipelet e ProcessingService realizzati come exploratory testing di SMILA, con relativa configurazione.

DummyPipelet: concatena a un attributo del Record un valore predefinito.

```
package it.polimi.chansonnier;

import org.eclipse.smila.blackboard.Blackboard;
import org.eclipse.smila.blackboard.path.Path;
import org.eclipse.smila.datamodel.id.Id;
import org.eclipse.smila.datamodel.record.Literal;
import org.eclipse.smila.datamodel.record.RecordFactory;
import org.eclipse.smila.processing.ProcessingException;
import org.eclipse.smila.processing.SimplePipelet;
import org.eclipse.smila.processing.configuration.PipeletConfiguration;
```

² <http://build.eclipse.org/rt/smila/javadoc/current/org/eclipse/smila/processing/SimplePipelet.html>

³ <http://build.eclipse.org/rt/smila/javadoc/current/org/eclipse/smila/processing/ProcessingService.html>


```

public class DummyPipelet implements SimplePipelet {
    /**
     * Nome della chiave di configurazione.
     */
    private static final String ATTRIBUTE_NAME = "ATTRIBUTE_NAME";

    /**
     * Nome dell'attributo dei Record da modificare.
     */
    private String _attributeName;

    @Override
    public Id[] process(Blackboard blackboard, Id[] recordIds)
        throws ProcessingException {
        for (Id id : recordIds) {
            try {
                // cerchiamo semplicemente un attributo sul Record stesso
                // avremmo potuto cercarlo all'interno di un altro attributo o
                // cercare un'annotazione su questo attributo, etc.
                final Path path = new Path(_attributeName);
                // tutta l'interazione con i dati avviene tramite la Blackboard
                if (blackboard.hasAttribute(id, path)) {
                    // costruiamo un nuovo valore (oggetto Literal che fa da
                    // wrapper a diversi valori base come stringhe, interi...)
                    String currentValue = blackboard.getLiteral(id,
path).getStringValue();
                    String newValue = currentValue + " ...edited by
DummyPipelet";
                    final Literal literal =
RecordFactory.DEFAULT_INSTANCE.createLiteral();
                    literal.setStringValue(newValue);
                    // la blackboard riceve il nuovo valore per questo Record
                    blackboard.setLiteral(id, path, literal);
                }
            } catch (final Exception e) {
                throw new ProcessingException("During execution of
DummyPipelet: " + e.toString());
            }
        }
        // gli Id dei Record ritornati verranno passati alla Pipelet successiva
        return recordIds;
    }

    /**
     * Riceve la configurazione alla creazione della Pipeline.
     */
    @Override
    public void configure(PipeletConfiguration configuration)
        throws ProcessingException {
        _attributeName = ((String)
configuration.getPropertyFirstValueNotNull(ATTRIBUTE_NAME)).trim();
        if (_attributeName.length() == 0) {
            throw new ProcessingException("Property " + ATTRIBUTE_NAME + "
is empty.");
        }
    }
}

```

DummyProcessingService: equivalente di DummyPipelet realizzato come servizio OSGi.

```

package it.polimi.chansonnier;

import org.eclipse.smila.blackboard.Blackboard;
import org.eclipse.smila.blackboard.path.Path;
import org.eclipse.smila.datamodel.id.Id;
import org.eclipse.smila.datamodel.record.Literal;
import org.eclipse.smila.datamodel.record.RecordFactory;
import org.eclipse.smila.processing.ProcessingException;
import org.eclipse.smila.processing.ProcessingService;
import org.osgi.service.component.ComponentContext;

public class DummyProcessingService implements ProcessingService {
    private String _attributeName = "Filename";

    protected void activate(ComponentContext context) {
        // chiamato dal framework OSGi all'attivazione
        // usato ad esempio per leggere la propria configurazione
    }

    @Override
    public Id[] process(Blackboard blackboard, Id[] recordIds) throws
ProcessingException {
        // la logica è identica a quella di DummyPipelet
        for (Id id : recordIds) {
            try {
                final Path path = new Path(_attributeName);
                if (blackboard.hasAttribute(id, path)) {
                    String currentValue = blackboard.getLiteral(id,
path).getStringValue();
                    String newValue = currentValue + " ...edited by
DummyProcessingService";
                    final Literal literal =
RecordFactory.DEFAULT_INSTANCE.createLiteral();
                    literal.setStringValue(newValue);
                    blackboard.setLiteral(id, path, literal);
                }
            } catch (final Exception e) {
                throw new ProcessingException("During execution of
DummyProcessingService: " + e.toString());
            }
        }
        return recordIds;
    }
}

```

Frammenti di Pipeline per l'invocazione:

```

<extensionActivity name="invokeDummyProcessingService">
    <proc:invokeService>
        <proc:service name="DummyProcessingService" />
        <proc:variables input="request" output="request" />
    </proc:invokeService>
</extensionActivity>

<extensionActivity name="invokeDummyPipelet">
    <proc:invokePipelet>
        <proc:pipelet class="it.polimi.chansonnier.DummyPipelet" />
        <proc:variables input="request" output="request" />
        <proc:PipeletConfiguration>
            <proc:Property name="ATTRIBUTE_NAME">

```

```

        <proc:Value>Filename</proc:Value>
      </proc:Property>
    </proc:PipeletConfiguration>
  </proc:invokePipelet>
</extensionActivity>

```

La configurazione globale di questi due componenti avviene tramite metadati OSGi:

```

// MANIFEST.MF:
Import-Package: junit.framework;version="[3.8.0,4.0.0)",
org.eclipse.smila.blackboard;version="0.7.0",
...
Export-Package: it.polimi.chansonnier;version="1.0.0.alpha"
SMILA-Pipelets: it.polimi.chansonnier.DummyPipelet
Service-Component: OSGI-INF/component.xml
// OSGI-INF/component.xml
<component name="dummyService">
  <implementation class="it.polimi.chansonnier.DummyProcessingService" />
  <service>
    <provide interface="org.eclipse.smila.processing.ProcessingService"/>
  </service>
  <property name="smila.processing.service.name"
value="DummyProcessingService"/>
</component>

```

Scelta architetturale

La configurazione di default di SMILA, ed anche gli use case principali, prevedono l'utilizzo di Pipelet e ProcessingService per la costruzione di Pipeline in BPEL, relegando l'utilizzo diretto di Web service in secondo piano. Vi sono due ragioni principali per questa scelta di design.

Per prima cosa, la restrizione dei metadati relativi ai Record da copiare negli oggetti del workflow solamente al sottoinsieme necessario per eseguire la logica BPEL. Parametri da passare a servizi Web, anche ad uno solo di essi, devono essere copiati in un oggetto che viaggerà attraverso tutto il percorso dei componenti Connectivity e Processing. Pipelet e ProcessingService possono invece recuperare solamente i dati a loro necessari tramite l'oggetto Blackboard.

Alternativamente, si può pensare a Pipelet e ProcessingService come a un livello di astrazione al di sopra dei Web service generici, pensati per poter adempiere al maggior numero di casi d'uso possibili. L'isolamento di essi tramite oggetti Java permette vantaggi non indifferenti, quali il cambiamento del back-end permesso dalla definizione di una propria interfaccia sopra di esso (la Pipelet/ProcessingService in questione), o l'iniezione di implementazioni alternative a scopo di test che ritornino risultati preprogrammati e che supportino l'esecuzione di unit test veloci (non legati a servizi remoti) e pertanto presenti in grande numero. In particolare i ProcessingService rimangono in esecuzione come servizi OSGi, e dunque danno la possibilità di introdurre un livello di cache fra le Pipeline e i servizi Web utilizzati dai ProcessingService.

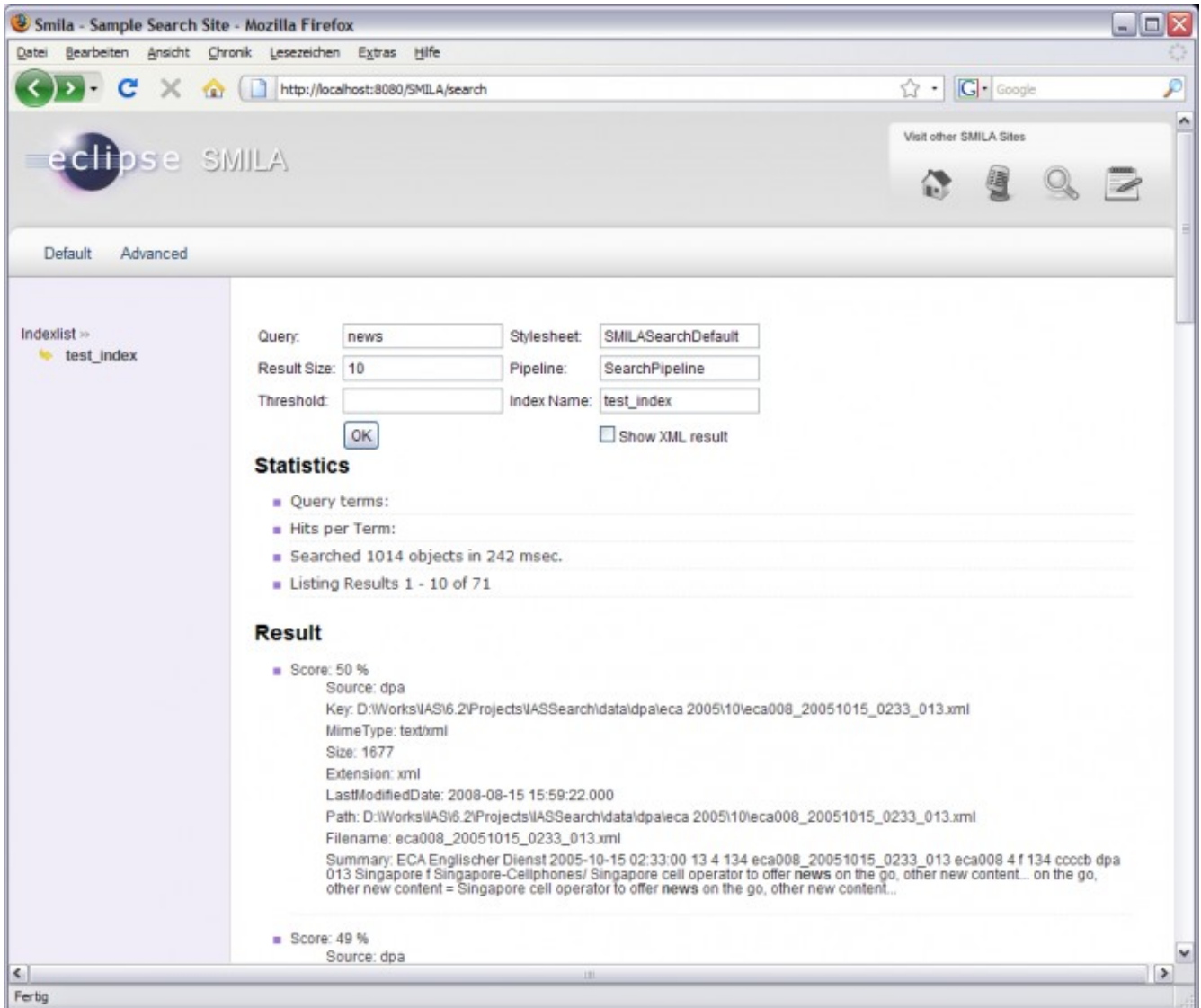
Gli annotatori relativi al caso di studio, siano essi wrapper di un'applicazione locale o di un servizio Web, verranno realizzati come Pipelet e ProcessingService.

Search

Così come SMILA organizza l'indicizzazione di insiemi di Record, prescrive anche l'utilizzo di un'infrastruttura di ricerca basata su Pipeline BPEL analoghe a quelle utilizzate per la ricerca, e sullo stesso modello dei dati costituito da Record e Id.

Interfaccia Web

L'interfaccia di default presentata da SMILA è di tipo Web, ed è generata tramite Servlet:



Una query utente genera un risultato costituito da una lista di Record, i quali vengono mostrati con un set di attributi e annotazioni. Questa interfaccia orientata all'utilizzo umano si appoggia su una più generale Api di ricerca. L'Api è disponibile attraverso una Facade costituita dal servizio OSGi SearchService.

I metodi di SearchService accettano in ingresso il nome di una Pipeline e un Record che incapsula la query richiesta dal client. Il servizio quindi esegue la Pipeline richiesta (differente da quelle utilizzate per l'indicizzazione) per ottenere un set di risultati, restituibile come oggetto Java o

in formato XML.

Search Pipelines

Le Pipeline di ricerca coordinano un insieme di oggetti per la produzione di un risultato. La struttura è analoga a quella già vista per le Pipeline di indicizzazione:

- SearchPipelet è l'interfaccia analoga a SimplePipelet. Il ciclo di vita di tali Pipelet è come sempre gestito da SMILA e ne limita l'uso ad oggetti di piccola portata.
- SearchProcessingService è l'interfaccia analoga a ProcessingService. Classi che implementano sono avviati come Declarative Service OSGi.
- SearchMessage è una classe wrapper per un insieme di Id, equivalent all'array Id[] utilizzato nelle signature di SimplePipelet e ProcessingService. Ha però un campo aggiuntivo che identifica il Record della query, distinguendolo dall'insieme di Record del risultato.

Sia SearchPipelet sia SearchProcessingService vengono chiamati in causa nelle Pipeline, ed accettano come parametri un riferimento alla Blackboard e un SearchMessage. Similarmente a Pipelet e ProcessingService, essi sono praticamente intercambiabili dal punto di vista dell'interfaccia.

SearchPipelet: [SearchMessage process \(Blackboard blackboard, SearchMessage message\)](#) ⁴

SearchProcessingService: [SearchMessage process \(Blackboard blackboard, SearchMessage message\)](#) ⁵

L'utilizzo di SearchPipelet e SearchProcessingService è molteplice: possono intervenire sulla query per inferire annotazioni da aggiungere alla ricerca prima che venga effettuata, oppure dopo di essa per modificare e arricchire il risultato, incapsulando così la business logic necessaria all'interno dell'Api anziché in un particolare front-end. Ad esempio HighlightingService si occupa di mettere in evidenza parti degli attributi secondo un transformer configurabile (nel caso di HTML, tramite tag che marcano parti di testo in grassetto).

La ricerca stessa è effettuata tramite un SearchProcessingService, di default l'implementazione inclusa nella distribuzione di SMILA LuceneSearchService. Nulla impedisce persino di utilizzare diversi servizi di ricerca ed aggregare i risultati; la concorrenza di tali operazioni necessita però di sincronizzazione poiché diversi oggetti agiscono sulla stessa Collection di Id, una struttura dati di base del linguaggio Java.

Storage

Un punto di grande interesse è l'implementazione dei vari Store utilizzati dai servizi di più alto livello, come ad esempio la Blackboard e i processi di ricerca.

⁴ <http://build.eclipse.org/rt/smila/javadoc/current/org/eclipse/smila/processing/SearchPipelet.html>

⁵ <http://build.eclipse.org/rt/smila/javadoc/current/org/eclipse/smila/processing/SearchProcessingService.html>

XmlStorage

XmlStorage è un collaboratore della Blackboard che si occupa di memorizzare attributi e annotazioni per ogni record. La sua collocazione non è critica in quanto l'Api della Blackboard permette di recuperare qualsiasi dato in esso memorizzato, ed allo stesso tempo si tratta di una quantità di informazione negligibile rispetto al ben più corposo BinStorage.

Il nome XmlStorage deriva dalla rappresentazione interna dei Record, utilizzabile nei workflow di indicizzazione e ricerca per filtrare Record tramite espressioni XPath. Un esempio di Record in forma XML è il seguente:

```
<Record>
  <id:ID>
    <id:Source>share</id:Source>
    <id:Key>some.html</id:Key>
  </id:ID>
  <A n="mimetype">
    <An n="filter">
      <V n="type">exclude</V>
      <An n="values">
        <V>text/plain</V>
        <V>text/html</V>
      </An>
    </An>
  </A>
  <L>
    <V>text/html</V>
  </L>
</A>
</Record>
```

Questo Record ha un Id a chiave multipla, costituita dal nome della sorgente dati *share* e dal file originante, *some.html*; ha un attributo *mimetype* di valore *text/html*. Tale attributo ha un'annotazione *filter* contenente un valore *exclude*. L'annotazione nidificata in essa contiene i valori *text/plain* e *text/html*, e potrà essere letta all'interno di workflow BPEL o dei servizi stessi per attivare l'esecuzione di business logic su questo Record.

BinStorage

Una implementazione di BinStorage è utilizzata dalla Blackboard per la gestione degli attachment. L'Api di questo storage è quindi differente rispetto agli altri, essendo orientata al byte e permettendo il recupero di attachment binary come array di byte o InputStream/OutputStream.

In particolare, questo tipo di accesso è possibile tramite BinaryStorageService o BlackboardFactory, entrambi servizi OSGi. La BlackboardFactory ha la possibilità di creare una Blackboard per ogni servizio che la richiede, ed è un'interfaccia a più alto livello del semplice BinaryStorageService (permette ad esempio di gestire automaticamente i diversi attachment per uno stesso record).

Le implementazioni disponibili di default per BinStorage sono basate su file system (IOHierarchicalManager e IOFlatManager), su EFS (Encrypting File System), o su JPA, permettendo quindi il mapping degli attachment verso un qualsiasi database relazionale. Quanto sia

consigliabile quest'ultima opzione dipende dalla mole di dati da gestire.

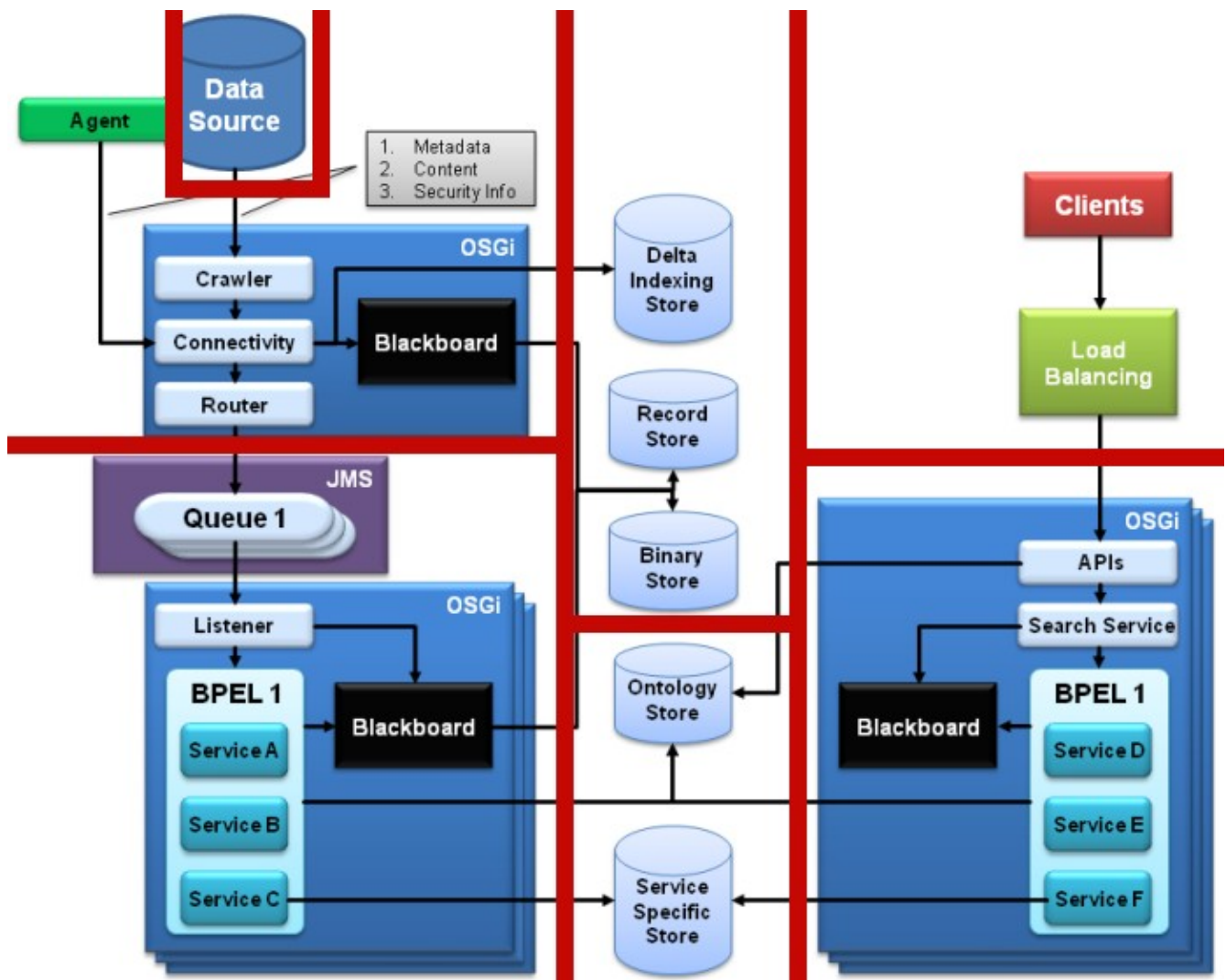
Service Specific Store (Lucene, Solr)

Allo scopo di ottimizzare la ricerca, un numero qualsiasi di Storage intermedi e di qualsiasi tecnologia può essere utilizzato per fare da ponte tra il componente Processing e il componente Search. Questi storage costituiscono l'output delle operazioni di Processing: tipicamente le pipeline di aggiunta, modifica o cancellazione di record specificano come ultima attività l'aggiornamento di questi Storage, che andranno poi a fare da input ai servizi del component Search.

Di default, Apache Lucene è utilizzato come ProcessingService per gestire un indice dei dati. Nulla impedisce di utilizzare un servizio Web, anche remoto, orchestrando il suo aggiornamento tramite BPEL nelle pipeline del componente Processing.

3.4.3 Distribuzione

I tre componenti principali di SMILA, Connectivity, Processing e Search, non hanno una stretta collocazione su di uno stesso nodo. Anche gli Storage non sono funzionalmente legati, purchè si fornisca un'interfaccia remota per accedervi: lo scopo della Blackboard è appunto di nascondere XmlStorage e BinStorage agli altri componenti.



Come si vede dalla segmentazione descritta in figura, ogni componente può girare su un nodo diverso. Ovviamente, Client e Data Source sono slegati in numero e collocazione da SMILA.

Connectivity

Il componente Connectivity è legato alle implementazioni dei suoi Crawler e Agent: ad esempio un Crawler orientato al Web necessariamente accede a Data Source remoti per funzionare, cioè i server HTTP che raggiunge ricorsivamente attraverso gli ipertesti incontrati. Altri crawler come quello orientato al file system invece necessitano di essere eseguiti sullo stesso nodo dei dati da loro accessibili, a meno di un'ulteriore astrazione che si ponga nel mezzo come Network File

System, SSHFS, o qualsiasi forma di Network Attached Storage o Storage Area Network.

Il caso di studio prevede l'utilizzo di un Agent personalizzato che estragga i video incorporati in un insieme di pagine di Youtube, ad esempio generabile tramite la ricerca Web. Per via di questioni legati alla presenza sul portale di un'alta quantità di materiale coperto da diritto d'autore, non esiste una Api pubblica per effettuare il download di tali dati; si userà quindi una forma di Web scraping per scaricare i video.

Il punto di connessione di Connectivity con Processing è un broker JMS. Come tale è naturale che sia possibile scegliere un broker remoto, raggiungibile su di una porta TCP. Non vi è un limite stretto al numero di Queue JMS che possono ricevere messaggi da un determinato componente Connectivity, né al numero di componenti Connectivity che possono puntare ad una determinata Queue.

Processing

Il componente Processing riceve messaggi contenente i Record da una Queue JMS e, in base alle specifiche di tale infrastruttura, ogni messaggio di una Queue può essere consegnato solo ad un Listener (questo differisce dall'utilizzo di Topic JMS, che permettono la sottoscrizione simultanea di più Listener). È quindi logico far sì che il JMS Broker giri sullo stesso nodo del componente Processing, in quanto è comunque strettamente legato ad esso.

Nel caso di studio, non è presente una particolare difficoltà di computazione nella fase di raccolta di informazioni dal crawler allo smistamento dei Record nelle pipeline, quindi Connectivity e Processing sono fatti girare nello stesso nodo e in particolare nella stessa Java Virtual Machine.

Pipeline

Le pipeline contengono singoli punti di estensione nella forma di chiamate a Web service o a Pipelet o ProcessingService. Mentre i Web service sono di norma remoti, Pipelet e ProcessingService sono contenuti in bundle OSGi locali: questa distinzione è importante poiché i Web service non hanno accesso agli attachment dei record.

Ciò non toglie che si possano costruire Pipelet e ProcessingService (entrambi Plain Old Java Objects che implementano una determinata interfaccia) che girino in locale ma utilizzino qualsivoglia metodo di comunicazione remota. Ad esempio, si possono generare varie classi stub a partire dal documento WSDL del servizio Web da utilizzare tramite Apache AXIS, e Pipelet o ProcessingService possono estrarre dalla Blackboard gli attachment necessari e chiamare i metodi remoti passando tali dati.

In effetti, l'integrazione programmatica di Web services, nella forma di Pipelet o ProcessingService che ne facciano da wrapper, è il punto di estensione più importante dell'architettura di SMILA, sia per quanto riguarda le Pipeline del componente Processing, sia per le Pipeline utilizzate dal componente Search. Questo tipo di integrazione di servizi remoti è definito come default e può utilizzare qualsiasi tipo di libreria disponibile in ambito Java, purché essa venga

inclusa nel bundle relativo.

Il caso di studio prevede l'utilizzo dell'Api di LyricWiki, un wiki pubblico nel quale sono presenti i testi delle canzoni da indicizzare. L'Api viene utilizzata per recuperare il testo di una canzone a partire dal titolo e dal nome dell'artista. Il testo sarà disponibile poi sia per l'indicizzazione nello Storage, sia nelle parti successive della Pipeline per successive elaborazioni.

Non vi è limite al tipo di interazioni che è possibile orchestrare verso servizi remoti, se non quello dato dall'impatto sulle performance. È quindi consigliabile ad esempio limitare l'invio o la ricezione di dati binari di grande dimensione.

Il caso di studio prevede anche l'utilizzo di alcuni servizi locali, sempre nella forma di Pipelet e ProcessingService. L'estrazione della traccia audio a partire dal video musicale e il riconoscimento della lingua e del genere sono servizi che non utilizzano risorse remote.

Search

La collocazione del componente Search è speculare a quella relativa al componente Processing: gli oggetti Java utilizzati dalle Pipeline sono per loro natura locali alla stessa JVM che fornisce il front-end, ma possono utilizzare qualsivoglia servizio remoto, ad esempio le Api REST di Solr che accedano agli indici di Lucene utilizzati dal componente Processing.

Storage

Qualsiasi sistema di Storage, anche attraverso la rete, può essere utilizzato per XmlStorage e BinStorage, purché si fornisca un'implementazione in Java delle necessarie interfacce e la quantità di dati trasmessi sia funzionalmente possibile. L'implementazione già presente per JPA promette di poter utilizzare un qualsiasi database relazionale (che però per dati binari potrebbe essere inadatto).

Per quanto riguarda i Service Specific Storage come Lucene, anche qui non vi è limitazione alle varie soluzioni integrabili, purché ne esista (o venga costruita) una Facade accessibile come Web service, ProcessingService o Pipelet.

Il caso di studio utilizza le implementazioni open source fornite dalla distribuzione standard di SMILA, fra cui l'adapter per Lucene per gli indici di ricerca e l'implementazione diretta su filesystem di BinStorage e XmlStorage.

Scalabilità

La caratteristica importante degli Storage è la loro caratteristica intrinseca di mantenere uno stato. Si tratta quindi di componenti essenzialmente difficili da duplicare per aumentare la scalabilità, ad esempio tramite sharding del database, o sistemi master-slave. In ogni caso, questa duplicazione è da gestire all'esterno di SMILA, i cui componenti devono vedere una Blackboard che idealmente dà accesso all'intero insieme di dati.

Gli altri componenti sono passibili di duplicazione per sostenere un traffico di dati elevato, non avendo uno stato intrinseco, e sono infatti pensati proprio per facilitare questo scopo. Alcuni

esempi pratici di duplicazione possono essere:

- Differenti componenti Connectivity mandano i propri risultati ad una medesima Queue JMS. Si tratta di un invio asincrono di messaggi. Il collo di bottiglia in questo caso è probabilmente il salvataggio in un BinStorage dei Record, ma tale operazione viene effettuata solo una volta per ogni Record finchè non si rende necessario accedervi tramite la Blackboard: l'overhead introdotto dal framework è minimo.
- Diversi coppie di componenti Connectivity e Processing girano ognuna su un nodo e sostengono richieste di indicizzazione da diversi Agent e Crawler. Tuttavia, ognuno è configurato per accedere ad uno stesso BinStorage su una macchina dedicata a fare da database. Inoltre, entrambi aggiornano lo stesso indice di Lucene come ultimo passo delle loro Pipeline.
- Similarmente è possibile il caso in cui diverse Pipeline di ricerca accedono allo stesso indice Lucene e/o BinStorage, ognuna da un diverso nodo. Si tratta di un accesso solitamente in lettura (nulla vieta tuttavia di riprocessare i record tramite Pipeline specifiche in seguito a una ricerca), ed è quindi possibile far accedere i componenti Search a particolari implementazioni della Blackboard che effettuino un caching aggressivo (ottimizzazione analoga all'uso di database slave).

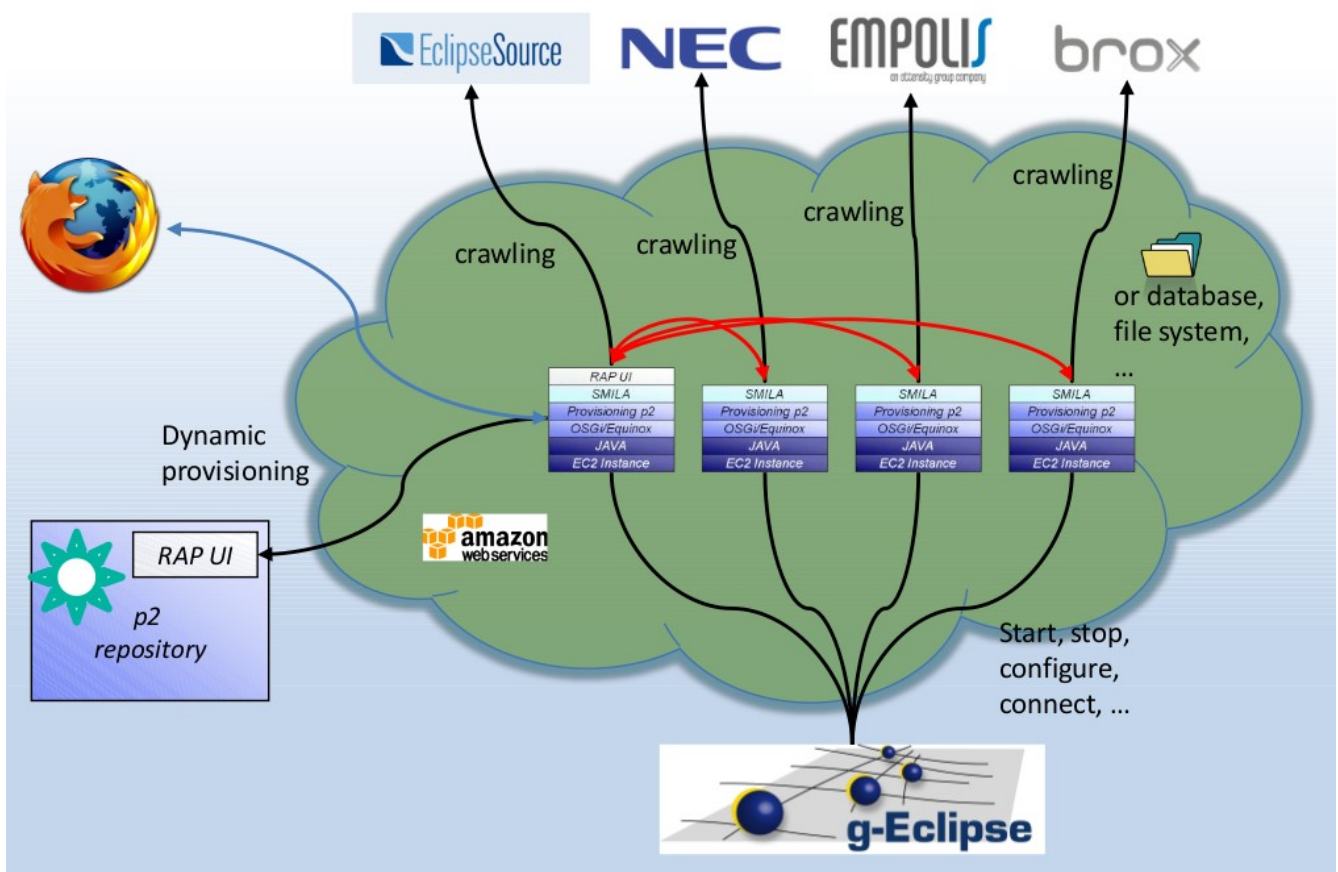
Altri mezzi di distribuzione

I bundle che costituiscono SMILA e i componenti da esso integrabili sono in se' sono difficili da distribuire in quanto la specifica OSGi non e' attrezzata per un tale scopo. I framework Eclipse ECF o Apache CXF permettono in teoria di distribuire i Bundle in modo trasparente, oppure di fornire un'interfaccia asincrona per le chiamate a metodi remoti. Tuttavia i componenti di SMILA sono più semplici da configurare per una distribuzione che segua le linee tracciate dall'architettura del sistema e precedentemente descritte. Ad esempio dividere diversi bundle che costituiscono uno stesso componente può essere problematico per la quantità di conversazioni che devono sostenere in modo remoto.

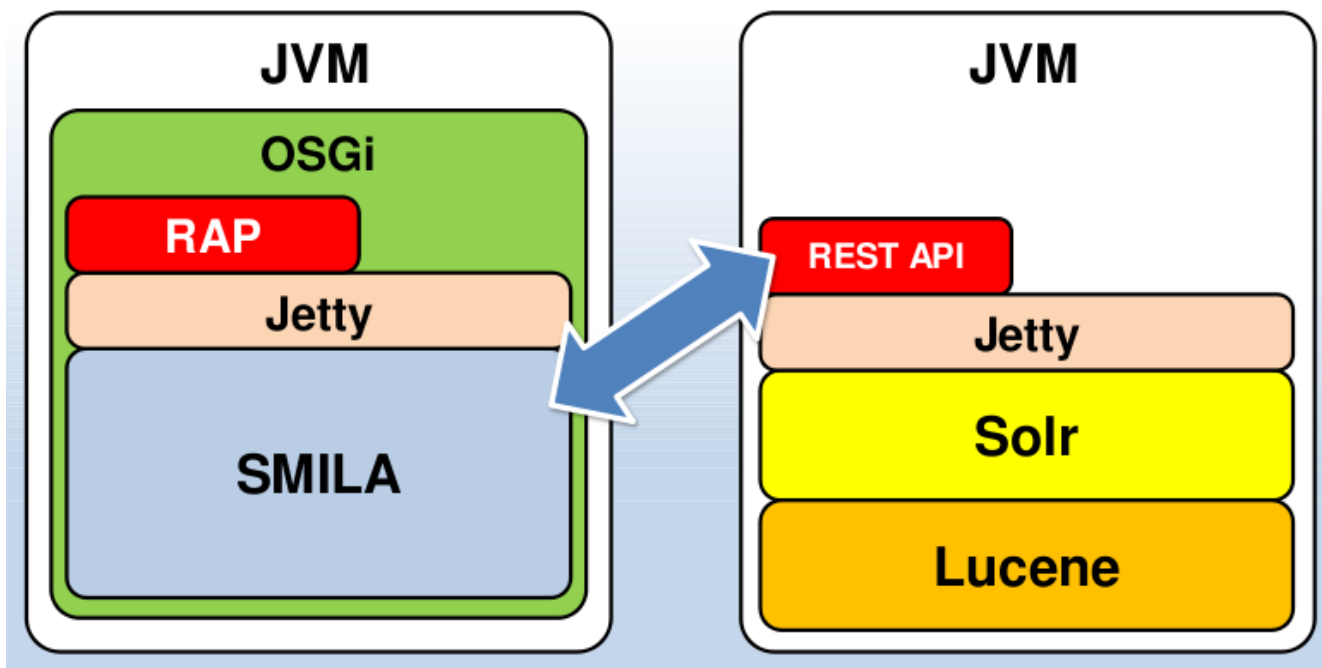
Cloud computing

È possibile utilizzare g-Eclipse, una piattaforma con diversi adapter per servizi di cloud computing, per far girare istanze multiple di SMILA su una cloud come Amazon EC2. L'utilizzo di tale piattaforma è stato dimostrato in una demo alla EclipseCon 2010.⁸

Nell'architettura citata, differenti JVM girano su diverse istanze di macchine di Amazon EC2. Ogni JVM ha un framework OSGi indipendente che esegue un sottoinsieme dei bundle di SMILA. Ad esempio, non è necessario eseguire i bundle relativi all'interfaccia Web nei nodi che fanno solamente crawling.



Il nodo utilizzato come front-end può quindi accedere ai servizi di storage degli altri nodi. In questo caso, tramite un'Api HTTP (REST) sopra agli indici Lucene dei nodi che effettuano il crawling. I servizi di Search saranno quindi eseguiti sul nodo front-end:



Questo è quindi un esempio di utilizzo che separa il componente Search da altre istanze di SMILA contenenti Connectivity (crawling di siti Web), Processing (indicizzazione) e sistemi di storage (Lucene). In particolare, i due tipi di nodi utilizzati sono replicabili indipendentemente: i primi per far fronte a un crescente numero di richieste utente, i secondi per indicizzare siti differenti.

Limiti di utilizzo

Un esempio di limite di utilizzo di SMILA è l'impossibilità di passaggio di attachment a Web service esterni via BPEL (mentre è possibile programmaticamente, all'interno di ProcessingService o Pipelet).

Sebbene questa possa sembrare una limitazione importante, nella visione di SMILA è intenzionale. Per prima cosa, gli sviluppatori ritengono che passare un attachment binario all'interno di un DOM non sia "ragionevole". Inoltre, lo scopo della Blackboard è di nascondere l'accesso ai Record, cosicché solamente gli Id e i metadati necessari al routing di essi all'interno delle varie pipeline debbano viaggiare in diversi nodi e componenti. L'astrazione che si costruirebbe fornendo trasparentemente attachment dentro alle Pipeline, così come si potrebbe configurare per gli attributi, sarebbe potenzialmente dannosa in quanto il loro peso è di molti ordini di grandezza superiore e andrebbero copiati nel workflow object ad ogni esecuzione di una pipeline. SMILA prescrive di gestire esplicitamente gli attachment tramite ProcessingService e Pipelet, il meccanismo di default per l'integrazione di annotatori.

Un'altra limitazione è l'utilizzo di JMS Queue, che spediscono messaggi a un solo componente Processing, invece di JMS Topic, che possono gestire più sottoscrizioni. Vi è comunque ampia possibilità di configurazione sia sul lato Router che sul lato Listener per gestire istanze multiple dei componenti Connectivity e Processing.

Infine, i bundle OSGi all'interno dello stesso componente sono progettati per essere eseguiti

nella stessa JVM: il modo più semplice per separare tali bundle è sostituirli con altre implementazioni che si occupino del remoting in modo trasparente. SMILA tuttavia, come è stato presentato in questo documento, presenta molte linee lungo le quali dividere e replicare i componenti per sfruttare un ambiente distribuito, e tali hack non si rendono di norma necessari.

4 Design

Chansonnier è un'applicazione Java costruita sulla piattaforma OSGi, ed in particolare al fianco di SMILA, framework per applicazioni di ricerca.

L'applicazione una volta compilata è quindi un insieme di bundle OSGi, alcuni dei quali appartenenti alla release di SMILA utilizzata, ed alcuni appartenenti al dominio `it.polimi.chansonnier`. All'insieme di bundle si affianca una cartella di configurazione dove i file XML sono divisi per bundle di riferimento, e una cartella workspace contenente lo stato dell'applicazione.

L'applicazione è realizzata tramite l'inserimento di unità personalizzate all'interno dei tre componenti principali di SMILA:

- **Connectivity**: Agent personalizzato per il recupero delle canzoni e interfaccia Web per la produzione di input per tale Agent (inserimento di link).
- **Processing**: `ProcessingService` di vario tipo per l'annotazione delle canzoni e pipeline specifica per organizzare il loro lavoro, eseguito all'aggiunta di una canzone; popolazione di un indice Solr delle canzoni per la loro ricerca..
- **Search**: interfaccia Web realizzata in Ajax per la ricerca di canzoni e navigazione dei risultati.

4.1 Modello dei dati

Il modello dei dati utilizzato ha come elemento atomico (record) la canzone.

La chiave univoca a livello di sorgente dati è il link della pagina Web contenente la canzone in una qualsiasi forma. La chiave completa del record è costituita quindi dal nome della sorgente ('youtube') affiancata a tale link.

Una canzone contiene una serie di attributi e attachment, alcuni estratti dal punto di origine dal quale proviene il Record indicizzato, ed altri ricavati da attributi generalmente esistenti all'interno del processo di indicizzazione, tramite il collegamento di diversi annotatori.

Il record costituente una singola canzone contiene questi attributi:

- **link**: link della pagina Web contenente il video musicale, che fa da identificatore per la singola canzone.
- **pageTitle**: titolo della pagina Web correlata (proveniente dalla pagina Web contenente il video)
- **description**: meta tag descrizione della pagina originale (proveniente dalla pagina Web contenente il video)

- **keywords**: meta tag keywords della pagina originale (proveniente dalla pagina Web contenente il video)
- **title**: titolo della canzone
- **artist**: artista che esegue la canzone.
- **lyrics**: testo della canzone.
- **language**: lingua della canzone.
- **languageConfidence**: parametro variabile da 0 a 1 che identifica la bontà di *language* come valore fuzzy.
- **emotion**: emozione primaria espressa dalla canzone
- **emotionConfidence**: parametro variabile da 0 a 1 che identifica la bontà di *emotion* come valore fuzzy.
- **image**: attributo multiplo contenente i nomi degli attachment rappresentanti immagini estratte dal video della canzone (o di altro genere).

I dati di tipo binario sono invece sotto forma di attachment:

- **original**: binario contenente la canzone nella sua forma originaria quando prelevata dal Web (solitamente video)
- **image{\$i}**: fotogrammi estratti dal video *original*. \$i è un numero intero positivo.

4.2 Annotatori

Gli annotatori utilizzati spaziano da servizi Web a librerie in codice Java nativo. Ogni entità esterna, sia essa un servizio Web o un software, è posta dietro a un livello di astrazione costituita dai ProcessingService personalizzati di Chansonnier.

Gli annotatori descritti qui sono quindi da intendere come driver dei ProcessingService di Chansonnier, e possono essere scambiati con un'altra possibile implementazione via file di configurazione.

4.2.1 LyricWiki

LyricWiki è un wiki pubblico ospitato sulla piattaforma wikia (la stessa di Wikipedia), che si occupa di raccogliere testi di canzoni più o meno famose tramite contribuzione e revisione volontaria da parte della sua comunità di utenti.

L'annotatore relativo a LyricWiki annota quindi una canzone con il proprio testo, effettuando una richiesta HTTP a LyricWiki ed estraendo i risultati dalla risposta.

LyricWiki ha anche un servizio Web pubblico e stateless per l'accesso ai dati contenuti nel wiki. Tale servizio espone due API, una di tipo SOAP e l'altra di tipo REST-like. Data la potenza

equivalente dei due metodi di accesso, si è scelto di privilegiare la semplicità di utilizzo e fare riferimento all'interfaccia di tipo REST.

Ad esempio, per ottenere i dati relativi alla canzone Stairway to heaven dei Led Zeppelin, la richiesta HTTP è una GET al seguente URL:

```
http://lyrics.wikia.com/api.php?func=getSong&artist=Led%20Zeppelin&song=Stairway%20to%20heaven&fmt=xml
```

che produrrà una risposta come:

```
<LyricsResult>
  <artist>Led Zeppelin</artist>
  <song>Stairway To Heaven</song>
  <lyrics>
    There's a lady who's sure all that glitters is gold
    And she's buying a stairway to Heaven
    When she gets there, she knows if the stores are all closed
    With a word she can get what she came for
    Ooh-ooh-hoo, hoo, ooh-ooh-ooh-ooh
    And she's buying a stairway t[...]
  </lyrics>
  <url>
    http://lyrics.wikia.com/Led_Zeppelin:Stairway_To_Heaven
  </url>
</LyricsResult>
```

Per questioni di diritti d'autore, il servizio Web non ritorna il testo completo ma solamente una frazione di esso che non infranga il principio di fair use.

4.2.2 Google Translate

Google Translate è un servizio Web del motore di ricerca Google accessibile anche programmaticamente, e che permette di ottenere traduzioni automatiche di testi da una lingua all'altra, ma soprattutto di riconoscere con un certo grado di affidabilità la lingua di appartenenza di un testo qualsiasi.

L'API utilizzata non è propriamente di tipo REST ma non ha nemmeno la pesantezza di un sistema SOAP. Una richiesta GET come:

```
http://ajax.googleapis.com/ajax/services/language/detect?v=1.0&q=hello%20world'
```

restituisce un risultato Json:

```
{
  "responseData" : {
    "language" : "en",
    "isReliable" : true,
    "confidence" : 0.919745
  },
  "responseDetails" : null,
  "responseStatus" : 200
}
```

dal quale si può estrarre facilmente il codice ISO della lingua trovata. Il servizio Web

richiede di includere un referrer anche fittizio negli header HTTP della richiesta, in quanto è pensato per l'utilizzo all'interno di applicazioni Web.

4.2.3 Synesketch

Le librerie Synesketch hanno la capacità di analizzare un testo (in lingua inglese) tramite reti semantiche allo scopo di scoprire qual è l'emozione dominante contenuta in esso.

In questo caso, non si tratta di un servizio Web ma di una libreria scritta in Java e che quindi può essere integrata facilmente all'interno dell'applicazione. Il bundle contenente il wrapper per Synesketch include quindi anche i Jar della libreria all'interno del proprio classpath.

A partire da un testo di qualche decina di parole, Synesketch produce un rapporto sul peso delle sei emozioni di base (Happiness, Sadness, Fear, Anger, Surprise, Disgust) stabilendo quindi anche qual è la principale. La divisione in sei emozioni di base ricalca il lavoro di Paul Ekman sull'emozione umana.⁹

Il driver Synesketch quindi ritorna anche un parametro di confidenza insieme all'emozione di maggior peso, allegando proprio il peso in questione al risultato.

Segue un esempio dei risultati ottenuti da Synesketch a partire da un testo, tramite il test unitario del suo wrapper.

```
/* *****  
 * Copyright (c) 2010 Giorgio Sironi. All rights reserved.  
 * This program and the accompanying materials are made available under  
 * the terms of the Eclipse Public License v1.0 which accompanies this  
 * distribution, and is available at http://www.eclipse.org/legal/epl-v10.html  
 * ***** */  
package it.polimi.chansonnier.driver.synesketch.test;  
  
import  
it.polimi.chansonnier.driver.synesketch.SynesketchemotionRecognitionService;  
import it.polimi.chansonnier.spi.EmotionRecognitionService;  
import it.polimi.chansonnier.spi.FuzzyResult;  
import junit.framework.TestCase;  
  
public class SynesketchEmotionRecognitionServiceTest extends TestCase {  
    private EmotionRecognitionService synesketch = new  
SynesketchEmotionRecognitionService();  
  
    public void testRecognizesHappiness() {  
        FuzzyResult emotion = synesketch.getEmotion("I'm so glad it's a  
sunny day!");  
        assertEquals("happiness", emotion.getValue());  
        assertGreaterThan(emotion.getConfidence(), 0.4);  
    }  
  
    public void testRecognizesSadness() {  
        FuzzyResult emotion = synesketch.getEmotion("I am full of sorrow and  
regrets...");
```

```

        assertEquals("sadness", emotion.getValue());
        assertGreaterThan(emotion.getConfidence(), 0.15);
    }

    public void testRecognizesFear() {
        FuzzyResult emotion = synesketch.getEmotion("I am shaking... Horror
movies scare me");
        assertEquals("fear", emotion.getValue());
        assertGreaterThan(emotion.getConfidence(), 0.3);
    }

    public void testRecognizesAnger() {
        FuzzyResult emotion = synesketch.getEmotion("I am mad at you!");
        assertEquals("anger", emotion.getValue());
        assertGreaterThan(emotion.getConfidence(), 0.2);
    }

    public void testRecognizesDisgust() {
        FuzzyResult emotion = synesketch.getEmotion("This is so
repulsive...");
        assertEquals("disgust", emotion.getValue());
        assertGreaterThan(emotion.getConfidence(), 0.18);
    }

    public void testRecognizesSurprise() {
        FuzzyResult emotion = synesketch.getEmotion("This is really
unexpected for me!");
        assertEquals("surprise", emotion.getValue());
        assertGreaterThan(emotion.getConfidence(), 0.75);
    }

    public void testDoesNotJudgeIfThereIsNoStrongestEmotion()
    {
        FuzzyResult emotion = synesketch.getEmotion("Hello.");
        assertLessThan(emotion.getConfidence(), 0.1);
    }

    private void assertGreaterThan(Double confidence, Double reference) {
        assertTrue("Confidence is " + confidence, confidence > reference);
    }

    private void assertLessThan(Double confidence, Double reference) {
        assertTrue("Confidence is " + confidence, confidence < reference);
    }
}

```

4.2.4 Ffmpeg

Ffmpeg è un software open source per la manipolazione di tracce multimediali sia audio che video. Annotatori basati su ffmpeg sono presenti all'interno di Chansonnier allo scopo di convertire il formato della traccia audio o video in un file "understandable" da altri annotatori, o per estrarre artefatti multimediali come singoli fotogrammi da mostrare durante la ricerca.

L'uso fatto di ffmpeg è tramite linea di comando, encapsulate tramite la classe `java.lang.Runtime` e il suo metodo `exec()`. La configurazione dell'annotatore necessita quindi di un parametro relativo al percorso assoluto del comando `ffmpeg` per funzionare.

Su di un sistema Linux/Unix, anche server, `ffmpeg` è solitamente disponibile per l'installazione tramite repository.

4.3 Servizi OSGi

I bundle comunicano tra di loro tramite la dichiarazione di servizi OSGi, solitamente nella forma di Declarative Service. I bundle `it.polimi.chansonnier` dichiarano un servizio che rispetta una certa interfaccia definita dal framework SMILA tramite un descrittore XML, cosicché il framework OSGi (di default Equinox, essendo SMILA un progetto della Eclipse Foundation) possa gestire il ciclo di vita dei servizi in quanto oggetti.

4.3.1 LinkGrabberAgent

`LinkGrabberAgent` è un'implementazione custom di un Agent, che rimane in esecuzione per processare una coda di link provenienti da una qualche interfaccia utente, e produce Record rappresentanti canzoni, o più in genere di video contenenti una traccia musicale che possono entrare poi nella fase di processing.

`LinkGrabberAgent` viene recuperato dall'Activator del bundle `it.polimi.chansonnier.servlet` al momento dell'attivazione dell'interfaccia Web. Successivamente i link inseriti dall'utente sono passati ad esso dalle servlet.

`LinkGrabberAgent` si deve occupare quindi di generare una chiave per il record prodotto (l'URI della pagina), e di estrarre gli elementi fondamentali dal codice HTML della pagina puntata dal link, quali gli attributi *pageTitle*, *keywords* e *description*, ma anche l'attachment *original*, estraibile in quanto risorsa inclusa da tale HTML. `YoutubeLinkGrabberAgent` aggiunge anche l'attributo *link* sebbene sia già all'interno della chiave, per semplicità di indicizzazione in Solr.

L'implementazione utilizzata come default in `Chansonnier` è `YoutubeLinkGrabberAgent`, che svolge questo lavoro attraverso su pagine del portale Youtube. Gli attachment *original* prodotti da questa implementazione sono quindi file multimediali in formato FLV, mentre gli attributi testuali provengono dai meta tag della pagina del video.

```
/* *****  
 * Copyright (c) 2010 Giorgio Sironi. All rights reserved.  
 * This program and the accompanying materials are made available under  
 * the terms of the Eclipse Public License v1.0 which accompanies this  
 * distribution, and is available at http://www.eclipse.org/legal/epl-v10.html  
 * *****/  
package it.polimi.chansonnier.agent;  
  
import it.polimi.chansonnier.utils.URLUtils;
```

```

import java.io.ByteArrayOutputStream;
...

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.eclipse.smila.connectivity.framework.AbstractAgent;
...

public class YoutubeLinkGrabberAgent extends AbstractAgent implements
LinkGrabberAgent {
    private RecordFactory _factory = RecordFactory.DEFAULT_INSTANCE;
    private Queue<String> _linksToProcess = new LinkedList<String>();
    private YoutubeGrabber _grabber;
    private final Log _log = LogFactory.getLog(YoutubeLinkGrabberAgent.class);
    ...

    @Override
    public synchronized void addLink(String link) {
        _linksToProcess.add(link);
    }

    @Override
    public void run() {
        try {
            while (!isStopThread()) {
                if (!_linksToProcess.isEmpty()) {
                    String newLink = _linksToProcess.poll();
                    Record newRecord = _createRecord();
                    newRecord.setId(_createId(newLink));
                    _setLink(newRecord, newLink);
                    _setPageTitle(newRecord, _getPageTitle(newLink));
                    _setDescription(newRecord, _getDescription(newLink));
                    _setKeywords(newRecord, _getKeywords(newLink));

                    _log.debug("it.polimi.chansonnier.agent.YoutubeLinkGrabberAgent:
downloading video");
                    _setOriginal(newRecord, _grabber.getVideo(newLink));

                    _log.debug("it.polimi.chansonnier.agent.YoutubeLinkGrabberAgent: completed
video");
                    getControllerCallback().add(getSessionId(),
getConfig().getDeltaIndexing(), newRecord, "424242");
                }
            } // while
        } catch (Throwable t) {
            ...
        } finally {
            ...
        }
    }

    private void _setLink(Record record, String link) {

```

```

        Attribute attribute = new AttributeImpl();
        attribute.setName("link");
        Literal l = new LiteralImpl();
        l.setStringValue(link);
        attribute.addLiteral(l);
        record.getMetadata().setAttribute("link", attribute);
    }

    // altri metodi _set*()...

    private void _setOriginal(Record record, InputStream video) throws
IOException {
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        byte[] b = new byte[4096];
        for (int n; (n = video.read(b)) != -1; ) {
            out.write(b, 0, n);
        }
        record.setAttachment("original", out.toByteArray());
    }
}

```

4.3.2 LyricsProcessingService

LyricsProcessingService ha due principali responsabilità molto legate fra loro. La prima è di estrarre valori per gli attributi *title* e *artist* a partire dal *pageTitle* presente nel record, mentre la seconda è di recuperare il testo della canzone così identificata e di includerlo nell'attributo *lyrics*.

L'implementazione di default in Chansonnier utilizza LyricWiki, un'enciclopedia a sviluppo pubblico contenente un vasto database di testi di canzoni. Questo annotatore effettua diversi tentativi di recuperare un testo provando diverse combinazioni di possibili attributi "inferred" *title* e *artist*, fermandosi quando il servizio Web restituisce un risultato .

```

package it.polimi.chansonnier.processing;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import it.polimi.chansonnier.spi.LyricsService;

import org.eclipse.smila.blackboard.Blackboard;
import org.eclipse.smila.blackboard.BlackboardAccessException;
import org.eclipse.smila.blackboard.path.Path;
import org.eclipse.smila.datamodel.id.Id;
import org.eclipse.smila.datamodel.record.Literal;
import org.eclipse.smila.processing.ProcessingException;
import org.eclipse.smila.processing.ProcessingService;

public class LyricsProcessingService extends AbstractProcessingService
implements ProcessingService {
    private LyricsService _lyricsService;

```

```

@Override
public Id[] process(Blackboard blackboard, Id[] recordIds)
    throws ProcessingException {
    try {
        if (this._lyricsService == null) {
            throw new ProcessingException("Unable to search lyrics:
LyricsService is not initialized.");
        }
        for (Id id : recordIds) {
            String pageTitle = blackboard.getLiteral(id, new
Path(getInputPath(blackboard, id))).toString();
            pageTitle = removeNoise(pageTitle);
            String[] pieces = pageTitle.split("-");
            String title = "Unknown";
            String artist = "Unknown";
            String lyrics = null;
            if (pieces.length == 2) {
                // Led Zeppelin - Stairway to heaven
                artist = pieces[0].trim();
                title = pieces[1].trim();
                lyrics = _lyricsService.getLyrics(title, artist);
                if (lyrics == null) {
                    // Stairway to heaven - Led Zeppelin
                    title = pieces[0].trim();
                    artist = pieces[1].trim();
                    lyrics = _lyricsService.getLyrics(title,
artist);
                }
            }

            if (lyrics != null) {
                Literal lyricsAttribute =
blackboard.createLiteral(id);
                lyricsAttribute.setStringValue(lyrics);
                blackboard.setLiteral(id, new
Path(getOutputPath(blackboard, id)), lyricsAttribute);
            }
            Literal artistAttribute = blackboard.createLiteral(id);
            artistAttribute.setStringValue(artist);
            blackboard.setLiteral(id, new Path("artist"),
artistAttribute);

            Literal titleAttribute = blackboard.createLiteral(id);
            titleAttribute.setStringValue(title);
            blackboard.setLiteral(id, new Path("title"),
titleAttribute);
        }
    } catch (BlackboardAccessException e) {
        throw new ProcessingException(e);
    }
    return recordIds;
}

private String removeNoise(String pageTitle) {
    pageTitle = removeTextBetweenParenthesis(pageTitle);
}

```

```

        pageTitle = removeBadChars(pageTitle);
        pageTitle = removeCommonExpressions(pageTitle);
        return pageTitle;
    }

    public String removeTextBetweenParenthesis(String text) {
        return text.replaceAll("\\([^(]*\\)", "");
    }

    private String removeBadChars(String text) {
        return text.replaceAll("[^A-Za-z0-9- ]{1}", "");
    }

    private String removeCommonExpressions(String text) {
        Pattern badWord = Pattern.compile("(with){0,1} lyrics",
Pattern.CASE_INSENSITIVE);
        Matcher matcher = badWord.matcher(text);
        text = matcher.replaceAll("");
        return text;
    }

    public void setLyricsService(LyricsService lyricsService) {
        _lyricsService = lyricsService;
    }

    public void unsetLyricsService(LyricsService lyricsService) {
        _lyricsService = null;
    }
}

```

4.3.3 EmotionProcessingService

EmotionProcessingService si occupa di aggiungere un attributo *emotion* alla canzone sotto esame, ricavato dall'attributo Lyrics e corredato dal parametro *emotionConfidence*.

L'implementazione di default utilizza le librerie in Java nativo Synesketch, che ricavano da un testo di media lunghezza l'emozione dominante e la confidenza di questo risultato.

```

package it.polimi.chansonnier.processing;

import it.polimi.chansonnier.spi.EmotionRecognitionService;
import it.polimi.chansonnier.spi.FuzzyResult;

import org.eclipse.smila.blackboard.Blackboard;
import org.eclipse.smila.blackboard.BlackboardAccessException;
import org.eclipse.smila.blackboard.path.Path;
import org.eclipse.smila.datamodel.id.Id;
import org.eclipse.smila.datamodel.record.Literal;
import org.eclipse.smila.processing.ProcessingException;
import org.eclipse.smila.processing.ProcessingService;

public class EmotionProcessingService extends AbstractProcessingService

```



```

implements ProcessingService {

    private EmotionRecognitionService _emotionRecognitionService;

    @Override
    public Id[] process(Blackboard blackboard, Id[] recordIds)
        throws ProcessingException {
        try {
            for (Id id : recordIds) {
                Path p = new Path(getInputPath(blackboard, id));
                Literal lyrics = blackboard.getLiteral(id, p);
                FuzzyResult emotion =
                    _emotionRecognitionService.getEmotion(lyrics.toString());
                Literal value = blackboard.createLiteral(id);
                value.setStringValue(emotion.getValue());
                blackboard.addLiteral(id, new
                    Path(getOutputPath(blackboard, id)), value);
                Literal confidence = blackboard.createLiteral(id);
                confidence.setFpValue(emotion.getConfidence());
                blackboard.addLiteral(id, new
                    Path(getOutputPath(blackboard, id) + "Confidence"), confidence);
            }
        } catch (BlackboardAccessException e) {
            throw new ProcessingException(e);
        }
        return recordIds;
    }

    public void setEmotionRecognitionService(
        EmotionRecognitionService emotionRecognitionService) {
        _emotionRecognitionService = emotionRecognitionService;
    }
}

```

4.3.4 LanguageProcessingService

LanguageProcessingService è simile in struttura ad EmotionProcessingService in quanto si basa ancora sul valore dell'attributo Lyrics. Questo servizio aggiunge l'attributo *language*, ricavato quindi dal testo della canzone, e la bontà di tale risultato nell'attributo *languageConfidence*.

L'implementazione di default utilizza l'Api di Google Translate, che è in grado di riconoscere la lingua di un testo ed indicare l'affidabilità del proprio risultato.

```

package it.polimi.chansonnier.processing;

import it.polimi.chansonnier.spi.FuzzyResult;
import it.polimi.chansonnier.spi.LanguageRecognitionService;

import org.eclipse.smila.blackboard.Blackboard;
import org.eclipse.smila.blackboard.BlackboardAccessException;
import org.eclipse.smila.blackboard.path.Path;

```

```

import org.eclipse.smila.datamodel.id.Id;
import org.eclipse.smila.datamodel.record.Literal;
import org.eclipse.smila.processing.ProcessingException;
import org.eclipse.smila.processing.ProcessingService;

public class LanguageProcessingService extends AbstractProcessingService
implements ProcessingService {
    private LanguageRecognitionService _driver;

    @Override
    public Id[] process(Blackboard blackboard, Id[] recordIds)
        throws ProcessingException {

        try {
            for (Id id : recordIds) {
                Path p = new Path(getInputPath(blackboard, id));
                Literal lyrics = blackboard.getLiteral(id, p);
                FuzzyResult language =
                _driver.getLanguage(lyrics.toString());
                Literal value = blackboard.createLiteral(id);
                value.setStringValue(language.getValue());
                blackboard.addLiteral(id, new
                Path(getOutputPath(blackboard, id)), value);
                Literal confidence = blackboard.createLiteral(id);
                confidence.setFpValue(language.getConfidence());
                blackboard.addLiteral(id, new
                Path(getOutputPath(blackboard, id) + "Confidence"), confidence);
            }
        } catch (BlackboardAccessException e) {
            throw new ProcessingException(e);
        }
        return recordIds;
    }

    public void setLanguageRecognitionService(LanguageRecognitionService
driver) {
        _driver = driver;
    }
}

```

4.3.5 FrameExtractionProcessingService

FrameExtractionProcessingService si occupa di estrarre fotogrammi a intervalli regolari da un video sotto indicizzazione, allo scopo di presentarli all'utente durante la ricerca.

L'implementazione di default presente in Chansonnier sfrutta l'applicazione open source ffmpeg, che deve essere quindi disponibile nella piattaforma ospite. L'api Runtime di Java richiama ffmpeg tramite linea di comando più volte allo scopo di produrre varie immagini conservate come attachment, ma anche passibili di ulteriori elaborazioni.

Vista la necessità di riconoscere la presenza di attachment come i frame estratti anche

dall'interfaccia Web, FrameExtractionProcessingService aggiunge anche all'attributo multiplo *image* dei valori costituenti i nomi degli attachment prodotti. Solo gli attributi (testuali e non binari come gli attachment) sono indicizzati all'interno di Solr e come tali sono disponibili all'interfaccia Web.

```
package it.polimi.chansonnier.processing;

import it.polimi.chansonnier.spi.FrameExtractionService;

import java.io.File;

import org.eclipse.smila.blackboard.Blackboard;
import org.eclipse.smila.blackboard.BlackboardAccessException;
import org.eclipse.smila.blackboard.path.Path;
import org.eclipse.smila.datamodel.id.Id;
import org.eclipse.smila.datamodel.record.Literal;
import org.eclipse.smila.processing.ProcessingException;
import org.eclipse.smila.processing.ProcessingService;

public class FrameExtractionProcessingService extends AbstractProcessingService
implements ProcessingService {

    private FrameExtractionService frameExtractionService;

    @Override
    public Id[] process(Blackboard blackboard, Id[] recordIds)
        throws ProcessingException {
        try {
            for (Id id : recordIds) {
                String outputPath = getOutputPath(blackboard, id);
                System.out.println(outputPath);

                File original = blackboard.getAttachmentAsFile(id,
getInputPath(blackboard, id));
                File frame = frameExtractionService.getImage(original,
"00:00:10");
                addImageAttribute(blackboard, id, outputPath + "1");
                blackboard.setAttachmentFromFile(id, outputPath + "1",
frame);

                File frame2 = frameExtractionService.getImage(original,
"00:00:30");
                addImageAttribute(blackboard, id, outputPath + "2");
                blackboard.setAttachmentFromFile(id, outputPath + "2",
frame2);

                File frame3 = frameExtractionService.getImage(original,
"00:00:50");
                addImageAttribute(blackboard, id, outputPath + "3");
                blackboard.setAttachmentFromFile(id, outputPath + "3",
frame3);
            }
        } catch (BlackboardAccessException e) {
```

```

        throw new ProcessingException(e);
    }
    return recordIds;
}

private void addImageAttribute(Blackboard blackboard, Id id, String
attachmentName) throws BlackboardAccessException {
    Literal literal = blackboard.createLiteral(id);
    literal.setStringValue(attachmentName);
    blackboard.addLiteral(id, new Path(getOutputPath(blackboard, id)),
literal);
}

public void setFrameExtractionService(
    FrameExtractionService shotDetectionService) {
    frameExtractionService = shotDetectionService;
}
}

```

4.3.6 SolrPipelet

SolrPipelet è una classe fornita da un bundle di SMILA, che fa da wrapper all'interfaccia REST di un'istanza di Solr in ambiente Java.

Questa pipelet produce automaticamente un documento che viene spedito tramite POST a Solr utilizzando l'Id del Record per produrre una “chiave” e trasformando gli attributi in campi del documento postato.

```

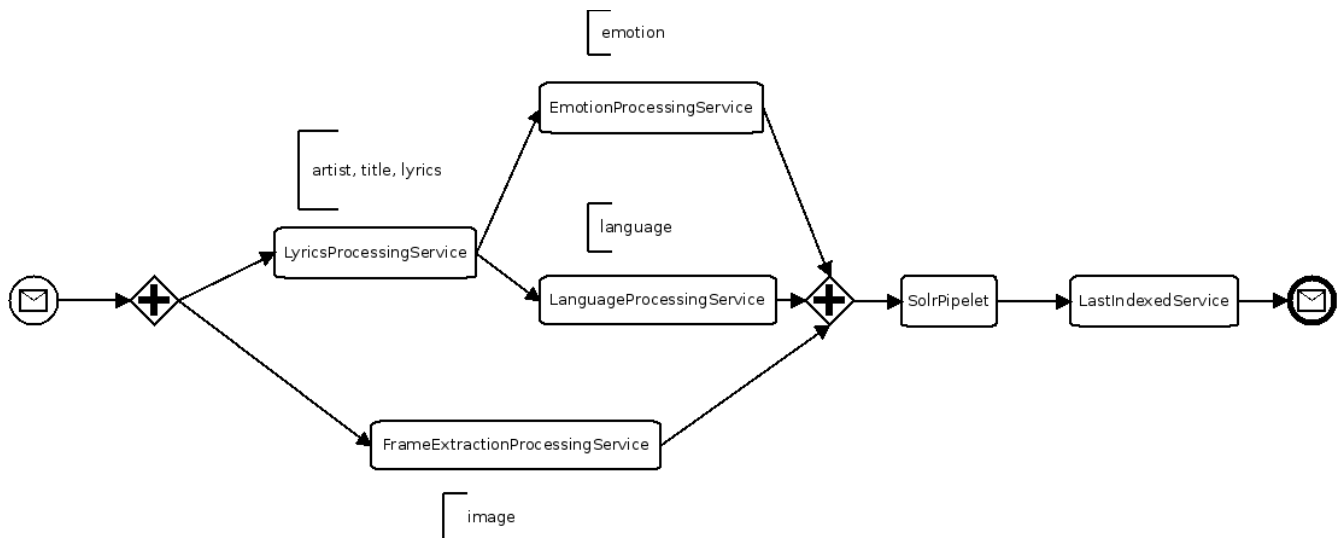
<?xml version="1.0" encoding="UTF-8"?>
<add commitWithin="10000" overwrite="true">
  <doc>
    <field name="id">%3CId+version%3D%221.0%22+xmlns%3D%22http%3A%2F
%2Fwww.eclipse.org%2Fsmila%2Fid%22%3E%3CSource%3Eyoutube%3C%2FSource%3E%3CKey
%3Ehttp%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3De8w7f0ShtIM%3C%2FKey%3E%3C%2FId
%3E</field>
    <field name="pageTitle">U2 - Beautiful Day (with Lyrics)</field>
    <field name="link">http://www.youtube.com/watch?v=e8w7f0ShtIM</field>
    <field name="lyrics">The heart is a bloom, shoots up through the stony
ground
There's no room, no space to rent in this town
You're out of luck and the reason that you had to care
The traffic is stuck and you're not [...]</field>
    <field name="artist">U2</field>
    <field name="title">Beautiful Day</field>
    <field name="emotion">happiness</field>
    <field name="emotionConfidence">0.06818181818181818</field>
    <field name="language">en</field>
    <field name="languageConfidence">0.9081169</field>
    <field name="image">image1</field>
    <field name="image">image2</field>
    <field name="image">image3</field>
  </doc>

```

4.3.7 LastIndexedService

LastAddedService è l'ultimo servizio della pipeline di indicizzazione, che tiene traccia degli ultimi record il cui processo di annotazione è quindi terminato. LastAddedService può essere quindi "richiamato" da un'interfaccia utente o di test per verificare se e quando una richiesta di indicizzazione è stata soddisfatta.

La pipeline di annotazione è un workflow che può essere parallelizzato:



L'implementazione della pipeline contenuta in Chansonnier svolge comunque i compiti in modo lineare, essendo il download del file il vero collo di bottiglia del processo.

4.4 Indice

Solr è un server di ricerca open source del progetto Apache Lucene. Utilizza Lucene internamente ed espone un'interfaccia HTTP che lo rende particolarmente adatto all'utilizzo in ambito Web.

Tutti gli attributi delle canzoni analizzate dal sistema sono indicizzati in Solr in modo asincrono (al momento dell'aggiunta al database e non della ricerca) e sono disponibili per la ricerca su un particolare campo. Solr si occupa delle operazioni orientate alla ricerca come la creazione di indici full text su alcuni campi, e di creare un id univoco per ogni record adatto alla trasmissione via URI o all'utilizzo nell'interfaccia per identificarne gli elementi.

Il modello dei dati è volutamente semplice (una mappa flat di attributi e attachment per il singolo record, non gerarchica) per permettere l'indicizzazione degli attributi testuali in una varietà di back end. SMILA permetterebbe di definire annotazioni su singoli attributi o addirittura su altre annotazioni e di navigare l'albero così ottenuto tramite espressioni XPath. Tuttavia, i servizi di interfaccia all'indicizzazione inclusi (SolrPipelet) permettono la ricerca solo sugli attributi presenti

al primo livello dell'albero, e definire un mapping dall'albero a una struttura piatta sarebbe oneroso e senza particolari vantaggi.

Solamente gli attributi sono immagazzinati nell'indice Solr:

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema name="example" version="1.2">
  <types>
    <fieldType name="string" class="solr.StrField" sortMissingLast="true"
omitNorms="true"/>
    <fieldType name="float" class="solr.TrieFloatField" precisionStep="0"
omitNorms="true" positionIncrementGap="0"/>

    <fieldType name="text" class="solr.TextField" positionIncrementGap="100">
      <analyzer type="index">
        <tokenizer class="solr.WhitespaceTokenizerFactory"/>
        <filter class="solr.StopFilterFactory"
          ignoreCase="true"
          words="stopwords.txt"
          enablePositionIncrements="true"
        />
        <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1" catenateWords="1" catenateNumbers="1" catenateAll="0"
splitOnCaseChange="1"/>
        <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.SnowballPorterFilterFactory" language="English"
protected="protwords.txt"/>
      </analyzer>
      <analyzer type="query">
        <tokenizer class="solr.WhitespaceTokenizerFactory"/>
        <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
        <filter class="solr.StopFilterFactory"
          ignoreCase="true"
          words="stopwords.txt"
          enablePositionIncrements="true"
        />
        <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1" catenateWords="0" catenateNumbers="0" catenateAll="0"
splitOnCaseChange="1"/>
        <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.SnowballPorterFilterFactory" language="English"
protected="protwords.txt"/>
      </analyzer>
    </fieldType>

    <fieldType name="uuid" class="solr.UUIDField" indexed="true" />
  </types>

  <fields>

    <field name="id" type="string" indexed="true" stored="true" required="true"
/>
```

```

<field name="uuid" type="uuid" indexed="true" stored="true" default="NEW"/>
<field name="link" type="string" indexed="true" stored="true"/>

<field name="pageTitle" type="string" indexed="true" stored="true"/>
<field name="artist" type="string" indexed="true" stored="true"/>
<field name="title" type="string" indexed="true" stored="true"/>
<field name="lyrics" type="string" indexed="true" stored="true"/>
<field name="description" type="string" indexed="true" stored="true"/>
<field name="keywords" type="string" indexed="true" stored="true"/>

<field name="emotion" type="string" indexed="true" stored="true"/>
<field name="emotionConfidence" type="float" indexed="true" stored="true"/>
<field name="language" type="string" indexed="true" stored="true"/>
<field name="languageConfidence" type="float" indexed="true" stored="true"/>
<field name="image" type="string" indexed="false" stored="true"
multiValued="true" />

  <field name="fullText" type="text" indexed="true" stored="false"
multiValued="true" />

</fields>

<uniqueKey>link</uniqueKey>

<copyField source="artist" dest="fullText"/>
<copyField source="title" dest="fullText"/>
<copyField source="lyrics" dest="fullText"/>

<solrQueryParser defaultOperator="OR"/>

</schema>

```

5 Implementazione

5.1 Metodologia

5.1.1 Source control

Tutto il codice prodotto è stato inserito in un sistema di controllo versione fin dall'inizio del progetto. Il sistema scelto è Git, un source control management system distribuito.

In Git ogni nodo, sia esso un server o una macchina di sviluppo, ha una copia dell'intero repository a propria disposizione in locale, in forma compressa. Il risultato è un ottimo tradeoff di spazio in confronto di tempo in quanto Git è uno dei più veloci sistemi di controllo versione disponibili, e permette di effettuare commit o creare nuovi branch di sviluppo senza utilizzare la rete (mentre sistemi centralizzati come CVS o Subversion richiedono invece di comunicare con il server).

Inoltre nel modello di sviluppo non esiste un repository definito univocamente come centrale. Chansonnier è partito da un repository locale sulla sua macchina di sviluppo, per poi solo successivamente essere esportato su di un repository nella stessa LAN su una macchina di staging, con il quale la macchina di staging stessa e la macchina di sviluppo si sincronizzavano utilizzandolo come punto di integrazione. In un momento ancora successivo, Chansonnier è stato rilasciato come prodotto open source su Github, che è diventato un altro repository remoto.

Nel repository di Chansonnier è contenuto tutto ciò che serve per compilare il prodotto, così da poter fare checkout del codice e build su qualsiasi macchina che abbia il JDK 6 installato. Ad esempio sono presenti nel repository Eclipse 3.5.2 provvisto di Delta Pack, e Ant 1.8. Il grosso quantitativo di file presenti non è un problema per Git una volta superata l'importazione iniziale, in quanto ovviamente traccia e trasmette ad altri repository solamente i file modificati.

5.1.2 Test-Driven Development

L'implementazione dell'applicazione è stata svolta tramite Test-Driven Development (TDD da qui in poi) sia a livello end-to-end (funzionalità dell'applicazione) che a livello funzionale (grafi di oggetti) ed unitario (singola classe).

Seguendo la tecnica TDD, i test devono essere scritti prima del codice di produzione che verrà da loro esercitato, affinché essi servano come strumento di design dell'Api. Inoltre le metodologie test-first come TDD assicurano che una suite di test di regressione sia presente fin da subito, e che sia in principio costruibile, spingendo il design verso la facilità di test e quindi verso una maggiore manutenibilità.

End-to-end (aka acceptance)

Il primo passo per sviluppare una funzionalità dell'applicazione è sempre definire un test a livello globale che la sfrutti e che fallisca. Vista l'interfaccia Web utilizzata, si è quindi utilizzata l'estensione HttpUnit per JUnit per effettuare richieste HTTP verso l'applicazione avviata in una sandbox, coadiuvata da Selenium per quanto riguarda le pagine di ricerca, che coinvolgono funzionalità JavaScript.

Lo stato iniziale dell'applicazione prima che ogni test case comprende un insieme di record vuoto e un indice anch'esso vuoto, per evitare che qualche forma di stato globale influisca sull'affidabilità dei test. Poiché Chansonnier utilizza un indice Solr che fa anch'esso parte dello stato, esso va resettato allo stesso modo ad ogni esecuzione.

I test end-to-end sono fondamentali per verificare la corretta integrazione dei vari servizi OSGi e il rispetto dei vari contratti definiti fra interfacce e classi di altri bundle che le implementano. Sono però per necessità abbastanza lenti: prima di ogni test case è necessario avviare l'intera applicazione e tutti i suoi servizi, cioè alcune decine di bundle, nel suo stato iniziale senza alcun dato immagazzinato o indicizzato. Per questo sono principalmente fatti girare in un ambiente di integrazione in un server di staging dedicato, e in un ambiente condiviso.

I test end-to-end costituiscono i test di accettazione per Chansonnier e definiscono quando una user story è stata correttamente implementata.

HttpUnit

La libreria HttpUnit estende il framework JUnit per permettere di effettuare richieste GET e POST verso qualsiasi URL (nel caso della suite di test di Chansonnier solitamente localhost), e di effettuare asserzioni sui tag e sul testo presenti nella pagina. HttpUnit si sostituisce così a un semplice browser che definisce richieste HTTP (oggetti `WebRequest`) e analizza oggetti `WebResponse`, analizzandone il contenuto per eseguire le asserzioni definite nel test.

Ad esempio `AddSongTest`, realizzato con HttpUnit, controlla che una volta aggiunto un link alla coda di Chansonnier la canzone appaia nella lista delle ultime canzoni indicizzate entro un timeout di 300 secondi (e sia quindi disponibile per la ricerca).

```
package it.polimi.chansonnier.test;

import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.impl.CommonsHttpSolrServer;
import org.apache.solr.client.solrj.impl.XMLResponseParser;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;
import org.apache.solr.common.SolrDocumentList;
import com.meterware.httpunit.GetMethodWebRequest;
import com.meterware.httpunit.PostMethodWebRequest;
import com.meterware.httpunit.WebConversation;
import com.meterware.httpunit.WebRequest;
import com.meterware.httpunit.WebResponse;
import com.meterware.httpunit.WebForm;
```

```

public class AddSongTest extends AcceptanceTest {
    public void testTheAddPageIsLoaded() throws Exception {
        WebConversation wc = new WebConversation();
        WebRequest req = new GetMethodWebRequest(
"http://localhost:8080/chansonnier/add" );
        WebResponse resp = wc.getResponse( req );
        WebForm add = resp.getForms()[0];
        assertEquals("add", add.getAction());
        assertEquals("post", add.getMethod());
        String[] parameters = add.getParameterNames();
        assertEquals(1, parameters.length);
        assertEquals("link", parameters[0]);
        assertEquals(1, add.getSubmitButtons().length);
    }

    public void
testGivenAYoutubeLinkAddsItToTheLastIndexedListAndTheRelatedSongIsSearchableThro
ughSolr() throws Exception {
        String link = "http://www.youtube.com/watch?v=GMDd4on20Yg";
        WebResponse resp = addVideoLink(link);
        assertTrue(resp.getText().contains("Success"));

        WebRequest req = new GetMethodWebRequest(
"http://localhost:8080/chansonnier/last" );
        assertWebPageContains(req, link, 300000);
        Thread.sleep(10000);

        SolrDocumentList docList = searchForSongs("*:");
        assertEquals(1, docList.size());
        SolrDocument song = docList.get(0);
        assertEquals("Green Day", song.get("artist"));
        assertEquals("Boulevard of Broken Dreams", song.get("title"));
        assertTrue(((String) song.get("lyrics")).contains("I walk a lonely
road"));
    }

    public void testFixturesCanBeAddedWithThePushOfAButton() throws Exception
{
        WebRequest req = new
GetMethodWebRequest("http://localhost:8080/chansonnier/fixtures");
        WebConversation wc = new WebConversation();
        WebResponse resp = wc.getResponse( req );
        assertEquals(1, resp.getForms()[0].getButtons().length);
        req = new PostMethodWebRequest(
"http://localhost:8080/chansonnier/fixtures" );
        resp = wc.getResponse( req );
        Thread.sleep(15000);
        SolrDocumentList result = searchForSongs("title:Hero");
        assertEquals(1, result.size());
        String id = (String) result.get(0).get("link");
        WebRequest image = new
GetMethodWebRequest("http://localhost:8080/chansonnier/attachment?id=" + id +
"&name=image1");
    }
}

```

```

        resp = wc.getResponse(image);
        assertEquals(200, resp.getResponseCode());
    }

    private SolrDocumentList searchForSongs(String queryString) throws
Exception {
        String url = "http://localhost:8983/solr";
        CommonsHttpSolrServer server = new CommonsHttpSolrServer( url );
        server.setParser(new XMLResponseParser());
        SolrQuery query = new SolrQuery();
        query.setQuery(queryString);
        QueryResponse rsp = server.query( query );
        return rsp.getResults();
    }
}

```

Selenium

Selenium è un'applicazione fondamentalmente diversa da HttpUnit, sebbene possa servire allo stesso scopo. La differenza fra i due sono nel modello di esecuzione: mentre HttpUnit si sostituisce a un browser interrogando direttamente un server HTTP, Selenium lancia un browser (come Firefox) e lo utilizza per accedere alle pagine Web da sottoporre a test, diventando effettivamente equivalente a un client reale.

La distribuzione di Selenium utilizzata in questo progetto consiste nell'accoppiata di Selenium RC (Remote Control) e il relativo Java driver. Selenium RC rimane in esecuzione come demone e lancia automaticamente i browser necessari a soddisfare le richieste dei suoi client, mentre il driver “gli manda” una serie di comandi da un test case per JUnit.

Un problema sorto nell'utilizzo di questa configurazione è stata l'esecuzione di Selenium RC su di un server senza interfaccia grafica (demone Xserver in ambiente Linux), ma solo con un interfaccia a linea di comando. Per permettere l'esecuzione dei test (e quindi del processo di build completo) in questo ambiente, si è utilizzato Xvfb (X virtual framebuffer), un Xserver adatto agli ambienti di test che non manda alcun output su display reali, ma li virtualizza in memoria permettendo quindi l'esecuzione di programmi come Firefox da linea di comando.

Xvfb fornisce un eseguibile, xvfb-run, che fa da wrapper ai programmi che necessitano di interfaccia grafica. Preponendo quindi xvfb-run si può lanciare il demone di Selenium RC in modo che faccia uso del display virtuale:

```

export DISPLAY=:99
sudo xvfb-run -e /home/giorgio/log/xvfb-run java -jar selenium-server.jar
-log /home/giorgio/log/selenium &

```

Una volta che Selenium RC è in esecuzione, è possibile effettuare degli screenshot allo schermo virtuale tramite alcuni programmi di utilità:

```

xwd -root -display :99 -out firefox && convert firefox firefox.png && scp
firefox.png Desmond:~

```

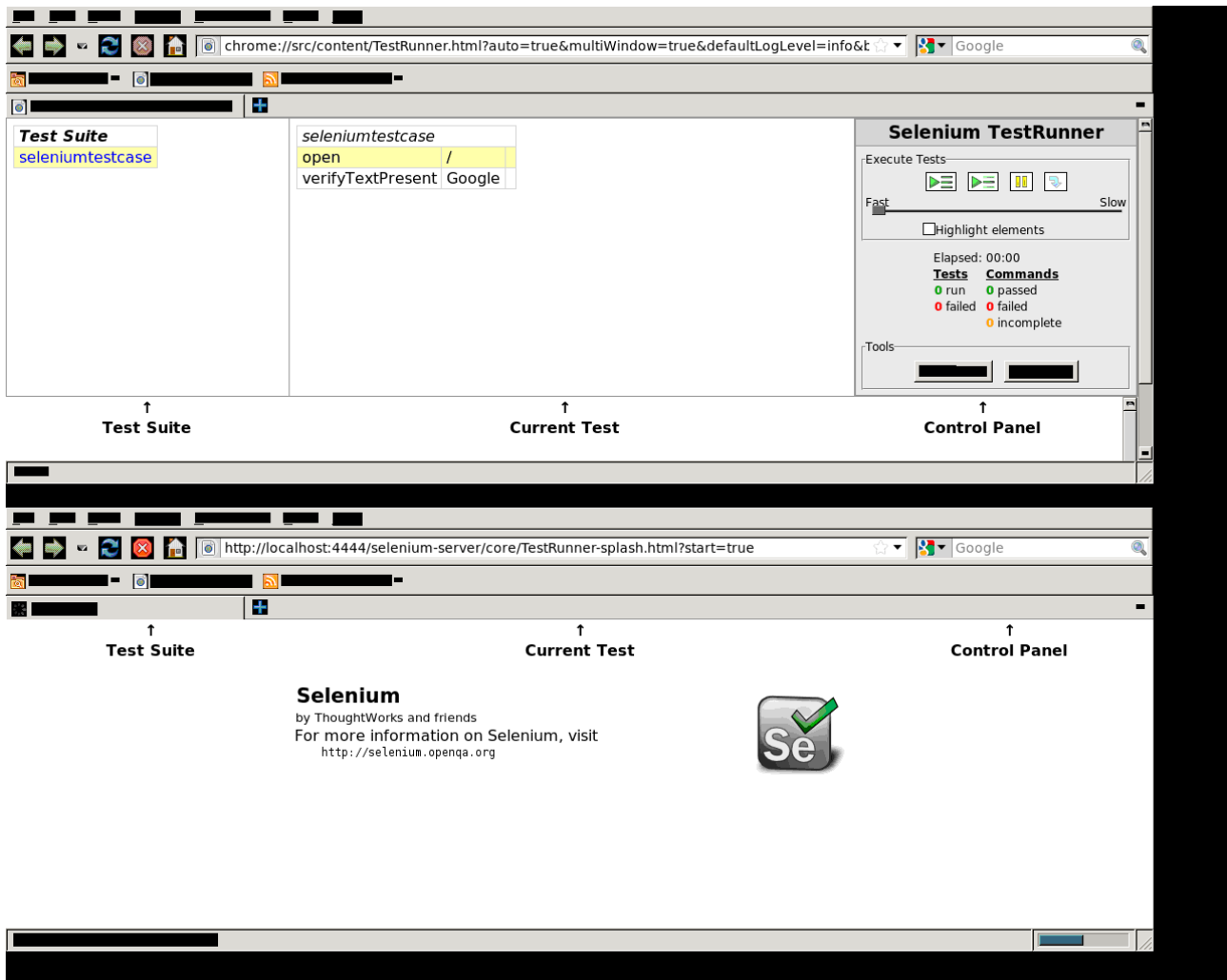


Illustrazione 6: Immagine di Firefox ottenuta dal display virtuale. Le caselle nere che sostituiscono i caratteri denotano l'assenza di alcuni font dalla macchina.

SearchSongTest è realizzato con Selenium: si occupa di aggiungere alcune canzoni, utilizzare l'interfaccia Ajax per effettuare delle ricerche e controllare che restituiscano i risultati corretti.

```
package it.polimi.chansonnier.test;

import it.polimi.chansonnier.fixtures.Fixtures;

import java.io.InputStream;

import com.meterware.httpunit.GetMethodWebRequest;
import com.meterware.httpunit.WebRequest;

public class SearchSongTest extends AcceptanceTest {
    public void testGivenAnAddedYouTubeLinkTheSongIsSearchable() throws
    Exception {
        InputStream beautifulDayFlv =
        Fixtures.class.getResourceAsStream("beautifulday.flv");
        fixtureManager.addSong("http://www.youtube.com/watch?v=e8w7f0ShtIM",
        beautifulDayFlv, "U2 - Beautiful Day (with Lyrics)");
        InputStream heroFlv =
```

```

Fixtures.class.getResourceAsStream("hero.flv");
    fixtureManager.addSong("http://www.youtube.com/watch?v=owTmJrtD7g8",
heroFlv, "Enrique Iglesias- Hero (with lyrics)");
    InputStream haloFlv =
Fixtures.class.getResourceAsStream("halo.flv");
    fixtureManager.addSong("http://www.youtube.com/watch?v=fSdgBse1o7Q",
haloFlv, "Beyonce-Halo Lyrics");

    WebRequest req = new GetMethodWebRequest(
"http://localhost:8080/chansonnier/last" );
    assertWebPageContains(req, "http://www.youtube.com/watch?
v=e8w7f0ShtIM", 20000);
    Thread.sleep(10000);

    selenium.open("/chansonnier/index.html");
    selenium.click("link=happiness");
        waitForPageToLoad();
        wrapped.verifyTrue(selenium.isTextPresent("(x)
emotion:\\"happiness\\""));
        wrapped.verifyTrue(selenium.isTextPresent("Beaut
iful Day"));
    wrapped.verifyTrue(selenium.isTextPresent("The heart is a bloom"));
    selenium.click("link=(x) emotion*");
    selenium.click("link=anger");
        waitForPageToLoad();
    wrapped.verifyTrue(selenium.isTextPresent("(x) emotion:\\"anger\\""));
    wrapped.verifyTrue(selenium.isTextPresent("Hero"
));

    wrapped.verifyTrue(selenium.isTextPresent("Would you dance"));
    selenium.click("link=(x) emotion*");
        waitForPageToLoad();
    selenium.click("link=en");
    wrapped.verifyTrue(selenium.isTextPresent("of 3 results"));
    selenium.click("link=Enrique Iglesias");
    wrapped.verifyTrue(selenium.isTextPresent("Hero"));
    wrapped.verifyFalse(selenium.isTextPresent("Beautiful Day"));

    selenium.refresh();
    waitForPageToLoad();
    typeAndEnter("query", "Enrique");
    waitForPageToLoad();
    assertOnlyOneHeroSongIsInTheResult();
    typeAndEnter("query", "Hero");
    typeAndEnter("query", "Let me be your hero");
    typeAndEnter("query", "Enrique Iglesias");
    waitForPageToLoad();
    assertOnlyOneHeroSongIsInTheResult();
}

private void typeAndEnter(String identifier, String value) {
    selenium.type(identifier, value);
    selenium.keyDown(identifier, "\\13");
}

```

```

private void assertOnlyOneHeroSongIsInTheResult() {
    wrapped.verifyTrue(selenium.isTextPresent("(x) fullText:Enrique"));
    wrapped.verifyTrue(selenium.isTextPresent("Hero"));
    wrapped.verifyFalse(selenium.isTextPresent("Beyonce"));
    wrapped.verifyTrue(selenium.isTextPresent("of 1 results"));
}
}

```

SolrJ

SolrJ è la più diffusa libreria Java che faccia da wrapper all'Api REST di Solr. Poiché SMILA integra già l'inserimento di dati in Solr tramite una Pipelet apposita e l'estrazione dei dati avviene tramite oggetti JavaScript XMLHttpRequest, SolrJ non è necessaria al funzionamento del codice di produzione di Chansonnier.

SolrJ viene invece utilizzata all'interno dei test di accettazione per effettuare richieste a Solr, e resettarne lo stato fra un test e l'altro all'interno dei metodi setUp().

I test relativi a queste richieste sostanzialmente controllano che l'indicizzazione di una canzone sia andata a buon fine, e dopo avere aggiunto alla coda la canzone da processare tramite l'interfaccia Web, aspettano conferma della fine del processo asincrono attendono fino a un time out prestabilito. Successivamente, effettuano una query prestabilita a Solr e si assicurano che i risultati corrispondano ai valori attesi per la canzone originaria.

AddSongTest, presentato precedentemente, utilizza SolrJ e HttpUnit per eseguire le proprie asserzioni.

Functional

L'impiego di test di accettazione durante lo sviluppo è un collo di bottiglia a causa delle loro performance, che sono condizionate dal dover costruire lo stato dell'applicazione da zero, coinvolgendo tutte le risorse reali utilizzate, come server HTTP per scaricare il video della canzone da indicizzare, o servizi Web a cui gli annotatori fanno riferimento.

Una volta quindi scritto un test di accettazione, è utile scendere di livello e predisporre un test funzionale, che eserciti un grafo di oggetti corrispondente a un componente dell'applicazione. In questo modo lo sviluppo del codice coinvolto può contare sul feedback di un test moderatamente veloce, il quale assicura che gli oggetti si integrino correttamente e che un intero componente funzioni al meglio. Si è quindi liberi di modificare il codice per aggiungere funzionalità o a scopo di refactoring “detecting” un guasto appena il test funzionale corrispondente è eseguito. Questo tipo di feedback non è possibile con dei test di accettazione, che non girano praticamente mai sulla macchina di sviluppo.

Nel caso di Chansonnier, i test funzionali coinvolgono la pipeline di annotazione che riceve in ingresso il video musicale e la sua pagina Web, e annota il record rappresentante la canzone con varie informazioni come lingua, testo, ed emozione. La pipeline contiene la logica più complicata e soggetta a cambiamenti dell'applicazione, e quindi il test funzionale relativo è eseguito spesso, ma

questi passa alla pipeline un record costruito usando un file locale (fixture) anziché prelevato dai server di YouTube.

Le asserzioni dei test relativi alla pipeline di annotazione sono analoghe ad alcuni test end-to-end, in quanto fanno uso di SolrJ per accedere direttamente all'indice e ai suoi dati. La differenza sta proprio nell'eliminare la traduzione da link in file multimediale, che coinvolge intensamente la rete.

Si elimina così il collo di bottiglia del download di un file multimediale, il cui peso è dell'ordine di qualche decina di megabyte. Il tempo di esecuzione di un test passa quindi dai 150-200 secondi di un singolo test di accettazione ai 10-15 di un test funzionale, che è qui di seguito presentato.

```
package it.polimi.chansonnier.test;

import it.polimi.chansonnier.fixtures.Fixtures;

import java.io.InputStream;
import java.util.Collection;

import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.SolrServerException;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;
import org.apache.solr.common.SolrDocumentList;
import org.eclipse.smila.datamodel.id.Id;

public class AddPipelineTest extends FunctionalTest {
    public static final String PIPELINE_NAME = "AddPipeline";

    @Override
    protected String getPipelineName() {
        return PIPELINE_NAME;
    }

    public void testSongsAreIndexedInSolr() throws Exception {
        InputStream heroFlv = Fixtures.class.getResourceAsStream("hero.flv");
        Id[] result = fixtureManager.addSong("http://www.youtube.com/watch?v=owTmJrtD7g8", heroFlv, "Enrique Iglesias- Hero (with lyrics)");
        assertEquals(1, result.length);
        InputStream haloFlv = Fixtures.class.getResourceAsStream("halo.flv");
        result = fixtureManager.addSong("http://www.youtube.com/watch?v=fSdgBselo7Q", haloFlv, "Beyonce-Halo Lyrics");
        assertEquals(1, result.length);
        Thread.sleep(15000);

        SolrQuery query = new SolrQuery();
        query.setQuery("title:Hero");
        QueryResponse rsp = solrServer.query(query);
        SolrDocumentList docList = rsp.getResults();
        assertEquals(1, docList.size());
        SolrDocument song = docList.get(0);
        assertEquals("http://www.youtube.com/watch?v=owTmJrtD7g8",
```

```

song.get("link"));
    assertEquals("Enrique Iglesias", song.get("artist"));
    assertEquals("Hero", song.get("title"));
    assertTrue(((String) song.get("lyrics")).contains("if I asked you to
dance"));
    assertEquals("anger", song.get("emotion"));
    assertTrue(((Float) song.get("emotionConfidence")) > 0.01);
    assertEquals("en", song.get("language"));
    assertTrue(((Float) song.get("languageConfidence")) > 0.2);
    Collection attachmentNames = song.getFieldValues("image");
    assertEquals(3, attachmentNames.size());

    String heroLink = "http://www.youtube.com/watch?v=owTmJrtD7g8";
    assertQueryGivesOnlyThisResult("fullText:Hero", heroLink);
    assertQueryGivesOnlyThisResult("fullText:Enrique", heroLink);
    assertQueryGivesOnlyThisResult("fullText:Iglesias", heroLink);
    assertQueryGivesOnlyThisResult("fullText:\"Enrique Iglesias\"",
heroLink);
    assertQueryGivesOnlyThisResult("fullText:\"Would you dance\"",
heroLink);
    assertQueryGivesOnlyThisResult("fullText:\"IF I ASKED YOU\"",
heroLink);

    query = new SolrQuery();
    query.setQuery( "title:Halo" );
    rsp = solrServer.query( query );
    docList = rsp.getResults();
    assertEquals(1, docList.size());
    song = docList.get(0);
    assertEquals("http://www.youtube.com/watch?v=fSdgBse1o7Q",
song.get("link"));
    assertEquals("Beyonce", song.get("artist"));
    assertEquals("Halo", song.get("title"));
    assertEquals("surprise", song.get("emotion"));
    assertEquals("en", song.get("language"));
    assertTrue(((String) song.get("lyrics")).contains("Remember those
walls I built?"));
    attachmentNames = song.getFieldValues("image");
    assertEquals(3, attachmentNames.size());
}

private void assertQueryGivesOnlyThisResult(String queryText, String
expectedLinkField) throws SolrServerException {
    SolrQuery query = new SolrQuery();
    query.setQuery(queryText);
    QueryResponse rsp = solrServer.query( query );
    SolrDocumentList docList = rsp.getResults();
    assertEquals(1, docList.size());
    SolrDocument song = docList.get(0);
    assertEquals(expectedLinkField, song.get("link"));
}
}

```


Integration

La definizione di test di integrazione qui utilizzata è quella seguente tratta da Growing Object-Oriented Software, guided by tests¹⁰:

We use the term integration tests to refer to the tests that check how some of our code works with code from outside the team that we can't change. It might be a public framework, such as a persistence mapper, or a library from another team within our organization. The distinction is that integration tests make sure that any abstractions we build over third-party code work as we expect. [...] we'll want integration tests to help tease out configuration issues with the external packages, and to give quicker feedback than the (inevitably) slower acceptance tests.

I tipici test di integrazione agiscono su classi che coinvolgono la rete o comunque risorse esterne al grafo di oggetti in memoria come il file system. In questo tipo di test vogliamo svolgere una piccola attività che coinvolge le risorse esterne in modo che il test fallisca o abbia successo in pochi secondi. Lo scopo del test è far sì che quando le nostre classi andranno a comporre le classi sotto integration test, che fanno da confine con l'esterno dell'applicazione, potranno contare su un loro corretto funzionamento. Allo stesso tempo possiamo continuare a “coprire” le nostre classi con test unitari che non coinvolgano risorse esterne e vengano completati in un tempo nell'ordine dei millisecondi.

Ad esempio, la classe YoutubeGrabber ricava un file multimediale a partire dal link alla pagina multimediale in cui è contenuto. Viene quindi testato solo che l'inizio dello stream ottenuto corrisponda a una fixture ottenuta manualmente:

```
/* *****  
 * Copyright (c) 2010 Giorgio Sironi. All rights reserved.  
 * This program and the accompanying materials are made available under  
 * the terms of the Eclipse Public License v1.0 which accompanies this  
 * distribution, and is available at http://www.eclipse.org/legal/epl-v10.html  
 * *****/  
package it.polimi.chansonnier.agent.test;  
  
import java.io.InputStream;  
  
import junit.framework.TestCase;  
  
import it.polimi.chansonnier.agent.YoutubeGrabber;  
import it.polimi.chansonnier.utils.URLUtils;  
import it.polimi.chansonnier.fixtures.Fixtures;  
  
public class YoutubeGrabberTest extends TestCase {  
    private YoutubeGrabber _grabber;  
  
    protected void setUp() throws Exception {  
        _grabber = new YoutubeGrabber();  
    }  
  
    public void testLedZeppelinSStairwayToHeavenIsDownloaded() throws  
Exception {
```

```

        assertFlvStartIsTheSame("http://www.youtube.com/watch?v=BcL---4xQYA", "stairway_to_heaven_start.dat");
    }

    public void testGreenDaySBasketCaseIsDownloaded() throws Exception {
        assertFlvStartIsTheSame("http://www.youtube.com/watch?v=GTwJo0HeNmU", "basketcase_start.dat");
    }

    public void testGreenDaySTimeOfYourLifeIsDownloaded() throws Exception {
        assertFlvStartIsTheSame("http://www.youtube.com/watch?v=IR6uz_VTCUo", "time_of_your_life_start.dat");
    }

    public void assertFlvStartIsTheSame(String pageUrl, String datFile) throws Exception {
        InputStream is = _grabber.getVideo(pageUrl);
        assertEquals(URLUtills.readStart(Fixtures.getAsFile(datFile)), URLUtills.readStart(is));
    }
}

```

I test di integrazione del driver relativo al servizio Google Translate assicurano che il servizio Web sia utilizzato correttamente.

```

package it.polimi.chansonnier.driver.googletranslate.test;

import
it.polimi.chansonnier.driver.googletranslate.GoogleLanguageRecognitionService;
import it.polimi.chansonnier.spi.FuzzyResult;
import junit.framework.TestCase;

public class GoogleLanguageRecognitionServiceTest extends TestCase {
    GoogleLanguageRecognitionService _service;

    protected void setUp() throws Exception {
        _service = new GoogleLanguageRecognitionService();
    }

    public void testRecognizesAnEnglishSentence() {
        FuzzyResult language = _service.getLanguage("This is an example of an English sentence that should be recognized by _service.");
        assertEquals("en", language.getValue());
        assertGreatherThan(language.getConfidence(), 0.9);
    }

    public void testSanitizesASentenceWithLineBreaks() {
        FuzzyResult language = _service.getLanguage("This is an example of an English sentence\n" + "that should be recognized by _service.");
        assertEquals("en", language.getValue());
    }

    public void testRecognizesAnItalianSentence() {

```

```

        FuzzyResult language = _service.getLanguage("Questa Ã una frase
italiana che dovrebbe essere riconosciuta come tale.");
        assertEquals("it", language.getValue());
        assertGreatherThan(language.getConfidence(), 0.9);
    }

    public void testRecognizesAFrenchSentence() {
        FuzzyResult language = _service.getLanguage("Viens, mon beau chat,
sur mon coeur amoureux; "
                                                    + "etiens les griffes de ta
patte,"
                                                    + "Et laisse-moi plonger
dans tes beaux yeux");
        assertEquals("fr", language.getValue());
        assertGreatherThan(language.getConfidence(), 0.4);
    }

    public void testReturnsEmptyStringForAnAmbiguosSentence() {
        FuzzyResult language = _service.getLanguage("Ciao ciao, hello my
friend, allons enfants de la Patrie");
        assertLessThan(language.getConfidence(), 0.5);
    }

    public void testReturnsEmptyStringForATooShortSentence() {
        FuzzyResult language = _service.getLanguage("Ciao");
        assertLessThan(language.getConfidence(), 0.1);
    }

    private void assertGreatherThan(Double confidence, Double reference) {
        assertTrue("Confidence is " + confidence, confidence > reference);
    }

    private void assertLessThan(Double confidence, Double reference) {
        assertTrue("Confidence is " + confidence, confidence < reference);
    }
}

```

Altri test di integrazione controllano la corretta configurazione della macchina su cui gira l'applicazione, come la presenza di un server Solr sulla sua porta standard:

```

package it.polimi.chansonnier.test;

import java.util.ArrayList;
import java.util.Collection;

import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.impl.CommonsHttpSolrServer;
import org.apache.solr.client.solrj.impl.XMLResponseParser;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocumentList;
import org.apache.solr.common.SolrInputDocument;

```

```

import junit.framework.TestCase;

public class SolrIntegrationTest extends TestCase {
    public void testSolrInstanceCanBeUsedForStorageAndSearch() throws
Exception {
        String url = "http://localhost:8983/solr";
        CommonsHttpSolrServer server = new CommonsHttpSolrServer(url);
        server.setParser(new XMLResponseParser());

        server.deleteByQuery( "*:*" );

        SolrInputDocument doc = new SolrInputDocument();
        doc.addField("id", "myId", 1.0f);
        doc.addField("title", "myDummyTitle", 1.0f);
        Collection<SolrInputDocument> docs = new
ArrayList<SolrInputDocument>();
        docs.add(doc);
        server.add(docs);
        server.commit();

        SolrQuery query = new SolrQuery();
        query.setQuery("*:*");
        QueryResponse rsp = server.query(query);
        SolrDocumentList docList = rsp.getResults();
        assertEquals("myDummyTitle", docList.get(0).get("title").toString());
    }
}

```

O ancora, la presenza del server di Selenium RC che renda possibile effettuare gli altri test.

```

package it.polimi.chansonnier.test;

import junit.framework.TestCase;

import com.thoughtworks.selenium.Selenium;

public class SeleniumIntegrationTest extends TestCase {
    WrappableSeleneseTestCase wrapped;
    Selenium selenium;

    public void setUp() throws Exception {
        wrapped = new WrappableSeleneseTestCase();
        wrapped.setUp("http://www.google.com/", "*chrome");
        selenium = wrapped.getSelenium();
    }

    public void tearDown() throws Exception {
        wrapped.tearDown();
    }

    public void testSeleniumRCIsStartedAndWorks() throws Exception {
        selenium.open("/");
        wrapped.verifyTrue(selenium.isTextPresent("Google"));
    }
}

```

Unit

Il design della singola classe è guidato da test a livello più fine, detti test unitari (unit test). I collaboratori (oggetti a cui il componente sotto test ha un riferimento sotto forma di campo o sotto forma di valore passato tramite stack) sono solitamente sostituiti per testare il componente in isolamento e rivelare facilmente una regressione (introduzione di un difetto che fa fallire il test) come appartenente a questa particolare classe. Inoltre, i test a livello unitario sono il più fine strumento di design, in quanto stabiliscono l'Api e il comportamento della classe sotto test.

Unit test in OSGi (Plugin tests)

Il test di servizi implementanti interfacce di SMILA è stato effettuato a un livello di integrazione intermedio (che sarebbe propriamente detto funzionale), fornendo loro servizi collaboratori di SMILA a basso impatto di istanziiazione, ma che comportano la necessità di avviare una piccola sessione del framework OSGi per ogni test case eseguito. Ad esempio, il test dei vari ProcessingService richiede un oggetto Blackboard che fornisca accesso a dei Record finti sui quali si vuole che un certo Processing Service svolga il proprio lavoro. La soluzione compresa nella suite di test di SMILA stesso e scelta anche per Chansonnier utilizza una particolare implementazione denominata TransientBlackboard, che memorizza i record solamente in memoria e non comporta un lento accesso al filesystem con tutte le dipendenze che esso comporta.

In questo caso, sostituire i collaboratori non avrebbe portato particolare giovamento sia per la difficoltà di produrre degli stub con risultati precaricati, data l'ampia Api della Blackboard, sia per l'assenza di ritorno sull'investimento. Infatti i collaboratori, essendo parte di SMILA e non codice parte di Chansonnier, non sono passibili di interventi di design o refactoring, e non cambiano praticamente mai, riducendo a zero la possibilità di introdurre regressioni in Chansonnier per loro responsabilità (sono equiparabili alle classi di java.util). Un punto a favore della sostituzione dei collaboratori è la maggior velocità di test che usano implementazioni leggere, ma tale punto è già “addressed” dalla Transient Blackboard, utilizzata infatti nella suite di test di SMILA stesso.

Un esempio di unit test che coinvolge servizi OSGi è LyricsProcessingServiceTest, che fornisce un'implementazione Stub di LyricsService con risultati predefiniti a LyricsProcessingService, e assicura il parsing di diverse forme del titolo della pagina di un video vengano interpretate correttamente nonostante il rumore contenuto.

```
package it.polimi.chansonnier.core.test;

import org.eclipse.smila.blackboard.BlackboardAccessException;
import org.eclipse.smila.blackboard.path.Path;
import org.eclipse.smila.datamodel.id.Id;

import it.polimi.chansonnier.processing.LyricsProcessingService;
import it.polimi.chansonnier.spi.LyricsService;

public class LyricsProcessingServiceTest extends ProcessingServiceTest
implements LyricsService {
```

```

LyricsProcessingService _lyricsProcessingService;
private static final String LYRICS = "The warden threw a party in the
county jail...";
private static final String TITLE = "Jailhouse Rock";
private static final String ARTIST = "Elvis Presley";

protected void init() throws Exception {
    _lyricsProcessingService = new LyricsProcessingService();
    _lyricsProcessingService.setLyricsService(this);
    _service = _lyricsProcessingService;
    inputAnnotationValue = "myPageTitle";
    outputAnnotationValue = "myLyrics";
}

public void testStoresLyricsAsAnAnnotationStartingFromThePageTitle()
throws Exception {
    final Id id = createNewRecord(ARTIST + " - " + TITLE);

    process(id);

    assertEquals(LYRICS, extractAttribute(id, outputAnnotationValue));
}

public void
testCreatesArtistAndTitleAttributesBasingOnWhichCombinationFindsTheLyrics()
throws Exception {
    final Id id = createNewRecord(ARTIST + " - " + TITLE);

    process(id);

    assertArtistAndTitleAttributesAreExtractedCorrectly(id);
}

public void testStoresLyricsAsAnAnnotationWhenTitleComesBeforeArtist()
throws Exception {
    final Id id = createNewRecord(TITLE + " - " + ARTIST);

    process(id);

    assertEquals(LYRICS, extractAttribute(id, outputAnnotationValue));
}

public void testRemovesParenthetizedNoiseFromPageTitle() throws Exception
{
    final Id id = createNewRecord(ARTIST + " - " + TITLE + " (with
lyrics!)");

    process(id);

    assertArtistAndTitleAttributesAreExtractedCorrectly(id);
}

public void testRemovesParenthetizedNoiseFromPageTitle2() throws Exception
{

```

```

        final Id id = createNewRecord(ARTIST + " - " + TITLE + " (song &
lyrics)");

        process(id);

        assertArtistAndTitleAttributesAreExtractedCorrectly(id);
    }

    public void testRemovesParenthetizedNoiseFromPageTitle3() throws Exception
{
        final Id id = createNewRecord(ARTIST + " - " + TITLE + " [ with
lyrics]");

        process(id);

        assertArtistAndTitleAttributesAreExtractedCorrectly(id);
    }

    public void testRemovesTheWordLyricsFromPageTitle() throws Exception {
        final Id id = createNewRecord(ARTIST + " - " + TITLE + " lyrics");

        process(id);

        assertArtistAndTitleAttributesAreExtractedCorrectly(id);
    }

    public void testRemovesTheWordLyricsFromPageTitleWithoutCaseSensitivity()
throws Exception {
        final Id id = createNewRecord(ARTIST + " - " + TITLE + " Lyrics");

        process(id);

        assertArtistAndTitleAttributesAreExtractedCorrectly(id);
    }

    public void testRemovesTheWordsWithLyricsFromPageTitle() throws Exception
{
        final Id id = createNewRecord(ARTIST + " - " + TITLE + " with
lyrics");

        process(id);

        assertArtistAndTitleAttributesAreExtractedCorrectly(id);
    }

    public void
testRemovesTheWordsWithLyricsFromPageTitleWithoutCaseSensitivity() throws
Exception {
        final Id id = createNewRecord(ARTIST + " - " + TITLE + " With
Lyrics");

        process(id);

        assertArtistAndTitleAttributesAreExtractedCorrectly(id);
    }

```

```

    }

    public void testRemovesAdditionalNonAlphanumericCharactersFromTitle()
throws Exception {
        final Id id = createNewRecord(ARTIST + " - " + TITLE + " ((With
Lyrics)))");

        process(id);

        assertArtistAndTitleAttributesAreExtractedCorrectly(id);
    }

    public void assertArtistAndTitleAttributesAreExtractedCorrectly(Id id)
throws BlackboardAccessException {
        assertEquals(ARTIST, extractAttribute(id, "artist"));
        assertEquals(TITLE, extractAttribute(id, "title"));
    }

    private Id createNewRecord(String pageTitle) throws
BlackboardAccessException {
        final Id id = createBlackboardRecord("youtube",
"http://www.youtube.com/dummy");
        setAttribute(id, new Path(inputAnnotationValue), pageTitle);
        return id;
    }

    private String extractAttribute(Id id, String attributeName) throws
BlackboardAccessException {
        return getAttribute(id, new Path(attributeName)).toString();
    }

    @Override
    public String getLyrics(String title, String artist) {
        if (title.equals("Jailhouse Rock")
&& artist.equals("Elvis Presley")) {
            return "The warden threw a party in the county jail...";
        }
        return null;
    }
}

```

Un altro esempio di unit test relativo ai ProcessingService è LanguageProcessingServiceTest, che assicura la corretta creazione degli attributi relativi al riconoscimento del linguaggio a partire da un collaboratore LanguageRecognitionService.

```

package it.polimi.chansonnier.core.test;

import it.polimi.chansonnier.processing.LanguageProcessingService;
import it.polimi.chansonnier.spi.FuzzyResult;
import it.polimi.chansonnier.spi.LanguageRecognitionService;

import org.eclipse.smila.blackboard.path.Path;
import org.eclipse.smila.datamodel.id.Id;
import org.eclipse.smila.datamodel.record.Literal;

```



```

public class LanguageProcessingServiceTest extends ProcessingServiceTest
implements LanguageRecognitionService {
    LanguageProcessingService _languageProcessingService;
    public static final String LYRICS = "Vamos a la playa...";

    protected void init() throws Exception {
        _languageProcessingService = new LanguageProcessingService();
        _languageProcessingService.setLanguageRecognitionService(this);
        _service = _languageProcessingService;
        inputAnnotationValue = "myLyrics";
        outputAnnotationValue = "myLanguage";
    }

    public void testAddsALanguageAttributeUsingTheLyricsOne() throws Exception
{
        final Id id = createBlackboardRecord("source", "item");
        Path p = new Path(inputAnnotationValue);
        setAttribute(id, p, LYRICS);

        process(id);

        Literal language = getBlackboard().getLiteral(id, new
Path(outputAnnotationValue));
        assertEquals("es", language.getStringValue());
        Literal confidence = getBlackboard().getLiteral(id, new
Path(outputAnnotationValue + "Confidence"));
        assertEquals(0.9, confidence.getFpValue());
    }

    @Override
    public FuzzyResult getLanguage(String textSample) {
        assertEquals(LYRICS, textSample);
        return new FuzzyResult("es", 0.9);
    }
}

```

Unit test senza OSGi (plain old JUnit tests)

Per quanto riguarda gli unit test veri e propri, che istanziano un oggetto della classe sotto test e verificano la produzione di risultati corretti da parte di esso, questi possono essere svolti al di fuori del framework OSGi purchè i bundle esportino i package Java contenenti le classi da testare.

Un bundle di test indipendente può importare quindi le classi facendo affidamento su OSGi solo per la risoluzione del proprio build path. L'istanziamento di un servizio avviene all'interno del codice di test, e presentando tale oggetto un'interfaccia adatta alla Dependency Injection, è semplice introdurre collaboratori appositi senza nessuna dipendenza verso le Api del framework.

Un esempio di unit test che non coinvolge OSGi a runtime è FfmpegFrameExtractionServiceTest, relativo al driver wrapper per ffmpeg utilizzato per estrarre fotogrammi dai video musicali. Questo test è sulla linea di confine fra unit e integration test, in

quanto coinvolge una risorsa esterna al grafo di oggetti (il file system).

```
package it.polimi.chansonnier.driver.ffmpeg.test;

import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;

import junit.framework.TestCase;

import it.polimi.chansonnier.spi.FrameExtractionService;
import it.polimi.chansonnier.driver.ffmpeg.FfmpegFrameExtractionService;
import it.polimi.chansonnier.utils.URLUtils;

public class FfmpegFrameExtractionServiceTest extends TestCase {
    private FrameExtractionService _service;

    protected void setUp() throws Exception {
        _service = new FfmpegFrameExtractionService();
    }

    public void testPngImageIsExtractedFromFlvVideoAtTheGivenSeekTime() throws
Exception {
        File image = _service.getImage(new File("fixtures/desmond.flv"),
"00:00:04");
        InputStream actual = new FileInputStream(image);
        assertEquals(URLUtils.readStart("fixtures/desmond.png"),
URLUtils.readStart(actual));
    }
}
```

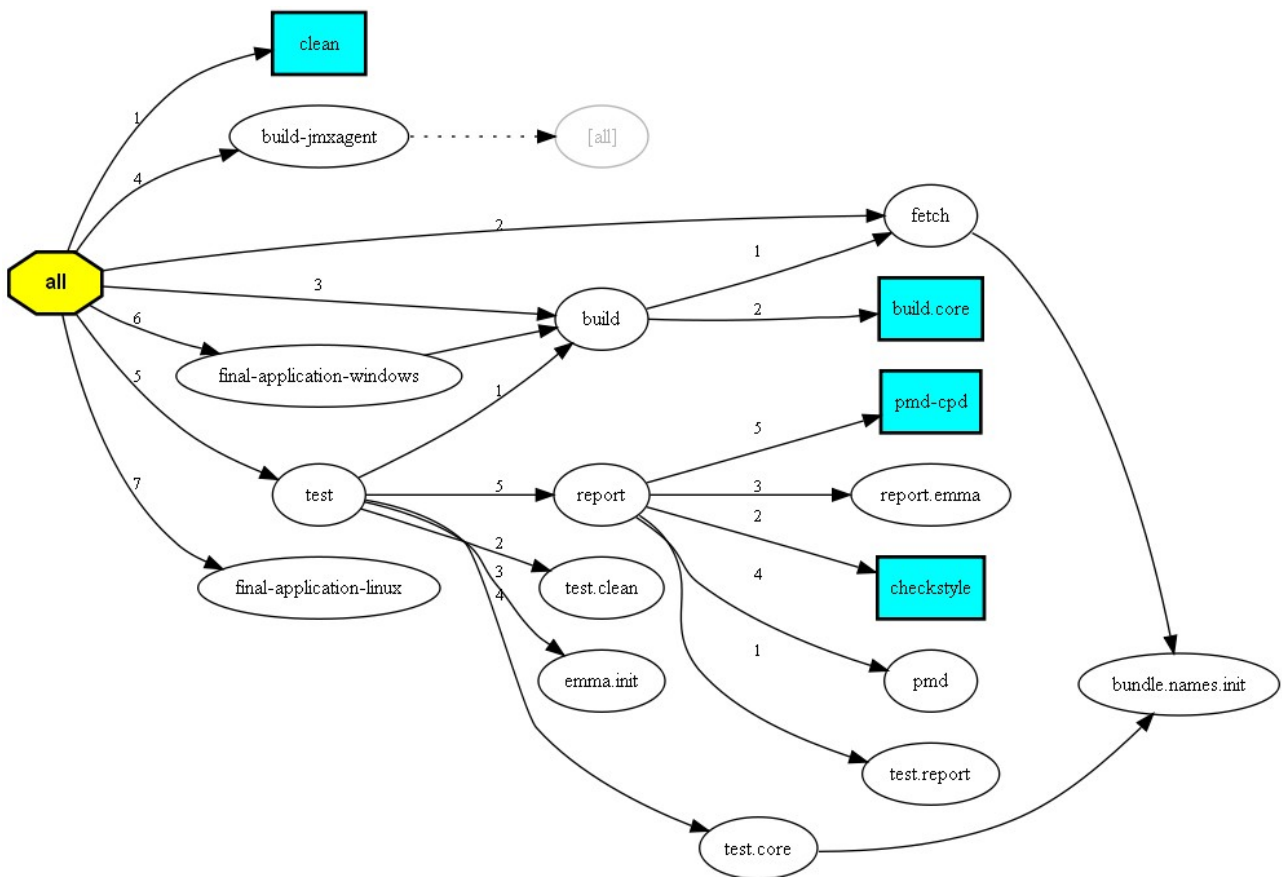
5.1.3 Automazione

Build

La produzione di un pacchetto installabile di Chansonnier è un processo automatico (push-button automation), che ricalca la struttura del processo di build di SMILA.

Un build file per Ant stabilisce i vari passi di questo processo, che sfrutta l'infrastruttura di Equinox per compilare il codice necessario, mentre un insieme di metadati presenti nei bundle definisce parametri quali l'elenco di bundle da integrare, e quali bundle di test devono essere eseguiti.

Il build file presenta diversi target, eseguibili in isolamento (ma con le dovute dipendenze) o come parte di un processo ex-novo:



I target più importanti, eseguiti in sequenza per default, sono:

- **clean**: prepara una cartella eclipse.build vuota, eliminando la precedente build se necessario.
- **fetch**: risolve le dipendenze fra i vari bundle segnalando eventuali problemi, e trasporta il loro codice all'interno di eclipse.build seguendo le informazioni dei loro build.properties
- **build**: compila il codice e produce i vari bundle sotto forma di Jar.
- **test**: per ogni bundle di test da eseguire, effettua il deployment di un'istanza dell'applicazione in una sottodirectory eclipse.test; successivamente esegue i test case e salva i report in formato XML.
- **final-application-linux**: se la test suite non ha prodotto errori, produce un archivio zip a partire dai bundle compilati.

Sebbene la complessità di un'applicazione OSGi e le caratteristiche di Java come linguaggio rendano fondamentale l'utilizzo di un'IDE come Eclipse per lo sviluppo, l'integrazione dei vari componenti e l'esecuzione dei loro test di regressione può avvenire anche su un server senza interfaccia grafica grazie all'automazione del processo di build.

Comunemente infatti, questo processo viene eseguito su una macchina Ubuntu Server dedicata allo scopo e che può dedicare le proprie risorse di Cpu e memoria a test di accettazione.

Il passo successivo di questa metodologia è un server di Continuous Integration, che compia

lo stesso lavoro ma in modo ciclico, estraendo la code base da testare direttamente da un repository di controllo versione (nel caso di Chansonnier il repository Git locale o il suo mirror su GitHub) a intervalli regolati. Grazie alla Continuous Integration changeset conflittuali provenienti da differenti sviluppatori possono essere scoperti il prima possibile (in questo caso “the earlier the better”).

Test suite

L'automazione dei test è una pratica fondamentale perché sia possibile eseguirli con regolarità e a costo zero, allo scopo di avere feedback sullo stato delle funzionalità dell'applicazione.

I test a livello fine come quelli unitari e funzionali sono relativamente semplici da mettere in atto, in quanto coinvolgono un grafo di oggetti che alla fine del test sarà semplicemente garbage collected. I test di integrazione e accettazione invece richiedono di gestire in automatico anche lo stato di risorse esterne quali l'istanza di Solr o di Selenium RC stesso perché un test possa essere eseguito.

Il ciclo di vita dei due demoni sta al di fuori della suite di test, in quanto è ragionevole ipotesi che vengano avviati allo startup della macchina contenente Chansonnier, sia essa di sviluppo o di produzione.

L'istanza di Solr utilizzata contiene l'indice di tutti i dati relativi alle canzoni processate che non siano già cancellati dal processo di build. Infatti all'inizio di un test di accettazione o funzionale gli storage sono di norma vuoti in quanto i task Ant di Equinox si preoccupano di effettuare il deployment di una nuova istanza dell'applicazione OSGi per ogni bundle di test da eseguire.

Per quanto riguarda l'esecuzione durante lo sviluppo, all'interno di Eclipse, allo stesso modo l'esecuzione di test case come JUnit Plugin Test comporta il deployment di un nuovo workspace e cartella di configurazione che fa da sandbox per il test e che può essere cancellato in tranquillità alla fine dell'esecuzione. I vari bundle non sono invece “redeployed” in quanto non contenenti stato, e il loro codice viene eseguito direttamente dalle loro cartelle bin/.

Solr

Come già stabilito precedentemente, l'istanza di Solr viene resettata ad ogni esecuzione di un test. Si utilizza SolrJ per tale scopo, eseguendo una query di delete su *:*(wildcard per indicare tutti i documenti presenti) e lasciando disponibile l'istanza dell'oggetto SolrServer creato ai test case perché possano usarlo per asserzioni.

Selenium

Dato che Selenium RC non contiene dati in sé e non ha quindi uno stato prestabilito a cui essere riportato all'inizio dell'esecuzione di un test, l'unico nodo rimanente è relativo al controllo che sia in esecuzione. Il setUp() dei test di accettazione crea quindi un'istanza interna di SeleneseTestCase, un'operazione che fallisce subito se Selenium RC non è in esecuzione per un qualche motivo. Si evita così di eseguire test di accettazione che coinvolgono dispendiosi download solo per poi fallire una volta passati alla fase di controllo dei risultati tramite l'interfaccia Web.

5.2 Pattern

5.2.1 Dependency Injection

La pratica di effettuare Dependency Injection dei collaboratori all'interno dei vari oggetti Java è fondamentale nel mantenere al minimo le dipendenze di ogni classe e la facilità di scrittura dei test. Dependency Injection è una forma di Inversion of Control, nella quale nessun componente dipende su altri componenti concreti, ma al contrario sia esso che il collaboratore da lui richiesto dipendono a tempo di compilazione da un'interfaccia; il primo avrà un campo di tipo uguale all'interfaccia, mentre il secondo ne sarà un'implementazione.

Ogni Bridge pattern precedentemente illustrato utilizza una qualche forma di Dependency Injection per inserire il driver specifico nell'oggetto facente parte dell'API. In generale, ogni oggetto collaboratore che viene iniettato anziché creato costituisce un seam utile per il testing in isolamento della classe principale. Al contrario, ogni oggetto creato all'interno di una ipotetica classe sotto test non può essere sostituito da un'implementazione stub specifica per il test, e quindi porterà la propria logica, e in alcuni casi la propria “pesantezza”, all'interno del test case.

Seguendo una tecnica di Dependency Injection, ogni collaboratore necessario ad una classe per funzionare non viene creato da essa, ma richiesto tramite la signature del costruttore o di un setter specifico. Idealmente nessun operatore new è utilizzato per classi contenenti business logic, poiché la creazione del grafo di oggetti è un aspetto separato dall'esecuzione di logica al loro interno. Utilizzare Dependency Injection comporta l'eliminazione dei pattern Singleton per favorire la costruzione del grafo di oggetti a più alto livello (ad esempio tramite configurazione o tramite una Factory).

Fra servizi OSGi

Per quanto riguarda i servizi OSGi, la Dependency Injection di altri collaboratori è incoraggiata e supportata senza introdurre nessuna dipendenza verso il particolare framework OSGi e nemmeno sulle interfacce OSGi standard. I servizi OSGi sono Plain Old Java Object, e richiedono altri POJO come collaboratori, con tutti i vantaggi del caso. Ad esempio le classi contenute nei bundle di Chansonnier e SMILA sono compilabili anche al di fuori dall'ambiente OSGi, una volta definito un classpath adeguato.

Il ciclo di vita dei servizi è quindi regolato dal framework OSGi, che si occupa di istanziare i servizi definiti e di passare loro i collaboratori necessari, prendendosi carico anche di crearli se non sono correntemente attivi.

```
<component xmlns="http://www.osgi.org/xmlns/scr/v1.1.0"
name="EmotionRecognitionService" immediate="true">
  <implementation
class="it.polimi.chansonnier.driver.synesketch.SynesketchEmotionRecognitionService" />
  <service>
    <provide interface="it.polimi.chansonnier.spi.EmotionRecognitionService"/>
```

```
</service>  
</component>
```

I servizi attivi sono solitamente implementazioni dell'SPI di SMILA, e costituiscono i punti di hook dove inserire le proprie funzionalità all'interno dei vari componenti connectivity o processing.

All'interno di un bundle

Alcuni collaboratori sono interni allo stesso bundle della classe richiedente, e non sono dichiarati come servizi OSGi ma incapsulati all'interno di "esso". Normalmente quindi verrebbero iniettati al momento della creazione dell'oggetto.

La soluzione adottata quindi è di fornire due costruttori, uno per la creazione di un oggetto nei test che accetti dei collaboratori come argomento, ed un altro vuoto conforme alle richieste di OSGi per i servizi che deve gestire senza ulteriore configurazione.

Un'alternativa è quella di definire una ComponentFactory come servizio aggiuntivo, così che la creazione di nuovi oggetti del servizio in questione sia gestita programmaticamente (da un metodo di tale ComponentFactory).¹¹ Questa soluzione comporta l'utilizzo di una classe e descrittore in più, ma rimuove le dipendenze di compilazione dal servizio principale verso i suoi collaboratori: il codice fa riferimento solo alle interfacce dei collaboratori, dovendo solamente maneggiarne delle istanze, e non crearle da zero.

5.2.2 Bridge

Ogni Processing Service è realizzato essenzialmente tramite l'implementazione di un Bridge Pattern. Infatti vi sono due responsabilità per ogni Processing Service che è utile separare in due classi indipendenti.

La prima responsabilità è relativa alla gestione di strutture dati proprie di SMILA e comporta una dipendenza su di esso e sul corrente modello dei dati. Ad esempio estrarre un valore sotto forma di stringa o intero dalla Blackboard o, al contrario, inserirlo è un esempio di questa responsabilità. Nella maggior parte dei casi, la classe implementante questa responsabilità espone un'API corrispondente all'implementazione dell'interfaccia `org.eclipse.smila.processing.api.ProcessingService`.

La seconda responsabilità riguarda invece la business logic vera e propria del servizio, sia essa scaricare il testo di una canzone o riconoscerne la lingua. Questa logica è eseguita su strutture dati native (String, URL) o al più definite da un modello del dominio in questione, ma di certo non riguardante SMILA. La terminologia più corretta per questo secondo tipo di classi è quella di driver.

Le interfacce implementate da questo tipo di classi sono il punto di congiunzione fra i ProcessingService e loro driver e fanno parte della Service Provider Interface di Chansonnier. (package `it.polimi.chansonnier.spi`)

Separare quindi questi due aspetti per fare sì che possano cambiare indipendentemente è proprio l'intento del Bridge pattern. Si veda il seguente esempio relativo a

FrameExtractionProcessingService, il quale estrae

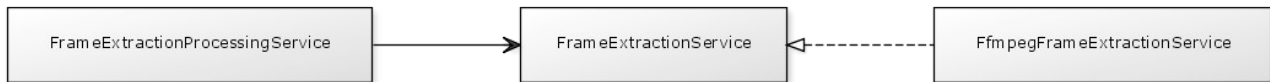


Illustrazione 7: Implementazione del Bridge pattern per FrameExtractionProcessingService

OSGi permette poi di definire FrameExtractionProcessingService come servizio avviato di default, e quindi disponibile per le pipeline di indicizzazione, tramite un descrittore XML. Inoltre tali metadati definiscono anche la sua dipendenza verso un driver FrameExtractionService:

```
<component xmlns="http://www.osgi.org/xmlns/scr/v1.1.0"
name="FrameExtractionProcessingService" immediate="true">
  <implementation
class="it.polimi.chansonnier.processing.FrameExtractionProcessingService" />
  <reference
    name="FrameExtractionService"
    interface="it.polimi.chansonnier.spi.FrameExtractionService"
    policy="static"
    cardinality="1..1"
    bind="setFrameExtractionService"
    unbind="unsetFrameExtractionService"/>
  <service>
    <provide interface="org.eclipse.smila.processing.ProcessingService"/>
  </service>
  <property name="smila.processing.service.name"
value="FrameExtractionProcessingService"/>
</component>
```

Allo stesso modo l'oggetto rappresentante il wrapper di Ffmpeg, che implementa FrameExtractionService, viene gestito tramite un descrittore dello stesso tipo presente nel proprio bundle:

```
<component xmlns="http://www.osgi.org/xmlns/scr/v1.1.0"
name="FrameExtractionService" immediate="true">
  <implementation
class="it.polimi.chansonnier.driver.ffmpeg.FfmpegFrameExtractionService" />
  <service>
    <provide interface="it.polimi.chansonnier.spi.FrameExtractionService"/>
  </service>
</component>
```

Grazie a questa modellizzazione ad esempio, si può sostituire l'implementazione tramite Ffmpeg cambiando solamente la "parte destra" del diagramma UML, oppure si può riciclare in progetti diversi tale componente (a destra) senza trascinare in tali progetti dipendenze su SMILA. In generale il primo caso è il più interessante per Chansonnier, in quanto sostituendo il bundle it.polimi.chansonnier.driver.ffmpeg con uno analogo che fornisca un'implementazione di FrameExtractionService, si cambia il back end effettivamente utilizzato per estrarre i frame dal video.

L'ulteriore livello di astrazione introdotto da classi Java al di sopra di servizi Web o eseguibili esterni è un vantaggio rispetto al modello più veloce di integrazione di servizi Web in SMILA, ovvero la loro diretta inclusione nella pipeline tramite codice BPEL. Mentre infatti la logica presente nella pipeline è testabile solo a livello end-to-end e non può essere separata dal servizio

Web reale, avere un livello di isolamento permette di cambiare tutti i back end sia a scopo di ottenere test veloci e isolati, sia di effettuare manutenzione del codice o sperimentare nuove implementazioni.

5.3 Tecnologie

5.3.1 OSGi

OSGi è un sistema di moduli dinamici per Java, sui quali sia SMILA che Chansonnier si basano per la propria architettura. OSGi è definito da una specifica che può essere soddisfatta da diversi framework OSGi presenti sul mercato commerciale e open source.¹² Per quanto riguarda Chansonnier e SMILA si farà sempre riferimento ad Equinox, il framework OSGi open source supportato dalla Eclipse Foundation sul quale SMILA, e quindi Chansonnier, girano.

Un'applicazione OSGi è composta da un insieme di bundle, ossia di JAR annotati con particolari metadati adatti a gestire "processi" come la risoluzione di dipendenze fra di essi e il versionamento delle varie classi in esso contenute.

Un bundle OSGi può avere ad esempio la seguente struttura:

```
it/      (1)
META-INF/
  MANIFEST.MF      (2)
OSGI-INF/      (3)
  ServiceDescriptor.xml      (4)
```

1. file .class organizzati secondo Reverse Domain Hierarchy; niente di diverso da un JAR
2. manifesto del bundle contenente i principali metadati, quali la lista dei package Java esportati e la loro versione, oppure la lista di package Java sulle quali questo bundle dipende.
3. cartella contenente dichiarazioni di servizi OSGi cosicchè il framework che ospita i bundle possa gestirne il ciclo di vita.
4. descrittore di un servizio OSGi

Un'applicazione definisce anche quali bundle devono essere attivati al proprio avvio, cosicchè i servizi in essi contenuti siano anch'essi "attivati". La comunicazione fra bundle si basa proprio su servizi esposti da essi, ed in generale OSGi definisce un architettura basata su servizi, ognuno con un ciclo di vita indipendente.

I servizi OSGi sono Plain Old Java Objects, classi definite dallo sviluppatore che non hanno dipendenze verso il framework OSGi o la specifica stessa. OSGi gestisce tramite reflection la loro istanziazione e passaggio di collaboratori ("dipendenze"), basandosi sui service descriptor.

5.3.2 BPEL

Web Services Business Process Execution Language (WS-BPEL, e semplicemente BPEL nel resto di questo documento) è un linguaggio di orchestrazione basato su XML, il cui scopo è coordinare diversi attori nell'esecuzione di un processo.

All'interno di Chansonnier BPEL è utilizzato per definire il processo di annotazione di un record (la singola canzone), configurando quindi la catena o albero di annotatori con parametri quale l'ordine della loro esecuzione, gli attributi da usare in input e gli attributi da produrre in quanto utilizzati dagli annotatori a valle o richiesti dall'indice Solr.

La pipeline di aggiunta di un record è quindi il cuore della configurazione di Chansonnier, ed è un'istanza di un executable business process. Si noti che operazioni non propriamente di processing come l'aggiornamento dell'indice Solr e della lista delle ultime canzoni indicizzate sono gestibili comunque da un'unico punto.

```
<?xml version="1.0" encoding="utf-8" ?>
<process name="AddPipeline"
targetNamespace="http://www.eclipse.org/smila/processor"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:proc="http://www.eclipse.org/smila/processor"
xmlns:rec="http://www.eclipse.org/smila/record">

  <import location="processor.wsdl"
namespace="http://www.eclipse.org/smila/processor"
  importType="http://schemas.xmlsoap.org/wsdl/" />

  <partnerLinks>
    <partnerLink name="Pipeline"
partnerLinkType="proc:ProcessorPartnerLinkType" myRole="service" />
  </partnerLinks>

  <extensions>
    <extension namespace="http://www.eclipse.org/smila/processor"
mustUnderstand="no" />
  </extensions>

  <variables>
    <variable name="request" messageType="proc:ProcessorMessage" />
  </variables>

  <sequence>
    <receive name="start" partnerLink="Pipeline"
portType="proc:ProcessorPortType" operation="process" variable="request"
      createInstance="yes" />

    <extensionActivity name="invokeLyricsService">
      <proc:invokeService>
        <proc:service name="LyricsProcessingService" />
        <proc:variables input="request" output="request" />
        <proc:setAnnotations>
```

```

        <rec:An n="it.polimi.chansonnier.processing.Input">
            <rec:V>pageTitle</rec:V>
        </rec:An>
        <rec:An
n="it.polimi.chansonnier.processing.Output">
            <rec:V>lyrics</rec:V>
        </rec:An>
    </proc:setAnnotations>
</proc:invokeService>
</extensionActivity>

<extensionActivity name="invokeEmotionProcessingService">
    <proc:invokeService>
        <proc:service name="EmotionProcessingService" />
        <proc:variables input="request" output="request" />
        <proc:setAnnotations>
            <rec:An
n="it.polimi.chansonnier.processing.Input">
                <rec:V>lyrics</rec:V>
            </rec:An>
            <rec:An
n="it.polimi.chansonnier.processing.Output">
                <rec:V>emotion</rec:V>
            </rec:An>
        </proc:setAnnotations>
    </proc:invokeService>
</extensionActivity>

<extensionActivity name="invokeLanguageProcessingService">
    <proc:invokeService>
        <proc:service name="LanguageProcessingService" />
        <proc:variables input="request" output="request" />
        <proc:setAnnotations>
            <rec:An
n="it.polimi.chansonnier.processing.Input">
                <rec:V>lyrics</rec:V>
            </rec:An>
            <rec:An
n="it.polimi.chansonnier.processing.Output">
                <rec:V>language</rec:V>
            </rec:An>
        </proc:setAnnotations>
    </proc:invokeService>
</extensionActivity>

<extensionActivity name="invokeFrameExtractionProcessingService">
    <proc:invokeService>
        <proc:service name="FrameExtractionProcessingService" />
        <proc:variables input="request" output="request" />
        <proc:setAnnotations>
            <rec:An
n="it.polimi.chansonnier.processing.Input">
                <rec:V>original</rec:V>
            </rec:An>

```

```

        <rec:An
n="it.polimi.chansonnier.processing.Output">
        <rec:V>image</rec:V>
        </rec:An>
    </proc:setAnnotations>
</proc:invokeService>
</extensionActivity>

    <extensionActivity name="invokeSolrPipelet">
        <proc:invokePipelet>
            <proc:pipelet
class="org.eclipse.smila.integration.solr.SolrPipelet" />
            <proc:variables input="request" output="request"
/>

            <proc:PipeletConfiguration>
                <proc:Property name="executionMode">
                    <proc:Value>ADD</proc:Value>
                </proc:Property>
                <proc:Property name="allowDoublets"
type="java.lang.Boolean">

                    <proc:Value>>false</proc:Value>
                </proc:Property>
                <proc:Property name="commitWithin"
type="java.lang.Integer">

                    <proc:Value>10000</proc:Value>
                </proc:Property>
                <proc:Property name="indexName">
                    <proc:Value>chansonnier</proc:Value>
                </proc:Property>
            </proc:PipeletConfiguration>
        </proc:invokePipelet>
    </extensionActivity>

    <extensionActivity name="invokeLastIndexedService">
        <proc:invokeService>
            <proc:service name="LastIndexedService" />
            <proc:variables input="request" output="request" />
        </proc:invokeService>
    </extensionActivity>

    <reply name="end" partnerLink="Pipeline"
portType="proc:ProcessorPortType" operation="process" variable="request" />
    <exit />
</sequence>
</process>

```

5.4 Interfaccia

5.4.1 Utente

SMILA ha un bundle già presente al suo interno che incapsula Tomcat, un Servlet Container

adatto quindi a servire Servlet e JSP. È stato quindi utilizzato questo approccio per costruire un'interfaccia Web portatile e automaticamente "remotizzabile".

L'interfaccia è quindi una Web application composta dalle seguenti parti:

```
configuration/org.eclipse.smila.tomcat/web-apps/chansonnier (1)
  js/          (2)
  style.css    (2)
  index.html
  ...
  WEB-INF
    web.xml (3)
Bundle it.polimi.chansonnier.servlet (4)
```

1. è la cartella di configurazione contenente l'equivalente di un WAR: ad esempio i file jsp e il descrittore web.xml sono all'interno di questa cartella.
2. sono file statici che vengono semplicemente ospitati dal server.
3. è il descrittore dell'applicazione che contiene il mapping delle servlet e altre configurazioni.

Il codice java precompilato (principalmente servlet) è incapsulato all'interno di un bundle (4) che esporta il package `it.polimi.chansonnier.servlet`, avendo cura di usare alcuni header specifici per Equinox per far sì che il class loader del bundle `org.eclipse.smila.tomcat` possa effettivamente vedere le servlet:

```
Eclipse-RegisterBuddy: org.apache.tomcat, org.eclipse.smila.tomcat
```

Un altro problema da risolvere nell'implementare un'interfaccia Web è stato l'introdurre riferimenti a servizi OSGi all'interno delle servlet (che hanno costruttore vuoto per definizione).

La soluzione adottata prevede di definire un Activator nel bundle delle servlet, che venga quindi istanziato e chiamato all'attivazione del bundle e quindi allo startup di Chansonnier. Tale Activator definisce alcuni ServiceTracker come membri statici della propria classe, e li rende accessibili con metodi pubblici altrettanto statici. Ogni ServiceTracker è rintraccia un differente servizio OSGi fra i bundle di SMILA e di Chansonnier, e può quindi essere richiamato dalla servlet tramite i metodi statici.

L'interfaccia utilizzata da un utente è quindi un qualsiasi browser.

Servlet

Le servlet presenti in Chansonnier hanno lo scopo, come descritto precedentemente, di interfacciarsi con SMILA e i componenti custom che sono stati integrati in esso. Si tratta quindi di use case che comprendono l'accesso allo storage, o l'"accodamento" di un file multimediale per il processo di annotazione:

- **AddServlet**: mostra un form html quando riceve richieste di tipo GET, e il aggiunge alla coda del LinkGrabberAgent il parametro link delle richieste POST, mostrando un messaggio di successo. L'operazione di processing è appunto asincrona e svolta con una coda, dato il tempo necessario per lo scaricamento dei file multimediali e la loro analisi.

- **LastServlet**: mostra la lista delle ultimi canzoni la cui indicizzazione è terminata con successo. È quindi un checkpoint cruciale per i test manuali o automatici, che possono semplicemente controllare in questa pagina la presenza della canzone che avevano aggiunto alla coda tramite il form.
- **AttachmentServlet**: interfaccia i client Web con il **BinaryStorage** tramite i parametri **id** (del record, cioè della canzone considerata) e **name** (del particolare attachment), restituendo in formato binario l'attachment. È usata ad esempio per mostrare i fotogrammi estratti dal video, conservati all'interno del **BinaryStorage**.

Ajax

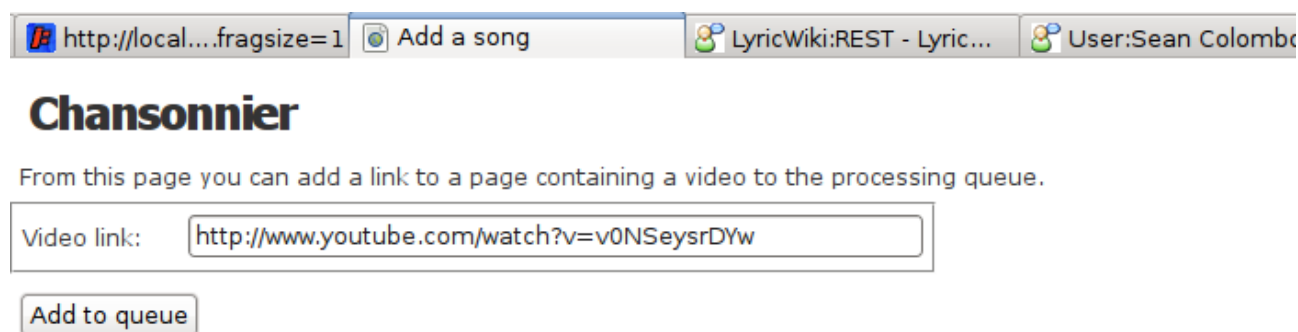
L'interfaccia di ricerca non fa utilizzo di servlet, né di SMILA stesso, escludendo il fatto che è l'istanza di Tomcat presente all'interno dell'applicazione OSGi a fare da host per i file dell'interfaccia utente.

L'interfaccia di ricerca ha quindi un approccio rich client, e consiste in file HTML, CSS e JavaScript. L'esecuzione della generazione di HTML avviene direttamente sul client, in quanto è un'accoppiata di libreria JavaScript ad effettuare richieste HTTP direttamente a Solr tramite la sua interfaccia REST, ricevendo risultati in formato JSON.

Le due librerie in questione sono **Ajax Solr** che fa da wrapper alla REST Api del server Solr, e **JQuery** che ha la funzione di framework JavaScript generico. JQuery gestisce quindi il comportamento dell'interfaccia, facendo da wrapper al Document Object Model e accettando routine da sottoscrivere a determinati eventi come il caricamento del documento.

5.4.2 Esempio di utilizzo

Il processo di discovery di nuove canzoni è guidato dall'utente, in quanto non è stato fattibile filtrare autonomamente i video del portale YouTube per identificarne quelli contenenti canzoni. Tramite l'interfaccia Web, è possibile aggiungere un link da sottoporre a Chansonnier:



The screenshot shows a web browser window with several tabs: 'http://local...fragsize=1', 'Add a song', 'LyricWiki:REST - Lyric...', and 'User:Sean Colomb'. The main content area has the title 'Chansonnier' in a large, bold, black font. Below the title, a message reads: 'From this page you can add a link to a page containing a video to the processing queue.' There is a text input field labeled 'Video link:' containing the URL 'http://www.youtube.com/watch?v=v0NSEysrDYw'. Below the input field is a button labeled 'Add to queue'.

Aggiunto il link, l'utente viene avvertito che l'indicizzazione richiede alcuni minuti.

Chansonnier

The link <http://www.youtube.com/watch?v=v0NSeysrDYw> was added to the queue.

You may want to monitor the [last indexed songs page](#) to discover when its processing phase has been comple

Una volta completata l'analisi e l'indicizzazione, la canzone compare nella pagine delle ultime canzoni indicizzate:

Chansonnier

This is the list of the last songs added to the index.

The last song added was *Love Generation*

All the songs added during this session are listed.

<http://www.youtube.com/watch?v=Ozp8rIdpDAU>

<http://www.youtube.com/watch?v=v0NSeysrDYw>

Da questo momento la canzone è disponibile per la ricerca.

L'interfaccia Web di Chansonnier presenta quindi l'elenco delle canzoni indicizzate e dei possibili filtri da applicare per effettuare una ricerca.


Chansonnier

Chansonnier

Current Selection
Viewing all songs!

Search

Emotions
[anger](#) [happiness](#) [surprise](#)

Languages


Artists
[883](#) [Beyonce](#) [Enrique Iglesias](#)
[Eros Ramazzotti](#) [Lucio Battisti](#) [The Killers](#)
[U2](#)

< 1 2 3 > displaying 1 to 3 of 9 results

Enrique Iglesias - Hero
 Let me be your hero. "(Whisper)" Would you dance if I asked you to dance? Would you run and never look back? Would you cry if you saw me cryin'? And would you [...]
[anger en Enrique Iglesias](#)

Beyonce - Halo
 Remember those walls I built? Well, baby, they're tumbling down And they didn't even put up a fight They didn't even make a sound I found a way to let you in But I never really had a doubt Standing in the light of your halo I got my angel now It's like I've been awake[...]
[surprise en Beyonce](#)

Eros Ramazzotti - Nuestra vida
 Toda mi historia he contado ahora yo me detendré archivaré mi pasado mi porvenir viviré Lo defenderé con la fuerza que siento en mi un sueño[...]
[es Eros Ramazzotti](#)

La ricerca basata su facet ad esempio permette di filtrare le canzoni presentate su di un attributo delle tag cloud, come ad esempio l'artista. La grandezza degli elementi della tag cloud è

correlata con il numero di canzoni che soddisfano il facet in questione.

Chansonnier

Chansonnier



Current Selection

(x) artist:883

Search

Emotions

Languages



Artists




883

< 1 > displaying 1 to 2 of 2 results

883 - nord sud ovest est

Ma perchè sei andata via Mi son persa nella notte perchè non m'hai detto che non eri mia Non lo so sarà il vento o sarai tu la voce che risponde ai miei perchè Dai galoppa più che puoi Corri vai non ti fermare che di strada ce [...]




it 883



883 - Quien seras

Las noches no se acabarán en la calle vacia, las llevo a casa junto a mi, haré una melodía. Escribo yo kilómetros de cartas solo para ti esper[...]

es 883



I facet presentati all'utente sono calcolati sull'insieme di canzoni corrente. Come si vede nella precedente immagine, essendo gli 883 un gruppo che scrisse canzoni solo in italiano e spagnolo, quando vengono presentate le loro canzoni solo queste due lingue sono disponibili per un ulteriore raffinamento. Infatti diversi facet possono essere combinati per restringere la ricerca:

Chansonnier

Chansonnier


Current Selection

(x) artist:883
(x) language:it

Search

Emotions

Languages



Artists




883

< 1 > displaying 1 to 1 of 1 results

883 - nord sud ovest est

Ma perchè sei andata via Mi son persa nella notte perchè non m'hai detto che non eri mia Non lo so sarà il vento o sarai tu la voce che risponde ai miei perchè Dai galoppa più che puoi Corri vai non ti fermare che di strada ce [...]

it 883



Ogni canzone viene presentata con alcuni fotogrammi estratti dal video da cui proviene, allo scopo di facilitare la sua identificazione:

Chansonnier

Chansonnier

Current Selection


(x) artist:883

(x) language:it

Search


Emotions

Languages



Artists

883



la notte perchè non m'hai detto che non eri
 ce che risponde ai miei perchè Dai galoppa
 la strada ce [...]

Per le canzoni in lingua inglese, viene effettuata anche un'analisi emozionale. In questo esempio vengono selezionate le canzoni che esprimono felicità:

Chansonnier

Chansonnier

Current Selection


(x) emotion:happiness

Search

Emotions

happiness

Languages



Artists


The Killers U2

< 1 > displaying 1 to 2 of 2 results

The Killers - Human

I did my best to notice, when the call came down the line Up to the platform of
 surrender, I was brought but I was kind And sometimes I get nervous, when I see
 an open door Clos[...]


happiness en The Killers



U2 - Beautiful Day

The heart is a bloom, shoots up through the stony ground There's no room, no space
 to rent in this town You're out of luck and the reason that you had to care, The traffic
 is stuck and you're not [...]

happiness en U2



I facet sono presenti anche all'interno del singolo risultato; selezionando per esempio un artista dai tag di una sua canzone (anziché dalle tag cloud), all'elenco di canzoni verrà applicato il filtro relativo:

Chansonnier

Chansonnier

Current Selection

(x) emotion:happiness


(x) artist:U2

Search

Emotions

happiness

Languages



Artists


U2

< 1 > displaying 1 to 1 of 1 results

U2 - Beautiful Day

The heart is a bloom, shoots up through the stony ground There's no room, no space to rent in this town You're out of luck and the reason that you had to care, The traffic is stuck and you're not [...]

happiness en U2



È possibile anche effettuare una ricerca full text sui vari campi dell'indice, se non si conosce a priori quale facet utilizzare:

Chansonnier

Chansonnier


Current Selection

(x) fullText:"gold"

Search

Emotions

Languages



Artists

Led Zeppelin


< 1 > displaying 1 to 1 of 1 results

Led Zeppelin - Stairway to Heaven

There's a lady who's ... [read more...](#)

en Led Zeppelin

Match: ... that glitters is **gold** And she's buying...



La parte di attributo che soddisfa la query viene presentata in evidenza nel risultato.

Diversi campi sono aggiunti all'indice full text, in modo che la stessa query di ricerca sia applicabile contemporaneamente ad artista, testo e titolo della canzone. In questo esempio la parola *sound* è presente sia nel testo della canzone di Beyonce, sia nel titolo di quella dei Coldplay.


+

Chansonnier

Current Selection
(x) fullText:"sound"

Search


Emotions
happiness surprise

Languages



Artists
Beyonce Coldplay

< 1 > displaying 1 to 2 of 2 results

Beyonce - Halo
Remember those walls I ... [read more...](#)
[surprise en Beyonce](#)
Match: ... They didn't even make a **sound** I found...



Coldplay - Speed Of Sound
How long before I get in?... [read more...](#)
[happiness en Coldplay](#)
Match: ...Speed Of **Sound**...



Si noti infine che è possibile applicare i filtri contemporaneamente alla ricerca full text, in modo ortogonale.


+

Chansonnier

Current Selection
(x) fullText:"sound"
(x) artist:Coldplay

Search


Emotions
happiness

Languages


Artists
Coldplay

< 1 > displaying 1 to 1 of 1 results

Coldplay - Speed Of Sound
How long before I get in?... [read more...](#)
[happiness en Coldplay](#)
Match: ...Speed Of **Sound**...



5.4.3 Api

È possibile anche specificare XML o JSON come formato dei risultati di una ricerca, utilizzando direttamente l'interfaccia REST di Solr. I dati restituiti saranno in una forma simile a:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<response>

<lst name="responseHeader">
  <int name="status">0</int>
  <int name="QTime">23</int>
  <lst name="params">
    <str name="indent">on</str>
    <str name="start">0</str>
    <str name="q">Title:*</str>
    <str name="version">2.2</str>
    <str name="rows">10</str>
  </lst>
</lst>
<result name="response" numFound="1" start="0">
  <doc>
    <str name="artist">U2</str>
    <str name="emotion">happiness</str>
    <str name="emotionConfidence">0.06818181818181818</str>
    <str name="lyrics">The heart is a bloom, shoots up through the stony ground
There's no room, no space to rent in this town
You're out of luck and the reason that you had to care,
The traffic is stuck and you're not [...]</str>
    <str name="pageTitle">U2 - Beautiful Day (with Lyrics)</str>
    <str name="title">Beautiful Day</str>
    <str name="id">%3Cid+version%3D%221.0%22+xmlns%3D%22http%3A%2F
%2Fwww.eclipse.org%2Fsmila%2Fid%22%3E%3CSource%3Eyoutube%3C%2FSource%3E%3CKey
%3Ehttp%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3De8w7f0ShtIM%3C%2FKey%3E%3C%2Fid
%3E</str>
  </doc>
</result>
</response>

```

O nella corrispondente rappresentazione in JSON:

```

{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "indent": "on",
      "start": "0",
      "q": "Title:*",
      "wt": "json",
      "version": "2.2",
      "rows": "10"
    }
  },
  "response": { "numFound": 1, "start": 0, "docs": [
    {
      "id": "%3Cid+version%3D%221.0%22+xmlns%3D%22http%3A%2F%2Fwww.eclipse.org
%2Fsmila%2Fid%22%3E%3CSource%3Eyoutube%3C%2FSource%3E%3CKey%3Ehttp%3A%2F
%2Fwww.youtube.com%2Fwatch%3Fv%3De8w7f0ShtIM%3C%2FKey%3E%3C%2Fid%3E",
      "pageTitle": "U2 - Beautiful Day (with Lyrics)",
      "lyrics": "The heart is a bloom, shoots up through the stony
ground\nThere's no room, no space to rent in this town\nYou're out of luck and
the reason that you had to care,\nThe traffic is stuck and you're not [...]",
      "artist": "U2",

```

```
"title": "Beautiful Day",  
"emotion": "happiness",  
"emotionConfidence": "0.06818181818181818"  
"language": "en",  
"languageConfidence": "0.14243"  
  }  
}
```

Il formato orientato alle macchine dei risultati è quindi la response standard di Solr, interpretabile da qualsiasi libreria relativa a Solr o direttamente utilizzabile tramite richieste HTTP.

6 Conclusione

L'utilizzo di strumenti open source ha permesso di integrare implementazioni robuste e standard di processi facenti parte della comune conoscenza ingegneristica¹³, quali la conversione di artefatti multimediali (video e fotogrammi) e l'analisi del testo (in questo caso emozionale e linguistica).

L'implementazione del caso di studio e il suo rilascio come applicazione open source¹⁴ su Internet è anche un'occasione per ripagare il progetto SMILA: Chansonnier è un'applicazione totalmente funzionante e liberamente distribuibile realizzata al di sopra di SMILA, che può aiutarne la diffusione e servire da esempio per lo sviluppo di applicazioni simili.

L'utilizzo di Web service quali LyricWiki, affiancati ai software open source integrati in Chansonnier, ha invece permesso di integrare fonti di informazione proveniente dalla comunità di utenti Internet quali i testi delle singole canzoni, in pieno stile Web 2.0 (crowdsourcing).

Chansonnier è quindi in grado di indicizzare canzoni e fornire annotazioni su di esse come la lingua e l'emozione associata (valide quindi anche per gli artisti ad esse legate con relazione 1 a N).

Le performance del processo di indicizzazione sono dell'ordine dei minuti per singola canzone, legate al download del file multimediale che fa da collo di bottiglia (YouTube dedica diversa banda in base alla popolarità del video, e i download provengono da nodi diversi di un Content Delivery Network).

Canzone	Durata del video	Peso del video (MB)	Tempo di indicizzazione	Note
Green Day – Boulevard of broken dreams	4:28	11,3	3:30	immagini contenenti il testo
Lucio Battisti – La canzone del Sole	5:06	13,1	3:50	video b/n
883 – Nord Sud Ovest Est	4:10	18,8	3:00	video 360p
U2 - One	5:22	30,5	4:20	video 360p
Led Zeppelin – Stairway to heaven	7:59	18,6	6:00	immagini fisse
Pooh – Uomini Soli	4:24	14,4	3:20	video 360p
Bob Sinclar – Love generation	3:39	14,8	2:40	video 360p
Michael Jackson - Thriller	13:42	73,7	11:00	video 360p

Per ridurre questi tempi si possono parallelizzare i download aggiungendo più link. La coda viene svuotata ogni 10 secondi, e la propria banda può essere sfruttata appieno.

Chansonnier realizza un sistema di ricerca testuale su contenuti multimediali: il paradigma utilizzato può comprendere sia la ricerca libera di testo sia la restrizione tramite filtri come artista, emozione e lingua. Attachment binari come le immagini estratte dai video musicali sono disponibili per successive analisi da parte di altri annotatori, e sono mostrate all'utente nei risultati di ogni ricerca..

L'API di Solr (XML o JSON, a scelta) presenta invece le stesse informazioni sotto forma di Web service, e permette l'utilizzo di Chansonnier all'interno di applicazioni di ricerca multidominio tramite l'aggregazione dei suoi risultati con altri motori di ricerca.

Bibliografia

- 1 Marco Brambilla, Alessandro Bozzon and Piero Fraternali. Conceptual Modeling of Multimedia Search Applications using Rich Process Models (9th International Conference on Web Engineering, 2002)
- 2 Crane – Pascarello, Ajax in Action, Manning 2006, chapter 1 “Rethinking the Web application”
- 3 Vossen – Hageman, Unleashing Web 2.0 – From concepts to creativity, Morgan Kaufmann 2007, chapter 1 ”A Brief History of the Web”
- 4 Tim O'Reilly, What Is Web 2.0, <http://oreilly.com/web2/archive/what-is-web-20.html>
- 5 Josuttis, SOA in Practice, O'Reilly 2007, chapter 1 ”Motivation”
- 6 Richardson – Ruby, RESTful Web Services, O'Reilly 2007, chapter 10 “The Resource-Oriented Architecture Versus Big Web Services”
- 7 Roy T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, 2000, chapter 5 “Representational State Transfer”
- 8 Searching the Cloud - the EclipseRT Umbrella! (<http://www.eclipsecon.org/2010/sessions/?page=sessions&id=1388>)
- 9 About Ekman <http://www.paulekman.com/about-ekman/>
- 10 Freeman – Pryce, Growing object-oriented software, guided by tests, Addison-Wesley 2010, chapter 1 “What Is the Point of Test-Driven Development?”
- 11 Hall – Pauls – McCulloch – Savage, OSGi in Action, Manning 2010, chapter 10 “Component Models”
- 12 What is OSGi, sito ufficiale <http://www.osgi.org/About/WhatIsOSGi>
- 13 Eric Raymond, The Magic Cauldron - <http://catb.org/~esr/writings/magic-cauldron/magic-cauldron-10.html>
- 14 <http://github.com/giorgiosironi/Chansonnier>