Politecnico di Milano
V Facoltà di Ingegneria
Corso di laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

VALORE AGGIUNTO E QUALITÀ NELLA COMPOSIZIONE DEI WEB MASHUP

Relatore: Prof. Maristella MATERA
Correlatore: Prof. Cinzia CAPPIELLO
Correlatore: Dott. Matteo PICOZZI

Tesi di Laurea di:
Francesca CIURLI Matr. 735183
Valeria CIAFARDINI Matr. 725070

Anno Accademico 2010-2011

Indice

Elenco delle figure . . . IV
Elenco delle tabelle . . . VII

1 Introduzione **1**
1.1 La qualità del mashup . . . 4
1.2 Struttura della tesi . . . 5

2 Stato dell'arte **7**
2.1 I mashup . . . 7
2.2 Tecnologie per l'integrazione dell'UI . . . 8
2.2.1 Linguaggio di composizione . . . 10
2.2.2 Stile di comunicazione . . . 10
2.2.3 Discovery e Binding . . . 11
2.2.4 Visualizzazione dei componenti . . . 12
2.3 Tecnologie di composizione . . . 13
2.3.1 Componenti del desktop . . . 13
2.3.2 Componenti del browser . . . 14
2.3.3 Portali Web . . . 14
2.4 I Web mashup . . . 15
2.5 Tool di sviluppo per i mashup . . . 16
2.5.1 Yahoo Pipes . . . 17
2.5.2 Intel Mash Maker . . . 17
2.5.3 Quick and Easily Done Wiki . . . 19
2.5.4 Damia . . . 20
2.5.5 JackBe Presto . . . 21
2.5.6 Marmite . . . 21

2.5.7 MashArt 22
2.5.8 ServFace 24
2.5.9 SOA4ALL 25
2.5.10 Caratteristiche dei tool a confronto 26
2.6 Generazione di reccomendations per la composizione dei mashup . 29
2.6.1 Mashup Advisor 29
2.6.2 MatchUp: Autocompletion for Mashups 30

3 DashMash: verso la composizione dei mashup orientata agli utenti finali 31
3.1 La piattaforma DashMash 31
3.2 Il modello dei componenti e della composizione 34
3.2.1 UISDL 35
3.2.2 XPIL, eXtensible Presentation Integration Language . . . 37
3.3 Composizione ed esecuzione del Mashup 39
3.4 Generazione del modello di composizione 42

4 Il modello di qualità 45
4.1 La misura della qualità 47
4.1.1 La qualità del componente 50
4.1.2 La qualità aggregata 51
4.1.3 L'algoritmo 53
4.1.4 La complessità 55

5 La generazione di raccomandazioni per la composizione del mashup 57
5.1 Il processo a supporto della composizione 57
5.2 Compatibilità e Similarità 58
5.3 Categorizzazione 62
5.4 Mining 66
5.4.1 Weka e la creazione del file ARFF 66
5.4.2 L'algoritmo PredectiveApriori 70
5.5 Il Valore aggiunto 70
5.5.1 Metriche 72

5.5.2 Regole di associazione 74
5.5.3 Algoritmo 78

6 L'implementazione 81
6.1 L'architettura 81
6.2 Esempio di reccomendation 84
6.3 Integrazione degli algoritmi nel front-end 96

7 Conclusioni 99
7.1 Estensioni future 101

Bibliografia 103

Elenco delle figure

2.1 Yahoo Pipes 17
2.2 Intel Mash Maker 18
2.3 Quick and Easily Done Wiki 19
2.4 JackBe Presto 22
2.5 Marmite 23
2.6 MashArt 24

3.1 La piattaforma DashMash: a)menù dei componenti, b) stato dello workspace, c) gestore delle connessioni 33
3.2 Il framework di composizione 36
3.3 Event Bus 40

4.1 Possibili configurazioni dei componenti 49
4.2 Modello di qualità per i componenti 50
4.3 Modello di qualità per i mashup 51

5.1 La similarità dei verbi con WordNet 61
5.2 Tabella Cateogorizzazione Funzionalità (Parte 1) 64
5.3 Tabella Cateogorizzazione Funzionalità (Parte 2) 65
5.4 Frequenza delle categorie applicando il principio di Pareto 69
5.5 Inizio della raccomandazione 77

6.1 Il mashup contenente solo l'API di Wikipedia 85
6.2 Il mashup contiene ora le APIs di Wikipedia e di Flickr 88
6.3 Il mashup contiene ora le APIs di Wikipedia, Flickr e Twitter 91
6.4 Il mashup contiene ora le APIs di Wikipedia, Flickr, Twitter e Google Ajax Search 93

6.5 Mashup ottenuto attraverso la raccomandazione 94
6.6 Mashup originale 95
6.7 L'interfaccia a supporto delle recommendations 97

7.1 Modello concettuale del repository DashMash 101

Elenco delle tabelle

2.1 Tecnologie di composizione a confronto 13
2.2 Tabella comparativa dei tool di sviluppo (a) 27
2.3 Tabella comparativa dei tool di sviluppo (b) 28
2.4 Tabella comparativa dei tool di sviluppo (c) 28

5.1 Matrice generica della compatibilità parallela 59
5.2 Tabella aggregazione regole 1==>1 76

Capitolo 1

Introduzione

La diffusione del Service Orientied Computing (SOC) ha fornito una grande quantità di servizi distribuiti e riutilizzabili. SOC ha dunque introdotto uno spostamento del paradigma di sviluppo software, specialmente nel dominio del Web, da uno sviluppo software orientato ai prodotti, ad una composizione di servizi orientata al consumatore. Questo cambiamento (o evoluzione) promuove la composizione di nuove applicazioni partendo da una grande disponibilità di servizi riutilizzabili. La composizione di servizi è stata tradizionalmente corredata da standard e tecnologie potenti (ad esempio BPEL, WSCDL, ecc) che comunque possono essere padroneggiate esclusivamente da esperti IT, capaci di programmare delle applicazioni. Soluzioni per una composizione di servizi orientate all'utente finale sono state scarsamente esaminate.

Recentemente, nel contesto del Web, è emerso un nuovo paradigma di composizione: i *Web mashup*. I mashup sono un particolare tipo di applicazioni Web che consentono l'integrazione, a differenti livelli, di servizi o API messi a disposizione in rete. I mashup sono composti da parti atomiche chiamate componenti che, sincronizzati e debitamente orchestrati, consentono l'integrazione di diversi contenuti generando un'applicazione a valore aggiunto.

I componenti possono essere di diverse nature e tipologie. In particolare i componenti possono essere tipicamente Web services, API remote, servizi proprietari (locali o remoti), per esempio servizi di accesso a database. Per poter essere integrato nelle pagine Web in generale, ed in particolare nelle piattaforme dedicate alla composizione dei mashup, ogni componente tipicamente necessita di un *wrapper*,

in grado di valutare l'interfaccia per l'invocazione dei suoi metodi ed eventuali descrizioni e annotazioni. Un wrapper è tipicamente uno script JavaScript che serve a mappare le funzionalità di un servizio in modo che possano essere effettivamente utilizzate. Attualmente nei mashup che si possono trovare in rete, per esempio su ProgrammableWeb.com [33], i componenti non hanno una struttura anche dal punto di vista dell'implementazione e il codice dei wrapper è *nascosto* all'interno dell'applicazione o pagina Web.

I mashup stanno progressivamente diventando una soluzione alternativa alla composizione di servizi, che può aiutare a realizzare il sogno di un Web programmabile persino da utenti che non sono programmatori (cioè da parte di coloro che solitamente sono ascritti alla categoria di utenti finali, che si servono dei contenuti, ma che non ne creano, in quanto non posseggono le conoscenze tecniche adeguate). Ci sono tante ricerche sugli strumenti di mashup, i così detti *mash-makers*, che forniscono interfacce grafiche per comporre servizi di mashup senza chiedere all'utente di scrivere del codice.

Dal punto di vista della programmazione orientata all'utente finale, permettere ad una classe allargata di utenti (non solo gli sviluppatori esperti) di creare le proprie applicazioni richiede la disponibilità di astrazioni intuitive e strumenti di sviluppo semplici, e un alto livello di assistenza. Alcuni progetti (ad esempio ServFace [19] e SOA4All [31]) si sono focalizzati sul semplificare la creazione di un'efficace presentazione dei servizi, per fornire un canale diretto tra l'utente e il servizio (offrendo spesso un'interfaccia di servizio programmabile). Questo approccio non ha comunque permesso la composizione di servizi multipli in un'applicazione integrata.

Anche in un contesto aziendale, dove la tendenza attuale è di fornire all'utente strumenti per la costruzione di applicazioni chiamate *situational application* ad esempio applicazioni che svolgono una funzione analitica ben precisa e sono sviluppate per un orizzonte temporale limitato, si trovano approcci analoghi a quelli descritti poco sopra. I mashup aziendali sono il supporto per l'attuale approccio mashup nelle internet aziendali, permettendo ai membri dell'impresa l'utilizzo di servizi interni, definiti dal dipartimento di IT per accedere alle informazioni finanziarie dell'impresa stessa, e di servizi pubblici, nonché di poterli comporre in modo innovativo e utile ad esempio per automatizzare alcune procedure buro-

cratiche. Se supportato da strumenti adeguati, il paradigma di mashup consente la programmazione orientata all'utente: utenti non esperti possono creare la loro applicazione analitica senza l'aiuto di uno sviluppatore esperto. Questo *modus operandi* incrementa la produttività. Inoltre ci sono molte funzionalità di filtraggio e composizione di informazioni che non sono supportate da applicazioni aziendali di lungo periodo, questo è dovuto per esempio a specifiche necessità e/o preferenze che caratterizzano le attività dei singoli individui, oppure per problemi di business inaspettati.

Strumenti adeguati possono dunque fare uso di raccomandazioni che possano assistere l'utente finale durante la composizione del mashup, ad esempio suggerendo l'inclusione di determinati servizi, che possano migliorare la qualità del mashup. Questa tesi propone una soluzione a questo problema e definisce metodi per la valutazione della similarità e della qualità di ogni singolo componente con lo scopo di favorire e agevolare l'utente durante la composizione.

Una ricerca precedente si è focalizzata sullo sviluppo di un ambiente di mashup, DashMash [14], per l'integrazione di componenti eterogenei (non solo dati e Rss feeds) a livello di presentazione. Il progetto di mashup è basato su una notazione visuale, attraverso la quale l'utente DashMash può visivamente, in modo intuitivo, creare la composizione del mashup. La composizione visuale genera automaticamente un modello di composizione che poi dirige l'esecuzione del mashup.

Nel contesto di questo progetto, il nostro lavoro di tesi si è concentrato sulla generazione di raccomandazioni a supporto della composizione. Tali raccomandazioni hanno l'obiettivo di supportare l'utente nella scelta di componenti *adeguati*, e cioè *compatibilià* sintatticamente e semanticamente con gli altri componenti già presenti nella composizione, e in grado di massimizzare la *qualità* della composizione. In parte, tali dimensione di raccomandazione, e in particolare la compatibilità e similarità dei componenti, sono già state affrontate da lavori di tesi precedenti. Come sarà meglio spiegato nel seguito, la presente tesi ha contribuito a estendere l'approccio precedentemente definito, enfatizzando maggiormente aspetti legati alla *qualità* e al *valore aggiunto* del mashup.

1.1 La qualità del mashup

A fronte di una opportuna analisi semantica, che individui componenti tra loro simili, si può ulteriormente guidare l'utente in scelte positive fornendo una dettagliata metodologia di analisi qualitativa dei componenti stessi. Fino ad ora i lavori di ricerca presenti in letteratura si sono focalizzati sulla definizione delle tecnologie che abilitano la composizione di mashup, mentre pochi sforzi sono stati rivolti ai concetti inerenti alla qualità di questo tipo di applicazioni. Da un lato i mashup non sono altro che applicazioni Web, quindi si potrebbe assumere che la loro qualità possa essere studiata con i ben consolidati metodi dell'ingegneria del Web. Nonostante tutte queste osservazioni, noi crediamo che questo specifico tipo di applicazioni derivi da particolari dinamiche che ne caratterizzano il processo di sviluppo e che richiedono, pertanto, un'analisi specifica e dedicata. Una delle caratteristiche fondamentali dei mashup è l'integrazione di servizi pronti all'uso (e possibilmente ad accesso pubblico), costituenti i mattoni su cui è costruito questo tipo di applicazioni. L'integrazione è quindi un fattore chiave, che introduce innovazione e, molto spesso, determina il successo nell'utilizzo di un mashup. Tuttavia, come ogni altro sistema *integrato*, la qualità dei singoli servizi incide fortemente sulla qualità della composizione finale. Questo aspetto si riscontra chiaramente se si considera che, omettendo tutte le logiche di sincronismo e coreografia, le funzionalità dell'applicazione finale e il suo comportamento derivano direttamente dal comportamento dei singoli servizi. Per questo la valutazione della qualità del mashup deve partire dalla valutazione qualitativa di ogni singolo componente. La qualità della composizione necessità anche di particolare attenzione [22]: quando integrati in un mashup infatti, i servizi possono giocare differenti ruoli che incidono sulla percezione della qualità globale da parte dell'utente; ciò fa capire che la qualità dell'integrazione non può essere una banale aggregazione della qualità dei singoli componenti, ma deve tenere conto del loro ruolo e del loro peso nel concerto collettivo di tutti i componenti dell'applicazione.

In questo lavoro di tesi verrà presentato un approccio alla valutazione della qualità di un mashup in cui differenti prospettive di qualità e la relativa valutazione, sono integrati nel ciclo di vita dei mashup stessi, costruendo un processo di sviluppo che tenga conto del dato qualitativo. Descriveremo quindi inizialmente

un modello di qualità per i componenti di un mashup e identificheremo come la qualità dei singoli componenti, rappresentata da un'insieme di attributi di qualità, possa essere sfruttata per la selezione di servizi che producano mashup di un certo livello di qualità e stiano alla base della valutazione della qualità del mashup finale. Mostreremo inoltre come il concetto di valore aggiunto possa fornire ulteriori elementi utili alla generazione delle raccomandazioni. Tale dimensione va oltre la qualità tecnica dei componenti e del mashup, e mira invece a dare un valore all'utilità percepita dall'utente. Nel seguito della tesi mostreremo come benchmark per tale dimensione possono essere ottenuti tramite l'analisi delle composizioni della comunità nel tentativo di estrarre associazioni tra componenti che altri utenti hanno ritenuto utili.
In generale, osservando le pratiche correnti, è possibile notare come durante la composizione di un mashup la selezione di servizi adeguati sia basata su requisiti funzionali, non considerando la qualità del servizio stesso. In questo lavoro di tesi, mostreremo invece che la qualità dei servizi selezionati, congiuntamente al modo in cui sono composti, ha un impatto diretto sulla qualità totale e sul valaore aggiunto del mashup finale. In accordo con ciò, mostreremo come i criteri di qualità che caratterizzano i mashup, e associazioni interessanti estratte dai mashup della comunità possano guidare la selezione dei componenti nei mashup stessi, durante la composizione da parte dell'utente finale.

1.2 Struttura della tesi

In questo paragrafo si descrive brevemente la struttura che si intende dare alla presente trattazione. Procediamo dunque a dare un elenco e una breve descrizione di tutti i capitoli:

- **Capitolo 2** - in questo capitolo viene descritto lo stato dell'arte, cioè lo scenario attuale dei mashup, dell'integrazione dell'UI, delle tecnologie di composizione e dei tool di sviluppo esistenti, che costituiscono il background della presente tesi;

- **Capitolo 3** - il capitolo terzo si occupa di descrivere nel dettaglio la piattaforma per la quale sono stati realizzati gli algoritmi di compatibilità, simila-

rità e qualità, cioè *Dash Mash*. L'attenzione e posta sulla struttura interna della piattaforma in modo da poter successivamente comprendere come la presente tesi costituisca un intervento volto a migliorarla e arricchirla;

- **Capitolo 4** - in questo capitolo ci si dedica invece a descrivere il modello di qualità al quale si è fatto riferimento per il calcolo della qualità, del valore aggiunto e del processo di reccomendation;

- **Capitolo 5** - il quinto capitolo presenta una visione generale dello scenario di calcolo di compatibilità, similarità e qualità esistenti che si sono sfruttate per giungere ai risultati di estensione del modello di qualità con il concetto di Added Value e una panoramica dei risultati ottenuti;

- **Capitolo 6** - questa sezione è dedicata invece, a descrivere l'implementazione degli algoritmi a partire dalla descrizione dell'architettura sottostante e da un cenno alle tecnologie utilizzate (le più salienti delle quali saranno poi più accuratamente descritte nell'Appendice A), fino a giungere alla descrizione di un esempio di reccomendation che utilizza i concetti di added value e regole di associaizone introdotti, volto a valorizzare il front-end dell'applicativo che è stato integrato;

- **Capitolo 7** - si conclude con un ultima parte dedicata a importanti riflessioni, sul tema trattato, che portano a trarre delle conclusioni;

- **Appendice A** - questa appendice raccoglie una breve rassegna delle tecnologie più importanti, sia per peculiarità, che per innovatività, che hanno condotto il nostro studio.

Capitolo 2

Stato dell'arte

Questo capitolo introduce e descrive brevemente cosa siano i mashup, cosa siano i componenti e come si strutturi la loro coreografia nel mashup. Tutte queste nozioni vengono contestualizzate ampiamente nei primi paragrafi di questo capitolo in cui ci dedichiamo al problema di integrazione dell'interfaccia utente, alle tecnologie di sviluppo e ai tool di composizione preesistenti; infine ci si dedica a esporre con un certo livello di generalità cosa si intenda per annotazioni semantiche, per coi concludere spiegando come si generano raccomandazioni a supporto della composizione dei mashup da parte degli utenti finali.

2.1 I mashup

Nel corso degli anni sono stati effettuati molteplici ricerche e sviluppi nel campo dell'integrazione di applicazioni e, più recentemente, di servizi secondo il paradigma SOA (Service Oriented Architecture). Il riutilizzo e l'integrazione di contenuti Web e di servizi forniti da terzi è già una realtà comunemente conosciuta con il nome di Web mashup [18]. I Web mashup [5, 28] sono siti o applicazioni Web sviluppate integrando interfacce utente, logica applicativa e contenuti (dati) ai quali è possibile accedere dal Web e che possono essere forniti, come già detto, da soggetti terzi. I primi mashup comparsi sulla scena non potevano fare affidamento su API come interfacce di programmazione, dal momento che gli effettivi fornitori di contenuti non erano nemmeno consapevoli del fatto che i loro siti Web fossero inseriti in altre applicazioni. I primi mashup con Google Maps [42], per esempio,

sono precedenti al rilascio ufficiale delle API di Google Maps. Le API per mashup disponibili pubblicamente sono in aumento. L'integrazione può essere eseguita "ad hoc", sfruttando qualsiasi linguaggio di programmazione supportato dalla fonte dei contenuti, sia lato client (e.g. AJAX) sia lato server (e.g. PHP, Java, ASP.NET). Il contenuto è tipicamente fornito in codice markup e integrato inserendo il rispettivo sito Web. Dal momento che tali tipi di "interfacce di componenti" non sono in genere molto stabili (si ricorda che il fornitore dei contenuti potrebbe non sapere che la sua applicazione è stata riutilizzata), lo sforzo maggiore nello sviluppo di mashup è dovuto al testing manuale e alla manutenzione. A causa della mancanza di framework di supporto adatti, la stabilità del codice non è assicurata (cioè ci potrebbero essere collisioni tra due codici sorgenti JavaScript), e potrebbero verificarsi conflitti tra componenti UI, cioè dotati di interfaccia grafica. Quindi la costruzione di un mashup Web resta un compito ad hoc lungo e impegnativo e la necessità di modelli di programmazione appropriati è evidente.

2.2 Tecnologie per l'integrazione dell'UI

Nell'integrazione dell'interfaccia utente è possibile individuare quattro temi in particolare:

- definire modelli e linguaggi per le specifiche dei componenti;
- definire modelli e linguaggi per le specifiche della composizione dei componenti;
- scegliere stili di comunicazione per l'interazione dei componenti;
- definire discovery e meccanismo di binding (anche durante il runtime) per l'identificazione dei componenti.

Anche tenendo conto di determinati principi guida, cioè, la semplicità, il formalismo, la leggibilità e la modularità delle specifiche, è possibile individuare alcune possibili soluzioni per la gestione dei componenti e delle interazioni con il loro livello di presentazione, ognuno caratterizzato da un determinato grado di complessità, tecnologia e programmazione cosiddetta *amichevole*:

- modello del componente: le applicazioni di integrazione dei componenti sono caratterizzate da una API (Application Programming Interface) e anche da un modello del componente. Nell'integrazione dei dati, lo schema della sorgente dei dati descrive il componente. In effetti, nell'integrazione dell'interfaccia utente a livello di presentazione, oltre al riutilizzo di classi di librerie, si rivela necessario un modello a componenti in grado di supportare le interazioni complesse e di coordinamento. Ogni singolo componente è descritto da una invocazione di servizi software di abilitazione dell'interfaccia e da un'interfaccia utente che consente l'interazione. Un'interfaccia utente del componente consente diversi livelli di interoperabilità;

- solo-GUI: tutte le interazioni con i componenti GUI di sola esecuzione sono effettuate mediante una componente logica per l'interfaccia utente. L'unico metodo per integrare un componente solo-GUI è conoscere la sua interfaccia utente ed essere in grado di tracciare i tratti di posizione del mouse e di capire cosa la UI del componente mette in evidenza, così da rendere possibile eseguire le azioni che causano la modifica dell'interfaccia utente;

- interfaccia nascosta: in molte applicazioni Web, il componente è dotato di un'interfaccia specifica che consente di controllare la sua interfaccia utente, ma non è pubblicamente descritta. Applicazioni Web di interazione permettono l'accesso e la manipolazione di contenuti e la visualizzazione della risposta mediante l'invio di richieste HTTP. Queste applicazioni obbediscono a un protocollo generale per l'interazione tra client e applicazioni, di solito difficili da identificare;

- interfaccia pubblicata: in questo caso ideale, il componente fornisce una descrizione pubblicata della sua interfaccia utente e una API in modo da poter effettuare manipolazioni a runtime. Un basso livello API potrebbe permettere il controllo dei singoli elementi dell'interfaccia utente. Una API di alto livello potrebbe esporre un set di soggetti e oggetti controllabile, così come le operazioni per modificare lo stato dell'entità.

2.2.1 Linguaggio di composizione

Un linguaggio di composizione sorge a sostegno dell'identificazione e della specificazione degli elementi coinvolti in una orchestrazione e della loro interazione. Per quanto riguarda l'integrazione dei dati, la composizione avviene spesso attraverso una vista SQL che permette di esprimere uno schema globale come un insieme di viste su uno schema locale. Nella integrazione delle applicazioni la composizione può essere descritta sia tramite linguaggi di programmazione general-purpose, come Java, sia attraverso linguaggi dedicati all'integrazione delle applicazioni, come i linguaggi di composizione del workflow o di servizio. Nella composizione dell'interfaccia utente ci potrebbero essere due tipi di soluzioni:

- linguaggi di programmazione general-purpose: gli sviluppatori possono adottare una terza generazione di linguaggi per la composizione delle applicazioni. Tali linguaggi sono molto flessibili, ma mancano di astrazione dei componenti a grana grossa (come strutture per la discovery e per il binding dei componenti o primitive di alto livello per la sincronizzazione dei componenti che l'interfaccia utente mostra);

- linguaggi di composizione specializzati: i linguaggi di alto livello sono tipicamente usati con una sintassi XML su misura per la composizione dei componenti dell'interfaccia utente a livello di descrizioni astratte o esterne. Il vantaggio principale di questi linguaggi è di avere un livello superiore di programmazione per il supporto delle composizioni, che sfrutta le caratteristiche del modello del componente.

2.2.2 Stile di comunicazione

Abbiamo già sottolineato l'importanza della comunicazione per l'integrazione dell'interfaccia utente: la necessità è quella di monitorare gli eventi all'interno di alcuni componenti dell'interfaccia utente, che possono aggiornare lo stato (e quindi, l'interfaccia utente) di altri componenti. È possibile distinguere tra due tipi di comunicazione:

- comunicazione mediata centralmente, in cui l'applicazione composita ha un coordinatore centrale che riceve gli eventi e le istruzioni riguardanti

le richieste per manipolare le interfacce utente dei componenti, dai singoli componenti;

- comunicazione diretta componente a componente, in cui l'applicazione composita è una coalizione di singoli componenti.

Queste soluzioni si differenziano nettamente dall'integrazione dei dati in cui i componenti sono di solito passivi e non avviano la comunicazione con l'applicazione che opera l'integrazione. L'integrazione dell'applicazione mostra la stessa differenza: un soggetto centralizzato (l'applicazione composita) richiama i componenti, se necessario. Un'ulteriore distinzione tra questi tipi di comunicazione in materia di integrazione dell'interfaccia utente è quella tra interazione RPC-style, in cui lo scambio di informazioni attraverso i componenti genera chiamate di metodo e dei dati restituiti, e interazione publish-subscribe, in cui le applicazioni comunicano in modo debolmente accoppiato tramite messaggi scambiati attraverso i messaggi dei broker.

2.2.3 Discovery e Binding

Un problema rilevante in materia di integrazione dell'interfaccia utente è la scoperta (*discovery*) dei componenti coinvolti nella composizione e capire come ottenere il binding, sempre dei medesimi componenti. Ci sono due diversi modi per farlo: il primo consiste nel definire la composizione staticamente in fase di progettazione o nel momento dell'implementazione e la seconda consiste nel fare questo in modo dinamico in fase di runtime.

Osserviamo che nell'integrazione dei dati e delle applicazioni l'associazione tra diverse fonti di dati in genere si verifica in fase di progettazione quando i dati dello schema globale sono definiti e la discovery viene eseguita dinamicamente da un middleware di integrazione. Tuttavia questo approccio manca di flessibilità a causa della difficoltà di interagire con componenti che siano stati recentemente soggetti a discovery e solo successivamente aggiunti all'integrazione iniziale.

Una soluzione ibrida di binding è utilizzata anche nel caso in cui il progettista dell'applicazione identifichi e testi un insieme di componenti potenziali e in seguito gli utenti selezionino un sottoinsieme di essi in fase di runtime in base al

compito che devono affrontare: in questo caso la discovery è statica, ma il riferimento è dinamico. Questo approccio ibrido rende anche possibile l'integrazione dell'interfaccia utente.

2.2.4 Visualizzazione dei componenti

Ci sono diverse soluzioni per la visualizzazione dei componenti, la cui distinzione è basata sul paradigma di rendering dell'interfaccia utente. Infatti, il componente può visualizzare la sua interfaccia utente in un primo tipo di soluzione, mentre in un secondo tipo di soluzione, l'applicazione composita riceve il codice di markup dell'interfaccia utente dei componenti e li renderizza. La seconda soluzione ha bisogno di una descrizione di markup, che deve essere interpretata con un motore di rendering in modo da ottenere una traduzione in elementi grafici: la specifica di markup descrive spesso le proprietà statiche dell'interfaccia utente, mentre i linguaggi di scripting dinamico ne forniscono il comportamento. Tale descrizione può essere fatta con i linguaggi orientati ai documenti o i linguaggi di interfaccia utente che si interfacciano appunto con un sofisticato modello di applicazione. Esempi di tale linguaggio sono XAML [50], XUL [52], UIXML [53], XIML [51]. È possibile avere due diverse soluzioni di rendering dell'interfaccia utente:

- component-rendering dell'interfaccia utente: il componente gestisce il rendering dell'interfaccia utente e il display, quindi l'applicazione composita è una collezione di interfacce utente di componenti. Questo è il caso delle applicazioni desktop classiche che individuano componenti eseguibili legati a librerie grafiche;

- un'interfaccia utente basata su markup: il componente restituisce il codice di interfaccia utente e delega il rendering dell'utente finale sia alle applicazioni composite sia all'ambiente funzionante, in grado di interpretare il codice dell'interfaccia utente dei componenti e di allocare spazio adeguato di layout per il rendering dei componenti stessi. In questo caso, l'interazione dell'utente con il componente può essere gestita direttamente attraverso una logica di creazione di script adatti incorporati nel codice del componente, oppure tramite l'applicazione composita, che è in grado di intercettare gli eventi generati dell'UI e li trasmette al componente per l'interpretazione.

2.3 Tecnologie di composizione

Illustriamo in tabella 2.1 un confronto tra le diverse tecnologie di interfaccia utente considerate nel contesto della composizione UI. Questo confronto è incentrato sui principali aspetti di integrazione UI precedentemente presentati. Le tecnologie a confronto sono brevemente descritte nel resto di questa sezione.

	Desktop UI components	Browser plug-in components	Web portals and portlets
Component model	Published, Programmable API	Basic interface	Public API
Composition language	General-purpose	Document markup code, Javascript	General-purpose
Communication style	Mediated intercomponent	Centrally mediated	Centrally mediated
Discovery and binding	static and dynamic	static	static and dynamic
Component visualization	rendered	rendered	markup-based

Tabella 2.1: Tecnologie di composizione a confronto

2.3.1 Componenti del desktop

Storicamente la composizione di interfaccia utente è nata per le applicazioni desktop, per creare un ambiente in cui le applicazioni sviluppate con linguaggi eterogenei potessero interagire. Pensiamo ad esempio ad ActiveX, che sfrutta la tecnologia Microsoft COM per incorporare una completa interfaccia utente nelle applicazioni host, e l'applicazione composita UI Block(CAB), che è un framework di interfaccia utente per la composizione in .NET con un servizio di contenitore che consente agli sviluppatori di creare applicazioni su moduli caricabili o plugin. In particolare, i componenti CAB possono essere utilizzati con qualsiasi .NET per costruire contenitori compositi ed effettuare comunicazioni componente-container. CAB fornisce un broker di evento per molti-a-molti, con accoppiamento di comunicazione tra i componenti sulla base di un modello publish-subscribe di eventi di runtime. Un altro esempio è Eclipse Rich Client Platform (RCP), che fornisce un simile framework, ma include anche una shell di applicazione di servizi di interfaccia utente. Esso offre inoltre un modulo basato su API che consente

agli sviluppatori di creare applicazioni all'interno di questa shell. Eclipse consente inoltre agli sviluppatori di personalizzare ed estendere i componenti dell'interfaccia utente tramite i cosiddetti punti di estensione, una combinazione di interfacce in linguaggi Java e di markup, come XML. I componenti desktop dell'interfaccia utente in genere utilizzano i linguaggi di programmazione general-purpose per componenti integrati (C # e Java per il CAB per RCP) perché le interfacce dei componenti sono API di programmazione specifiche del linguaggio. I componenti possono gestire i loro propri rendering dell'interfaccia utente, ma potrebbero anche sostenere stili di comunicazione flessibile. Entrambi i binding in fase di progettazione e di esecuzione sono supportati, basandosi su meccanismi di reazione specifici del linguaggio. Molte delle tecnologie per i componenti desktop dell'interfaccia utente sono dipendenti dal sistema operativo. Sebbene CAB e RCP non dipendano direttamente dal sistema operativo, si affidano rispettivamente al loro ambiente di runtime. La mancanza di interfacce che riconoscano la tecnologie e siano descrittive, rende l'interoperabilità tra i componenti attuata con diverse tecnologie difficile da raggiungere.

2.3.2 Componenti del browser

L'esperienza di navigazione nel browser spesso coinvolge caratteristiche di interfaccia utente avanzate. Le principali tecnologie utilizzate per creare componenti di interfaccia utente nascosti in interfacce basate sul markup sono applet Java con controlli di tipo ActiveX. Dopo la definizione del binding dei componenti durante la scrittura della pagina da parte del Web designer, il browser scarica i componenti a runtime e li istanzia. Questi componenti spesso rendono disponibile il loro proprio rendering, con una piccola comunicazione ulteriore tra di loro e la pagina Web che li contiene. L'interfaccia esterna di questo tipo di componenti è molto semplice e solitamente richiede solo la configurazione di opportuni parametri quando si incorporano i componenti all'interno del codice di markup.

2.3.3 Portali Web

Lo sviluppo di portali Web distingue esplicitamente tra componenti UI (portlets) e applicazioni composite (portals). I portlets sono componenti di applicazioni

Web; generano frammenti di documenti di markup che aderiscono a regole fissate, in modo da facilitare l'aggregazione di contenuti nei portali server per formare in ultimo documenti compositi. I portali server tipicamente fanno sì che gli utenti possano customizzare pagine composite e rendono disponibili svariati tipi di personalizzazioni. Analogamente alle servlet Java, i portlet implementano una specifica interfaccia Java per l'API standard dei portlet, che è stata intesa per aiutare gli sviluppatori a creare portlet che possano conformarsi ai portali server. Per i portlet Java, le applicazioni dei portali sono basate sul linguaggio di programmazione Java, sebbene con le parti Web, uno sviluppatore Web programma le proprie applicazioni in .NET. L'applicazione portale aggrega tutti gli output di markup dei suoi portlet e gestisce la comunicazione in una tipologia centralmente mediata. I portlet permettono anche sia il binding statico che quallo dinamico; a runtime, l'applicazione portale può rendere i portlet disponibili in un registro per la selezione e il posizionamento da parte degli utenti. Sebbene i portlet e i portali Web abbiano obiettivi e architetture similari, non sono interoperabili. I Web service per portlet remoti (WSRP) indirizzano questa richiesta al protocollo di livello esponendo portlet remoti come servizi Web; la comunicazione tra portali e portlet avviene tramite SOAP, il che significa che gli sviluppatori possono costruire portali e portlet con differenti linguaggi e framework a runtime.

2.4 I Web mashup

I mashup sono particolari applicazioni Web che consentono l'integrazione, a differenti livelli, di servizi o API messi a disposizione in rete. I mashup sono composti da parti atomiche chiamate componenti che, sincronizzati e debitamente orchestrati, consentono l'integrazione di diversi contenuti generando un'applicazione a valore aggiunto.

I componenti possono essere di diverse nature e tipologie. In particolare i componenti possono essere tipicamente Web Services, API remote, Servizi proprietari (locali o remoti) [29] per esempio servizi di accesso a database. Ogni componente necessita di un wrapper per poter essere integrato nelle pagine Web in generale, ed in particolare nella piattaforma.

Per quanto riguarda il modello dell'interfaccia utente del componente e la specificazione esterna i Web mashup offrono una API pubblicata, ma un'interfaccia nascosta. Il linguaggio di composizione e costituito da linguaggi di programmazione general-purpose. Lo stile di comunicazione è mediato in maniera centralizzata. In relazione a discovery e binding si evidenzia un binding di tipo statico, mentre circa la visualizzazione dei componenti si può asserire che generalmente essa è basata sul markup.

In quanto alle caratteristiche peculiari di questi componenti enumeriamo compatibilità, similarità e qualità. La compatibilità è una corrispondenza prettamente sintattica e riguarda i tipi dei parametri di input e output e delle operazioni/eventi di ogni componente. Se c'è matching esatto allora i componenti sono compatibili. La similarità valuta sempre i medesimi parametri di cui sopra, però dal punto di vista semantico e viene calcolata tramite l'utilizzo di WordNet e delle ontologie. Infine la qualità è data da un insieme di caratteristiche che vengono assegnate al componente e che ne costituiscono una sorta di strumento valutativo. Queste verranno ampiamente trattate nel quarto capitolo per poi essere estese con il concetto di Added Valure ovvero, come vedremo, un'altra misura di qualità che fa parte della mashup composition quality che ci porta a valutare un componente in base a quanto quest'ultimo, integrato con gli altri componenti del mashup, aggiunga valore alla composizione finale. In altre parole, vedremo come un concetto così difficilmente valutabile, poichè astratto e soggettivo dal punto di vista della percezione di aggiunta di valore per l'utente finale, possa essere definito e catalogato sulla base della popolarità di associazioni e sulla scissione su tre livelli di qualità: dati, logico, interfaccia utente.

2.5 Tool di sviluppo per i mashup

Per velocizzare l'intero processo di sviluppo dei mashup, ma anche per consentire agli utenti finali dei mashup (anche inesperti) di creare proprie applicazioni Web, di recente sono emersi numerosi strumenti e framework di sviluppo. Questi strumenti vengono solitamente forniti con una varietà di caratteristiche e una miscela di approcci di composizione. Uno sguardo da vicino ad essi ci permette di identificare le questioni aperte e le sfide della ricerca che caratterizzano il fe-

nomeno mashup. Ai fini della presentazione, abbiamo selezionato gli approcci più popolari o rappresentanti strumenti per creare mashup volti all'utente finale.

2.5.1 Yahoo Pipes

Yahoo Pipes [9] prende uno o più componenti RSS, li analizza e restituisce un nuovo componente come risultato. Ciò permette di avere uno strumento di rappresentazione grafica che fornisce agli sviluppatori la possibilità di procedere manipolando il singolo componente. Yahoo Pipes si concentra in particolare sull'integrazione dei dati piuttosto che sull'integrazione dell'interfaccia utente. Ne mostriamo un'immagine in figura 2.1.

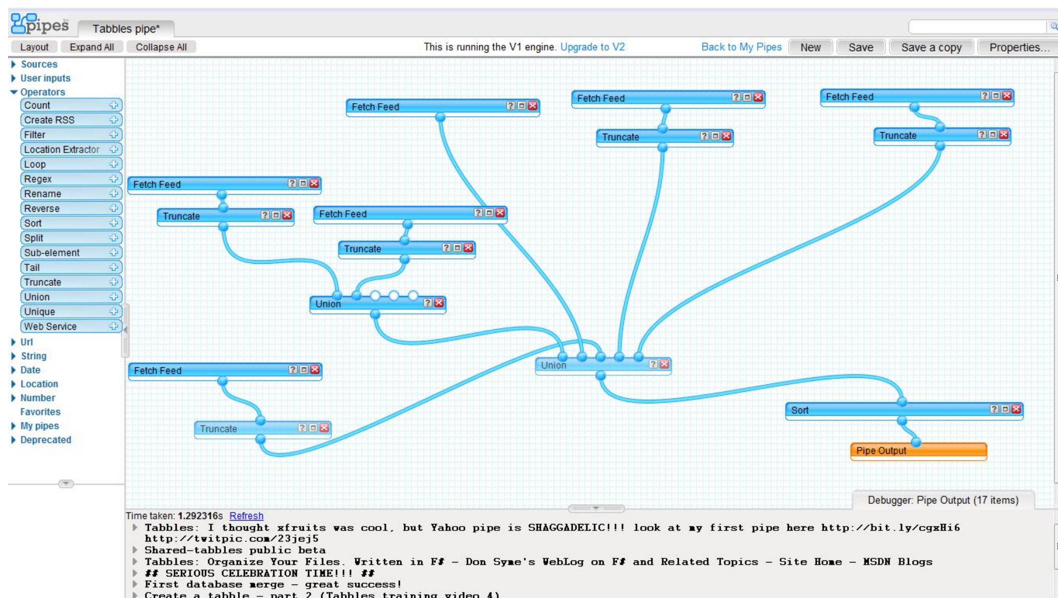

Figura 2.1: Yahoo Pipes

2.5.2 Intel Mash Maker

Intel Mash Maker [25] è un browser per il Web Semantico. Può essere utilizzato come un browser Web, eccettuato il fatto che Intel Mash Maker comprende il significato semantico delle pagine Web. Questo fatto rende possibile la presenza di alternative, più dispendiose, rappresentazione di dati e anche l'*intelligente* combinazione dei dati con altre informazioni. Con Intel Mash Maker, e similmente con

altre applicazioni, è possibile creare Web Mashup, ma Intel Mash Maker (figura 2.2) è provvisto di un tool per la creazione di un Web mashup predisposto, basato sull'interpretazione semantica delle pagine Web. Risulta inoltre provvisto di strumenti che visualizzano i contenuti semantici. In conclusione, questo framework ha l'obiettivo di ottenere una soluzione che renda disponibile un livello di astrazione che a sua volta permette la componentizzazione di componenti eterogenei tramite il collegamento di eventi di alto livello e di operazioni, in modo tale che l'utente finale possa pensare alla composizione come ad una operazione funzionale ed astratta, senza la necessità di doversi occupare dell'implementazione dei componenti o degli adattatori e della tecnologia di comunicazione che è introdotta da essi. D'altro canto, chi mette a disposizione un componente ha soltanto la necessità di esporre i propri eventi e metodi con un semplice adattatore mappando gli eventi e i metodi stessi con un concetto di livello superiore.

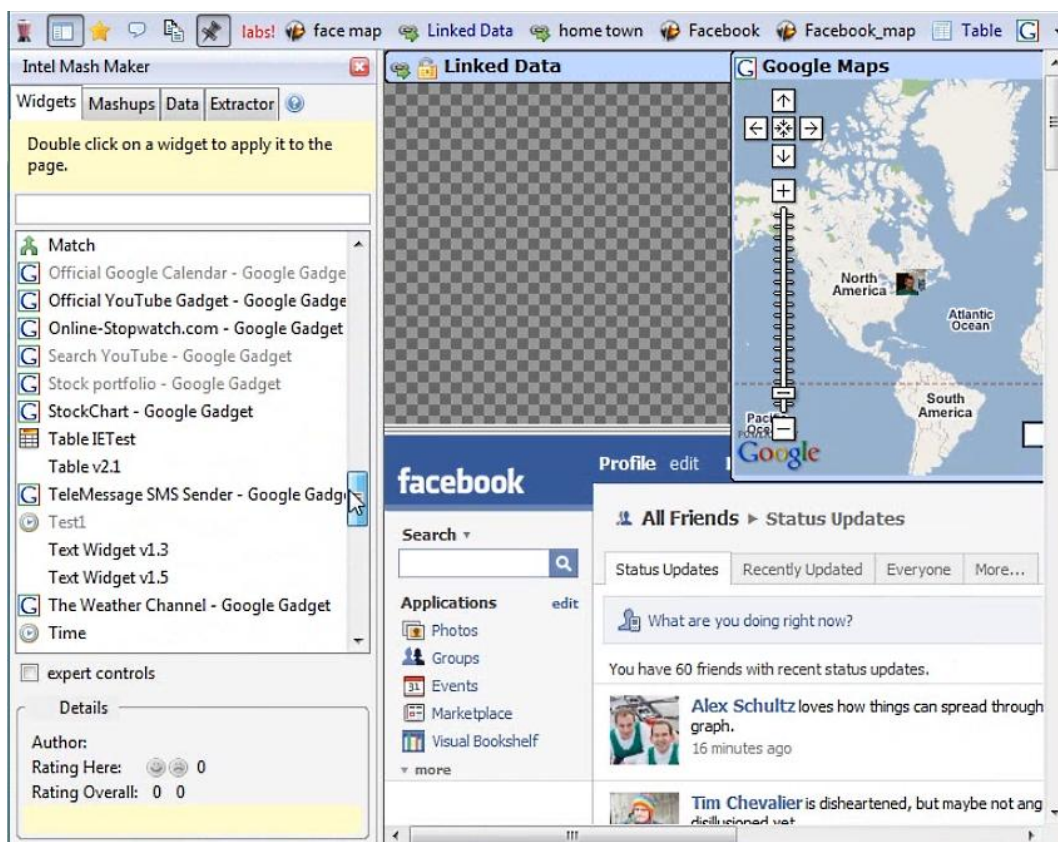

Figura 2.2: Intel Mash Maker

2.5.3 Quick and Easily Done Wiki

QEDWiki [23] è la proposta IBM per un *creatore di mashup* wiki-based, completamente in esecuzione all'interno del browser e con un client di accesso che permette l'interazione con il mashup Hub di IBM. L'Hub supporta la creazione dei data-feed e widget di interfaccia utente e incorpora dati del Mashup per applicazioni Intranet (Damia) per l'assemblaggio dei dati e la loro manipolazione. Avendo tutte le caratteristiche tipiche di un ambiente wiki, permette agli utenti di modificare, visualizzare immediatamente e condividere facilmente mashup. I mashup sono assemblati da JavaScript da Widget basati su PHP, il cui cablaggio determina il comportamento del mashup. I Widget rappresentano componenti applicativi che potrebbero o meno avere la propria interfaccia utente. Per assemblare un mashup, un utente seleziona un layout di pagina (un modello HTML) e poi trascina degli widget sulla griglia della pagina (figura 2.3) e li configura in modo interattivo.

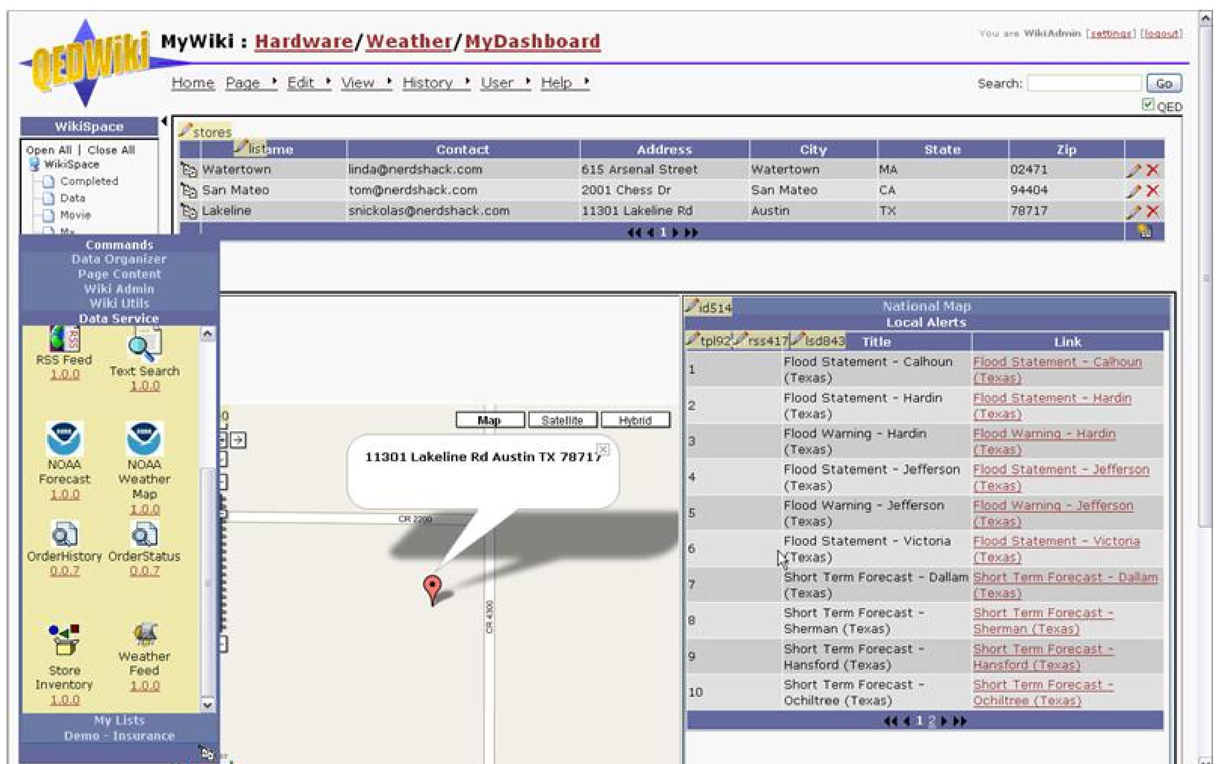

Figura 2.3: Quick and Easily Done Wiki

2.5.4 Damia

Mediante un interfaccia basata sul Web, IBM DAMIA [21], rende possibile un agevole utilizzo dei tool che gli sviluppatori e gli utenti IT possono utilizzare per assemblare velocemente i componenti dei dati da Internet e una vasta gamma di fonti di individuazione dei dati. I benefici di questo servizio includono la capacità di aggregare e di trasformare una consistente varietà di componenti a livello di dati e di contenuto, che possono essere utilizzati nei nuovi mshup. Damia fornisce inoltre la possibilità di svolgere le seguenti attività:

- importare XML, Atom, e feeds RSS;
- assemblare componenti sia da Internet che da fogli di sviluppo Excel (il database costituisce un supporto per questa applicazione);
- importare dati da file locali in formato XML e fogli di sviluppo Excel;
- aggregare e trasformare un'ingente varietà di dati o di contenuti dei componenti in nuovi servizi organizzativi.

Quando si costruisce una applicazione Web completa che rende disponibile un'interfaccia utente, sono richiesti ulteriori tool o tecnologie al fine di mostrare il contenuto dei dati messo a disposizone da DAMIA. I costruttori di applicazioni mashup e i lettori dei componenti che si servono di Atom e RSS possono essere utilizzati come il livello di presentazione nella costruzione di applicazioni Web. DAMIA presenta la seguente composizione:

- un'applicazione Web basata sul browser per l'assemblaggio, la modifica e la presentazione dei mashup;
- servizi per la manipolazione, la custodia e il recupero dei componenti dei dati creati nell'ambito dell'innovazione nello stesso modo che sulla rete Internet. In aggiunta alla creazione dei componenti dei dati a partire da varie sorgenti, Damia può pubblicare informazioni come ad esempio fogli di sviluppo Excel oppure documenti XML nel formato mashup;
- una repository per la condivisione e la raccolta dei componenti o delle informazioni create da DAMIA;

- servizi per l'amministrazione dei componenti e delle informazioni rigurdanti i mashup;

- ricerca delle capacità e tool per la segnalazione e la valutazione dei mashup.

2.5.5 JackBe Presto

JackBe Presto [24] è una piattaforma di Enterprise Mashup che include funzionalità per la creazione e la gestione di mashup aziendali. Esso fornisce un supporto per gli sviluppatori e gli utenti dell'applicazione nella'attività di creazione, dall'inizio fino a personalizzare e condividere mashup Enterprise Apps. Le funzionalità di JackBe Presto sono le seguenti:

- motori di servizi di accesso per i servizi Web, SQL, RSS e Web clipping;

- compositori/creatori di mashup tra cui un grafico, uno strumento drag-and-drop (figura 2.4) e un linguaggio di marcatura per *declarative enterprise mashup*;

- connettori mashup per gli strumenti di enterprise più popolari, fra cui Microsoft Excel, HP Systinet e Oracle WebCenter;

- API mashup per JavaScript, Java, REST, C#, e NET;

- Enterprise Mashup Markup Language (EMML).

Nel marzo 2010 JackBe ha lanciato una versione cloud-based del suo prodotto Presto ospitato su Amazon EC2.

2.5.6 Marmite

Marmite [20] è uno strumento creato da Jeffrey Wong e Jason I Hong. Si presenta come una soluzione che consente agli utenti di creare i propri mashup, senza avere prerequisiti in termini di competenze di programmazione. Permette di:

- accedere a Web Service API;

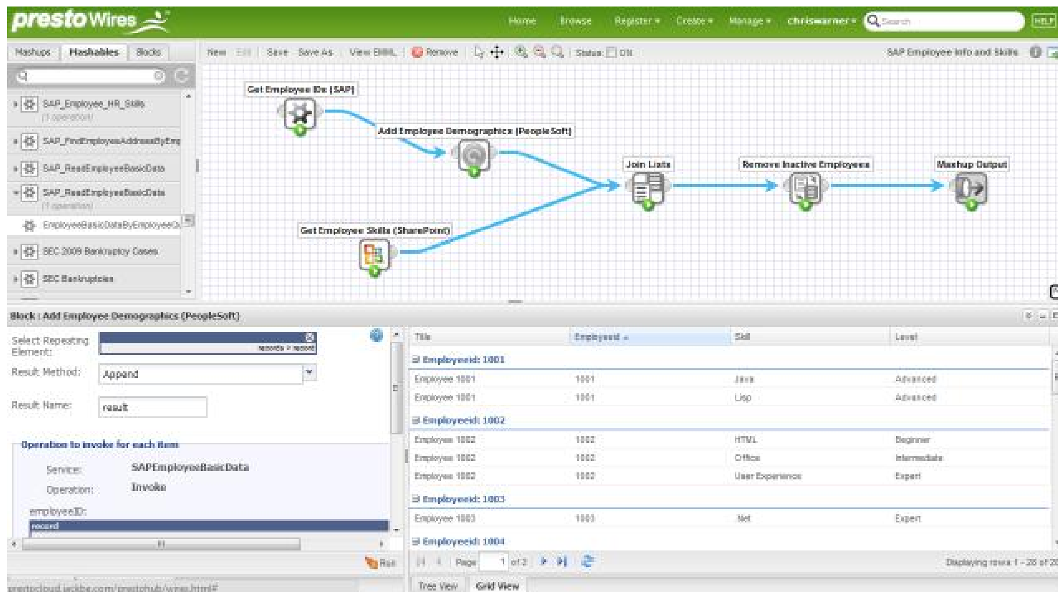

Figura 2.4: JackBe Presto

- combinare Web Service API con screen-scrape orientati alla programmazione;
- presentare uno strumento composto da un menù di operatori, una visualizzazione del flusso di dati e una vista dei dati (figura 2.5).

2.5.7 MashArt

Di seguito è riportata una descrizione generale di MashArt [27, 49]. MashArt (figura 2.6) consiste in uno strumento avanzato che da la possibilità all'utente di componentizzare le proprie applicazioni in maniera relativamente semplice ed efficiente mediante specifici metodi. Innanzitutto vi è un metodo di run-operation invocato dal framework che permette di effettuare un'operazione che è descritta all'interno del mio componente. Il parametro *operation* descrive l'operazione che dovrà essere eseguita. Successivamente ci si servirà di un metodo chiamato *load* il quale andrà a posizionare il componente all'interno del *div* corrispondente nella pagina html. Infine verrà sollevato un evento. Gli eventi generano dei parametri come output, mentre le operazioni successive prendono dei parametri come input.

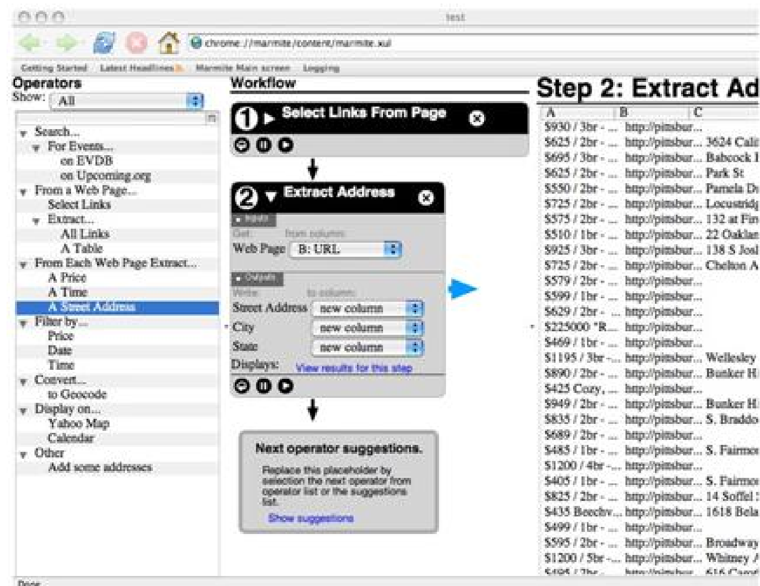

Figura 2.5: Marmite

I parametri generati dagli eventi vengono riconosciuti e servono per identificare univocamente il componente specifico che si vuole andare a caricare ed il *div* in cui lo si vuole caricare. Tale è dunque in breve il comportamento di base di mashart, una piattaforma per i mashup e lo sviluppo tramite host di applicazioni Web di alto livello e che permette anche l'analisi approfondita di tali applicazioni.

MDL

Per istanziare il modello di componente descritto, utilizziamo MDL, un linguaggio di descrizione astratto e non legato alla tecnologia per i componenti UI (simili ai servizi WSDL per il Web). Data un'applicazione che vogliamo componentizzare, MDL consente di definire un nuovo componente per descrivere quali sono gli eventi e le operazioni che caratterizzano il componente e per definire i tipi dei dati e il costruttore. Non c'è un esplicito costrutto di linguaggio per lo stato del componente, come quello che è posto a mano internamente al componente, per esempio il suo UI. D'altra parte, MDL ci consente di descrivere i cambiamenti di stato nella forma di eventi e operazioni.

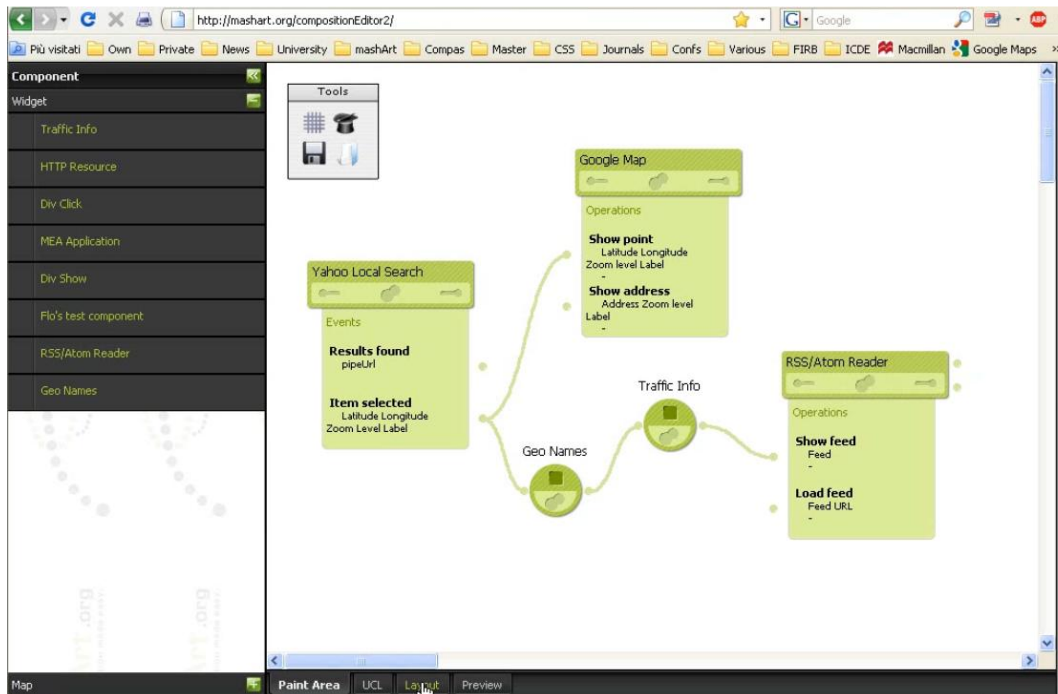

Figura 2.6: MashArt

2.5.8 ServFace

Il progetto ServFace [19] mira a creare una metodologia di reingegnerizzazione dei processi basata sul modello per uno sviluppo integrato di processi per applicazioni basate sui servizi. L'insieme di Service Annotations identificato nel progetto ServFace sono raccolte nel ServFace Annotation Model. Insieme a servizi di descrizione tecnica quali WSDL, rende disponibili gli input necessari per un meccanismo di inferenza di una interfaccia utente automatizzata, che generi interfacce utente di alta qualità per l'interazione tra utenti umani e Web services annotati. Per la composizione di servizi annotati nella direzione di applicazioni complesse, sono investigati in ServFace due alternativi approcci di modellazione:

- Presentation-oriented service composition, nel quale l'applicazione viene modellata visualmente tramite la composizione delle applicazioni dell'interfaccia utente per parti che sono generate utilizzando le annotazioni dei servizi. Inoltre, un tool ServFace Builder è attualmente in costruzione: dovrà integrare un motore per l'inferenza per generare interfacce utente a partire

da servizi annotati. Questo approccio è il solo ad essere più orientato allo sviluppo da parte dell'utente finale;

- Task-oriented service composition: è supportato da un tool chiamato MARIAE [54]. Rende disponibile una nuova soluzione per sfruttare i task di modellazione (rappresentati in notazione ConcurTaskTrees) e i modelli di interfaccia utente (in linguaggio MARIA) per il design e lo sviluppo di applicazioni interattive basate su servizi Web per svariati tipi di piattaforme (desktop, smartphone, vocali, ecc.). Il tool è in grado di importare automaticamente descrizioni di servizi e annotazioni e di supportare l'associazione interattiva di compiti di servizi base con le operazioni di sevizi Web. Inoltre un discreto numero di trasformazioni semi-automatiche sfrutta l'informazione in descrizioni di servizi e annotazioni di questo tipo per derivare dei *front-end* utilizzabili su più apparecchi.

2.5.9 SOA4ALL

Il Progetto di Ricerca Europeo SOA4All [31], propone un'innovativa piattaforma di realizzazione dei servizi fruibile dal maggior numero possibile di utenti grazie alla combinazione delle tecnologie Web 2.0, del context management e della semantica in un'unica soluzione completa e user-friendly. SOA4All mira a realizzare un mondo in cui miliardi di parti espongono e consumano servizi attraverso la tecnologia avanzata del Web. L'obiettivo principale del progetto è quello di fornire un framework globale e le infrastrutture che integrano i progressi tecnici complementari ed evolutivi in una piattaforma di servizio coerente e indipendente dal dominio di consegna. Al fine di massimizzare l'impatto del progetto sul mondo reale, l'utilità della piattaforma SOA4All sarà dimostrata attraverso la collaborazione con le principali imprese europee. In un contesto più ampio, SOA4All contribuisce in modo significativo a rafforzare la competitività del Software e del settore IT a livello europeo. Il framework SOA4All supporta un mondo in cui vi sono un numero massiccio di componenti e servizi da esporre e utilizzare, obiettivo da realizzare tramite una piattaforma coerente e indipendente dal dominio. L'architettura complessiva di SOA4All può essere strutturata in quattro parti principali:

- SOA4All Studio: Una ricca piattaforma Web che fornisce agli utenti una vista unificata che copre l'intero ciclo di vita dei servizi, tra cui le fasi di progettazione, di esecuzione e analisi post-mortem;
- SOA4All Distributed Service Bus: La spina dorsale infrastrutturale attorno alla quale tutti i componenti SOA4All possono comunicare e collaborare unendo spazi semantici e funzionalità di Enterprise Service Bus;
- SOA4All Services Platform: Il gruppo di servizi che forniscono le funzionalità di base e attività, come la Service ranking e la selezione, il servizio di *discovery*, il Servizio di adeguamento, il Servizio di Composizione, il servizio di esecuzione e la reasoning Engine;
- Business Services: I servizi effettivamente forniti dagli utenti finali.

2.5.10 Caratteristiche dei tool a confronto

In questo breve paragrafo riportiamo una tabella comparativa(composta da tre parti (a), (b), (c)) [35] che raccoglie e paragona tutte le caratteristiche di schematizzazione più generali per ciascuna delle piattaforme descritte.

Le dimensioni per il confronto riguardano:

- *Component Model*: indica su che modello si basa il componente, se sia orientato ad un modello rivolto a logiche di business, oppure se si ponga come scopo finale una modellazione volta all'interfaccia utente;
- *Composition Model*: valuta se il modello di composizione sia interamente lato server o se invece sia determinato da specifici eventi;
- *Development Environment*: discrimina i tool in di sviluppo in base al loro ambiente di sviluppo. Le principali distinzioni sono ambiente di sviluppo basato sul browser oppure basato sul server;
- *Programming Efforts*: permette di categorizzare lo sforzo fatto nella programmazione su tre livelli: basso, medio o alto;
- *Composition Runtime*: valuta se il runtime della composizione sia interamente lato server o se invece sia determinato da specifici eventi;

- *Browser Compatibility*: questo parametro valuta la compatibilità del tool con i browser attualmente più utilizzati;
- *Component Lifecycle Management*: esprime la presenza o meno di una gestione del ciclo di vita di ciascun componente della piattaforma;
- *User-defined Component Support*: indica se nella piattaforma sia presente un qualche tipo di supporto per i componenti definiti dall'utente;
- *Event Bus*: valuta la presenza o l'assenza di un bus sul quale viaggino gli eventi;
- *Service Adaptor*: valuta se vi sia o meno una funzionalità che permetta l'adeguamento dei servizi;
- *Cross-domain Handler*: categorizza alternativamente i tool come server-side-proxy o plug-in a seconda della tipologia di handler cross-domain che essi posseggono;
- *Authentication*: infine questa caratteristica indica la presenza o meno di un qualche tipo di autenticazione nella piattaforma.

Caratteristiche	YahooPipes	IntelMashMaker	QEDWiki
Component Model	business logics	-	business logics
Composition Model	server-side	event-driven	server-side
Development Environment	server-based	browser-based	browser-based
Developer Assistances	low	low	medium
Programming Efforts	low	medium	high
Composition Runtime	server-side	browser-side	browser-side
Browser Compatibility	all	Firefox	All
Component Lifecycle Management	yes	no	-
User-defined Component Support	no	no	yes
Event Bus	no	no	yes
Service Adaptor	yes	no	yes
Cross-domain Handler	server-side-proxy	plug-in	server-side-proxy
Authentication	no	no	no

Tabella 2.2: Tabella comparativa dei tool di sviluppo (a)

Caratteristiche	Damia	JBe Presto	Marmite
Component Model	UI	business-logic	UI
Composition Model	server-side	server-side	event-driven
Development Environment	browser-based	browser-based	browser-based
Developer Assistances	medium	high	medium
Programming Efforts	medium	high	low
Composition Runtime	browser-side	browser-side	-
Browser Compatibility	all	all	-
Component Lifecycle Management	yes	no	yes
User-defined Component Support	yes	no	yes
Event Bus	-	no	-
Service Adaptor	yes	no	no
Cross-domain Handler	server-side-proxy	-	server-side-proxy
Authentication	no	no	no

Tabella 2.3: Tabella comparativa dei tool di sviluppo (b)

Caratteristiche	MashArt	ServFace	SOA4All
Component Model	UI	UI	UI
Composition Model	event-driven	event-driven	-
Development Environment	browser-based	browser-based	browser-based
Developer Assistances	medium	medium	high
Programming Efforts	medium	medium	medium
Composition Runtime	browser-side	browser-side	-
Browser Compatibility	Firefox	all	all
Component Lifecycle Management	yes	no	yes
User-defined Component Support	yes	no	yes
Event Bus	yes	no	yes
Service Adaptor	yes	no	-
Cross-domain Handler	server-side-proxy	plug-in	-
Authentication	no	no	no

Tabella 2.4: Tabella comparativa dei tool di sviluppo (c)

2.6 Generazione di reccomendations per la composizione dei mashup

In questa sezione verranno presentati due lavori comparabili con quello svolto in questa tesi. Essi si focalizzano sulla generazione di raccomandazioni a supporto dello sviluppo di mashup, ma non sono orientati alla qualità, uno degli argomenti chiave di questo lavoro di tesi. Come già accennato nell'introduzione, il nostro lavoro è infatti tra i primi ad affrontare il problema della qualità per la composizione di mashup. Tuttavia, i due lavori qui descritti meritano attenzione poiché, anche se da una diversa angolatura, affrontano il problema di facilitare la scelta di componenti e di pattern di composizione per la creazione di mashup. In questo senso, essi possono perciò essere confrontati con il lavoro di questa tesi.

2.6.1 Mashup Advisor

Un tentativo nel creare un ambiente di sviluppo di mashup che faccia delle recommendation è Mashup Advisor [15]. Esso si rivolge ad un pubblico che non deve necessariamente fare parte della categoria dei programmatori e cerca di rendere più rapida ed efficace la composizione di servizi in un mashup. Tutto ciò viene reso possibile da un sistema di raccomandazioni generate automaticamente sulla base di considerazioni semantiche e statistiche sull'uso di componenti in composizioni già prodotte in passato dagli stessi o da altri utenti. Mashup Advisor fornisce delle raccomandazioni a fronte di un cambio di stato durante la composizione del mashup o a fronte di una richiesta esplicita da parte dell'utente. Il processo di raccomandazione è formato da due fasi: nella prima, il tool genera una lista ordinata di componenti in base ad una valutazione, mentre nella seconda viene calcolato il piano migliore per la scelta del prossimo componente in base al mashup parziale già presente. L'architettura di questo tool si compone di quattro componenti: un repository manager, un semantic-matcher, un output ranker e un planner. Il primo analizza la repository e calcola informazioni statistiche sull'uso di componenti in altre composizioni. Il gestore della statistica calcola le statistiche d'uso dei concetti all'interno di Mashup Advisor. Per il calcolo probabilistico, il semantic-matcher può essere configurato in modo da tenere in considerazione quanto due

concetti siano semanticamente simili. Il semantic matcher è usato sia dal planner che dall'ouput ranker e serve a trovare il migliore matching per un dato concetto, calcolando un *punteggio di similarità*. L'obiettivo dell'output ranker è triplice: in primo luogo, questo modulo identifica un insieme di candidati che hanno degli output coerenti con il mashup parziale che l'utente sta componendo e che quindi possono essere aggiunti; in secondando luogo, assegna un punteggio di rilevanza ad ogni candidato; infine, classifica i candidati in base al loro punteggio. L'output ranker seleziona i componenti dal repository manager, escludendo i concetti che appaiono nel mashup parziale dell'utente, sia in forma di output che in forma di input. Per ogni candidato di output, il punteggio è stimato con la probabilità condizionata che quell'output possa entrare a fare parte del mashup, sapendo come è strutturato il mashup parziale del momento. Una volta che l'output ranker raccomanda un numero di concetti di output rilevanti, l'utente può selezionare qualsiasi concetto; Mashup Advisor calcolerà quindi la miglior combinazione per il concetto selezionato. Il tool usa un pianificatore per calcolare il più alto valore di utilità per raggiungere uno scopo (il componente selezionato), a partire dallo stato del mondo (i concetti del mashup parziale).

2.6.2 MatchUp: Autocompletion for Mashups

Il sistema MatchUp [6] si propone di supportare uno sviluppo dei mashup che sia rapido, a richiesta e intuitivo; ciò grazie ad un innovativo meccanismo di autocompletamento. L'intuizione che ha guidato lo sviluppo di MatchUp è che i mashup, anche se sviluppati da diversi utenti, hanno delle caratteristiche in comune: usano delle classi di componenti simili, come è simile il modo in cui vengono legati gli stessi componenti. MatchUp analizza questa similarità per predire, dato un mashup parzialmente composto, quale saranno i più probabili, e potenziali completamenti (includendo componenti e connessioni non ancora inseriti) per quel particolare mashup parziale. Questo sistema è realizzato grazie ad un nuovo algoritmo di ranking, il cui risultato finale è una lista dei *top-k* completamenti da cui gli utenti possono scegliere in base alle proprie necessità.

Capitolo 3

DashMash: verso la composizione dei mashup orientata agli utenti finali

L'approccio descritto per la piattaforma DashMash cerca di rispondere alle necessità spiegate nell'introduzione, incoraggiando la composizione di mashup aziendali in varie direzioni. La più importante è il paradigma di composizione intuitiva che nasconde all'utente la complessità della composizione dei servizi. Questo è possibile grazie al meccanismo per la definizione automatica e basata sul sistema di coppie di servizi per flusso di dati e il controllo di sincronizzazione dei servizi, mentre permette all'utente di customizzare le coppie per aggiungere maggior interattività al mashup composto. Queste funzionalità sono state implementate fornendo modelli e astrazioni che verranno descritti nei paragrafi seguenti. Questi aspetti e le corrispondenti soluzioni saranno descritte nel resto del capitolo.

3.1 La piattaforma DashMash

La piattaforma DashMash [14, 8, 7], costituisce un esempio di piattaforma mashup end-user-oriented, che si pone l'obiettivo di colmare le lacune che in genere impediscono agli utenti finali di sfruttare appieno il potenziale del mashup a livello di strumenti di innovazione. DashMash offre un paradigma di composizione intelligibile e facile da usare, che permette anche agli utenti inesperti di comporre un

mashup proprio. Il suo paradigma è efficace e aumenta la soddisfazione degli utenti finali. Su questa piattaforma si basano gli algoritmi che stanno alla base della presente trattazione, in quanto detti algoritmi cooperano nel perseguire l'obiettivo di fornire possibilità sempre maggiori per gli utenti. DashMash è una piattaforma di mashup Web che consente la creazione di cruscotti per l'accesso e l'analisi di informazioni. In contrasto con l'approccio tradizionale alla costruzione di cruscotti, basato su motori analitici rigidi e monolitici, DashMash si propone di offrire un'insieme di servizi (component services) che l'utente può opportunamente comporre in base a specifiche necessità. DashMash permette all'utente di comporre servizi senza che egli debba necessariamente padroneggiare aspetti tecnici, come ad esempio il passaggio di messaggi e dati che molti utenti non conoscono.

In DashMash la composizione è applicata a servizi adeguatamente registrati e descritti che chiamiamo componenti. I componenti di DashMash possono essere specifici per un dominio o generici. Il primo tipo consiste in servizi creati adhoc per specifici domini per i quali deve esser costruito il cruscotto, per esempio per fornire accesso a sorgenti di dati interne all'organizzazione. Generalmente i componenti di dominio specifici sono costruiti internamente (ad esempio dal dipartimento IT aziendale), mentre i componenti generici corrispondono a servizi pubblici (ad esempio mappe e RSS Feeds). Data la natura *aperta* di DashMash (figura 3.1) (e dei mashup in genere) la natura interna o esterna del componente non costituisce un fattore discriminante.

Dato un repository di servizi pronti all'uso (adattati e descritti), DashMash fornisce un ambiente di tipo *sandbox* dove i servizi possono essere combinati in base a determinate logiche di integrazione. Se da un lato può limitare ciò che l'utente può fare, d'altro canto permette un processo leggero di composizione, rende possibile la generazione automatica dei modelli di composizione e incrementa il grado di controllo sulla qualità del mashup finale. Gli utenti sono liberi i comporre vari tipi di applicazioni soddisfacendo diverse necessità.

La caratteristica principale che distingue la piattaforma DashMash dagli altri ambienti per i mashup è il meccanismo di composizione intuitiva. Come mostreremo in dettaglio nei paragrafi a venire, DashMash è equipaggiato da un ambiente visuale che permette all'utente di combinare i servizi trascinando le icone dei componenti in un riquadro di composizione. La piattaforma è in grado di genera-

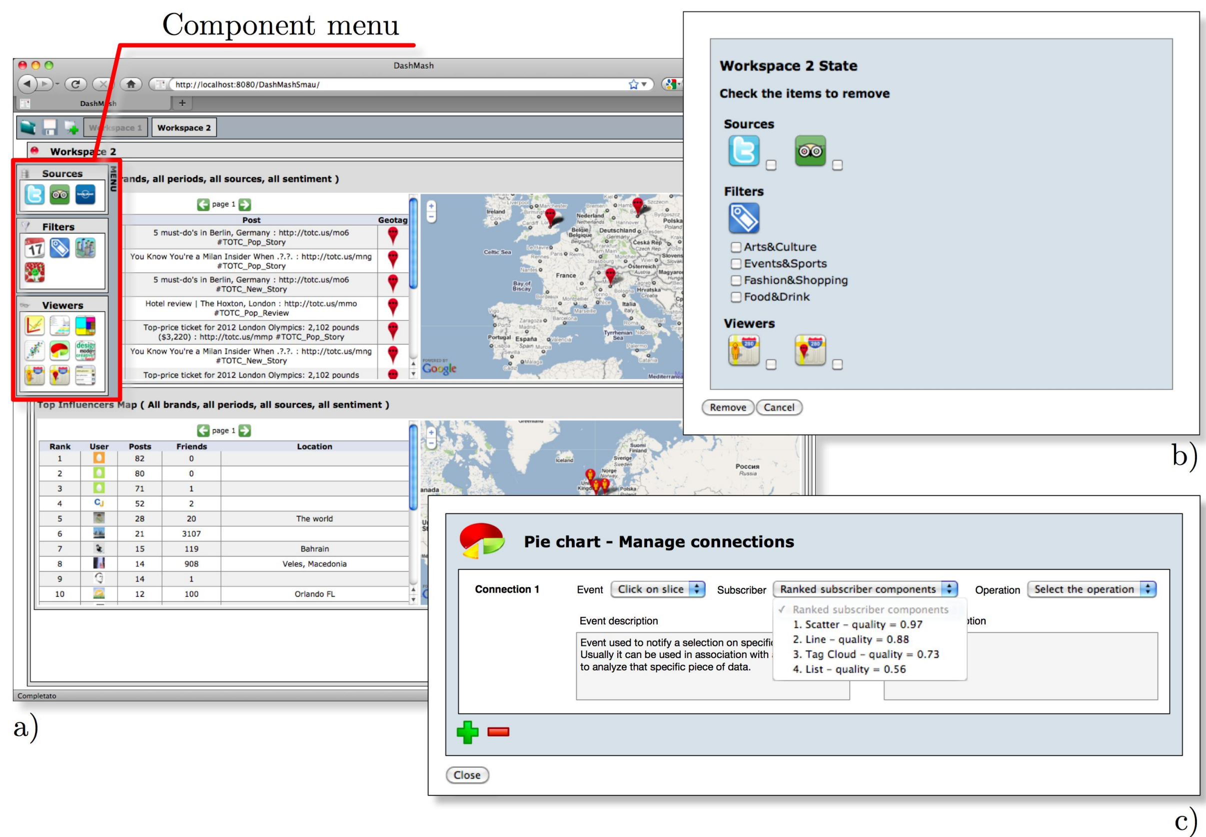

Figura 3.1: La piattaforma DashMash: a)menù dei componenti, b) stato dello workspace, c) gestore delle connessioni

re automaticamente coppie di servizi basati sulla classificazione dei componenti e delle possibili coppie parametri-operazioni che possono essere definite tra essi. Oltre a queste coppie di default la piattaforma supporta la definizione di altre coppie per soddisfare necessità di composizione inaspettate. La loro definizione comunque è anche basata su rappresentazioni intuitive degli elementi coinvolti (chiamati parametri di eventi e operazioni di servizio). L'organizzazione della piattaforma è centrata su un paradigma per l'orchestrazione di componenti che è gestita da un framework intermedio incaricato di gestire sia la definizione della composizione di mashup che l'esecuzione della composizione stessa. A differenza di altre piattaforme dove il progetto del mashup è tenuto separato dall'esecuzione del mashup, in DashMash le due fasi e il rispettivo ambiente sono strettamente interconnessi. L'effetto risultante è che le azioni di composizione dell'utente sono intercettate e automaticamente tradotte in un modello di composizione; questo modello e ogni aggiornamento su di esso vengono eseguiti immediatamente. Gli utenti sono perciò in grado di definire e provare la loro composizione in maniera interattiva e ripetitiva.

3.2 Il modello dei componenti e della composizione

Per raggiungere gli obbiettivi descritti, in primo luogo DashMash trae vantaggio dai modelli utilizzati per la descrizione sia dei componenti che della composizione. Sfruttiamo l'astrazione definita nel progetto Mixup [4], un motore mashup Web la cui composizione logica è basata su un modello a eventi: eventi generati dall'interazione dell'utente con un componente (ad esempio la selezione di una parola su un grafico che rappresenta una nuvola piena di parole) possono essere mappati ad operazioni di uno o più componenti sottoscritti a tali eventi (ad esempio il filtraggio di informazioni sulla base della parola selezionata): l'occorrenza dell'evento perciò causa un cambiamento di stato nei componenti sottoscritti. Il mappaggio tra i componenti viene espresso attraverso l'utilizzo di listener di eventi, ingredienti base per la logica di composizione. I listener di eventi specificano l'evento generato da un particolare componente (componenti *Pubblicatori*) e le

operazioni che il componente ricevente (componente *Sottoscrittore*) deve eseguire.

La composizione logica descritta sopra (figura 3.2) richiede che ogni componente sia descritto attraverso un preciso modello. Attraverso l'indicazione del *binding* con gli attuali servizi/API, il modello del componente specifica gli eventi che un componente può generare e quindi comunica al mondo esterno i cambiamenti dello stato del componente. Data la descrizione astratta di un componente, un modello di composizione specifica quali componenti sono correlati in una composizione di mashup e il modo in cui vengono accoppiati attraverso i listener che mappano gli eventi alle operazioni. I modelli di componenti e di composizione vengono memorizzati in un registro di componenti della piattaforma come descrittori XML. In seguito mostreremo come i due livelli di descrizione forniscono un modello uniforme per la combinazione di servizi e anche permettono la programmazione ad alto livello e la generazione di codice. Un'applicazione composita perciò consiste in uno o più componenti, adeguatamente descritti e un modello di composizione che rappresenta i *binding* definiti. Il framework di esecuzione inoltre offre supporto per:

- la traduzione delle azioni di composizione dell'utente in un modello di composizione;
- la sincronizzazione dei componenti del mashup risultante, come specificato nel modello di composizione creato;
- la gestione del layout dell'applicazione composita attraverso la generazione di template HTML che includono indicatori per agganciare ed eseguire i componenti.

3.2.1 UISDL

In questo paragrafo viene descritta la nuova sintassi con cui si presentano i componenti che popolano la piattaforma *DashMash*, che discende concettualmente dalla sintassi MDL, ma che risulta innovativa per caratteristiche che permettono maggiori performance e danno la possibilità di avere modularità. Le parti salienti del file UISDL sono la descrizione del componente e le operazioni che permettono al

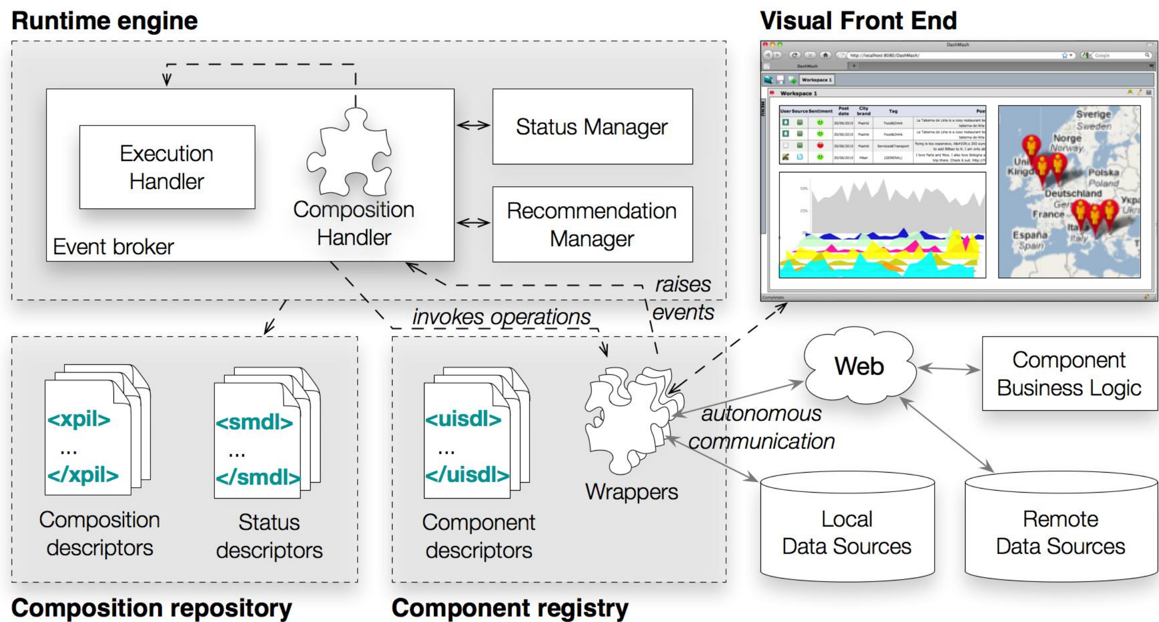

Figura 3.2: Il framework di composizione

framework di interagire con il componente. In breve non si deve fare altro che avere un componente *taggato* in base alle operazioni e agli eventi che esso comprende. A loro volta sia le operazioni che gli eventi possono avere degli output, mentre solo le operazioni possiedono degli input; per ovviare a questa discrepanza si è scelto di assimilare input e output alla più semplice definizione di parametri. Alla luce di questa breve delucidazione non si può fornire migliore chiarimento che un esempio visivo di come sia strutturato un componente, come mostrato di seguito.

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

<uisdl version="0.1">
    <component id="componente_di_prova" name="componente_di_prova" description="
        Composition Handler is very useful component that..." adapter="" address=
        "">

        <operation name="getData" description="this operation get data from.."
            dialog-descripion="get data from..." address="getData">

            <param name="param1" description="parametro 1" direction="input" type
                ="xsd:string" validation-regex="*" validation-minvalue="0"
                validation-maxvalue="5"/>
```

```
            <param name="param2" description="parametro 2" direction="output"
                type="xsd:string"/>
        </operation>

        <operation name="getData2" description="this operation get data from.."
            address="getData">

            <param name="param1" description="parametro 1" direction="input" type
                ="xsd:string" validation-regex="*" validation-minvalue="0"
                validation-maxvalue="5" />
            <param name="param2" description="parametro 2" direction="output"
                type="xsd:string"/>
        </operation>

        <event name="dateUpdated" address="dateUpdated" dialogdescripion="on date
             updated...">

            <param name="response" direction="output" type="xsd:string"/>
        </event>

        <event name="dateRemoved" address="dateUpdated">

            <param name="response" direction="output" type="xsd:string"/>
        </event>
    </component>
</uisdl>
```

3.2.2 XPIL, eXtensible Presentation Integration Language

Gli elementi di composizione descritti possono essere specificati in un determinato linguaggio di composizione, cioè eXtensible Presentation Integration Language (XPIL). Così il linguaggio contiene due insiemi di elementi XML: il primo espone i componenti in uso, e il secondo descrive i modelli di composizione. Il container Xpil è l'elemento originario del documento XPIL. Questo ha diversi attributi:

- Language namespace: http://www.openxup.org/2006/08/xpil/integration
- Xmins:tns: questo attributo contiene il riferimento namespace del componente. Il valore è un URL che può inequivocabilmente identificarlo.

Components: <component>
Questo elemento punta a un componente di rappresentazione del livello ed è definito da questi attributi:

- Id: questo elemento specifica un unico id per il componente nel documento XPIL;

- Ref: questo elemento specifica la posizione di un documento XPIL o UISDL. Il valore può essere un path, una URL, e questo può essere un riferimento relativo o assoluto. Se il valore è riferito per un documento XPIL, il componente è un componente made-up, ma questo sarà trattato come un singolo componente nella composizione. L'elemento componente può contenere una lista di proprietà.

Di seguito mostriamo un esempio di componenti:

```
...
<component ref="../components/compositionHandler/compositionHandler.uisdl"
id="compositionHandler" address="compositionHandler"/>
<component ref="../components/dataService/dataService.uisdl"
id="dataService" address="dataService"/>
<component ref="../components/saveService/saveService.uisdl"
id="saveService" address="saveService"/>
...
```

Event listeners: <listener>
Questo elemento specifica un event listener che collega un evento del componente ad un'operazione o ad un altro componente ed è caratterizzato da questi attributi:

- Id: questo elemento specifica un unico id per l'ascoltatore in un documento XPIL;

- Publisher: questo elemento contiene l'id del componente della risorsa che vara gli eventi;

- Event: questo attributo specifica il nome dell'evento. Il valore si riferisce al nome o attributo del tag di evento nel descrittore del componente. La combinazione tra questo valore e l'attributo publisher identifica l'evento.

- Subscriber: questo attributo contiene l'id del componente che deve far funzionare un'operazione perché l'evento sia lanciato;

- Operation: questo attributo specifica il nome dell'operazione che dovrà essere effettuata dopo la ricezione della notifica di un evento. L'elemento può

contenere XSLT/XQuery per la trasformazione del dati e per la loro conversione. Inoltre, questo può contenere script o riferimenti a codici esterni per aggiungere una logica di integrazione addizionale.

Di seguito mostriamo un breve esempio di event listener:

```
...
<listener id="7" publisher="dataService"
event="dataReady"
subscriber="viewerBox1_viewerPieChartHC1"
operation="getData"/>
<listener id="8" publisher="viewerBox1_viewerPieChartHC1"
event="changeParameters"
subscriber="compositionHandler"
operation="changeViewerParameters"/>
...
```

XPIL rende dunque disponibili metodologie di supporto e rappresentazioni dello schema di composizione mostrando quali elementi sono coinvolti nella composizione e come essi interagiscono (attraverso coppie <event- publisher, operation-subscriber>). Inoltre, XPIL rende possible la descrizione formale e parametrica degli eventi.

3.3 Composizione ed esecuzione del Mashup

Il framework DahMash è un modulo client scritto in Javascript. L'event broker intercetta gli eventi e li distribuisce ai moduli incaricati di gestirli. Gli eventi possono essere relativi alla definizione dinamica della composizione (ad esempio il trascinamento dell'icona di un componente in un'area di composizione) o alle azioni degli utenti e del sistema verificatesi durante l'esecuzione del mashup (questi eventi possono causare modifiche dello stato di alcuni componenti). L'utente interagisce con l'ambiente visuale del DashMash (figura 3.3), in quanto può selezionare i componenti presenti nella barra degli strumenti e aggiungerli alla composizione di mashup. L'aggiunta di un componente si ottiene spostando l'icona in un pannello di composizione chiamato viewer box, che è un contenitore implementato con un <div> HTML che aiuta a contestualizzare l'effetto delle azioni di composizione su alcune sorgenti dati.

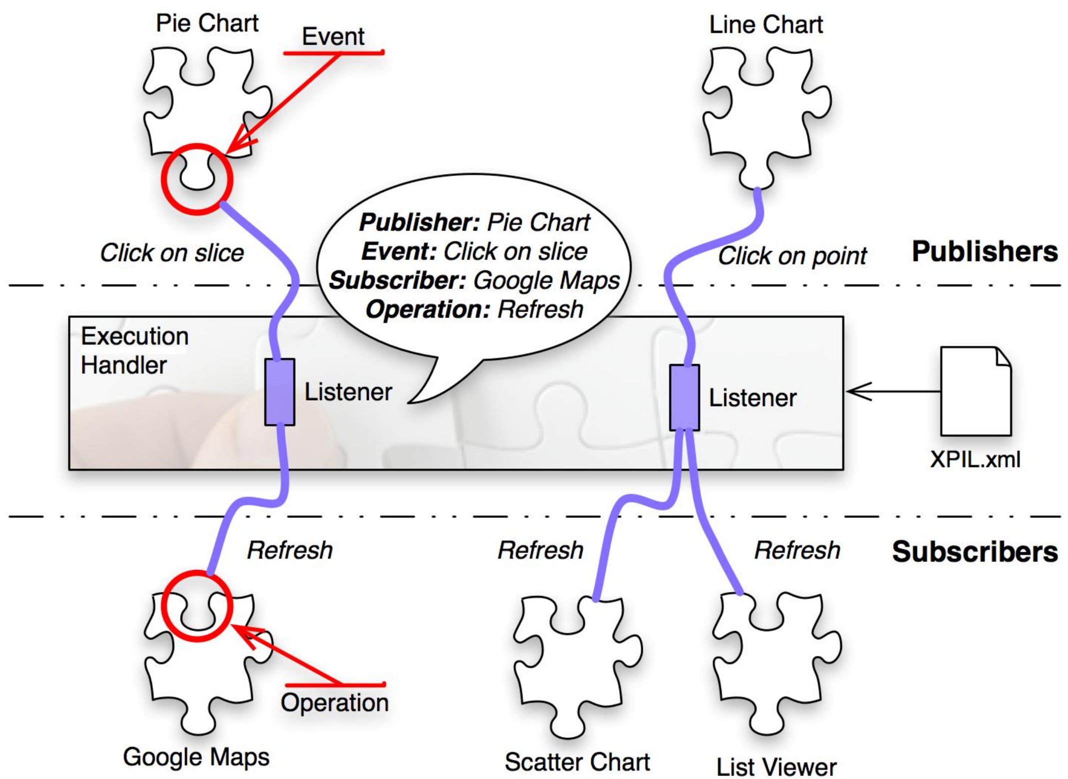

Figura 3.3: Event Bus

Il composition handler gestisce gli eventi di composizione. Trasforma gli eventi in listener e crea o aggiorna (se già esistente) il modello di composizione corrente, aggiungendo puntatori ai nuovi componenti e i listener necessari per collegarli ai componenti già esistenti. Nei paragrafi successivi verrà meglio approfondito il meccanismo spiegato sopra, che comunque risulta del tutto automatico. Il composition handler inoltre invia gli eventi di composizione allo status manager, che ha il compito di mantenere alcune proprietà che possono essere utili per recuperare lo stato di un mashup definito in precedenza, ed eseguirlo più tardi. Queste proprietà dipendono dallo specifico componente, ad esempio i parametri di configurazione che possono essere applicati. Si possono riferire ad esempio a valori di default oppure a valori specifici (il valore dei parametri per un'interrogazione sulle sorgenti dati) a proprietà di layout (ad esempio il colore usato per mostrare i valori di un grafico) o a qualsiasi altra proprietà che l'utente può o vuole settare per controllare lo stato del componente, il contenuto o l'aspetto. Quando l'aggiornamento della composizione e dello stato è terminata, l'execution handler ricarica la composizione e visualizza il mashup. Durante l'esecuzione del mashup i listener vengono attivati dagli eventi corrispondenti. Ogni componente pubblicatore (*Publisher*) notifica all'execution handler il verificarsi di un dato evento. In base ai listener specificati nel modello di composizione l'execution handler a turno informa i componenti sottoscritti, e questo scatena l'esecuzione delle operazioni corrispondenti. La rappresentazione del mashup è ottenuta dall'integrazione di ogni componente dell'interfaccia utente in base a dei template di layout predefiniti, dove ogni componente è eseguito all'interno di un <div> HTML. Gli utenti possono immediatamente vedere il risultato della composizione poiché il cruscotto cambia il proprio stato non appena un nuovo componente viene aggiunto oppure appena viene applicata una nuova configurazione.

Vale la pena notare che il composition handler è anch'esso un componente del mashup: ad ogni azione di composizione dell'utente si genera un evento del composition handler che viene notificato e poi gestito in fase di esecuzione. Un effetto interessante di questa scelta architetturale è che, la gestione degli eventi, qualsiasi sia la natura degli eventi è incapsulata all'interno di un singolo modulo (l'execution handler). Un aspetto ancora più importante e totalmente in linea con la filosofia dei mashup è che la logica della composizione automatica è programma-

bile e perciò flessibile: certamente dipende dall'insieme dei listener predefiniti, la cui conoscenza è all'interno del composition handler, ma essendo un componente del mashup, può essere facilmente tolta e sostituita.

3.4 Generazione del modello di composizione

Per consentire la definizione automatica del *binding* dei componenti, così che all'utente non venga richiesto di programmare esplicitamente i listener, iniziamo con una classificazione dei componenti. E' possibile identificare le seguenti tre classi si componenti:

- i data services sono in grado di recuperare dati dalle sorgenti dati disponibili. Vengono utilizzati principalmente per accedere a database/datawarehouse interni che possono memorizzare dati eterogenei, per esempio contenuti estratti da siti Web che pubblicano informazioni pertinenti, persino commenti generati dagli utenti (ad esempio blog e forum). Un singolo data service è necessario per accedere ai data source interni; quando è richiesta l'integrazione di sorgenti multiple, si può sfruttare un layer di integrazione lato server;

- i filtri aggiungono condizioni di selezione sulle sorgenti dati e consentono un raffinamento interattivo sull'insieme di dati che devono essere analizzati. Eseguono operazioni di filtraggio per ripulire i contenuti delle sorgenti Web sulla base di criteri selezionati (ad esempio le parole che interessano oppure contenuti più recenti).

- i viewers supportano la visualizzazione di dati che possono essere estratti attraverso i data services da sorgenti interne e da generiche sorgenti esterne accessibili attraverso componenti generici. Poiché la visualizzazione dei dati implica alcune forme di aggregazione, i viewers possono anche fornire logiche di trasformazione. Perciò i viewers possono essere semplici visualizzatori di tabelle o di grafici di ogni tipo (ad esempio quelli forniti attraverso le Google Chart API) o un qualsiasi servizio di visualizzazione/aggregazione che possa essere utile per un dominio specifico (ad esempio nuvole di parole per l'analisi del sentiment o mappe per la localizzazione di punti di interesse);

- componenti generici che possono essere integrati per rendere più efficienti i componenti di analisi. Data la natura *aperta* dei mashup, possono riferirsi a differenti funzionalità, ad esempio le mappe, i calendari, il recupero di contenuti multimediali, perfino altri data sources (ad esempio Rss feeds) che possono completare le informazioni estratte attraverso i data services, e possibilmente essere scandagliati attraverso i servizi di analisi, con aspetti addizionali utili. Per questi aspetti, la capacità dell'utente di creare innovazioni è largamente favorita.

Questa classificazione di servizi ci permette di identificare alcuni comportamenti di default che ogni mashup basato su questi servizi dovrebbe avere e che perciò possono essere gestiti automaticamente dalla piattaforma attraverso l'aggiunta di listener nel modello di composizione. Per esempio, i filtri sono produttori di parametri usati per selezionare dati attraverso il data service. I viewers sono consumatori di dati, cioè elaborano i contenuti (possibilmente filtrati) estratti attraverso il data service e producono dati aggregati. Alcuni componenti come ad esempio i data services possono essere sia consumatori che produttori. Oltre al comportamento del flusso dati è necessario modellare la sincronizzazione tra il composition handler e i differenti componenti per permetterci di gestire gli eventi di composizione. La composizione difatti è trattata come l'esecuzione di un mashup. La principale sincronizzazione tra differenti classi di componenti è basata su alcuni eventi sollevati che vengono gestiti dall'execution handler, da cui deriva un semplice modello di composizione. Per esempio, per gestire l'aggiunta di una sorgente dati in un viewer box, il composition handler pubblica l'evento `updateDataService` che scatena l'operzione `setDataSource`, attraverso la quale il data service è configurato per recuperare i dati dalla sorgente dati aggiunta e visualizzarli nel viewr box dove la composizione ha luogo, attraverso un viewer. Analogamente viene definito un listener tra il composition handler e il componente filtro per gestire l'aggiunta o la modifica di un filtro. L'aggiunta di un viewer corrisponde a due differenti listener: un listener che accoppia il composition handler e il data service, per gestire la costruzione delle query aggregate, e un listener che accoppia il data service allo specifico componente viewer per informare il viewer stesso quando il data service ha completato l'estrazione dei dati aggregati necessari e li rende disponibili all'ambiente di esecuzione lato client. I listener relativi al composition handler vengono

aggiunti di default ad ogni composizione. In questo modo il composition handler è in grado di ascoltare qualsiasi evento di composizione. Un altro tipo di listener che deve essere aggiunto automaticamente si basa sullo specifico componente che l'utente inserisce nel mashup. Ad esempio dei listener relativi ai viewer possono dipendere dai parametri o dalle operazioni esposte dal servizio specifico. Infine alcuni listener possono essere aggiunti dall'utente finale per avere sincronizzazioni aggiuntive. Questo significa che l'utente deve dare indicazioni sull'accoppiamento dei componenti. Comunque questo ambiente visuale è ancora in grado di supportare utenti inesperti attraverso dialog-windows intuitive. La conoscenza del possibile mappaggio è codificata all'interno del composition handler e può essere facilmente configurata con il vantaggio risultante che DashMash può essere facilmente adattato a domini con eventualmente diversi servizi e classificazioni. In estremo il DashMsh può anche lavorare in assenza di classificazioni dando all'utente la libertà di accoppiare i componenti in qualsiasi maniera.

Capitolo 4

Il modello di qualità

A questo punto è doveroso dettagliare i concetti di qualità fin ora esistenti e trattati in numerosi articoli per avere le basi necessarie alla comprensione dei passi logici che ci hanno portato all'estensione di tali modelli con l'aggiunta del concetto di Valore Aggiunto. In particolare è utile ridefinire i Mashup:

I Mashups sono applicazioni Web che integrano all'interno di una pagina Web due o più risorse eterogenee a diversi livelli dello stack di applicazione, cioè dati, logica applicativa, e livello di interfaccia utente, possibilmente mettendoli in comunicazione tra loro.

Il motivo di questa raffinazione della definizione data fin ora è quella di porre l'accento sul fatto che i mashup hanno, come abbiamo già detto, una propria interfaccia utente e mirano a fornire un valore aggiunto dato dall'integrazione di più servizi esistenti piuttosto che fare questo attraverso la comune codifica a partire da zero. Se trascuriamo il contenuto statico probabilmente aggiunto dallo sviluppatore del mashup durante l'integrazione (per esempio nel modello di layout del mashup), si posso individuare due distinte ed indipendenti problematiche di calcolo della qualità, nello specifico:

- Componenti: dato che un mashup di dati riutilizza, funzionalità apllicative e/o intefacce utente, la qualità di questi contenuti influenza certamente la qualità del mashup finale. PIù bassa è la qualità dei componenti scelti, più bassa è la qualità del risultata della composizione. Anche se lo sviluppatore

è consapevole della scarsa qualità di un componente, non è sempre possibile scegliere un componente migliroe, ad esempio, perchè non sono disponibili altri componenti che offrono le stesse funzionalità.

- Composizione: mentre di solito non è possibile migliorare la qualità dei componenti di terze parti, dall'altra parte risulta relativamente semplice degratare la qualità potenziale di un mashup, cioè la massima qualità che il mashup potrebbe avere integrando lgi stessi componenti, se la laogica di composizione del mashup non è ben realizzata. Tra l'altro vale la pena mettere a fuoco il fatto che la logica di composizione è in effetti l'unica parte del mashup che non viene riutilizzata e che viene realizzata dalla sviluppatore di solito a partire da zero ogni qualvolta viene realizzato un nuovo mashup. La fase di composizione può essere così suddivisa:

  – Integrazione dati: integrare i diversi componenti può richiedere l'interazione dei data set dei componenti stessi, ad esempio quando un componente è configurato per fornire l'input ad un altro componente. In questo caso è probabile che si richieda una riformattazione dei dati, una pulizia dei dati, un'aggiunta dati ecc.

  – Servizio di orchestrazione e sincronizzazione UI: questo sottotask è il passaggio di dati da un componente ad un altro, cioè la creazione della logica necessaria all'orchestrazione tra i servizi che devono essere sincronizzati oppure alla sincronizzazione delle logiche di interfaccia tra i componenti. Dal momento che le interazioni tra servizi sono tipicamente basati sull'invocazione mentre quelle tra UI sono basate sugli eventi risulta non banale mixare i due aspetti.

  – Lauout: infine, uno degli aspetti più cruciali per il sucesso di qualsiasi composizione è l'aspetto grafico. La pratica comune di sviluppo di mashup è l'utilizzo di modelli HTML. Anche se in prima analisi può risultare un compito piuttosto facile lo sviluppo e quindi l'utilizzo di modelli buoni e che siano in grado di ospitare in modo trasparente interfacce utente di terze parti (possibilmente con le impostazioni personalizzate CSS) è ancora una volta tutt'altro che banale.

Detto questo, ed individuate quindi le peculiarità che caratterizzano uno studio di qualità su queste particolari Web applications, le sfide rimangono quella di valutare la qualità in modo da tener conto di ciò che l'utente percepisce e quindi la sua prospettiva o goal Mashup in relazione a ciò che invece emerge dalla composizione creata, e di dare la possibilità per lo sviluppatore di agire sulla logica di composizione al fine di risolvere problemi legati alla qualità nella maggior misura possibile[47].

4.1 La misura della qualità

Come già accennato nella parte introduttiva è utile fornire una vasta gamma di dati di qualità, che in seguito daranno all'utente non solo informazioni sul dato di qualità complessivo, ma anche sui singoli pesi delle specifiche caratteristiche qualitative. Per spiegare meglio questo concetto è opportuno fare un piccolo esempio. Supponiamo che un utente debba fare la ricerca di un gruppo di ristoranti in un raggio di 3 km attorno alla sua abitazione: si troverebbe a scegliere tra un ricco gruppo di componenti possibili (o addirittura di mashup possibili), proposti ad esempio da Yahoo, da Google o da Bing alternativamente. L'utente medio farà pertanto una analisi della qualità anche in relazione ai suoi bisogni: ad esempio potrebbe servirgli una ricerca molto dettagliata e precisa, ma non troppo costosa, oppure una ricerca molto veloce, anche se costosa. In questo caso è facile vedere che importanza cruciale assume il dato di qualità non solo aggregato, ma anche scisso in tutte le possibili componenti in modo che l'utente possa fare tutte le sue considerazioni, a seconda delle proprie necessità. Nella ampia rosa di possibili valutazioni di qualità, si è scelto di analizzarne un sottoinsieme abbastanza corposo che isola quelle caratteristiche qualitative che sono di maggiore importanza per poter dare dei riscontri concreti all'utente. Si andrà ora a dettagliare il concetto di qualità che, come abbiamo visto pocanzi, è un problema che si propone su due livelli: componente e composizione. In particolare le due parti fondamentali si possono ridefinire come segue: la qualità intrinseca del componente o qualità locale e la qualità dell'intero mashup già componentizzato o qualità globale. Date queste specifiche caratterizzazioni è facile pensare che nello specifico vi sia interesse a

spiegare dettagliatamente i ruoli dei parametri e delle metriche secondo i quali verrà calcolata la qualità del mashup.

Le caratteristiche di qualità esposte sopra sono certamente un consistente gruppo di possibili scelte; ai nostri scopi se ne sceglie un sottoinsieme di specifico interesse, lasciando eventualmente delle possibili estensioni nella sezione relativa agli sviluppi futuri.

Dopo queste premesse si comincia ad introdurre il funzionamento vero e proprio dell'algoritmo per i dati di qualità. In breve l'algoritmo di qualità si basa sull'individuare all'interno del mashup una struttura a grafo in cui i nodi sono i componenti e gli archi sono i collegamenti tra i componenti (i *binding*); in particolare questi archi sono orientati e seguono il flusso di output-input dei componenti in cascata. Per comprendere meglio la struttura e i criteri di importanza di un mashup, è bene descrivere i ruoli principali che un componente può assumere; questi ruoli sono essenzialmente tre e li elenchiamo qui di seguito [3]:

- slave: rappresenta il caso in cui il comportamento di un componente è condizionato dal comportamento di un altro componente; tipicamente il suo stato cambia in relazione al comportamento di un componente master;

- filter: questa categoria ha un ruolo marginale rispetto agli altri due. Il suo scopo è offrire meccanismi di accesso selettivo ai contenuti che un componente può mostrare, cioè l'utente può scegliere quale contenuto visualizzare all'interno dell'applicazione. I componenti filter, cioè, aiutano a ridurre il dataset di altri componenti;

- master: Identifica il componente con cui l'utente interagisce maggiormente, per questo la sua importanza è considerata più alta rispetto gli altri due ruoli.

Apriamo una breve parentesi sulle possibili configurazioni (figura 4.1) nelle quali possiamo trovare i componenti:

- slave-slave: è la configurazione più semplice. Un mashup che integra questo tipo di componenti permette all'utente di interagire con i componenti

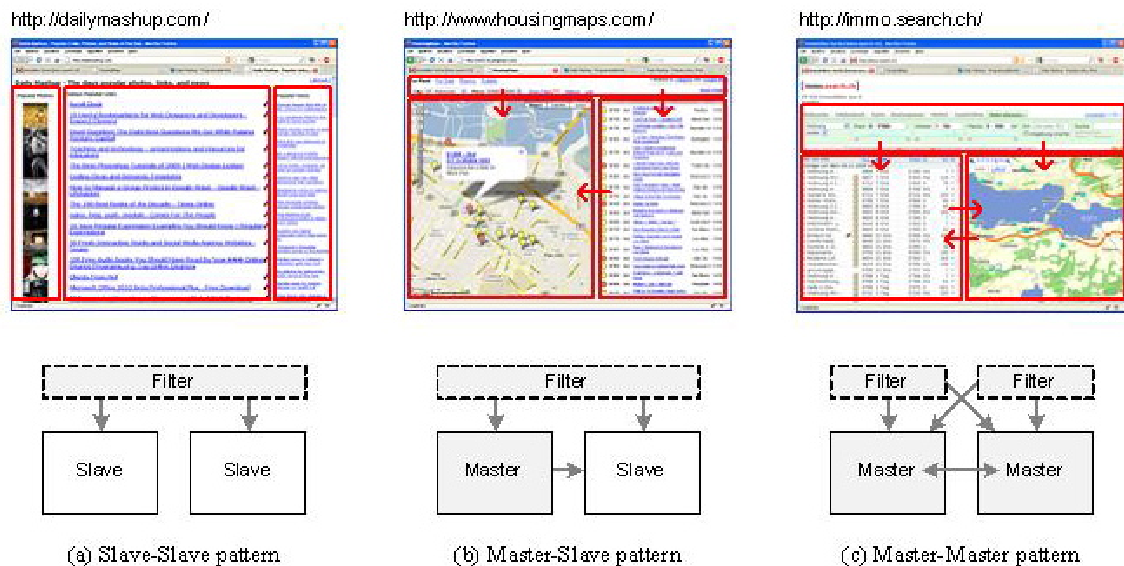

Figura 4.1: Possibili configurazioni dei componenti

presenti, ma in maniera isolata: non c'è flusso di informazioni tra un componente e l'altro. Possono essere usati dei filtri per selezionare il dataset mostrato all'utente;

- master-master: è il percorso più completo, dove oltre ai componenti filtro, gli altri componenti svolgono il ruolo di master. Ciò significa che gli effetti dell'azione eseguita su un componente si ripercuotono su tutti gli altri, che si sincronizzano in accordo ai dati selezionati. Ciò significa che un componente master svolge il ruolo di slave quando l'azione viene eseguita sull'altro componente;

- master-slave: questo è il pattern più diffuso. E' basato su tutti e tre i ruoli visti sopra. Un componente filtro permette all'utente di restringere il set di dati mostrati simultaneamente dagli altri componenti. Il componente master permette all'utente di eseguire l'azione principale, per esempio selezionare il dato di interesse. Il componente slave si sincronizza automaticamente con la

selezione fatta sul componente master, per esempio visualizzando i dettagli dell'elemento selezionato.

Alla luce di ciò, potremo esporre il funzionamento dell'algoritmo relativo al calcolo della qualità. Innanzitutto si calcola tramite una funzione apposita l'importanza di ciascun nodo, in grado di riflettere il ruolo di mastre o slave dei vari componenti. Tale importanza dipende direttamente dal numero di cammini possibili che attraversano il nodo e che collegano le entrate e le uscite dal grafo o mashup. Successivamente si calcola la qualità globale del mashup come media pesata secondo l'importanza delle qualità di ciascun componente.

4.1.1 La qualità del componente

Ci dedichiamo nello specifico alla descrizione del calcolo della qualità del singolo componente, anche svincolato dal mashup.

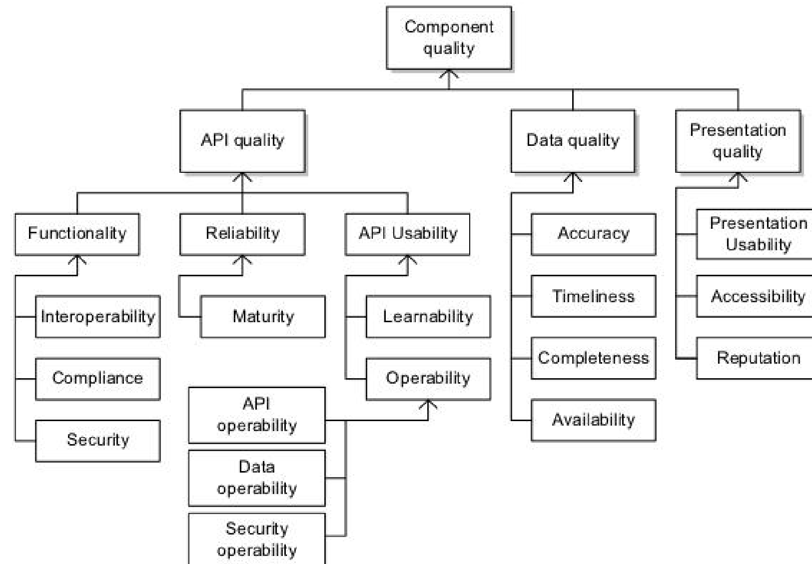

Figura 4.2: Modello di qualità per i componenti

Le dimensioni di qualità del singolo componente (figura 4.2), che non dettaglieremo ulteriormente, in quanto rimandiamo all'ampia documentazione presente[47],

si riferiscono quindi alla fase iniziale, quando il progettista deve selezionare i componenti da integrare. Questa qualità è dunque riconducibile a quella assoluta di un componente, cioè la valutazione delle sue metriche indipendentemente da gli altri componenti scelti per quel particolare mashup. Tale qualità non è altro che una media pesata, secondo pesi accuratamente determinati tramite accurate sperimentazioni, delle singole qualità del componente. Il fatto di tenere disaggregate tutte queste caratteristiche di qualità fornisce anche l'eventuale possibilità di valutare i dati finali del mashup in maniera disaggregata sempre facendo una media pesata a seconda dell'importanza del nodo.

4.1.2 La qualità aggregata

La qualità riferita alla composizione intesa come un involucro di componenti chiuso e non disaggregabile introduce diverse dimensioni di valutazione che mostriamo riassunte in figura 4.3[48].

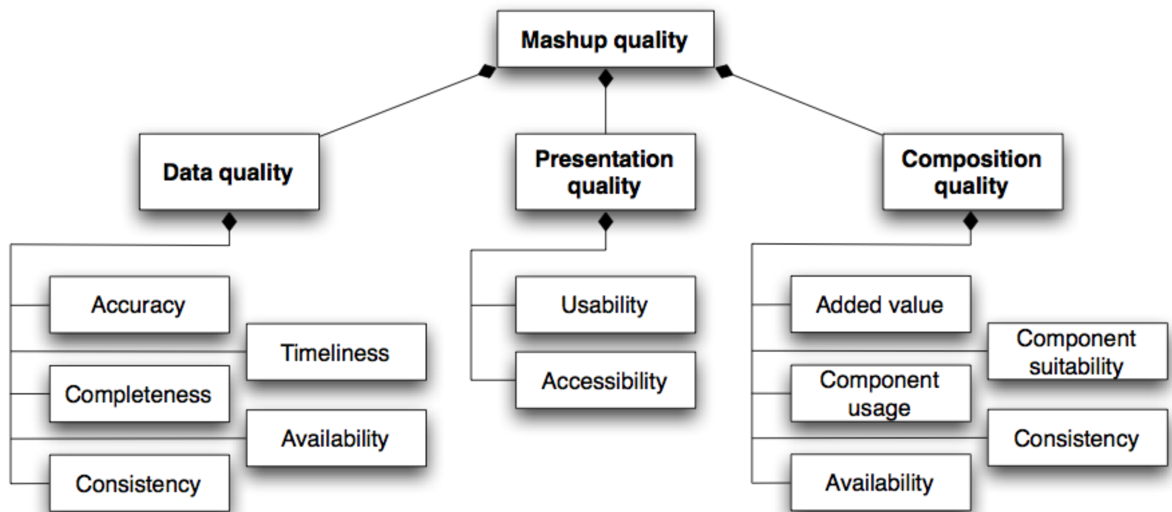

Figura 4.3: Modello di qualità per i mashup

Per quanto riguarda il ***data quality***, esso viene valutato esamindando metriche di *accuratezza*, che si riferiscono alla probabilitù che i dati del mashup siano

corretti, di *completezza*, che indicano la capacità del dataset di fornire informazioni desiderate, di *timeliness*, che valutano la freshness del dataset, di *consistenza*, che identificano situazioni di conflitto tra i dateset del mashup e di *disponibilità*, che si riferiscono alla probabilità che il mashup sia in grado di fornire i dati desiderati.

Il ***presentation quality*** invece si suddivide in *usabilità* e *accessibilità*: la prima dimensione è molto ampia, valuta la facilità di utilizzo dei mashups da parte di un determinato set di utenti e può essere suddivisa a sua volta in altre sottodimensioni quali la comprensibilità, l'apprendibilità e l'operabilità, dimensioni che quindi si riferiscono alla facilità di apprendimento e di utilizzo dei meccanismi di interazione; la seconda dimensione invece è riferita alle proprietà grafiche della pagina, nello specifico quelle stabilite dallo standard W3C.

La terza ed ultima categoria è invece la ***composition quality*** che deve valutare l'adattabilità delle componenti del mashup in riferimento alle specifiche desiderate; si divide in *valore aggiunto*, *adattabilità dei componenti*, *uso dei componenti*, *consistenza* e *disponibilità del mashup*. L'adattabilità dei componenti si riferisce all'appropriatezza delle caratteristiche dei dati e dei componenti rispetto all'obiettivo per il quale il mashup è stato progettato, l'uso dei componenti punta invece a verificare se il componente è usato in modo improprio o meno, sempre in riferimento al goal, la consistenza valuta se la composizione è compatibile sia da un punto di vista semantico che da un punto di vista sintattico, problemi di questa natura potrebbero infatti portare alla produzione di risultati inaccurati, la disponibilità verifica che il mashup sia effettivamente disponibile durante un certo intervallo di tempo e per ultimo introduciamo il valore aggiunto, tema principale della discussione, il quale valuta se la quantità di caratteristiche e di dati offerti dall'intero mashup è maggiore della quantità di dati e caratteristiche fornite da ogni singola componente; se questo valore è maggiore è possibile affermare che il mashup fornisce un valore aggiunto, in caso contrario non è possibile affermare ciò, tratteremo largamente questa definizione nel capitolo 5.

Risulta evidente l'essenzialità di valutare la centralità o importanza di un componente all'interno del mashup, in modo da valutare i suoi dati di qualità anche in

relazione al suo ruolo all'interno del mashup; se un componente di scarsa qualità occupa un ruolo marginale nel mashup, non vi sono grandi problemi, perché il mashup potrebbe utilizzarlo poco e non subire gravi cali di affidabilità o performance. Lo studio della topologia del mashup risulta quindi molto utile per l'utente perché valuta il dato di qualità locale in relazione alla topologia, in modo da rendere più accurato il dato di qualità globale, che viene valutata non solo come semplice media delle qualità dei componenti che lo costituiscono, ma come media pesata delle qualità locali di questi in relazione a specifiche metriche che ne valutano i livelli di centralità e connessione. Ricordiamo che tutti questi studi vengono operati agevolmente tramite l'utilizzo della libreria JUNG [41]. In relazione a questo aspetto ci sono in particolare utili tutti i costrutti relativi ai grafi messi a disposizione dalla libreria JUNG in particolare il metodo che restituisce l'elenco dei successori di un nodo, che verrà ampiamente utilizzato all'interno dell'algoritmo di qualità, nell'ambito delle funzioni di importanza e per il calcolo del numero di cammini passanti per un determinato nodo e che collegano gli input agli output. Risulta evidente che queste misure legate ai mashup, intesi come rete, sono fondamentali per fornire ulteriori funzionalità ad un utente, soprattutto qualora, per esempio, egli voglia sostituire un componente malfunzionante, che abbia alta *importanza* con un altro componente simile.

4.1.3 L'algoritmo

Proponiamo a seguire lo pseudocodice che concerne l'algoritmo di qualità e che in particolare illustra le funzioni di basilare utilità che calcolano rispettivamente il numero di cammini che collegano gli input con gli output, passanti per un dato nodo, e l'importanza di un determinato nodo.

Forniamo alcune spiegazioni preliminari: dato C un insieme di componenti, utilizziamo le seguenti funzioni:

- mashupToGraph(m): funzione che dato un mashup, lo converte in grafo, facendo in modo che i componenti diventino nodi e che i binding diventino archi;

- numCammini(nodoA, nodoB): funzione che dati due nodi, calcola il numero di cammini che li collega;

- localQuality(ci): funzione che calcola la qualità di un singolo componente, valutandone i tag stringa e i tag numerici;
- get[](ci): funzioni che estraggono i valori dei tag numerici e di quelli in formato stringa (dataformat, language, security);
- weightedAverage(accuracy, availability, interoperability, security, timeliness, reputation, completeness): funzione che a partire dai dati di qualità, fa una media pesata secondo opportuni pesi;
- EvaluateSecurity(tagSec): funzione che valuta un dato numerico di sicurezza a partire dal tag di sicurezza;
- EvaluateInteroperability(tagLing, tagDf): funzione che valuta un dato numerico di interoperabilità a partire dai tag di linguaggio e dataformat;
- allNodesWeigtedAverage (Importanza(ci), localQuality(ci)): funzione che dati i valori di qualità dei nodi e la loro importanza calcola la qualità globale del mashup come media pesata secondo l'importanza di ciascun nodo delle qualità dei nodi stessi;
- Importanza(ci): funzione che calcola l'importanza del nodo ci come numero dei cammini passanti per quel nodo che collegano il nodo iniziale e il nodo finale (cioè nodo input e nodo output, nodi accessori del grafo-mashup).

localQuality(c_i)
begin
$getAccuracy(ci) = accuracy;$
$getAvailability(c_i) = availability;$
$getCompleteness(c_i) = completeness;$
$getReputation(c_i) = reputation;$
$getTimeliness(c_i) = timeliness;$
$getTagling(c_i) = tagLing;$
$getTagdf(c_i) = tagDf;$
$getTagsec(c_i) = tagSec;$
$security = EvaluateSecurity(tagSec);$

$interoperability = EvaluateInteroperability(tagLing, tagDf)$;
return localQuality $= weightedAverage(accuracy, availability, interoperability, security,$
$timeliness, reputation, completeness)$;
end
globalQuality(m)
begin
$mashupToGraph(m)$;
return globalQuality $= allNodesWeightedAverage(Importanza(c_i), localQuality(c_i))$;
end
Importanza(c_i)
begin
$Importanza = NumCammini(nodoIniziale, c_i) \times NumCammini(c_i, nodoFinale)$;
return Importanza;
end

4.1.4 La complessità

La complessità dell'algoritmo di qualità si condensa essenzialmente nella funzione *NumCammini*. L'analisi sul grafo si riduce essenzialmente ad una analisi in profondità del grafo, cioè una analisi di tipo *Depth First*. Viene infatti esplorato il grafo andando, in ogni istante dell'esecuzione dell'algoritmo, il più possibile in profondità: gli archi del grafo vengono esplorati a partire dall'ultimo vertice scoperto v che abbia ancora degli archi non esplorati uscenti da esso. Una volta terminata la visita di tutti gli archi non esplorati del vertice v si ritorna indietro per esplorare tutti gli archi uscenti a partire dal vertice da cui un successore di v era stato precedentemente scoperto. Il processo di esplorazione continua fin quando tutti i vertici del grafo non siano stati esplorati.

La complessità del nostro algoritmo si risolve pertanto in una complessità del tipo $\Theta(|V| + |E|)$, dove con |E| si indica il numero degli archi presenti nel grafo, mentre con |V| si indicano il numero dei nodi presenti nel grafo.

Capitolo 5

La generazione di raccomandazioni per la composizione del mashup

Come già asserito nel titolo, questo capitolo si pone l'obiettivo di illustrare i vari passi che compongono l'algoritmo implementato, che supporta il calcolo di compatibilità, similarità e qualità globale dei mashup con particolare focus sul concetto di qualità di un componente integrato in una determinata composizione sulla base del valore che esso aggiunge a questa. Si rimanda al capitolo successivo e all'appendice A per la descrizione dell'implementazione, dell'architettura di riferimento e delle tecnologie utilizzate.

5.1 Il processo a supporto della composizione

Per avere una visione d'insieme elenchiamo qui di seguito i passi di supporto alla composizione, che scandiremo successivamente, di sezione in sezione, rimandando a documentazione per quanto riguarda i concetti già largamente descritti in articoli pubblicati e ai quali si è fatto riferimento, e dettagliando opportunamente quel che è stato il nostro apporto nell'estensione delle metriche del modello di qualità esistente descritto fin'ora e nella logica di reccomendation da noi introdotta.
Come primo passo per supportare la composizione, si va ad analizzare la compatibilità tra due componenti generici: in questa prima fase si opera a livello sintattico, valutando se due componenti dal punto di vista sintattico hanno gli stessi operazioni, eventi, input ed output. L'analisi sintattica di compatibilità vie-

ne poi compattata e sintetizzata all'interno di una matrice di compatibilità, che si forma di elementi dal valore booleano che determinano in maniera esclusiva se ci sia o meno compatibilità. A questo punto nel sottoinsieme di componenti che sono risultati tra loro compatibili si vanno ad individuare i valori percentuali di similarità, che vanno a determinare e ad individuare l'interscambiabilità tra questi componenti e le possibili alternative di scelta tra componenti simili. Sulla base di questo si suggeriscono, a partire dall'alberatura di associazione le cui fasi di creazione verranno descritte qui di seguito, i componenti che risultando avere più occorrenze di associazione all'interno dei mashup più popolari (repository ProgrammableWeb). All'interno di questo sottoinsieme si effettua il calcolo delle misure di qualità che abbiamo descritto nel capitolo precedente che individuano a loro volta i valori qualitativi di ogni componente o gruppo già aggregato di componenti e in particolare aggiungendo la valutazione della dimensione di valore aggiunto come discriminate nella scelta dell'utente tra un componente e un altro da integrare nella composizione per avvicinarsi in misura massima al *Goal Mashup* i tag relativi al calcolo della qualità,

5.2 Compatibilità e Similarità

L'algoritmo per la valutazione della ***Compatibilità*** tra componenti ha come risultato la matrice di compatibilità. La valutazione della compatibilità viene fatta in base ai tipi di ciascun input, output o operazione, evento di ciascun componente, cioè solamente a livello sintattico. Si ottiene compatibilità solo nel caso in cui ci sia una corrispondenza perfetta tra tutti i tipi presenti. A rigore di precisione è bene fare una distinzione tra parametri obbligatori e parametri opzionali; questi ultimi infatti non costituiscono un ostacolo per la compatibilità, in quanto essendo solo opzionali assumono un ruolo meno importante rispetto a quelli obbligatori, i quali invece costituiscono il nucleo fondante per la valutazione della compatibilità. Per rendere le cose più semplici si può supporre di avere una matrice che sarà riempita di 0 e 1; si avrà uno 0 nel caso di assenza di compatibilità, mentre si avrà 1 in caso contrario. Si ottiene così una matrice sparsa, che accentuerà tanto più questa caratteristica, quanto più numerosi risulteranno i componenti da analizzare.

L'analisi di similarità sarà pertanto condotta semplicemente per quelle coppie di elementi, in corrispondenza dei quali si trova un 1 nella matrice di compatibilità.

La matrice di compatibilità risulta strutturata su confronti tra coppie di operazioni ed eventi e varia a seconda che si stia analizzando la compatibilità seriale o quella parallela. Per avere un riferimento chiaro proponiamo un esempio generico nelle tabella 5.1 (la tabella di compatibilità può essere anche seriale). L'utente dunque non solo potrà visionare i dati disaggregati, ma anche il valore aggregato di compatibilità[1].

	$(O\|E)_{1_{c1}}$	$(O\|E)_{2_{c1}}$	...	$(O\|E)_{n_{c1}}$	...	$(O\|E)_{1_{cj}}$	$(O\|E)_{2_{cj}}$	...	$(O\|E)_{n_{cj}}$
$(O\|E)_{1_{c1}}$	1	1	1	0	0	1	0	0	1
$(O\|E)_{2_{c1}}$	0	0	0	1	0	0	0	1	0
...	1	1	1	0	0	1	0	0	1
$(O\|E)_{n_{c1}}$	0	0	0	1	0	0	0	1	0
...	0	0	0	0	1	0	0	1	0
$(O\|E)_{1_{cj}}$	1	1	1	0	0	1	0	1	0
$(O\|E)_{2_{cj}}$	0	0	0	1	0	0	0	1	0
...	1	1	1	0	0	1	0	0	1
$(O\|E)_{n_{cj}}$	1	1	1	0	0	1	0	1	0

Tabella 5.1: Matrice generica della compatibilità parallela

Dopo aver ottenuto la matrice di compatibilità, si va a stabilire la matrice di ***Similarità*** relativamente a quel sottoinsieme di componenti che sono risultati compatibili. La similarità che si ottiene con l'algoritmo formulato prevede due parti distinte che vanno poi a comporre la matrice di similarità: una prima parte prevede appunto l'utilizzo di WordNet mentre la seconda è composta tramite l'uso delle ontologie. Come sottoinsiemi dei synset (Appendice A: WordNet) si sono individuati verbsynset e nounsynset che rispettivamente permettono di concentrarsi specificamente sui verbi e sui nomi. La parte di similarità ottenuta con WordNet è atta soprattutto a fornirci un livello di similarità dei tag "operazione" ed "evento" che sono posti nei componenti e che individuano l'azione che il componente supporta. L'algoritmo usato per questa valutazione prevede di partire da due synsets in principio e di prendere le wordform di ciascuno di questi due e successivamente

effettuare un confronto e valutare quante corrispondenze si individuavano. Si passava dunque ad individuare gli hyperonimi di ciascuna delle componenti facenti parte di entrambe le wordform. In questo modo si procede pensando a WordNet come ad un grande e articolato albero semantico, che assume forme particolari a seconda dei vocaboli che si stanno analizzando. Individuati gli hyperonimi si ripete l'analisi di individuazione delle corrispondenze dei due insiemi di hyperonimi in modo da valutare la similarità ad un ulteriore livello di dettaglio. All'interno degli hyperonimi si includono le wordform in modo da tenere traccia dei livelli precedenti e non perdere l'eventuale similarità tra due concetti o meglio tra due termini che sono uno sottoclasse dell'altro. Per quanto riguarda i nomi si di risale gli hyperonimi fino al terzo livello, cioè quattro livelli totali di dettaglio, uno fornito dalle wordform e tre ulteriori livelli di dettaglio forniti dagli hyperonimi. Per ottenere dei risultati fruibili e sensati da WordNet si danno dei pesi a ciascun livello indagato in modo da trovare dei risultati ottimali e da ridurre al minimo il livello di errore tra la similarità attesa e la similarità stimata con Wordnet. E' utile avere dei coefficienti sia per i verbi, che per i nomi, dal momento che le operazioni sono generalmente espresse o come semplice verbo, o come verbo e oggetto (ad esempio *find* o *find restaurant*). Per chiarire questo procedimento mostriamo in figura un esempio relativo alla similarità tra verbi i cui pesi scelti sulla base delle prove effettuate sono seguenti:

- peso relativo alle wordform: 0.1
- peso del primo livello di hyperonimi: 0.8
- peso del secondo livello di hyperonimi: 0.1

A partire da questi risultati si ottengono delle tabelle di similarità che come è stato provato[1] mostrano risultati di similarità soddisfacenti ed inerenti a quello che realmente ci si aspetterebbe di individuare.

Il calcolo della similarità dei parametri dei componenti, cioè gli input e gli output, viene determinata tramite l'utilizzo delle ontologie (Appendice A). Di nuovo si deve immaginare di avere un albero, solo che in questo caso l'albero viene costituito tramite le ontologie, invece che con Wordnet. È noto che tutti i concetti di una ontologia discendono più o meno direttamente dal concetto "thing",

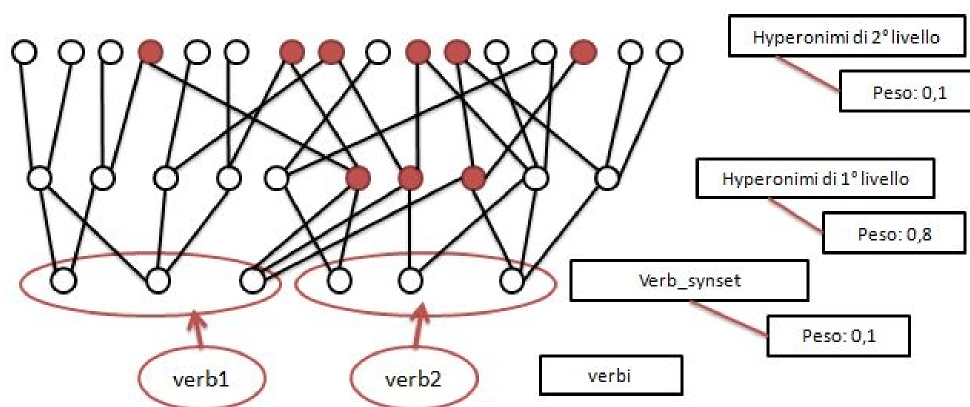

Figura 5.1: La similarità dei verbi con WordNet

a partire da questo si diramano tutte le componenti lessicali e si specializzano o nella stessa ontologia o in ontologie più specifiche. Risalendo l'albero generato dai due termini che si stanno esaminando, si incontrerà prima o poi un nodo in comune; una volta raggiunto il nodo si stimerà la distanza dei due termini dal nodo individuato (esempio in figura 5.1).

A questo punto si riesce a calcolare la *similarità finale*, cioè quella composta congiuntamente dalla similarità dei parametri (input e output) e quella delle operazioni, o equivalentemente quella componente di similarità che si appoggia alle ontologie e quella che si appoggia a WordNet, tramite una media pesata tra queste due. Questa metodologia permette di aggiungere il primo tassello per fare *reccomendation* nei confronti dell'utente che sta componentizzando il suo mashup, eventualmente anche se si tratta di un utente con scarsa competenza, metodologia che verrà ampliata con le reccomendation guidate dalle regole di associazione e dal calcolo del valore aggiunto, concetti ai quali si giungerà attraverso i passi logici descritti qui di seguito.

5.3 Categorizzazione

A partire dallo scenario descritto fino ad ora si è impostato un problema di ricerca per dettagliare il modello di qualità e il calcolo della stessa. Per fare questo si è reso necessario fin da subito un repository al quale poter attingere per i dati necessari allo studio. Lo scenario odierno offre un unico repository sufficientemente ampio, che è stato utilizzato come dataset per la nostra attività di studio: www.programmableweb.com. Quando si parlerà di mahup analizzati e mining sui dati ci si riferirà quindi a questo repository che vantando una raccolta di 6280 mashups e 4292 APIs suddivisi per descrizione, popolarità, rating, tag e categoria, ad oggi risulta essere la più vasta raccolta Web Mashup.
Analizzando il modello di qualità a cui ci si è riferiti fino a questo momento emergono delle incompletezze soprattutto per quanto riguarda l'aspetto della qualità di composizione. Mentre data quality e presentation quality mostrano e danno spazio a metriche piuttosto sistematiche in riferimento ad accuratezza, completezza, dimensioni temporali, affidabilità, consistenza e usabilità, accessibilità come da figura 4.3 (Modello di qualità per i Mashup). Il composition quality, descritto come abbiamo visto dalle dimensioni di valore aggiunto, idoneità, utilizzo, affidabilità e consistenza dei componenti, si discosta dalle metriche di qualità dei dati e qualità di presentazione perchè più astratto. Si pensi per esempio alla facilità di quantificazione della completezza di un mashup oppure l'usabilità dello stesso rispetto al valore aggiunto che la sincronizzazione di più API possa dare all'utente finale rispetto all'utilizzo singolo di ogni componente del mashup. Come primo step dello studio si sono analizzati i mashup del repository di Programmable Web effettuando una scrematura innanzitutto in base al rispetto della definizione di Mashup quindi sincronizzazione di due o più APIs quindi applicazioni che utilizzano contenuti di più sorgenti diverse (tipicamente di terzi) ottendendo un qualche servizio completamente nuovo. Ottenuto il dataset più attinente possibile alle necessità del nostro studio, si s̀ono raccolti dati inerenti alla popolarità alle API e alle categorie di API occorse più di frequente. In particolare si è analizzato per ogni API quali funzionalità venissero offerte e raccolte queste si è effettuato un lavoro di elevamento a gruppi di funzionalità o macro-funzionalità che ci hanno permesso una categorizzazione delle stesse. Ad esempio la categoria che ha tota-

lizzato la maggior percentuale di occorrenze nei mashup del repository, *Mapping*, ha tutta una serie di funzionalità di ricerca itinerari: percorso in auto, percorso con i mezzi percorso a piedi ecc. tutti catalogati nella macro-funzionalità *itinerari*; tutto quel che riguarda i servizi di ricerca di una posizione (con indirizzo, con latitudine/longitudine, ecc) sono racchiusi dalla macro-funzionalità *mapping* e così via per tutte le categorie e tutte le funzionalità.

Come si può notare dalla tabella 5.2,3 le categorie risultano in alcuni casi molto generiche es. *Others* o *Reference*, si è scelto questo per rimanere coerenti con il repoisitory di riferimento e le categorie presenti in esso. Naturlamente un repository pensato per uno studio di questo tipo avrebbe della categorie molto più specifiche per permettere delle associazioni tra macro-funzionalità a sua volta più specifiche. Come si descriverà nel sesto capitolo il repository ideale conterrebbe una collezione di categorie scelte sulla base delle funzionalità più utilizzate e che al massimo caratterizzino la/le macro-funzionalità associate. Dai risultati è emerso come nello scenario dei Web Mashup odierni alcune categorie di funzionalità e quindi di API siano nettamente superiori in numero di occorrenze rispetto ad altre (tabella n. 5.2,3) ad esempio Mapping, Shopping, Social, Video, Photo occupano da sole la percentuale del 59% del totale delle categorie. Questo risultato vista la fase di pre-analisi per la scrematura del data set era un risultato che ci si aspettava in quanto la grande maggioranza dei Mashup ad oggi attivi forniscono servizi di mapping, integrazione con il mondo social, piuttosto che servizi di acquisto on line o visualizzazione di immagini e video contestualizzati in mappe customizzate. Il fatto però che una categoria abbia più occorrenze in assoluto all'interno del dataset di Mashup analizzato non è necessariamente indice di maggior qualità dei componenti appartenenti a questa categoria più popolare, rispetto ad altri. Infatti quesi risultati non hanno la presunzione di rappresentare il fatto che la popolarità è misura di qualità bensì che una maggior occorrenza di una determinata API di categoria a nel data set di mashup considerato, può essere interpretata come indice di interesse dell'utente finale alla suddetta categoria di componenti rispetto a componenti della categoria b meno popolare di a. Di conseguenza le funzionlità offerte dalla categoria a sono di maggior interesse per l'utente finale rispetto a quelle di categoria b nella misura in cui la categoria a occorre in percentuale

Category	Functionality	% Occorrenze di categoria	% Occorrenza di categoria most popular	Value Of Category
Advertising	Create Advertisers (also for blogging, web publishing, forum, wiki, comunity) Manage Advertisers (publishers, insertion orders, line items, campaigns, creatives, pixels, target profiles etc) Search Engine	1,05%	0,99%	1,02%
Answers	Search Engine Make Questions	0,25%	0,00%	0,12%
Blog Search		0,66%	1,99%	1,32%
Blogging	Archive RSS Custom Feeds and management tools to bloggers, podcasters, and other web-based content publishers Create Blogs (post new blogs) Manage Blogs (edit or delete existing posts, query for posts, see followers)	0,97%	2,32%	1,64%
Bookmarks		1,50%	3,97%	2,74%
Calendar	Create Calendar Events Manage Calendar Events (edit, update, quering, invite)	0,34%	0,66%	0,50%
Chat	IM	0,51%	0,33%	0,42%
Database	Data management (upload dati Quering Download Synchronization data	0,63%	0,33%	0,48%
Email	Compose and Send messages Manage folders Notification Migration	0,27%	0,00%	0,14%
Enterprise	Create Account Synchronization data	0,58%	0,33%	0,46%
Events		0,90%	0,33%	0,62%
Fax		0,01%	0,00%	0,00%
Feeds	Download feed (Atom, RSS)	0,51%	0,99%	0,75%
File Sharing		0,06%	0,00%	0,03%
Financial	Visualization finantial data Aggregation feeds news	0,19%	0,00%	0,09%
Food		0,07%	0,00%	0,03%
Games		0,20%	0,00%	0,10%
Government		0,50%	0,33%	0,41%
Internet	Download information (access etc) Quering Custom gadget	3,07%	1,32%	2,20%
Job Search		0,26%	0,66%	0,46%
Mapping	Mapping (Search by address, Latitute/Longitude etc) Geolocalizzazione Visuale satellite Visuale mappa Visuale ibrida Visuale earth(3D models, KML) Elevation data for all locations (including depth locations on the ocean floor) Zoom Street View Creazione Itinerari (auto, mezzi pubblici, pedonali) Traffico Integrazione Info(foto,webcam,wiki) Draw markers and lines Geotag	26,46%	34,11%	30,28%
Media Management		0,33%	0,33%	0,33%
Medical	Create and Manage medical records (history etc) Search Engine	0,09%	0,00%	0,04%
Messaging		1,87%	0,33%	1,10%
Music	Search Engine Quering (structured and unstructured query by charts, similaries, genres, ratings, artists) Create Catalog of Music (artists, album, tracks, videos, ratings)	3,70%	3,97%	3,84%
News		1,29%	0,99%	1,14%
Office	Searh Engine Create Documents (upload) Manage Documents List documento on "GData" feeds	0,51%	0,33%	0,42%
Other	Interactivity Choose look and feel RealTime Data Create Chart (Pie, Line, Culomn, Graph etc) Manage Contacts Quering Synchronization IM Font Library Custom Web Portal Custom Gadgets Virtual Keyboard OnScreen Traffic Search Engine	2,68%	1,66%	2,17%
Payment	Create Account Choice differently-priced subscription	0,33%	0,33%	0,33%
Photos	Create Web Album (upload photos and video) Manage Web Album Create Collage Manage Photos (list, delete, update) Print Photos Order Prints Blogging	7,00%	8,94%	7,97%
PIM		0,11%	0,00%	0,06%
Project Management		0,15%	0,00%	0,08%
Real Estate		0,40%	0,33%	0,37%
Recommendations		0,41%	0,00%	0,21%

Figura 5.2: Tabella Cateogorizzazione Funzionalità (Parte 1)

Reference	Create Library Manage Library (modify library collections, ratings, labels, reviews) Autentication Search Engine Search Engine via Feeds Get Library collections Get Public Reviews Diacritics simbol	**1,94%**	**1,32%**	**1,63%**
Retail		**0,00%**	**0,00%**	**0,00%**
Search	Search Engine (Web,Local,Blog,Images,Videos,Books) Customer Search Engine Inteegrate results (in GoogleMaps API or in web site or MU) Custom Gadets (sidebar) Redefine Desktop UI Download (pdf) Documentation and blog for developers Repository Quering	**7,38%**	**8,94%**	**8,16%**
Security	Authorization and Autentication (Google Account) Prevention from pishing e malware Download crypht table Captcha Verification	**0,43%**	**0,00%**	**0,21%**
Shipping		**0,33%**	**1,66%**	**1,00%**
Shopping	Autentication Search Engine Comparization products Quering Charge Credits Card Process orders Products List Manage data Manage item	**8,14%**	**10,93%**	**9,53%**
Social	Autentication Create Account Control access Insert multimedial contents (videos, images, lists) Sharing (updates, photos, videos, status, conversation, documents) Community Social Network (connected contacts, update information from them) Blogging Custom Gadets (sidebar) Search Engine	**9,98%**	**2,65%**	**6,32%**
Sports	Show statistics data	**0,09%**	**0,00%**	**0,05%**
Storage	Data Storage	**1,35%**	**0,66%**	**1,00%**
Tagging		**0,03%**	**0,00%**	**0,02%**
Telephony		**2,13%**	**0,00%**	**1,06%**
Tools	Create and Manage content Manage archive Monitoring Upload/download attachments Custom gadgets Machine learning and data mining Add comment	**1,03%**	**0,00%**	**0,51%**
Travel	Trip planner Quering Search Engine	**0,38%**	**0,33%**	**0,35%**
Utility	Translate Quering Manage information	**0,76%**	**0,00%**	**0,38%**
Video	Upload Videos Create User Account (Comments, playlists and subscriptions) Search Engine	**6,23%**	**5,96%**	**6,09%**
Weather	Weather information	**0,52%**	**1,32%**	**0,92%**
Widgets	User preferences Create and manage gadgets	**1,28%**	**0,33%**	**0,80%**
Wiki		**0,13%**	**0,00%**	**0,06%**
		100,00%	100,00%	100,00%

Figura 5.3: Tabella Cateogorizzazione Funzionalità (Parte 2)

minore della b. Ad esempio notoriamente il Social Network Facebook, che non è un mashup, ma integra contenuti social, non è un'applicazione di qualità però, secondo i dati forniti da Mark Zuckerberg (fondatore), il numero degli utenti attivi nel luglio 2011 è 750 milioni. A questo punto lo scenario che si presenta è sufficientemente dettagliato per poter improntare sulla base di questi dati uno studio sulla ricerca di risultati di qualità riferibili al valore aggiunto che l'integrazione di API di categorie diverse può dare rispetto ad altre configurazioni di categorie di funzionalità passeremo quindi ad illustrare lo studio di mining che ci ha portato hai risultati di associazione tra componenti e ad una valutazione sistematica della dimensione di Added Value.

5.4 Mining

Dopo aver analizzato il repository di ProgrammableWeb e le sue categorie si è deciso di analizzare eventuali associazioni presenti tra le categorie in questione per poter in seguito effettuare una raccomandazione dei componenti durante la creazione dei mashup. E' evidente che dato un repository cosi vasto la probabilità di ottenere delle associazioni veritiere sia molto alta, il campione di dati difatti è molto numeroso e le capacità di calcolo attuali ci permettono di formulare diverse regole di associazione.

5.4.1 Weka e la creazione del file ARFF

Il software utilizzato per svolgere lavoro di mining si chiama Weka, acronimo di Waikato Environment for Knowledge Analysis, è stato sviluppato in Nuova Zelanda, scritto in Java e permette di applicare metodi di apprendimento automatici ad un set di dati; attraverso questi è quindi possibile avere una previsione dei nuovi comportamenti dei dati. Il dataset è composto da valori e attributi presenti all'interno di una relazione, le istanze corrispondono alle righe e gli attributi alle colonne. Weka utilizza il formato ARFF, cioè Attribute Relationship File Format, per leggere ed analizzare il dataset; all'inizio del file vengono definiti gli attributi della relazione, che potranno essere valori binari, classi di valori, variabili booleane, e successivamente alla definizione degli attributi viene definita la sezione

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
```

Per elaborare il file ARFF si è partiti da un database popolato attraverso un crawler, con i dati sui mashup disponibili nel repository di ProgrammableWeb. Per definizione un crawler è un software che analizza i contenuti della rete (o di un database) in modo metodico e automatizzato. Il crawler da noi utilizzato è stato adattato alle esigenze di dati emerse dall'impostazione di questo lavoro sperimentale. Ciò è stato realizzato attraverso l'implementazione della connessione al repository di ProgrammableWeb e del download delle informazioni sui mashup. I dati di dettaglio delle caratteristiche dei mashup vengono raccolte in un database progettato adhoc, di cui si illustano le tabelle più significative per le successive fasi di mining:

- *made*, che contiene per ogni mashup tutte le api da cui è composto (una tupla per ogni api del *mashup X*, quindi, se X ha 5 componenti, avremo 5 tuple per di dettaglio di questo mashup);

- *apis*, che contiene per ogni API del repository la categoria di appartenenza e altre informazioni di dettaglio ad esempio tags,data,descrizione,protocollo ecc.;

In prima analisi è stato effettuato del mining partendo dalle APIs, era stato ritenuto corretto infatti ragionare sulle singole componenti, salvo poi in un secondo step cambiare metodologia e decidere di effettuare mining ad un livello di aggregazione maggiore, lavorando quindi sulle categorie. Sono state considerate tutte

le categorie, con ripetizioni, che costituiscono ogni singolo mashup, quest'utimo infatti non ha una categorizzazione precisa e ciò discende direttamente dalla definizione di mashup, ovvero un'integrazione di servizi offerti all'utente, sincronizzati fra loro. Ogni servizio, nel nostro studio, è strutturalmente definito come oggetto o componente, il quale si traduce in un'API che può essere categorizzata secondo le più comuni categorie, come parlato in precedenza. E' stato ritenuto utile, per seguire questo approccio con livello di aggregazione su categorie, realizzare un join tra le tabelle *made* e *apis* appena descritte. E' stata quindi ottenuta una nuova relazione che presenta per ogni mashup non solo le api che lo compongono (tabella *made*) ma anche la diretta associazione con la categoria di ogni componente. Questo ha permesso la creazione del file che è stato dato in input a Weka per ottenere le associazioni di categoria più popolari. Prima di procedere alla creazione del file è stato estratto un campione delle categorie piů rilevanti e frequenti su un totale di sessanta categorie e tra queste ne sono state scelte diciannove.

```
Advertising, Blogging, Bookmarks, Events, Internet, Mapping,
Messaging, Music, News, Other, Photos, Reference, Search,
Shopping, Social, Storage, Tools, Video, Widgets
```

Successivamente sono state definiti otto attributi categoria; analizzando infatti i mashup presenti nel repository si è osservato che nella maggior parte dei casi, il numero massimo di categorie per mashup è tre; questo è possibile riscontrarlo anche dalla seguente immagine Si nota infatti che a partire dalla categoria 4, i valori riferiti a questa categoria mancano per il 91% dei casi, quelli relativi alla categoria 5 per il 94%, la categoria 6 è mancante nel 97% dei casi e le categorie 7 e 8 sono mancanti entrambi nel 98%. Se fossero state considerate piu' categorie, i valori percentuali sarebbero arrivati anche al 100%; eventualmente si sarebbe potuto analizzare anche un numero inferiore di attributi, ma al fine di ottenere un'analisi il piu' precisa possibile e che potesse considerare la maggior parte dei casi, si è deciso di considerare questo numero di categorie.
Una volta definiti questi dati, è stato sviluppato un tool che è in grado di automatizzare la creazione del file ARFF, a partire da una tabella relazionale. Eseguendo il tool, dopo aver invocato start viene generato il file e la schermata che viene visualizzata è la seguente Una volta elaborato il file, si puň notare che ogni ri-

ga corrisponde ad un mashup analizzato, e successivamente al nome del mashup vengono visualizzate le categorie presenti in esso, espresse attraverso un indice compreso tra 1 e 19, invece nel caso di categorie mancanti, viene visualizzato il simbolo '?'. Al fine di ottenere dei risultati il più attendibili possibili si è deciso di applicare il *Principio di Pareto* o *Legge 80/20*, una legge empirica che è sintentizzata nella frase *la maggior parte degli effetti è dovuta ad un numero ristretto di cause*; naturalmente i valori 80% e 20% sono ottenuti mediante osservazioni empiriche e sono solo indicativi, ma è interessante sapere come numerosi fenomeni abbiano una distribuzione statistica in linea con questi valori. Si è quindi deciso di adottare questo principio e sono stati generati due file, uno comprendente l'80% dei valori del file ARFF e l'altro contentente il rimanente 20%. Si è potuto notare che la maggior parte dei risultati ottenuti dall'analisi del file contentente l'80% dei valori, sono state riscontrate anche nell'altro file, questo a validazione della correttezza delle regole ottenute attraverso il mining.

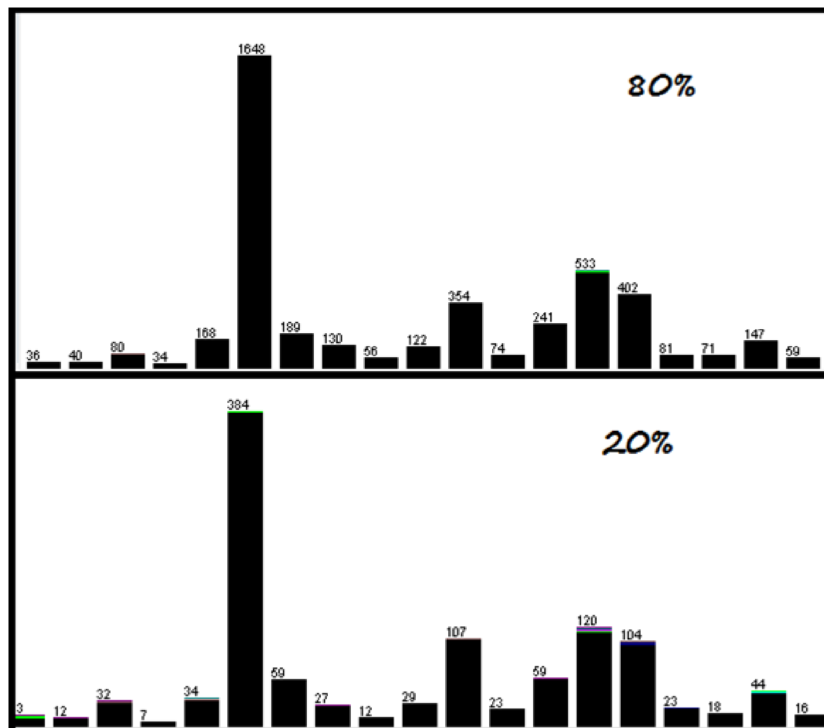

Figura 5.4: Frequenza delle categorie applicando il principio di Pareto

5.4.2 L'algoritmo PredectiveApriori

E' un algoritmo associativo, ordina le regole in base all'accuratezza attesa prevista; quest'accuratezza si basa sulla correzione del valore di confidenza, vengono cioè visualizzate le migliori n regole in accordo all'accuratezza prevista e in accordo alle regole non sussunte da una regola con almeno lo stesso valore di accuratezza prevista. Questo algoritmo viene adattato alle regole di associazione generando item sets frequenti considerando tutti i dati come corpo delle regole. Per ogni algoritmo è necessario settare dei valori, come è possibile osservare nell'immagine: il valore booleano di *car* indica se la classe di regole di assocazione è un estratto delle regole generali, il valore *ClassIndex* rappresenta l'indice della classe di attributi, se settato a -1, l'ultimo attributo sarŕ preso come classe di attributi, e l'ultimo valore *NumRules* indica ovviamente il numero di regole che si desidera ottenere.

5.5 Il Valore aggiunto

Come già discusso le tre principali dimensioni del modello di qualità per i mashup sono: dati, presentazione e composizione. Il valore aggiunto fa parte della composition quality proprio perchè si considerano le misurazioni riferite all' intera composizione, non al componente preso singolarmente. Il valore aggiunto può essere visto come il totale delle caratteristiche fornite e/o dai dati offerti dal mashup che non si otterrebbero se la composizione fosse costituita da componenti diversi oppure da singoli componenti. Più concretamente:

Una particolare composizione fornisce added value se almeno per ogni componente, il set di caratteristiche o dati offeriti dal mashup è maggiore della somma delle caratteristiche o dati offerti dal singolo componente.

Partendo da questa definizione di base del concetto di valore aggiunto si rendere necessario avere una più sistematica valutazione e validazione del concetto dettagliadolo per renderlo quanto più possibile vicino a cosa realmente viene percepito dagli utilizzatori, sviluppati e più in generale dal mondo reale. Innanzitutto è opportuno suddividere su tre livelli di astrazione il concetto di valore aggiunto, proprio come in precedenza è stato fatto per il modello di qualità dei mashup:

- Logic Layer
- Data Layer
- UI Layer

La valutazione del ***livello logico*** può essere vista sotto due ottiche differenti. Quella che considera le funzionalità rappresentate da un alto livello di astrazione ovvero un *Use Cases* level oppure abbassare il livello di astrazione alle *API Operations*. Nel primo caso si otterrà una valutazione molto più vicina a ciò che effettivamente l'utente percepisce accompagnata però da una complessità di misurazione sicuramente maggiore che utilizzando la seconda ottica. La minor complessità in questo caso è data dalla possibilità di fare parsing sul codice ottenendo così le operazioni atomiche offerte dal componente considerato, a discapito di un allontanamento da ciò che realmente l'utente finale percepisce (banalmente un utente può riconoscere l'esistenza di alcune funzionalità all'interno di un componente ed può ignorarne altre, che comunque sono previste dall'implementazione). Nella sezione sulle metriche si andrà a dettagliare questo concetto mostrando i supporti e le scelte concettuali prese a favore di questa seconda visione, *API Operations*, che è stata utilizzata per le recommendation.

Per valutare il valore aggiunto a ***livello di dati*** si può innanzitutto considerare le tipologie di dati fornite (dati strutturati, non strutturati, dati numerici ecc) e di conseguenza la possibilità di eseguire join e intersezioni fra domini di dati di tipo diverso e quindi la quantità di itnerrogaizioni (query) aggiuntive che è possibile effettuare considerando i dati derivanti dalla composizione nel suo insieme piuttosto che quelli offerti da ogni componente preso singolarmente. In altre parole va a valutare quanti dati provengono da differenti sorgenti o media e quindi contribuiscono ad una maggior ricchezza totale.

Il ***livello UI*** è forse il più complicato sistematizzare nella quantificazione in quanto il concetto di visualizzatori o viewers è da un lato molto più astratto rispetto a quello di funzionalità o dato e soprattutto l'interfaccia utente tende ad essere associata ai componenti del mashup. Talvolta infatti più visualizzatori fanno riferimento ad un unico componente viceversa a volte un visualizzatore contiene più sorgenti dati. La difficoltà sta proprio nel definire cosa è UI cosa

non lo è all'interno del mashup senza fare confusione con caratteristiche proprie delle API (esempio Google Maps è risultata una delle poche API disponibili che fornisce una UI proprietaria) o funzionalità che non sono però parte dell'UI. Per risolvere questa criticità si è proposto di valutare il content enrichment definito come ricchezza di componenti quali foto, video, testo, musica etc. racchiudendoli in un concetto di Presentation Media che chiariremo nei capitoli successivi. La valutazione del *Presentation Media* sarà quindi trattata suddividendola tra Logic e Data layer.

5.5.1 Metriche

Resta a questo punto da definire come procedere con il calcolo effettivo delle misure di Valore aggiunto sopracitate. Partendo dalle metriche più intuitive che riguardano il valore aggiunto in termini di data layer avremo:

$$Dj = \frac{\sum_{k=0}^{N} \% didatidellasorgente(k)joinedconaltridatasource}{Num.Totaledisorgentidati}$$

$$Di = \frac{\sum_{k=0}^{N} \% didatidellasorgente(k)intersecaticonaltridatasource}{Num.Totaledisorgentidati}$$

Maggiori sono le misure di Added Value Dj, Di, più il livello di integrazione dei dati è alto e quindi la composizione così come considerata, per quanto concerne il data layer, ha un alto valore aggiunto per l'utente.

Per quanto riguardo il logic layer, come è stato indicato, la scelta concettuale è stata quella di effettuare la valutazione sulla base della visione *API Operations*, la quale ha permesso, tramite parsing sul codice, oppure come nel nostro caso di studio, attraverso un'analisi dei risultati derivanti dalla valutazione di compatibilità e similarità , di ottenere non solo le funzionalità e le operations delle api/componenti del mashup, ma anche di averne una valutazione in termini di compatibilità, similarità e, come vedremo nello step successivo, di enrichment sul componente in quanto tale all'interno di un'integrazione di servizi Web. Facciamo qualche esempio per illustrare meglio il concetto di metrica per il livello logico. Un mashup con:

- Due o più componenti incompatibili tra loro avrà un valore aggiunto pari a zero, la logica di integrazione viene meno;

- Due o più componenti simili forniranno un apporto sufficiente in base a quanto questi siano integrati a livello di dati;

- Due o più componenti dissimili e incompatibili danno un valore aggiunto nullo;

- Due o più componenti dissimili e compatibili possono aggiungere valore nella misura in cui c'è integrazione a livello dati e funzionalità (binding e topologia del mashup)

- Due o più componenti simili ma incompatibili danno un valore aggiunto nullo;

- Due o più componenti simili e compatibili possono aggiungere valore nella misura in cui c'è integrazione a livello dati e funzionalità (binding e topologia del mashup)

Si arriva infine alla quantificazione dell'AddedValue per lo User Interface Layer che, come precedentemente detto, ricade sui due livelli precedenti sotto la definizione di *Presentation Media* Added Value. Esso verrà quantificato sulla base della presenza all'interno dell'intero mashup, di uno o più componenti che offrono contenuti multimediali (foto, audio, video, ecc.), osservando che in realtà, più un mashup è ricco di contenuti, maggiore può essere il valore aggiunto per l'utente finale; verrà quindi dato un peso maggiore nella valutazione dell'Added Value al fatto che un componente integri o meno *media contents* con altri componenti del mashup, con annessa misura percentuale, piuttosto che alla valutazione del data layer o del logic layer. Tutto ciò, come descritto nella sezione successiva, sarà supportato da regole di associazione tra categorie, che restringono il campo di possibili componenti raccomandabili sui quali effettuare il calcolo delle metriche di valore aggiunto e qualità (e quindi il tempo di elaborazione della reccomendation).

5.5.2 Regole di associazione

Una volta definito il modello di qualità del mashup, come calcolare il valore aggiunto che una certa configurazione fornisce, fine a se stessa oppure in relazione ad un'altra composizione di confronto, si è pensato di spingersi nell'estensione del concetto di reccomendation non più sulla base della sola qualità dei componenti,del mashup e della similarità, compatibilità[1], bensì aggingendo delle regole di associazione basate su uno studio sui grandi numeri delle caratteristiche dei mashup più popolari (e quindi non necessariamente di qualità maggiore rispetto ad altri).

Come abbiamo visto siamo partiti dal mining sul nostro repository di riferimento e dopo aver dato in input a Weka il file ARFF creato precedentemente, sono state ottenute diverse regole di associazione strutturate in questa forma:

```
cat=16 cat=14 6 ==> cat=14 6    acc:(0.94607)
cat=6 cat=15 cat=15 5 ==> cat=15 5    acc:(0.93259)
cat=14 cat=14 19 ==> cat=14 18    acc:(0.92997)
```

che viene letta in questo modo: la categoria A implica la categoria B. Nel primo membro viene infatti indicata la categoria che si sta prendendo in considerazione, l'ordine nel quale essa viene considerata e la frequenza di occorrenza di quella combinazione; nel secondo membro sono presenti gli stessi valori ed inoltre viene visualizzato il valore di accuratezza riferito a quella regola. L'accuratezza rappresenta la percentuale di mashup del campione, la cui classe predetta dal modello coincide con la classe reale. Sono state ottenute 4999 regole, ma ne sono state prese in considerazione 1290, cioè è stato estratto il campione di regole con accuratezza maggiore o uguale a 0,5 in quanto sarebbe stato poco ragionevole analizzare un campione di dati aventi un valore di accuratezza inferiore al 50%. Osservando il file si è osservato che ricorrevano regole strutturate in questa maniera: un'API implica un'altra API 1==>1, 1==>2, 2==>1, 2==>2, 3==>1, 3==>2, 4==>1 ed infine 4==>2. Attraverso un opportuno algoritmo che sfrutta il metodo Java Split() che consente di isolare l'antecedente dell'implicazione dal conseguente. Questo ha permesso di ottenere classi di associazioni divise per classi, ad esempio associazione 1:1, 1:2, 2:1 ecc.

A questo punto all'interno delle classi erano però presenti regole con medesime combinazioni di categorie, e quindi attraverso un'ulteriore elaborazione classe per classe, sono state accorpate le regole simili, sommando i valori di occorrenza ed effettuando una media dei valori di accuratezza per ottenerne il valore definitivo in riferimento a quell'associazione ora unica e storata nella tabella delle regole di associazione come viene illustrato dallo pseudo codice dell'algoritmo per l'accorpamento delle regole.

popolDbRules()
begin
$Array[]rules;$
$for(inti = 0; i < N; i++)\{$
$rules = AllAssociationRulesClass(i);$
$doInsert(calcTotAccuracy(rules));$
$\}$
end

con $forti = 0aN$ indica lo scorrimento delle N classi di assciazione (1:1,1:2,2:1,2:2,3:1 ecc.)

calcTotAccuracy(r)
begin
$Array[]accuracy;$
$for(inti = 0; i < N; i++)\{$
$accuracy[i] = r[i] + [sum(getReplicateRulesAcc()) \div numReplicateRules];$
$\}$
return accuracy[i];
end

con $r[i] + [sum(getReplicateRulesAcc()) \div numReplicateRules]$ che indica il concatenamento dei dati di categoria della regola di associazione, con il nuovo valore di accuratezza mediato sul numero delle replicazioni.

E' stata quindi creata una tabella che racchiude le regole di associazione suddivise per tipologia di associazione (ad esempio 1:1, 2:1, 3:2 ecc.), chiamata *Asso-*

ciationRules che farà parte del repository di Mashup ideale che proporremo nelle conclusioni. L'utilizzo del repository di ProgrammableWeb, essendo organizzato in modo poco chiaro ed efficiente, porta a questa necessità di reingegnerizzazione che vedremo successivamente come proposta per il futuro.

id	cata	catb	accuratezza
1	Shopping	Shopping	0.6747
2	Storage	Internet	0.6538
3	Tools	Social	0.78209

Tabella 5.2: Tabella aggregazione regole 1==>1

Una volta definite in modo chiaro le associazioni, il passo successivo è stato quello di analizzare le regole ottenute, ragionando attraverso una struttura ad albero che potesse essere in grado di esprimere il concetto di espansione delle classi di associazione.

Come detto in precedenza, queste regole permettono di esprimere reccomendation per le categorie di assocazione, raccomandazione che può essere espressa sotto tre forme: analizzando mashup frequenti, e cioè svolgendo mining sulle categorie come in questo caso, analizzando la similarità tra binding ed analizzando la compatibilità dei componenti. Per esempio i primi due casi possono essere spiegati attraverso il seguente esempio: è stata scelta una regola di assocazione dalla tipologia una categoria implica due categorie.
Messaging ==> Bookmarks Music 0,82
Messaging ==> Bookmarks Social 0,82
Shopping ==> Shopping Shopping 0,64
Events ==> Mapping Search 0,60
Tools ==> Shopping Mapping 0,60
Reference ==> Social Social 0,55
Reference ==> Social Video 0,55

che può essere visualizzata quindi in questa forma ad albero

Come è possibile osservare, si nota che ci sono più strade possibili per ogni categoria considerata, in questo caso si è scelto di considerare la strada *Tools* implica *Mapping*; a questo punto si prosegue andando ad esplorare le regole di

Figura 5.5: Inizio della raccomandazione

associazione presenti nel caso due categorie implicano una categoria, cioè Tools e Mapping implicano una categoria, partendo ora da due categorie invece che una come è successo all'inizio .

60. cat=Social cat==Photos 3 ==> cat=Photos 3 acc:(0.86698)
63. cat=News cat=Photos 3 ==> cat=Bookmarks 3 acc:(0.86698)
64. cat=Search cat=Video 3 ==> cat=Search 3 acc:(0.86698)
66. cat=Tools cat=Mapping 3 ==> cat=Internet 3 acc:(0.86698)

...

172. cat=Tools cat=Mapping 2 ==> cat=Mapping 2 acc:(0.81849)
173. cat=Tools cat=Social 2 ==> cat=Mapping 2 acc:(0.81849)
175. cat=Advertising cat=Shopping 2 ==> cat=Shopping 2 acc:(0.81849)

...

1146. cat=Mapping cat=Reference 3 ==> cat=Reference 2 acc:(0.59982)
1147. cat=Mapping cat=Tools 3 ==> cat=Search 2 acc:(0.59982)
1149. cat=Music cat=Music 3 ==> cat=Social 2 acc:(0.59982)

Le ulteriori regole ottenute permettono di espandere di un altro step l'albero, si nota che ci sono tre strade possibili, osservando la regola 66 si ottiene la raccomandazione sulla categoria *Internet*, prendendo invece spunto dalla regola 172 si può notare che la categoria raccomandata è nuovamente quella di *Mapping* ed

invece analizzando la terza scelta notiamo che al secondo membro è presente la categoria *Search*. Oltre alla tipologia *2==>1* è possibile però prendere in considerazione anche il caso in cui due categorie ne implicano altre due, e le regole ottenute osservando quest'altra opportunità sono le seguenti

383. cat=Tools cat=Mapping 2 ==> cat=Shopping cat=Mapping 2 acc:(0.81849)

...

1215. cat=Search cat=Video 3 ==> cat=Search cat=Search 2 acc:(0.59982)
1217. cat=Shopping cat=Shopping 3 ==> cat=Shopping cat=Shopping 2 acc:(0.59982)
1218. cat=Shopping cat=Shopping 3 ==> cat=Shopping cat=Shopping 2 acc:(0.59982)
1222. cat=Social cat=Video 3 ==> cat=Internet cat=Social 2 acc:(0.59982)
1223. cat=Tools cat=Mapping 3 ==> cat=Internet cat=Mapping 2 acc:(0.59982)

...

A questo punto si può procedere ulteriormente andando a considerare il caso in cui al primo membro sono presenti tre categorie e il caso in cui ne sono presenti invece quattro; in questo studio si è potuto notare che il numero massimo di categorie considerate per regola è pari a sei, e l'ultimo caso è proprio quello in cui quattro categorie ne raccomandano altre due, si lascia libera scelta all'utente se interrompere la raccomandazione o se invece considerare altre possibili combinazioni ed aggiungere quindi ulteriori categorie non prese in considerazioni negli step precedenti.

5.5.3 Algoritmo

L'algoritmo che accorpa quanto detto fino ad ora prevede quattro fasi di elaborazione per giungere alla reccomendation:

- Acqusizione dei dati *AS IS Mashup* : categorie di componenti presenti nella composizione al momento della valutazione e processo di supporto alla composizione (reccomendation);
- Accesso al db, tabella *AssociationRules* contentente le regole di associazione e creazione alberatura di reccomendation;

- Valutazione qualità, compatibilità, similarità e valore aggiunto del componente i rispetto al componente j del sottoinsieme di componenti di categoria x proposti al passo 2;
- Scelta dell'utente e di nuovo fase 1 fino al termine della composizione.

Mostiamo qui in seguito lo pseudo codice d'implementazione dell'algoritmo di reccomendation e valutazione di valore aggiunto fin'ora descritta concettualmente e che è stata integrata nell'architettura che dettaglieremo del successivo capitolo.

compReccom()
begin
$getComponentsAsIs() = [catComp1, catComp2, catComp3, ..., catCompN];$
$Array[]arrayRecc = getAssocitionRules();$
$for(inti = 0; i < N; i++)\{$
$arrayQual[i] = [(localQuality(arrayRec[i]) + AddedValue(arrayRec[i]) \div 2];$
$\}$
return compReccom $= calcMax(arrayQual);$
end

Rimandando al calcolo della local quality nella sezione 5.5.3. Inoltre prima di ripetere l'algoritmo è possibile richiamare la funzione di calcolo globalQuality() che calcola la qualità del mashup con l'aggiunta del nuovo componente.

Capitolo 6

L'implementazione

Questo capitolo chiarisce le specifiche caratteristiche dell'architettura e in generale gli strumenti che ci hanno guidato nell'implementazione ed integrazione con quanto già esistente di tutto il lavoro svolto nell'ambito della generazione di reccomendations. Ci si soffermerà dapprima sul back-end dell'applicativo, cioè sull'architettura lato server e successivamente sul front-end, nel paragrafo dedicato alle prove pratiche si chiarificherà invece con esempi pratici la realizzazione di alberature di reccomendation fondendo tutti i concetti fin'ora descritti e introdotti.

6.1 L'architettura

L'architettura scelta è la comune architettura a tre livelli, che viene di buona norma utilizzata in situazioni analoghe alla nostra. Nell'architettura a tre livelli (detta anche thin client) il client non comunica direttamente con il server del database ma con un server dell'applicazione. In questo modo il client svolge solo il compito di interfaccia utente e la logica dell'applicazione viene inserita nel server applicativo. Questa soluzione è sicuramente più modulare: se si modifica la base di dati sottostante, il server dell'applicazione richiede a sua volta delle modifiche, ma l'interfaccia utente può anche restare invariata. Server applicativo e server di database possono risiedere nella stessa macchina o su macchine diverse collegate in rete. Nell'architettura a 3 livelli (detta 3-tier architecture), esiste un livello intermedio, cioè si ha generalmente un'architettura condivisa tra:

- un client, cioè il computer che richiede la risorsa, dotata di un'interfaccia utente (generalmente un navigatore Web) incaricato della presentazione;
- il server d'applicazione (detto anche middleware), incaricato di fornire la risorsa ma facendo riferimento ad un altro server;
- il server di dati, che fornisce al server dell'applicazione i dati di cui ha bisogno.

Nell'architettura a 3 livelli, le applicazioni a livello del server sono de localizzate, il che significa che ogni server è specializzato in un compito (server Web/server database ad esempio). L'architettura a tre livelli permette :

- un elevato livello di flessibilità;
- un elevato livello di sicurezza, dato che questa può essere definita in maniera indipendente per ogni servizio e ad ogni livello;
- delle performance soddisfacenti, dato che condivide dei compiti tra i differenti server.

Descriveremo dunque le seguenti componenti che vanno a costituire l'architettura a supporto della nostra elaborazione:

- Web client;
- Web server;
- Application server;
- Database.

In particolare all'interno del Web client si ha una interfaccia utente molto reattiva, a motivo dell'utilizzo di AJAX.

Per quanto riguarda il Web server specifichiamo che in esso sono situati componenti, che nel nostro studio ricordiamo essere downloadati attraverso un crawler di collegamento a ProgrammableWeb, le ontologie necessarie alla valutazione di compatibilità e similarità[1], e il Web service, questo in modo da rendere snella e fruibile l' applicazione. Inoltre questa struttura architetturale favorisce futuri

aggiornamenti e manutenzioni di tutto l'apparato, anche perché in questo modo chi aggiunge nuovi componenti ha la possibilità di annotarli anche considerando le regole di associazione e le ontologie che sono già presenti sul Web server.

Si parla ora della terza componente elencata, cioè l'application server. All'interno dell'application server si trovano altre parti fondamentali per l'algoritmo di reccomendation:

- il WordNet dictionary, nella sua versione più attuale. Questo permette di manutenere in maniera agevole l'applicativo e di aggiornare con facilità la piattaforma con le versioni via via più aggiornate di WordNet;
- il reasoner server (Pellet [36]), che di nuovo sarà agevolmente manutenibile;
- entity bean, (un tipo di Enterprise JavaBean, un componente J2EE lato server, che rappresenta i dati persistenti conservati in un database);
- session bean, (un tipo di Enterprise JavaBean che rende possibili operazioni, come calcoli o accessi al database);
- WorkManager, che è un elemento fondamentale per la gestione di tutto l'apparato e viene supportato dall'utilizzo dell'application server Jboss [38].

Il quarto aspetto da descrivere è il database, che presenta le seguenti caratteristiche, che sono basilari per rendere ottimale il nostro algoritmo:

- analisi di utilizzo. Nel momento in cui un utente rimpiazza un componente possiamo tenere traccia di questa azione: pertanto i dati salvati ci permettono di aiutare gli altri utenti nelle loro scelte, il che rende il tutto decisamente *user friendly*;
- cache similarity, che non è altro se non il salvataggio dei dati relativi alla similarità;
- gestione dei processi;
- gestione degli utenti, rendendo possibile un certo livello di personalizzazione, ad esempio mostrando allo specifico utente solo un gruppo di componenti appartenenti ad una tipologia che lui solitamente utilizza (ad esempio un creatore di mashup, che crea mashup solo relativi all'ambito del turismo).

- gestione associazioni, che rende possibile la consultazione e l'aggiornamento in tempo reale delle regole di associazione per ottenere reccomendation abbinate a processi di calcolo delle misure di qualità ed Added Value, che tengono conto dell'evoluzione delle composizioni create/popolari.

Ribadiamo, infine, che tutta la nostra architettura ruota sull'utilizzo dell'application server (nel nostro caso Jboss).

6.2 Esempio di reccomendation

In questa sezione si valuteranno i principali risultati ottenuti e si commenterà il loro impatto. La sezione dedicata alle prove è stata introdotta anche per far comprendere più agevolmente come si presenteranno i risultati all'utente finale e come egli potrà sfruttarli in modo ottimale. Come già detto in precedenza, è stata utilizzata un'integrazione tra le metriche *qualità* di *added value* e le *regole di associazione* di categorie, per arrivare ad ottenere da un lato una valutazione del valore aggiunto che il mashup fornisce all'utente e dall'altro una raccomandazione da dare all'utente sulla composizione del mashup. Per questo secondo punto si procede in questo ordine: come prima cosa viene considerato un sottoinsieme dei componenti del mashup analizzato, di modo da ottenere così un mashups incompleto rispetto a quello di partenza e successivamente viene sviluppata una reccommendation per validare quanto più la composizione del mashup si avvicina a quello realmente esistente.
E' stato analizzato quindi il mashup http://blachan.com/shahi, un semplice ed intuitivo dizionario visivo composto da quattro componenti:

- Flickr
- Google Ajax Search
- Yahoo Image
- Wikipedia

Come primo step si parte quindi dal sottoinsieme formato dalla singola API di Wikipedia di categoria Reference, ottenendo così un falso mashup a causa della

presenza di una sola API, si prova quindi a dare una raccomandazione attraverso le regole di associazione ottenute e infine si procede alla scelta del componente tramite le metriche di added value definite precedentemente.

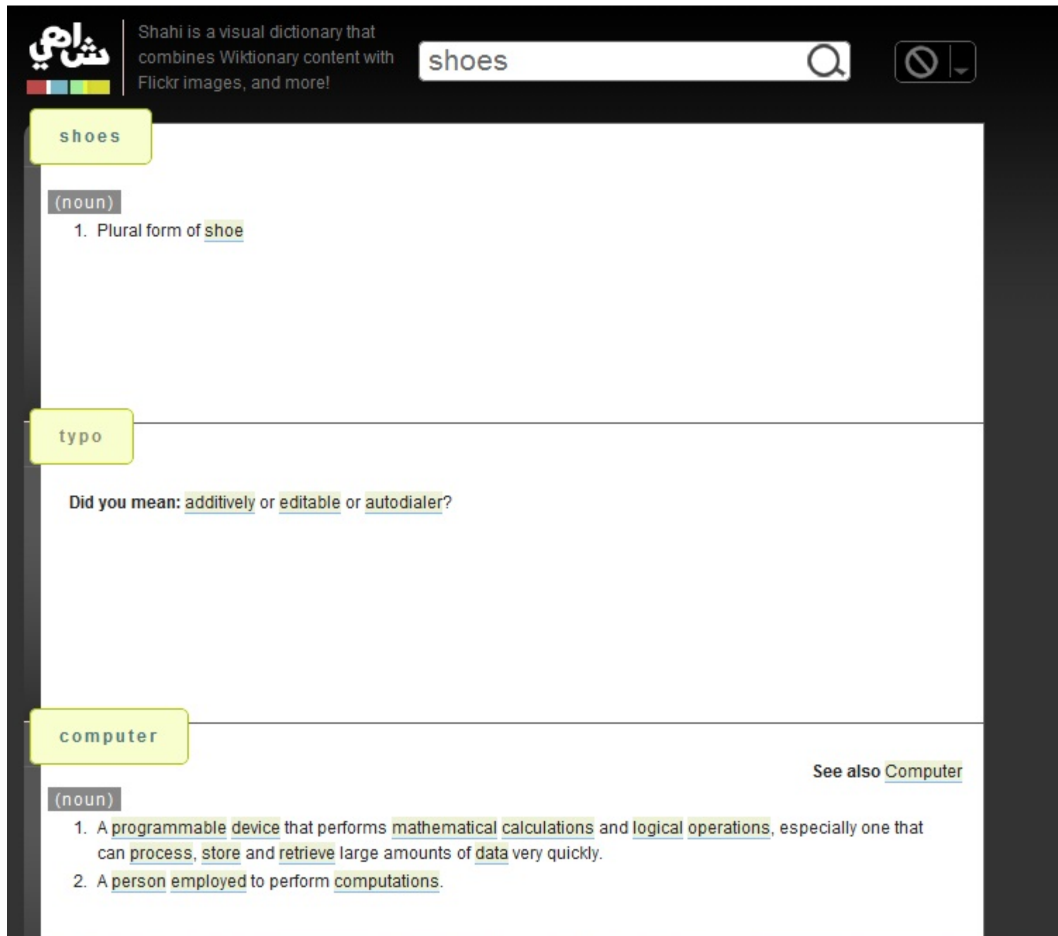

Figura 6.1: Il mashup contenente solo l'API di Wikipedia

A partire dalla categoria Reference viene ottenuta la raccomandazione alle APIs di categoria Photos tramite la regola

```
cat=Reference ==> cat=Photos    acc:(0.55)
```

Si ipotizza che il repository contenga le seguenti APIs di categoria Photos

- **BlueMelon** Photo sharing and storage service, Photos
- **Buzznet** Photo sharing,Photos

- **Flickr** Photo sharing service, Photos
- **Google Picasa** Photo management and sharing service, Photos
- **Instagram** Mobile photo sharing service, Photos

Si procede quindi alla valutazione iniziale del valore aggiunto

- **Presentation Media** I contenuti multimediali di partenza riguardano solo Wikipedia, quindi in questo caso il content è rappresentato solo da testo
- **Logic Layer** Le funzionalità di Wikipedia sono:
  - **Ricerca e accesso al database di mediawiki**
  - Login per effettuare modifiche
  - Supporto del Javascript client
  - Ampio range di formati
- **Data Layer**
  - Di = 0
  - Dj = 0

Verranno ora quindi aggiunti uno alla volta ognuno di questi componenti e saranno ricalcolati i valori di Added Value per poi andar a raccomandare il componente con maggior valor aggiunto fruibile. Ora si osserverà come cambia la valutazione delle metriche di Added Value inserendo due delle APIs di categoria Photos presenti nel repository.

GooglePicasa

- **Presentation Media** I contenuti multimediali aggiunti inserendo questa API riguardano *foto*, quindi per ora i contents del mashup saranno testi e foto.
- **Logic Layer** Le funzionalità di Google Picasa sono:
  - Integration with Picasa WebAlbums
  - Create Album

  - Upload photos
  - Comment phtos
  - Desktop Applications
  - Uploding to PWA
  - Login
  - Power digital photo frame

- **Data Layer**
  - Di = 0
  - Dj = 0

Flickr

- **Presentation Media** I contenuti multimediali aggiunti inserendo questa API riguardano *foto*, quindi per ora i contents del mashup saranno testi e foto.
- **Logic Layer** Le funzionalità di Flickr sono:
  - Login
  - Photo management
  - Sharing Photos
  - **Search Photos**
- **Data Layer**
  - Di = 0
  - Dj = 0

In questo caso la scelta ricade su Flickr, il dominio di funzionalità risulta infatti più attinente, avendo in comune Wikipedia e Flickr la funzionalità di ricerca.

Si procede quindi con un successivo step di raccomandazione:

```
cat=Photos cat=Reference ==> cat=Social  acc:(0.66753)
```

Figura 6.2: Il mashup contiene ora le APIs di Wikipedia e di Flickr

Si ipotizza, come prima, che il repository contenga le seguenti APIs di categoria Social:

- **Linkedin** Business social networking platform, Social
- **Twitter** Microblogging service, Social
- **Facebook** Social networking service, Social
- **Foursquare** Social networking and city exploration, Social
- **MySpace** Social networking service, Social

Si procede quindi alla valutazione del valore aggiunto nel caso in cui vengano inserite due delle Social Api presenti nel repository.

Twitter

- **Presentation Media** I contenuti multimediali aggiunti da questa API sono di tipo social, e in ogni caso sono testuali.
- **Logic Layer** Le funzionalità di Twitter sono:
  - **Search in data Base**
  - Login
  - User information
  - Status data
  - **Share url**
  - Update timelines
  - Multiple Data formats
- **Data Layer**
  - Di = 0
  - Dj = 0

Facebook

- **Presentation Media** I contenuti multimediali aggiunti da questa API sono di tipo social, e in ogni caso sono testuali.
- **Logic Layer** Le funzionalità di Facebook sono:
  - **Share Page**
  - Social connections
  - Profile information
  - Publish activities
  - Share social context utilizing profile
  - Share/Add Friend
  - Share Group
  - Share Event data
  - **Share Photos**
- **Data Layer**
  - Di = 0
  - Dj = 0

Per quanto riguarda le funzionalità aggiuntive, è possibile notare che queste si equivalgono, in numero, tra Facebook e Twitter, ma è noto che Facebook non permette di visualizzare gli status degli utenti iscritti, attraverso una ricerca effettuata con tag o parole chiave, per problematiche relative alla privacy, per questo motivo la scelta del componente si sposta su Twitter.

Si passa al successivo step di raccomandazione:

```
cat=Social cat=Reference ==> cat=Search  acc:(0.59)
```

Si ipotizza nuovamente, che il repository contenga le seguenti APIs di categoria Search:

- **Ebay** eBay Search service, Search
- **Google Search** Search services, Search

Figura 6.3: Il mashup contiene ora le APIs di Wikipedia, Flickr e Twitter

- **Yahoo Search** Search services, Search
- **Google Ajax Search** Web search components, Search
- **Yahoo Image Search** Image search services, Search
- **Bing** Online Search services, Search

Si procede quindi alla valutazione del valore aggiunto nel caso in cui vengano inserite due delle Social Api presenti nel repository.

Google Ajax Search

- **Presentation Media** I contenuti multimediali aggiunti da questa API sono anche in questo caso di tipo foto.
- **Logic Layer** Le funzionalità di Google Ajax Search sono:
  - **Search(Web,Local,Blog,Images,Videos)**
  - Integrate results in GoogleMaps API
  - Cache
  - **Show results**
  - **Integrate results in Web site MU**
- **Data Layer**
  - Di = 0
  - Dj = 0

Yahoo Image Search

- **Presentation Media** I contenuti multimediali aggiunti da questa API sono anche in questo caso di tipo foto.
- **Logic Layer** Le funzionalità di Yahoo Image Search sono:
  - **Access to Yahoo!'s Image Search**
  - **Search in Internet for images**

- **Data Layer**
  - Di = 0
  - Dj = 0

Google Ajax Search risulta più valido, dal punto di vista dell'added value, rispetto ai domini di funzionalità introdotti fin ora dall'utente nel mashup, per questo motivo tra le due APIs è stata scelta quest'ultima.

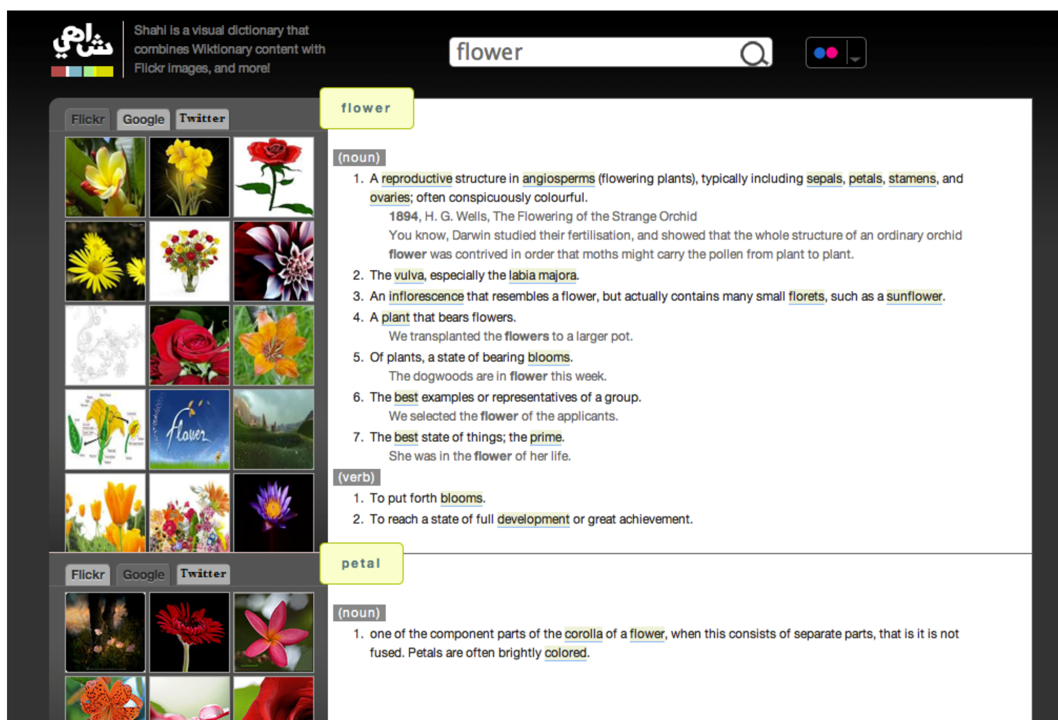

Figura 6.4: Il mashup contiene ora le APIs di Wikipedia, Flickr, Twitter e Google Ajax Search

Si arriva quindi all'ultimo step di raccomandazione:

```
cat=Social cat=Reference cat=Search  ==> cat=Search
```

Le APIs di categoria Search sono le medesime di prima, ed esssendo stata scelta nel passo precedente l'API di Google Ajax Search, in quest'ultimo passo verrà scelta l'altra API e cioè quella di Yahoo Image Search.

Google Ajax Search risulta più valido, dal punto di vista dell'added value, rispetto ai domini di funzionalità introdotti fin ora dall'utente nel mashup, per questo motivo tra le due APIs è stata scelta quest'ultima.

In definitiva, il mashup composto grazie alla scelta guidata effettuata dall'utente tramite le reccommendation basate sull'Added Value è quello presente nella figura sottostante. La differenza rispetto a quello originale è data dall'aggiunta dell'API di Twitter che dà nel complesso un apporto maggiore; infatti dal punto di vista multimediale, quindi per quanto riguarda il livello *Presentatiion Media*, sono stati aggiunti contenuti di tipo *social*. La compatibilità di funzionalità si riscontra ugualmente nei risultati dati dalle metriche di Added Value per Presentation Media, Logic e Data Layer con o senza la presenza della API Twitter, il tutto fermo restando la non unicità del percorso di raccomandazione.

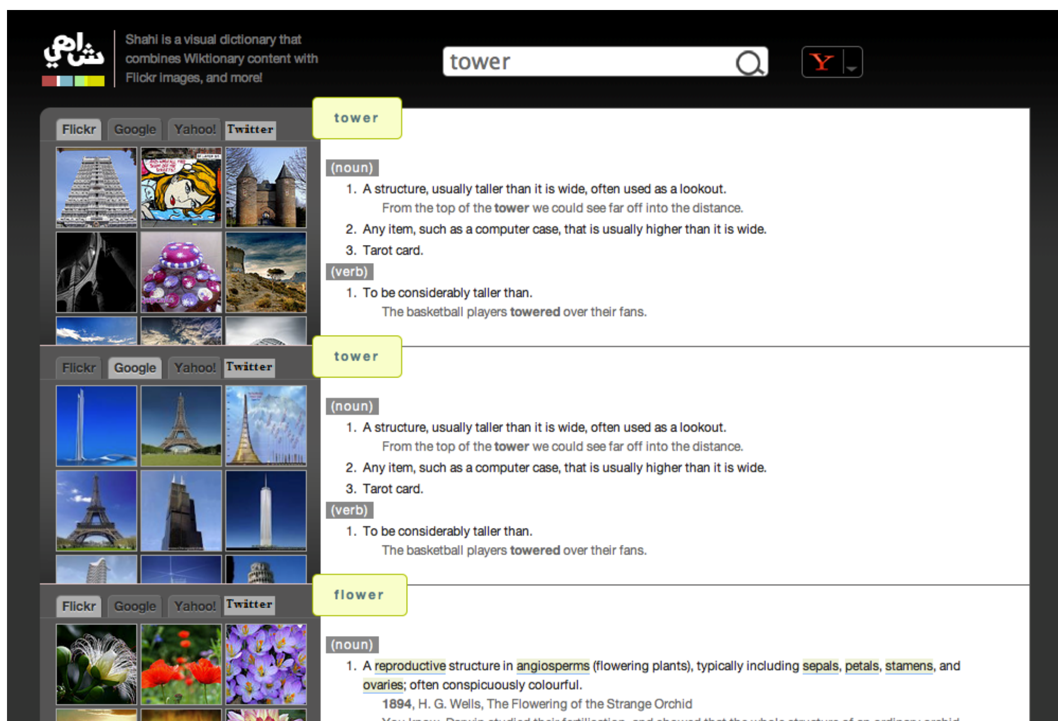

Figura 6.5: Mashup ottenuto attraverso la raccomandazione

Shahi is a visual dictionary that combines Wiktionary content with Flickr images, and more!

tower

Flickr Google Yahoo!

tower

(noun)

1. A structure, usually taller than it is wide, often used as a lookout.
   From the top of the **tower** we could see far off into the distance.
2. Any item, such as a computer case, that is usually higher than it is wide.
3. Tarot card.

(verb)

1. To be considerably taller than.
   The basketball players **towered** over their fans.

Flickr Google Yahoo!

tower

(noun)

1. A structure, usually taller than it is wide, often used as a lookout.
   From the top of the **tower** we could see far off into the distance.
2. Any item, such as a computer case, that is usually higher than it is wide.
3. Tarot card.

(verb)

1. To be considerably taller than.
   The basketball players **towered** over their fans.

Flickr Google Yahoo!

flower

(noun)

1. A reproductive structure in angiosperms (flowering plants), typically including sepals, petals, stamens, and ovaries; often conspicuously colourful.
   1894, H. G. Wells, The Flowering of the Strange Orchid
   You know. Darwin studied their fertilisation, and showed that the whole structure of an ordinary orchid

Figura 6.6: Mashup originale

6.3 Integrazione degli algoritmi nel front-end

Questa sezione spiega come gli algoritmi descritti si integrino all'interno del front-end, cioè come si presenterà l'interfaccia utente. In questo modo risulterà definitivamente chiaro come il sistema di raccomandazione generato dal connubio tra gli algoritmi esistenti e i nostri algoritmi di calcolo della qualità e reccomendation possano aiutare l'utente finale nell'operazione di composizione di un mashup. Innanzitutto come si può vedere in figura 6.7 l'interfaccia si può dividere in due zone: sulla sinistra abbiamo una zona più estesa, che contiene un browser che contiene l'elenco degli workspace e per ogni workspace l'elenco dei componenti contenuti in esso, mentre sulla destra abbiamo un'area denominata *See Also*, nella quale vengono visualizzate di volta in volta le informazioni di dettaglio, che l'utente sceglie di vedere navigando negli workspace.

Ogni workspace viene contrasegnato dal suo valore in termini di qualità tramite un numero di stelline che va da 1 a 5 (5 ottima; 1 scarso). Selezionando lo workspace a cui si è interessati, nella parte del *See Also* verranno visualizzate tutte le composizioni simili allo workspace di partenza e i relativi dati di qualità. Analogamente per i componenti che sono contenuti in ciascuno workspace abbiamo il contrassegno delle stelline che indica i livelli di qualità del componente. Di nuovo selezionando lo specifico componente l'utente finale potrà visualizzare, sempre nella sezione di schermo *See Also*, i componenti emersi dalla consultazione delle regole di associazione. Questo sotto insieme di componenti visualizzati avrà, per ciascun componente, la visualizzazione dei dati di qualità, similarità e valore aggiunto, calcolati seguendo gli algoritmi mostrati nei capitoli precedenti. Cliccando i tasti *Similarity Details* e *Added Value Details* sarà possibile visualizzare il dettaglio della valutazione di ogni sotto dimensione, per esempio per l'Added Value: Data Layer, Logic Layer e UI Layer valutation. L'aggregazione delle valutazioni di qualità, similarità e valore aggiunto viene incanalata, attraverso opportuna mediatura, in una visualizzazione ancora una volta su un numero di stelline da 1 a 5 per restare coerenti all'interfaccia utente esistente. Così intuitivamente l'utente durante la composizione ha la percezione di una reccomendation qualitativamente molto interessante per qualità, similarità e valore aggiunto (5 stelline) potendola confrontare con altre soluzioni giudicate dalla valutazione meno interessanti

(ad esempio 1 stellina), se pur degne di nota poichè raccomandate dalle regole di associazione sulle composizioni comuni e popolari.

Come si può constatare osservando figura 6.7, l'interfaccia utente proposta è abbastanza intuitiva e si presta all'utilizzo da parte di utenti finali anche inesperti.

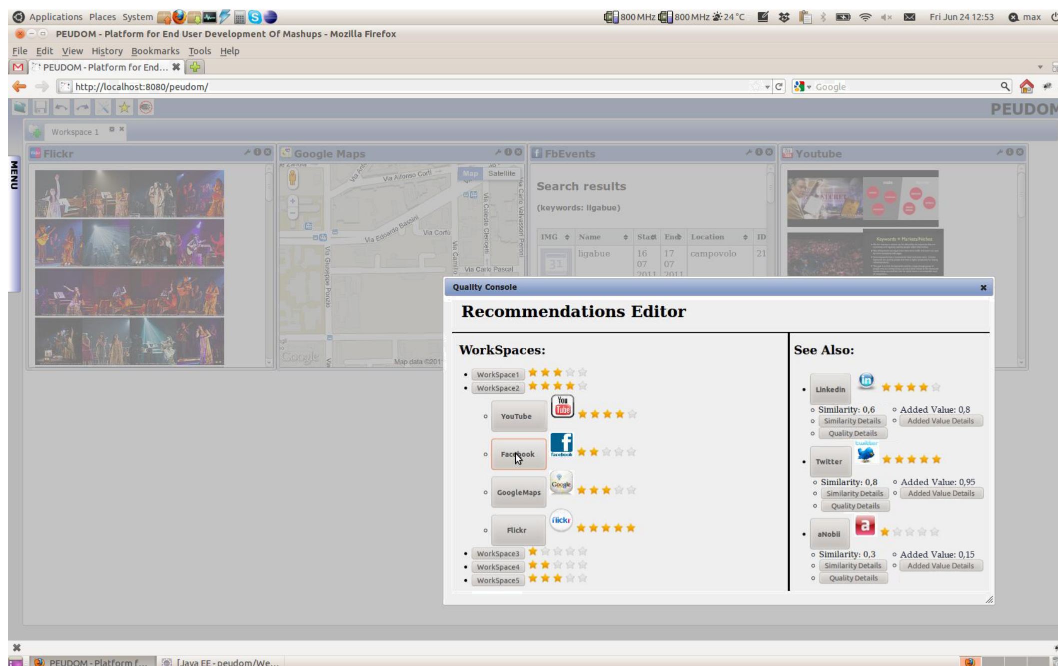

Figura 6.7: L'interfaccia a supporto delle recommendations

Nell'esempio mostrato si sta navigando all'interno del Workspace2. Come si nota l'utente ha quattro componenti, di cui uno, il componente Facebook ha qualità inferiore rispetto alle sue aspettative. Selezionando il componente nella sezione *See Also* compaiono tre componenti simili, con il dato aggregato di qualità, similarità e valore aggiunto. A sua discrezione l'utente potrà visualizzare i dettagli (cioè i dati disaggregati) relativi alle dimensioni di valutazione e scegliere in base non solo al risultato aggregato (numero stelline) ma in base a cosa egli è più interessato: un maggior valore aggiunto? un componente che sostituisca in tutto e per tutto uno già inserito nella composizione? La gestione dei dati di reccomendation in questo modo è molto libera e customizzabile risulta peraltro chiaro come l'interfaccia e gli algoritmi ad essa sottesi possano fattivamente coadiuvare l'utente finale nella creazione del suo mashup tramite l'utilizzo delle recommendations.

Capitolo 7

Conclusioni

In questa parte conclusiva della tesi verranno esposte le conclusioni relative al lavoro svolto. L'obiettivo era proprio quello di partire dalle pratiche correnti che si sono fin oggi basate su reccomendation di servizi orientate al requisito funzionale dell'utente, e di estenderle introducendo una valutazione dell'importanza del servizio in quanto parte di una composizione. In altre parole, ci si è proposti di quantificare nella maniera più sistematica possibile il concetto di valore aggiunto dei componenti di un mashup riducendo il livello di astrazione di tale dimensione di qualità. Nel tentativo di realizzare ciò che è stato espresso pocanzi, si è resa necessaria l'analisi di un campione più vasto possibile di mashup (ProgrammableWeb) per poter verificare le effettive preferenze dell'utenza disaggregando la misura di qualità da quella di interesse. E' emersa infatti la peculiarità che la qualità di un mashup o, più in generale di un'applicazione Web, non sia condizione necessaria per la popolarità. Si pensi per esempio a *Facebook*: la qualità non è certo il suo aspetto più rilevante e al quale deve la sua notorietà; è invece l'interazione degli utenti via chat, il messaging e la condivisione di contenuti che dà il vero valore aggiunto a questa applicazione Web. E' facile capire come in questo scenario la qualità passi in secondo piano rispetto al valore aggiunto che l'integrazione delle funzionalità offre all'utente finale.
Proprio alla luce di questo, il lavoro si è orientato verso lo studio, attraverso mining, del comportamento comune rispetto all'interesse di tipologie di mashups piuttosto che altre. E' emerso come determinate categorie di componenti siano integrate con maggiore frequenza rispetto ad altre nei mashups utilizzati e questo

viene riproposto nella generazione dalle reccomendation basate sulle regole di associazione estratte. Naturalmente le percentuali di occorrenza delle categorie nei mashups sono un valore destinato a cambiare nel tempo; il vantaggio del nostro approccio è dato dal fatto che, essendo stato creato un algoritmo di elaborazione dei dati attraverso mining, la creazione delle regole di associazione per la reccomendation muta nel tempo seguendo la variazione degli utilizzi effettivi delle categorie.
Il filo conduttore del nostro lavoro, una volta raggiunti i risultati sopra citati, è stato quello di considerare come principale il *Goal Mashup* e di tenere quindi conto dell'utilità percepita dall'utente e del suo interesse verso un servizio di integrazione piuttosto che un altro. Quindi il nostro lavoro ha dimostrato come le reccomendation che tengono conto delle dimensioni di valore aggiunto e associazione di categorie popolari si avvicinano a quello che l'utente percepisce utile. Esse infatti si basano sul fatto che la comunità frequentemente ha ritenuto interessante certe associazioni, a partire dagli stessi gusti (cioè medesimi inserimenti di componenti fatti fino all'*As Is* del mashup valutato). Dunque possiamo riassumere la nuova visione di reccomendation come una sinergia tra:

- Soluzioni di Mashup più frequenti e regole di associazione;
- Similarità rispetto ai binding creati (*As Is* del Mashup);
- Similarità e compatibilità dei diversi componenti;
- Replacement per massimizzare la qualità complessiva della composizione

Le prime tre componenti possono essere raggruppate sotto la definizione di *Enrichment* del mashup che si appoggia alle regole strutturate e definite dalla fase di categorizzazione e data mining in associazione alle metriche già esistenti di qualità, similarità e compatibilità e quelle introdotte di Added Value.

7.1 Estensioni future

Si conclude con uno scenario futuro di ottimizzazione della raccomendation e valutazione di qualità, con la reingegnerizzazione del repository dei compoenti DashMash. Presentiamo in figura 7.1 il modello concettuale già progettato per ottimizzare gli algorimi di reccomendation introdotti.

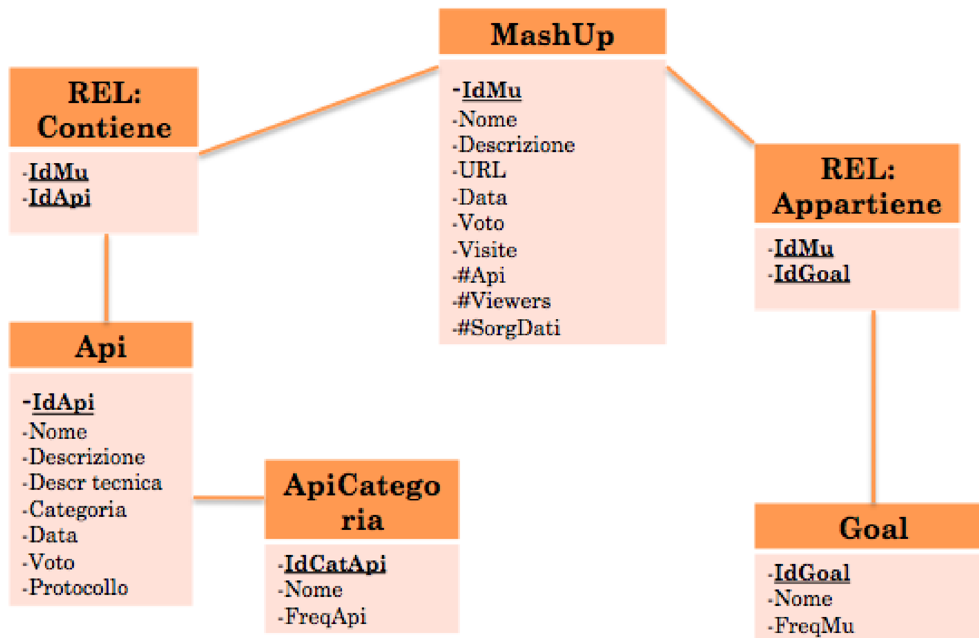

Figura 7.1: Modello concettuale del repository DashMash

Tale modello può essere utilizzato per organizzare la base di conoscenza delle composizioni create dagli utenti della piattaforma in modo che da queste composizioni possano essere estratte le regole di associazione che al momento, in mancanza di una vasta collezione di mashup, estriamo dalla collezione di ProgrammableWeb. Le informazioni di categoria e di *Goal Mashup* permettono infatti creare regole di associazione estratte dalle composizioni disponibili con i contenuti del repository e quindi di ottenere reccomendation basate sullo stile di composizione della comunità degli utenti della piattaforma. Lo scenario futuro mostra un ulteriore margine di estensione del modello di reccomendation in quanto è possibile considerare al-

tre dimensioni di composition quality, quali *component suitability* e *component usage*. Tali dimensioni mostrano un alto livello di astrazione, che sistematizzato avvicinerebbe ulteriormente la reccomendation al *Goal Mashup* dell'utente finale.

Bibliografia

[1] Massimo Gatti, Maddalena Losi. *Compatibiltà, Similarità e Qualità nella composizione dei Web Mashup*. 2010

[2] Hui Guo, Anca Ivan, Rama Akkiraju, Richard Goodwin. *Learning ontologies to improve the quality of automatic Web service matching*. 29/10/2009.

[3] Florian Daniel, Jin Yu, Boualem Benatallah, Fabio Casati, Maristella Matera, Regis Saint-Paul. *Understanding UI Integration: A survey of problems, technologies*. 2006.

[4] Florian Daniel, Jin Yu, Boualem Benatallah, Fabio Casati, Maristella Matera, Regis Saint-Paul. *A Framework for Rapid Integration of Presentation Components*. 2007.

[5] Florian Daniel, Maristella Matera. *Mash-up Context-Aware Web Application: a Component-Based Development Approach*. 2006.

[6] Serge Abiteboul, Ohad Greenshpam, Tova Milo, Neoklis Polyzotis. *MatchUp: autocompletion for mashups*. 2009.

[7] Matteo Picozzi, Marta Rodolfi, Cinzia Cappiello, Maristella Matera. *Quality-based Recommendations for Mashup Composition*. 2010.

[8] Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Gabriele Sprega, Donato Barbagallo, Chiara Francalanci. *DashMash: a Mashup Environment for End User Development*. 2010.

[9] Yahoo! Pipes. *http://pipes.yahoo.com/pipes/*. 2011.

[10] Roberta Ferrario, Nicola Guarino. *Towards an Ontological Foundation for Services Science*. 2009.

[11] Nicola Guarino, Claudio Masolo, Alessandro Oltramari, Aldo Gangemi, Laure Vieua. *La Prospettiva dell'Ontologia Applicata*. 2003.

[12] Repository dI ontologie Swoogle. *www.Swoogle.com*. Data ultimo accesso: 22/06/2011.

[13] Alessandro Cosola, Maddalena Losi. *Mutare le applicazioni Web in componenti Mashup: proposte, modelli e soluzioni*. 2006.

[14] Gabriele Sprega, Matteo Picozzi. *DashMash: a Mashup Environment for End-User Development*. 2010.

[15] Hazem Elmeleegy, Anca Ivan, Rama Akkiraju. *MashupAdvisor: A Recommendation Tool for Mashup Development*. 2008.

[16] A.Kim, Y. Lee. *Quality Model for Web Services*. 2005.

[17] Eclipse. *http://www.eclipse.org/*. 2011.

[18] Open Mashup Alliance. *http://www.openmashup.org/oma- docs/v1.0/index.html*. 2010.

[19] ServFace. *http://141.76.40.158/servface/*. 2011.

[20] J. Wong, J. I. Hong. *Making mashups with marmite: towards end-user programming for the Web*. 2007.

[21] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, A. Singh. *Damia: data mashups for intranet applications*. 2008.

[22] L. Zeng, B. Benatallah, M. Dumas. *Quality Driven Web Services Composition*. 2003.

[23] IBM. Qedwiki. *http://services.alphaworks.ibm.com/graduated/qedwiki.html*. 2011.

[24] JackBe. Jackbe presto. *Technical report, http://www.jackbe.com/*. 2011.

[25] Intel. Intel Mash Maker. *Technical report*. 2010.

[26] G. Fischer. *End-user development and meta-design: Foundations for cultures of participation*. 2009.

[27] F. Daniel, M. Matera, M. Weiss. *Web mashups: leveraging user innovation. Technical report, Politecnico di Milano*. 2009.

[28] F. Daniel, M. Matera. *Quando l'utente guida l'innovazione: Il Web mashup*. 2010.

[29] Sven Casteleyn, Florian Daniel, Peter Dolog, Maristella Matera. *Engineering Web Applications*. 2009.

[30] D. Barbagallo, C. Cappiello, C. Francalanci, and M. Matera. *Applied Semantic Technologies: Using Semantics in Intelligent Information Processing*. 2010.

[31] SOA4All. *http://www.soa4all.eu/*. 2011.

[32] Web Semantico. *http://www.semanticweb.org/*. 2011.

[33] ProgrammableWeb. *http://www.programmableweb.com/*. 2011.

[34] S. Bechhofer, C. Goble. *Towards Annotation using DAML+OIL*. 2001.

[35] X.Liu, Q.Zhao, G.Huang, H.Mei. *A Mashup Framework for Web-delivered Service Composition*. 2011.

[36] Pellet. *http://clarkparsia.com/pellet/*. 2011.

[37] WordNet. *http://wordnet.princeton.edu/*. 2011.

[38] JBoss. *http://www.jboss.org/jbossas/downloads/*. 2011.

[39] JQuery. *http://jquery.com/*. 2011.

[40] WonderWeb. *http://wonderweb.semanticweb.org/*. 2011.

[41] JUNG. *http://jung.sourceforge.net/*. 2011.

[42] GoogleMaps. *http://www.google.com/apis/maps/*. 2011.

[43] Devis Bianchini, Michele Melchiori. *Mashing-up Semantic-enhanced Services*. 2010.

[44] P.Hitzler, M.Krötzsch, S.Rudolph. *Foundations of Semantic Web Technologies*. 2009.

[45] Search Computing. *http://www.search-computing.it/*. 2011.

[46] Devis Bianchini, Cinzia Cappiello , Valeria De Antonellis , Barbara Pernici. *P2S: A Methodology to Enable Inter-organizational Process Design through Web Services*. 2009.

[47] C. Cappiello, F. Daniel, M. Matera. *A quality model for mashup components*. 2009.

[48] C. Cappiello, F. Daniel, M. Matera, C. Pautasso. *Information quality in mashups*. 2010.

[49] F. Daniel, F. Casati, B. Benatallah, M.-C. Shan. *Hosted universal composition: models, languages and infrastructure in mashart*. 2009.

[50] XAML. *http://en.wikipedia.org/wiki/xaml*. 2011.

[51] A. Puerta, J. Eisenstein. *Ximl: A universal language for user interfaces*. 2010.

[52] XUL. *http://en.wikipedia.org/wiki/xul*. 2011.

[53] UIXML. *http://download.oracle.com/*. 2011.

[54] MARIAE. *http://giove.isti.cnr.it/tools/mariae/*. 2011.