POLITECNICO DI MILANO
Dipartimento di Elettronica e Informazione
Corso di Laurea in Ingegneria Informatica

Uno strumento di analisi prestazionale per servizi di ricerca nel contesto del Search Computing

Relatore: Prof. **Alessandro Siro CAMPI**

Tesi di laurea di:
Ilaria MAGRUCCI
Matr. 711370

Anno Accademico 2009-2010

Sommario

Indice delle figure

1 INTRODUZIONE

1.1 Abstract

Quando ci si trova a dover lavorare con i servizi web, come nel caso del progetto SeCo, può essere molto utile avere a disposizione uno strumento che effettui calcoli di tipo prestazionale; delle informazioni riguardanti il tempo di risposta o la quantità di dati restituiti possono essere utili per discriminare riguardo la scelta di un servizio per compiere un determinato tipo di interrogazioni. Lo strumento sviluppato, quindi, consente la registrazione di servizi, la definizione dei pattern di accesso ad essi e la formazione di insiemi di query di test. Mediante l'utilizzo di questo insieme di elementi, verrà fornito all'utente un insieme di metriche prestazionali che gli potranno essere utili per prendere decisioni sulla scelta di un servizio. Viene fornita, poi, una funzionalità di creazione automatica di un insieme di query di test. Sfruttando, quindi, alcune informazioni inserite dall'utente, il sistema riesce a formare un insieme di interrogazioni i cui risultati possano definire gli estremi di funzionamento del servizio. Lo strumento sviluppato al momento è completamente indipendente dall'attuale implementazione del progetto SeCo ma si pensa che, in un secondo tempo, possa venire integrato nel sistema come, ad esempio, funzionalità aggiuntiva in cascata alla registrazione di un servizio.

1.2 L'architettura di SeCo

Search Computing (SeCo) è un progetto finanziato dall'European Research Council(ERC), che risponde alla Call for "IDEAS Advanced Grants" del 2008, un programma dedicato al supporto delle ricerche basate sull'investigazione. SeCo è iniziato il 10 novembre 2008 e durerà 5 anni, fino al 31 ottobre 2013.

Il search computing si concentra sulla costruzione di risposte a complesse query di ricerca come "Dove posso seguire una conferenza che riguarda il mio campo di ricerca vicino ad una spiaggia assolata?" interagendo con una costellazione di servizi di ricerca cooperanti, usando criteri di ranking e join dei risultati come fattori dominanti per la composizione dei servizi.

Altri esempi di query che possono venire risolte tramite search computing sono: "Chi sono i più forti competitor europei in campo software?", "Chi è il migliore dottore per curare l'insonnia nell'ospedale vicino?".

Le informazioni per rispondere a queste domande sono disponibili sul Web, ma nessun sistema software può né accettare una query di questo genere né calcolare una risposta.

Il search computing è una nuova scienza multi-disciplinare che fornisce le astrazioni, i fondamenti, i metodi, e gli strumenti richiesti per rispondere a queste e molte simili interrogazioni.

Mentre i sistemi attuali rispondono a query generiche o specifiche per un certo dominio, search computing consente di rispondere a interrogazioni tramite una costellazione di servizi selezionati dinamicamente e cooperanti.

L'idea è semplice ma necessita, per la sua realizzazione, di un ampio studio, sia in termini di quantità di lavoro svolto, che di varietà di campi di interazione. C'è bisogno di nuove teorie che

facciano da fondamento, radicate in discipline formali come matematica, statistica e teoria dell'ottimizzazione. Per esprimere query e per scoprire servizi sono necessari un nuovo linguaggio e dei paradigmi descrittivi.

Nuove interfacce e protocolli possono essere d'aiuto per catturare le preferenze di ranking e consentire il loro raffinamento. La conoscenza del dominio semantico aiuta ad arricchire la conoscenza terminologica riguardo agli oggetti che vengono ricercati.

Il ranking è sempre relativo agli individui e al contesto, quindi lo studio del comportamento personale e sociale è essenziale. Le implicazioni economiche e legali del search computing vanno capite e padroneggiate.

Riassumendo, search computing è uno sforzo multi-disciplinare che richiede l'aggiunta a solidi principi software del contributo da altre scienze come la matematica, la ricerca operativa, la psicologia, la sociologia, la rappresentazione della conoscenza, le human-computer interfaces, le scienze economiche e legali.

Il search computing spazia su diversi campi di ricerca. Per prima cosa si può parlare di una teoria del search computing, intesa come lo studio delle astrazioni che coprono i concetti di servizi tradizionali o di servizi di ricerca e le interazioni tra essi. Sempre nell'ambito della teorizzazione del search computing rientra il design di operazioni base sui servizi, gli algoritmi per effettuare tali operazioni e il calcolo della complessità in termini di spazio e tempo di questi ultimi.

In cascata a questo studio c'è la modellazione statistica dei servizi di ricerca, ovvero la costruzione di stimatori statistici ad esempio del numero e della qualità dei risultati prodotti dai servizi o dalle operazioni sui servizi. Una branca importante del progetto SeCo, poi, si occupa dello studio dei metodi di ottimizzazione, vale a dire la costruzione di modelli di costo per le computazioni sui servizi di ricerca.

Dal momento che nel search computing ha una valenza molto importante il ranking, un campo di ricerca è proprio quello delle tecniche di esposizione di proprietà "ranking specific" del servizi, con lo scopo di guidare la selezione e la composizione di questi ultimi.

Grande spazio viene, ovviamente, dato alla costruzione di linguaggi per consentire l'espressione di query su servizi di ricerca. Tali linguaggi devono essere sviluppati in modo tale da incorporare gli aspetti relativi al ranking e le strategie per trattare con questi ultimi.

Dal momento che il ranking di un servizio si basa notevolmente sul giudizio dell'utente, grande importanza viene data alla costruzione di nuovi e originali paradigmi focalizzati sull'espressione delle preferenze. L'interazione con l'utente deve essere semplice, leggera e allo stesso tempo coinvolgerlo, collegando una richiesta alla successiva.

Il join dei servizi di ricerca richiede, inoltre, la risoluzione di problemi semantici. La ricerca nell'ambito del search computing deve incorporare alcuni risultati derivandoli tramite reasoning automatico, inferenza su ontologie e semantic web services.

Il search computing, inoltre, va a considerare anche un ranking di alto livello, ovvero inerente al servizio di ricerca e non solo al singolo risultato. Questo tipo di ranking, che può essere chiamato "ranking dei ranking" è un altro parametro di cui viene tenuto conto nell'orchestrazione dei servizi.

I sistemi di search computing comportano molti step computazionalmente pesanti: discovery dei servizi e selezione, ottimizzazione del piano delle query, ranking e merging dei risultati, interazione con le risorse ontologiche, modifiche adattative del piano delle query. La natura voluminosa di molti step, che può essere assimilata alle operazioni algebriche sui grafi, rende il search computing un'applicazione adatta ad essere assegnata a clouds di computing systems, in cui i nodi sono indistinguibili tra di loro.

Fattori come le strategie di ricerca correlata ai profili utente o alle interazioni passate dell'utente

possono migliorare significativamente la qualità percepita da ogni utente. Sia gli aspetti individuali che quelli sociali sono ingredienti chiave del search computing.

Si pone l'attenzione anche allo sviluppo di un metodo per specificare, progettare, assemblare e fare deploy di applicazioni software di search computing. Il metodo dovrebbe rivolgersi separatamente a service design e service orchestration, dove la prima attività produce gli ingredienti più appropriati da utilizzare (e riutilizzare) dalla seconda attività.

Il successo del search computing dipende anche dallo sviluppo di un'economia dei servizi di ricerca. Gli sviluppatori di applicazioni e i proprietari delle sorgenti di dati avranno bisogno di modelli di business adatti, basati su schemi pubblicitari, pay-per-query, tariffe di sottoscrizione, micro-billing, e così via.

Il search computing solleva nuovi problemi riguardo al controllo di come i dati vengono usati. Ad esempio, l'uso di un servizio di ricerca potrebbe essere concesso ad un'applicazione di service computing, purché il proprietario del servizio possa tenere traccia di tutte le query che coinvolgono i propri dati e limitare il tipo di informazione che viene resa visibile alle query.

Il planning di evoluzione del progetto SeCo comprende cinque fasi. Attualmente il progetto si trova nei primi due step della sua evoluzione. Nei primi mesi è stata definita un'infrastruttura di search computing che supporti le operazioni base per registrare i servizi, per registrare le query, per esprimere le query e le loro proprietà formali, per associarle con strategie di esecuzione, per calcolare il costo di ogni strategia di esecuzione e per determinare il ranking globale combinando i ranking locali per ogni singolo servizio di ricerca. I problemi da coprire, mano a mano che il progetto evolve, cresceranno in complessità, da situazioni semplici che caratterizzano combinazioni di due servizi di ricerca a situazioni più ampie aperte ad un numero arbitrario di servizi. Gli scenari considerati in questa prima fase iniziale sono prettamente statici, ma mano a mano che il progetto prenderà forma diventeranno sempre più dinamici e anche adattativi.

Parallelamente allo sviluppo del prototipo di infrastruttura, è stato portato avanti lo studio relativo a metodi e linguaggi che comprende la definizione di query su servizi di ricerca attraverso astrazioni di linguaggio e formalismi descrittivi. Sono proprio questi linguaggi che renderanno possibile lo sviluppo di applicazioni di search computing. L'enfasi sull'astrazione del linguaggio, quindi, si evolverà gradualmente in un insieme di metodologie, strumenti e best practices che aiutino gli sviluppatori nel progettare delle applicazioni software che beneficino del search computing.

Mano a mano che il progetto andrà avanti la scelta dei servizi diventerà sempre più dinamica e verrà migliorata l'efficacia del search computing grazie all'uso di vocabolari, tassonomie e ontologie sui domini selezionati. Come esperimento, è stato già preso in considerazione un particolare dominio su cui studiare e fare test, ovvero quello della bio-informatica. Nei mesi successivi si continuerà con gli esperimenti al fine di generalizzare ed astrarre alcuni metodi e tecniche generali.

Sia lo studio inerente le human-computer interfaces, l'evoluzione adattativa del sistema, che i modelli di business e di sicurezza devono ancora essere presi in esame in maniera concreta.

1.3 Il profilatore: descrizione, obiettivi, tecnologia e integrazione nell'architettura

Nell'ottica di una scelta dinamica dei servizi di ricerca, sorge spontaneo il problema di discriminare sulla bontà di un servizio rispetto ad un altro. Criteri di giudizio, in questo frangente, possono essere

il tempo di risposta del servizio, o la bontà dei risultati prodotti, o ancora la probabilità di failure del servizio durante l'interrogazione. Una riflessione di questo genere porta, quindi, alla necessità di collezionare informazioni riguardo l'utilizzo dei motori di ricerca, monitorando le performance di ricerca. Argomento della tesi, dunque, è lo sviluppo di uno strumento che riesca a collezionare un buon insieme di informazioni prestazionali che facciano da discriminante nella scelta di un servizio. Essendo ancora in una fase di progetto che coinvolge interrogazioni di tipo statico, le informazioni del profilatore sono fruibili dall'utente, il quale poi effettuerà la scelta del servizio più adatto alle sue esigenze, mentre non viene presa nessuna decisione in modo dinamico.

1.4La tecnologia utilizzata

Dal punto di vista della tecnologia, l'applicazione è stata realizzata in Java, utilizzando come tool di sviluppo Eclipse. Come database è stato scelto MySQL 5.1 con il relativo connector per gestire la comunicazione con Eclipse. Per la gestione della persistenza si è scelto di usare EJB 3 (Enterprise Java Bean), mentre come application server si è utilizzato JBoss v4.2.

Gli Enterprise JavaBean (EJB) sono i componenti che implementano, lato server, la logica di business all'interno dell'architettura Java EE. Le specifiche per gli EJB definiscono diverse proprietà che questi devono rispettare, tra cui la persistenza, il supporto alle transazioni, la gestione della concorrenza e della sicurezza e l'integrazione con altre tecnologie, come JMS, JNDI, e CORBA. Lo standard attuale, EJB 3 appunto, differisce notevolmente dalle versioni precedenti ed è stato completato nella primavera del 2006.

Le specifiche EJB intendono fornire una metodologia standard per implementare la logica di funzionamento delle applicazioni di tipo enterprise, applicazioni che, ad esempio, forniscono servizi via Internet. Per realizzare applicazioni di questo tipo è necessario affrontare una serie di problematiche tecniche che possono rivelarsi molto complesse e laboriose da risolvere. Gli Enterprise JavaBean intendono fornire una soluzione a questi problemi in modo da semplificare lo sviluppo di questo tipo di applicazioni.

Le specifiche EJB descrivono in dettaglio come realizzare un application server che fornisca persistenza, elaborazione delle transazioni, controllo della concorrenzialità e molte altre funzionalità.

Le specifiche, inoltre, definiscono il ruolo del contenitore di Enterprise JavaBean e di come far comunicare il contenitore con gli EJB.

Esistono tre tipi di EJB:

- Entity beans: il loro scopo è di inglobare gli oggetti sul lato server che memorizzano i dati. Gli entity bean forniscono la caratteristica della persistenza dei dati:
    - Persistenza gestita dal contenitore (CMP): il contenitore si incarica della memorizzazione e del recupero dei dati relativi a un oggetto utilizzando una tabella di una base di dati.
    - Persistenza gestita dal bean (BMP): in questo caso è il bean a occuparsi del salvataggio e recupero dei dati a cui fa riferimento, il salvataggio può avvenire in una base di dati o con qualsiasi altro meccanismo perché è il programmatore che si incarica di realizzare il meccanismo della persistenza dei dati.
- Session beans: gestiscono l'elaborazione delle informazioni sul server. Generalmente sono una interfaccia tra i client e i servizi offerti dai componenti disponibili sul server. Ne esistono di due tipi:

    - con stato (stateful). I bean di sessione con stato sono oggetti distribuiti che posseggono uno stato. Lo stato non è persistente, però l'accesso al bean è limitato ad un unico client.
    - senza stato (stateless). I bean di sessione senza stato sono oggetti distribuiti senza uno stato associato, questa caratteristica permette un accesso concorrente alle funzionalità offerte dal bean. Non è garantito che il contenuto delle variabili di istanza si conservi tra diverse chiamate ai metodi del bean.
- Message driven EJBs: sono gli unici bean con funzionamento asincrono. Tramite il Java Message Service (JMS), si iscrivono a un argomento (topic) o a una coda (queue) e si attivano alla ricezione di un messaggio inviato all'argomento o alla coda a cui sono iscritti. Non richiedono una istanziazione da parte dei client.

E' stata, inoltre, utilizzata la libreria HttpClient, insieme a JUnit, per formulare le chiamate http e per fare parsing delle risposte nella fase di test.

Oltre a Java, come linguaggi sono stati usati HTML, per la realizzazione delle pagine jsp, e, per comodità di interazione con il sistema, è stata inserita una piccola parte di codice JavaScript.

1.5L’integrazione con SeCo

Il profilatore è un componente completamente scisso dall’attuale implementazione di SeCo, scelta adottata per mantenerlo indipendente da eventuali cambiamenti radicali che il progetto di base può ancora subire, vista la fase di sviluppo in cui ci si trova. Il progetto di base, infatti, è ancora suscettibile di modifiche importanti su concetti basilari, tecnologie, scelte implementative di varia natura. Attualmente, quindi, il profilatore può essere visto come un tool che fa calcoli prestazionali su servizi. Per fare queste operazioni si avvale di uno strumento, lo scce, che si occupa dell’invocazione di un buon insieme di servizi. Si basa, inoltre, sull’applicazione di concetti analoghi a quelli adottati nel progetto SeCo, come servizi e pattern, mantenendo, però, un buon livello di autonomia. Si pensa che, in un futuro, il tool potrebbe essere integrato realmente nel progetto, consentendo l’esecuzione di test sulle performance di un servizio registrato, al fine di stabilire, staticamente o dinamicamente, quale sia il servizio più appropriato per svolgere un certo tipo di invocazione.

2 SEARCH COMPUTING

2.1 Introduzione

L'utilizzo dei pattern nell'ambito della gestione dei dati non è una novità, basti pensare all'utilizzo che si fa dei data marts nell'ambito del data warehousing. Nello stesso modo, i Service Marts sono semplici schemi che si riferiscono a "Web objects" nascondendo la struttura dei dati sorgente e presentando una semplice interfaccia, composta da attributi di input, output e rank; gli attributi possono avere valori multipli e possono venire clusterizzati mediante repeating groups. Quando si accede agli oggetti attraverso i Service Marts, i risultati solo liste di oggetti che possiedono un rank e che sono suddivise in chunk.

Il Search Computing consente di rispondere a domande attraverso una costellazione di servizi di ricerca cooperanti che sono correlati per mezzo di operazioni di join. Il Search Computing ha lo scopo di rispondere a query su diversi campi semantici di interesse; quindi il Search Computing riempie il dislivello tra i sistemi di ricerca generalizzati, che non sono in grado di trovare informazioni che coinvolgano argomenti multipli, e i servizi di ricerca specifici per un dominio che non possono andare oltre il limite del loro dominio.

Già in precedenza si sono fatti alcuni esempi di query tipiche che possono essere risolte tramite Search Computing.

Nel framework del Search Computing, si definisce Service Mart l'astrazione dei dati per la pubblicazione e la composizione delle sorgenti dei dati. L'obiettivo di un Service Mart è facilitare la pubblicazione di classi speciali di servizi software, chiamati servizi di ricerca, le cui risposte sono liste di oggetti con rank. Ogni Service Mart è mappato su un "Web object" disponibile su Internet; quindi, potremo avere dei Service Mart per "hotel", "voli", "dottori" e così via. Coppie di Service Mart, inoltre, possono essere potenziate con delle "connessioni" per supportare la loro unione. I Service Mart e le loro connessioni costituiscono una rete di risorse che può essere utilizzata come interfaccia di alto livello per esprimere query di ricerca.

2.2 Service Marts

I Service Mart sono astrazioni; pubblicare un Service Mart comporta creare un ponte tra una descrizione astratta a diverse implementazioni concrete dei servizi. Implementare un Service Mart, quindi, può richiedere il mapping verso diverse sorgenti di dati, ognuna configurata o come un Web Service, o come un API, o come una collezione di dati materializzati. Il concetto di Service Mart, quindi, offre un'astrazione per dare una visione "regolare" al mondo, insieme con una metodo e una tecnologia associata per costruire tale view regolare a partire da sorgenti di dati concrete.

2.2.1 Livello concettuale

Un Service Mart è un'astrazione che descrive una classe di oggetti Web. Ogni definizione di Service Mart, quindi, include un nome e un insieme di attributi esposti. I Service Mart hanno attributi atomici e gruppi ripetitivi, che consistono in un insieme non vuoto di sotto-attributi che complessivamente definiscono una proprietà. Gli attributi atomici hanno un solo valore, mentre i gruppi ripetitivi sono multi – valore. Per esempio, un Service Mart "Movie" ha gli attributi singoli ("Title", "Director", "Score", "Year", "Languages") e i gruppi ripetitivi ("Genres", "Openings", "Actors"), ognuno con sotto – attributi. Lo schema di un gruppo ripetitivo è introdotto da un livello di parentesi:

Movie (Title, Director, Score, Year, Genres(Genre), Language, Openings(Country, Date), Actors(Name))

Altri esempi di Service Mart possono essere

Cinema(Name, Address, City, Country, Phone, Movies(Title, StartTimes))

Restaurant(Name, Address, City, Country, Phone, Url, Rating, Category(Name))

I gruppi ripetitivi modellano delle proprietà multi – valore tra le istanze di Service Mart. Oltre ad aggiungere potere espressivo ai Service Mart, modellano anche delle relazioni 1:M o M:N, elementi concettuali il cui scopo è connettere oggetti del mondo reale. Concetti come "acts-in" tra "actor" e "movie" vengono modellati dai gruppi ripetitivi, mettendo gli attori come gruppo ripetitivo dei film o i film come gruppo ripetitivo degli attori (o entrambi).

2.2.2 Livello logico

A livello logico ogni Service Mart è associato con uno o più specifici access patterns. Un access pattern descrive il modo in cui si può accedere al Service Mart. E' una signature specifica del Service Mart, con la caratterizzazione di ogni attributo come input (I) o output (O), in relazione al ruolo che gioca l'attributo nella chiamata al servizio. Nel contesto dei database logici, un assegnamento di etichette I/O agli attributi di un predicato è chiamato decorazione, e quindi gli access pattern possono essere considerati dei Service Mart decorati. Inoltre, un attributo di output è progettato come rande (R) se il servizio produce i suoi risultati in un ordine che dipende dal valore di quell'attributo. Per facilitare la composizione dei servizi, si assume che tutti gli attributi rande restituiscano un valore normalizzato nell'intervallo [0…1]. Per esempio, per il Service Mart "Movie" si possono avere i seguenti access patterns:

Movie1 (Title(O), Director(O), Score(R), Year(O), Genres.Genre(I), Language(O), Openings.Country(I), Openings.Date(I), Actors.Name(O))

Movie2 (Title(I), Director(O), Score(R), Year(O), Genres.Genre(O), Language(O), Openings.Country(I), Openings.Date(I), Actor.Name(O))

In tutti i casi lo "Score" è un attributo di output usato per fare ranking dei film che sono presentati in ordine di score discendente, cioè con il film con il più alto punteggio per primo. "Country" e "Date" sono parametri di input, che sono usati per estrarre film di uno specifico paese dopo una specifica data. Nel primo access pattern i film sono cercati fornendo come input anche uno dei loro generi. Nel secondo access pattern, i film sono ricercati fornendo in input anche il titolo. Possono essere usati anche altri access pattern per accedere ai film fornendo il regista o un attore. La scelta

di un access pattern è una limitazione sul modo in cui si possono ottenere i dati, tipicamente imposta dalle interfacce dei servizi esistenti. Alle volte gli access pattern hanno più attributi del Service Mart originale. Si considerino i cinema e i ristoranti: il loro indirizzo è una caratteristica dell'oggetto sottostante, ma gli utenti che cercando cinema e indirizzi tipicamente forniscono al servizio un indirizzo in input (la propria abitazione o il posto in cui si trovano) e vogliono cercare il più vicino. Quindi, le versioni U degli attributi "Address", "City" e "Country" denotato la posizione dell'utente e le versioni T/R degli stessi attributi rappresentano la posizione del cinema / ristorante:

Cinema1 (Name(O), UAddress(I), UCity(I), UCountry(I), TAddress(O), TCity(O), TCountry(O), TPhone(O), Distance(R), Movie.Title(O), Movie.StartTimes(O))

2.2.3 Connection patterns

I connection pattern introducono una tecnica di accoppiamento dei Service Mart. Ogni pattern ha un nome concettuale e quindi una specifica logica, che consiste di una sequenza di semplici predicati di confronto tra coppie di attributi o sotto – attributi di due servizi, che sono interpretati come un'espressione Booleana congiuntiva, e quindi possono essere implementati mediante join dei risultati restituiti chiamando le implementazioni dei servizi. I connection pattern possono essere diretti o indiretti.

Per esempio, Movies e Cinemas sono connessi attraverso il connection pattern non orientato "Shows" che usa il join sui titoli:

Shows(Movie, Cinema): [(Title=Title)]

L'interpretazione dei join nei connection pattern è esistenziale: se il titolo del film è uguale al titolo di un qualsiasi film proiettato nel cinema allora il predicato è soddisfatto e quindi le due istanze del film e del cinema sono composte senza effettuare nessuna selezione sui sub – attributi (nell'esempio, se un titolo del cinema soddisfa il join allora tutti i film proiettati al cinema vengono selezionati).

Si considera, quindi, una connessione diretta tra cinema e ristoranti: un pattern diretto può essere usato a partire dal primo verso il secondo (la query ricerca prima i cinema e poi i ristoranti vicini). La connessione è specificata come la congiunzione di espressioni predicato, che collegano l'indirizzo del cinema alla posizione in input di un servizio relativo ai ristoranti, di modo che, dopo aver determinato un cinema (vicino all'indirizzo dell'utente) il servizio sarà invocato usando la posizione del cinema come input per la ricerca del ristorante:

DinnerPlace(Cinema, Restaurant): [(TAddress=UAddress), (TCity=UCity), (TCountry=UCountry)]

Logicamente, i connection pattern sono espresso tra coppie di tipi ordinate di attributi compatibili. Un connection pattern deve essere supportato da una coppia di access pattern. Tutti gli attributi di entrambi gli access pattern selezionati devono avere delle etichette che possano combaciare; se sia l'operando di destra che quello di sinistra hanno una label O, allora il pattern è non direzionato, altrimenti se l'operando di sinistra ha etichetta O e quello di destra ha etichetta I allora il pattern è diretto da sinistra a destra.

Visualmente, i Service Mart e i connection pattern logici possono essere presentati come grafi delle risorse, in cui i nodi rappresentano i mart e gli archi rappresentano i connection pattern logici. Il modello del Web del Search Computing, quindi, presenta una semplificazione della realtà, vista attraverso dei grafi delle risorse potenzialmente molto grandi. Tale rappresentazione rende possibile la selezione di concetti interconnessi che supportano la creazione e l'estensione dinamica di query multi – dominio.

2.2.4 Livello fisico

Al livello fisico dei Service Mart si modellano le service interface, in cui ogni service interface viene mappata su una sorgente di dati concreta. Una service interface potrebbe non supportare alcuni degli attributi di un Service Mart, ad esempio perche a una sorgente potrebbe mancare una proprietà.

Un service interface è una unità di invocazione e come tale va descritta non solo con il suo schema concettuale o la decorazione logica, ma anche con le sue propietà fisiche. Ci sono un gran numero di opzioni per caratterizzare servizi data – intensive, sia in termini di performance che di qualità. Le interfacce dei servizi sono descritte attraverso quattro tipi di parametri:

- Descrittori di ranking: classificano la service interface o come un servizio di ricerca (produce risultati ranked) o come un servizio esatto (produce oggetti non ranked). I servizi esatti sono associati ad una selectivity, che è un numero positivo che esprime il numero medio di tuple prodotte da ciascuna chiamata. Quando a un servizio di ricerca è associato un access pattern che ha uno o più attributi di output segnati come R, allora il ranking viene detto esplicito, altrimenti viene detto opaco.
- Chunk descriptor: hanno a che fare con la produzione di output da parte di una service interface. Il servizio è chunked quando può essere invocato ripetutamente e ad ogni invocazione viene restituito un nuovo insieme di oggetti, tipicamente in numero fissato, di modo da fornire un ritrovamento progressivo di tutti gli oggetti del risultato; in questo caso espone una chunk size (numero di tuple nel chunk). Ovviamente, se il servizio è rande, allora il primo chunk contiene gli oggetti con ranking più alto.
- Cache descriptors: si occupano delle invocazioni ripetute ad un servizio. Un modo molto efficiente di velocizzare le invocazioni ad un servizio consiste nel caching lato richiedente delle risposte restituite per un dato input, e quindi usare queste risposte memorizzate invece di invocare il servizio. Questa politica, però, non è accettabile con molti servizi, ad esempio quelli che offrono risposte real – time. I parametri, quindi, indicano se una service interface può fare caching e in tal caso qual è il cache decay, cioè il tempo trascorso tra due chiamate alla sorgente che rende l'uso delle risposte memorizzate tollerabile.
- Cost descriptors: si occupano di associare ogni chiamata a servizio con un costo; questo può anche essere espresso come tempo di risposta o costo monetario.

2.3 Search Computing Framework

Le applicazioni di Search Computing sono costruite sfruttando un approccio a framework configurabile. Il Service Mart Framework fornisce le basi per wrappare e registrare le sorgenti di dati; il Service Invocation Framework maschera i problemi tecnici dell'interazione con un Service Mart registrato. Lo User Framework fornisce funzionalità e strumenti di memorizzazione per registrare gli utenti con differenti ruoli e autorizzazioni. Il Query Framework supporta la gestione e la memorizzazione delle querce che possono essere eseguite, salvate, modificate e pubblicate così che gli altri utenti possano vederle. Il Query Processing Framework è un componente centrale dell'architettura che fornisce un servizio per eseguire query multi . dominio. Il Query Manager si prende cura di separare le query in sotto – query e collegarle alle relative data source; il Query Planner produce un piano di esecuzione ottimizzato per le query che detta la sequenza di step per eseguire e query. Infine, l'Execution Engine esegue il piano di esecuzione sottoponendo le chiamate ai servizi designati attraverso il Service Invocation Framework, costruisce quindi i risultati combinando gli output prodotti dalle chiamare ai servizi, calcolando il ranking globale e producendo un risultato ordinato secondo l'importanza complessiva.

Per ottenere una specifica applicazione di Search Computing l'architettura general – purpose appena descritta va personalizzata con l'aiuto di tool orientati a programmatori, utenti esperti e utenti finali.

- Service Publishers: registrano le definizioni del Service Mart nel service Repository e dichiarano i connection pattern utilizzabili per eseguire il join. I service publishers hanno il compito di implementare mediatori, wrapper o componenti di materializzazione dei dati, al fine di rendere le sorgenti di dati compatibili con l'interfaccia standard del Service Mart e con il comportamento atteso.
- Expert Users: configurano le applicazioni di Search Computing selezionando i Service Mart di interesse, scegliendo le sorgenti di dati per supportate il Service Mart e connettendoli attraverso i connection patterns.
- End Users: usano le applicazioni di Search Computer configurate dagli utenti esperti. Interagiscono sottoponendo query, ispezionando risultati e facendo evolvere il loro bisogno di informazioni in relazione ad un approccio esploratore di ricerca di informazione che viene chiamato Liquid Query.

2.4 Registrazione e adattamento dei Web Service

Nel Search Computing, le sorgenti di dati dovrebbero produrre output rande e l'estrazione dei dati dovrebbe essere eseguita incrementalmente, mediante "chunks".

Si distinguono tre scenari differenti:

- I dati possono essere interrogati per mezzo di Web service o combinando i risultati di Web service differenti.
- I dati sono disponibili sul Web ma devono essere estratti da siti Web mediante wrappers.

- I dati non sono accessibili direttamente e vanno prima materializzati.

I risultati restituiti da una chiamata ad una service interface espongono un formato di interscambio scritto in JSON (JavaScript Object Notation), un formato per lo scambio di dati facile da leggere e scrivere per gli umani e comodo per il parsing e la generazione da parte delle macchine.

Il formato deriva direttamente dalla descrizione concettuale del Service Mart, quindi tutte le istanze di un Service Mart condividono lo stesso formato di interscambio, senza badare alla service interface che li produce. Si riporta un'istanza JSON di "movie":

{"title": "Highlander",
"director": "Russell Mulcahy",
"score": "0.7",
"year": "1986",
"genres": [{"genre": "action"}],
"openings": [{"country": "US", "date": "31-10-1986"}],
"actors": [{"name": "Christopher Lambert"},
{"name": "Sean Connery"}]}

2.4.1 Web Services

La tipica implementazione del servizio è un Web service reale registrato nella piattaforma. I Web service restituiscono il loro output in un formato arbitrario, quindi lo sviluppatore dell'implementazione del servizio deve definire una serie di trasformazioni sui risultati e raggrupparle in un'implementazione remora del servizio che nasconda le caratteristiche peculiari di ogni sorgente remota.

Per affrontare il bisogno di combinare i risultati di differenti Web service è stato costruito il Service Mart Framework che contiene dei moduli software predefiniti utili per manipolare i dati. Tali moduli si occupano sia della conversione in JSON che della manipolazione della chunk size.

2.4.2 Web pages

Il secondo tipo di sorgenti che si può voler usare sono le pagine HTML. Il Web è ricco di informazioni di buona qualità memorizzate in siti Web HTML. I wrapper sono dei programmi particolari che possono rendere disponibili i dati pubblicati nel Web. Nel contesto dei Service Mart, i wrapper possono essere usati per catturare i dati che sono pubblicati dai Web server in formato HTML, dal momento che in questi casi una conversione di dati è necessaria per supportare l'integrazione delle sorgenti di dati, questi devono essere riordinati in relazione allo schema normalizzato del Service Mart. Un altro uso tipico dei wrapper nel Search Computing ha luogo quando un servizio risponde con documenti HTML che devono essere tradotti nello schema normale e codificati in JSON. Vi sono molti modi per costruire wrapper e, in particolare, è stato scelto di utilizzare Lixto.

2.4.3 Database materializzati

Anche se la maggior parte delle implementazioni dei servizi richiedono una chiamata ad un servizio remoto, in alcuni casi dati riassunti e materializzati possono aver bisogno di essere memorizzati nel sito del motore. La materializzazione di dati è un processo generale, che può essere applicato alle sorgenti per trasformare il loro formato, per eliminare ridondanze, per aumentare la loro qualità e così via; la materializzazione dei dati è anche molto utile per supportare l'esecuzione efficiente delle query. Dal momento che la cattura di un dato snapshot di dati è intrinseca al processo di materializzazione, l'approccio può essere usato solo con dati che cambiano raramente.

E' stato sviluppato un materializzatore che è specifico per l'utilizzo nell'ambito del Search Computing. Il materializzatore è un componente software i cui obiettivi sono leggere sorgenti di dati arbitrarie e organizzare i dati in un formato normalizzato, adatto all'esportazione dei dati in accordo con una definizione di Service Mart.

2.4.4 Applicazioni e casi d'uso

Si descrive una interazione di ricerca che riguarda i film, i cinema e i ristoranti "vicini a", e il raffinamento e l'esplorazione dei risultati attraverso l'applicazione di filtri locali addizionali. Si supponga che l'utente sottometta una query chiedendo in cinema con un film con punteggio alto di un certo genere, vicino a un buon ristorante vegetariano. Una volta che l'utente ha inserito i parametri di ricerca, la query viene eseguita e i risultati mostrati in una tabella di risultati. La pagina dei risultati è anche arricchita con delle opzioni di interazione che l'utente può scegliere. Una volta che i risultati sono mostrati l'utente può interagire con essi attraverso i comandi disponibili. Alcune operazione (come opzioni di visualizzazione ed espansioni a nuovi servizi) richiedono che l'utente selezioni un sottoinsieme delle istanze risultato; la selezione viene eseguita per mezzo di checkboxes. Si possono inoltre applicare filtri locali sui valori delle colonne e aggiungere dimensioni di ricerca (come l'orario dei trasporti pubblici).

2.5 Ottimizzazione data – driven della composizione dei servizi di ricerca

Per agevolare la composizione dei servizi di ricerca per risolvere query multi – dominio è stato sviluppato un framework concettuale che va a sfruttare diverse risorse quali WordNet e lo Stanford Natural Language Parser. E' stato tentato un approccio che fa leva su WordNet e le suo funzionalità per annotare query, servizi e domini. Stabilito che la semantica ottenuta in questo modo è troppo limitata, è stato aggiunta una quantità significativa di feedback utente e raffinamento manuale nella definizione di servizi domini e queries. Sicuramente, questo framework, una volta sviluppato in tutti i suoi componenti sarà un banco di prova molto interessante per misurare il livello di feedback e di raffinamento e per misurare allo stesso tempo la deviazione tra l'elaborazione automatica e l'elaborazione guidata dal feedback umano. Nel seguito si andrà ad analizzare come si comporta tale framework nel risolvere la query "dove posso seguire una conferenza scientifica sui database vicino ad una spiaggia bellissima raggiungibile con voli economici".

2.5.1 Architettura generale e flussi di esecuzione

Nell'ambito della risposta a query multi – dominio di identificano due flussi di attività principali: il flusso di registrazione (che si occupa della dichiarazione e della descrizione dei domini, e della descrizione dei servizi di ricerca e la loro associazione ai domini) e il flusso di esecuzione della query (che si occupa dell'effettiva pro cessazione delle query.

Nel flusso di registrazione si affrontano i seguenti problemi: la rappresentazione semantica, la memorizzazione, la gestione e l'accesso ai domini e le loro descrizioni; la descrizione semantica, la memorizzazione, la gestione e l'accesso ai servizi di ricerca; il clustering di servizi sulla base della similarità; il mapping dei servizi nei domini; e la definizione dei condizioni di join ammissibili tra servizi.

Nel flusso di esecuzione della query si affrontano i seguenti problemi: la definizione di interfacce apposite per la presentazione di query utente multi – dominio; la divisione di query in sub – queries; il mapping delle sub – queries nei domini; il mapping di sub – queries su domini dati verso servizi di ricerca associati per definire query di basso livello; la generazione di query plans e la loro valutazione rispetto a diverse metriche di costo per scegliere il più promettente per l'esecuzione; la generazione e l'esecuzione di piani di esecuzione delle query; e la trasformazione e il rendering dei risultati per il consumo da parte dell'utente.

La registrazione viene eseguita dagli sviluppatore, e il framework li può aiutare nella selezione dei domini e dei servizi, annotandoli e creando mapping tra essi. Le queries sono eseguite dagli utenti il cui feedback aiuta nel risolvere ambiguità e confermare interpretazioni.

2.5.2 Flusso di registrazione

Il flusso di registrazione comprende tutte le attività coinvolte nella registrazione dei domini, nella descrizione dei domini e dei servizi di ricerca.

Il domain framework si occupa dei domini e delle loro definizioni e affronta i problemi delle annotazioni semantiche, della memorizzazione, della gestione e dell'accesso ai domini e alle loro descrizioni.

L'intera infrastruttura del motore di ricerca multi – dominio è basata sul concetto di dominio. Intuitivamente, si considera un dominio come un campo di interesse per l'utente, come musica, sport, arte, turismo, computer science e così via. Ogni dominio è associato ad una label distintiva e associato con una insieme di WordNet synset: ogni synset può essere associato con più domini, e tale associazione sarà in seguito caratterizzata da una distribuzione di probabilità. Questo consente di caratterizzare un dominio sulla base dei termini più frequentemente utilizzati per descrivere i concetti in quel dominio, e viceversa di identificare per ogni synset la lista dei domini a cui si riferisce. Uno dei compiti più interessanti nel search computing è di investigare se i domini di Wordnet hanno un sufficiente potere discriminante per aiutare a partizionare le query e associarle a specifici motori di ricerca e sorgenti di dati.

Si assume che i domini siano organizzati come una tassonomia, in una struttura ad albero di relazioni tra domini e sottodomini. L'informazione sui domini viene resa disponibile agli altri componenti attraverso un' API che espone le interfacce per interrogare e aggiornare la struttura del

dominio (cioè la creazione, la cancellazione, e l'aggiornamento delle informazioni del dominio, includendo i synset e i servizi associati).

Per quanto riguarda l'esempio di partenza, sono richiesti i domini delle Conferenze Scientifiche, delle Spiagge e dei Voli. Le Spiagge sono incluse in un dominio più generale delle Risorse Geografiche ed entrambe le Risorse Geografiche e i Voli sono trovati come sotto – domini dei Viaggi.

2.5.3 Il framework dei servizi di ricerca

Il framework dei servizi di ricerca definisce un modello concettuale del servizio di ricerca e affronta l'annotazione semantica, la memorizzazione, la gestione e l'accesso ai servizi di ricerca.

La funzione centrale svolta dal framework è di rendere possibile l'annotazione delle interfacce di richiesta / risposta dei servizi. Tale fase di annotazione usa il vocabolari di Wordnet ed etichetta ciascun servizio, le sue operazioni e i parametri di input – output di ogni operazione.

Il service repository espone servizi in termini di operazioni; ogni operazione è descritta per mezzo di un set di attributi funzionali o non – funzionali. Gli attributi che la qualificano sono l'id, il nome, il descrittore (annotazione Wordnet), il nome del servizio, i descrittori di input e output e i tipi, il tempo medio di risposta e il costo di interazione con le operazioni. In aggiunta, diversi parametri descrivono gli aspetti che qualificano i servizi di ricerca: la descrizione del ranking, la descrizione del caching, la descrizione del decay, la descrizione del chunking.

I parametri più importanti per quanto riguarda la QoS sono la scalabilità, la capacità (il limite delle richieste concorrenti per garantire le performance), la performance (misura della velocità nel completare una richiesta di servizio), il tempo di risposta, la latenza, il throughput, l'affidabilità, la robustezza, l'accuratezza, e la completezza. Questa informazione è cruciale quando è il momento di scegliere tra diversi servizi e ottimizzare l'esecuzione della query sulla base di modelli di costo.

Il processo di registrazione è semi – automatico: un utente interagisce con il service registration framework per aggiungere uno specifico servizio. Se la sua descrizione WSDL non è disponibile, una descrizione va esplicitamente definita. Il sistema analizza il nome del servizio, delle operazioni e dei parametri e cerca di assegnarli ad un dominio, associando un possibile termine e un possibile synset in WordNet. Questo processo è semi – automatico dal momento che richiede differenti interazioni con l'utente (ad esempio per ogni termine il sistema mostra all'utente possibili synset differenti e gli chiede di scegliere il più appropriato). Una volta che questa fase è finita, il sistema colleziona automaticamente informazioni implicite o nascoste.

Nell'esempio proposto, la registrazione dei servizi relativi alle conferenze può portare ad integrare diversi servizi, ad esempio con scelte differenti si parametri di input ed output, dal momento che le conferenze possono essere cercate per argomento, nome, posizione, data di inizio o alcuni di questi input in combinazione. Un servizio di questo tipo, pertinente al nostro running example può essere:

ConfSearch (topic, nameX, placeX, dateY)

dove topic è un parametro di input, name, place e date sono parametri di output e il servizio e i parametri sono poi annotati da descrittore Wordnet. I risultati sono collezioni di triplette di nomi, posti e date associate ad una dato argomento di input, il risultato può essere sia ranked che non; il

ranking può prendere in considerazione l'importanza relativa data da una comunità di utenti alla conferenza stessa; il servizio può essere caratterizzato in termini di disponibilità e di accuratezza.

2.5.4 Analizzatore del servizio

L'analizzatore di servizio affronta i seguenti problemi: il clustering dei servizi disponibili, sulla base della loro similarità; il mapping di servizi sui domini; e la definizione di connessioni di join tra servizi. Questa parte del framework richiede che le altre parti siano implementate come prototipo, quindi si trova in una fase estremamente prematura.

Lo scopo del processo di clustering è il raggruppamento di tutti i web service disponibili e le loro associazioni probabilistiche che un insieme di domini. Il processo di clustering sfrutterà la disponibilità di annotazione di tipo synset per i servizi e i loro parametri di input / output, e farà uso anche della presenza della semantica associata con i web service associati per associarli a uno dei domini di WordNet. Mentre il processo di clustering sarà periodicamente effettuato, ci si immagina un processo incrementale di aggiunta di un servizio ad un cluster esistente. Agli sviluppatori sarà chiesto di verificare la correttezza delle scelte effettuate dal sistema. In aggiunta gli verrà offerto di espandere il dominio ontologico dei domini di WordNet con altri termini e relazioni.

Durante il clustering dei servizi e l'associazione con i domini, sarà anche costruita dell'informazione riguardo al "grado di appartenenza" di ogni servizio al corrispondente cluster. Tale informazione potrà poi venire usata nella fase di selezione dei servizi di ricerca.

Il secondo maggiore compito dell'analizzatore dei servizi è la definizione dei cammini di join ammissibili tra i servizi. L'obiettivo di questo compito è di identificare, per ogni coppia di servizi che può essere invocata per risolvere una query, gli attributi di join che saranno usati per comporre i loro risultati. Una soluzione possibile a questo problema richiede per prima cosa la classificazione dei servizi in ogni cluster, calcolato sulla base delle loro interfacce. In questo modo, per ogni coppia di classi dipendenti da domini differenti, si possono identificare parametri che hanno lo stesso tipo e annotazioni, che sono candidate per essere qualificate come attributi di join. Quindi, il processo di accoppiare i servizi viene progressivamente eseguito, con l'aiuto degli sviluppatori, che possono dire se i cammini di join identificati dal sistema possono essere usati per connettere domini, e , se così, come elementi dei cammini di join dovrebbero essere accoppiati e le condizioni di join essere completamente qualificate.

Per quanto riguarda l'esempio, l'obiettivo di questo passo è identificare che servizi come ConfSearch e Flights possono essere connessi facendo match su posto, date di inizio e di fine e volo; inoltre, la posizione della conferenza è usata come destinazione di un viaggio da un luogo scelto dall'utente, la data di inizio viene usata per designare la data per il primo viaggio, e la data di fine viene usato come data di ritorno. Il servizio dei voli viene finalmente identificato in uno associato con ranking, e specificamente con un criterio di ranking basato sul costo totale del volo.

2.5.5 Analisi della query

Una query di alto livello è la specifica di un bisogno di informazione di un utente ad un alto livello di astrazione. Si assume che le query di alto livello siano delle descrizioni dei bisogni dell'utente in linguaggio quasi – naturale, che possono richiedere informazioni da domini multipli. L'unica

restrizione che viene imposta è che le queries devono consistere in un insieme di noun phrases, cioè frasi la cui testa sia un nome o un cognome, opzionalmente accompagnato da modificatori.

Il componente di analisi della query decompone le query di alto livello in sub – queries, ognuna rappresentante un obiettivo di ricerca in un dominio specifico. La query dell'esempio, quindi, potrebbe diventare Q1 = "DB scientific conference?" Q2 = "Place closet o a beautiful beach?", e Q3 = "Place reachable with cheap flight? "

Per il parsing della query in linguaggio naturale si utilizza un tool open source sviluppato dallo Stanford Natural Language Processing Group.

All'uscita del parser su ottiene una rappresentazione ad albero delle frase che è adatta per il problema di dividere le query in sub – query per essere assegnate a domini differenti.

2.5.6 Query to domain and service mapping

Si tratta di un componente che affronta i problemi di mappare le sotto – query nei domini e di mappare le sotto – query in associati servizi di ricerca per definire queries di basso livello.

L'operazione di mappare una query su un dominio può andare a buon fine solo se: ogni sotto – query comprende solo richieste ad un dominio; e se le parole usate nella sotto – query non sono ambigue, il che consente una definizione chiara della loro semantica.

Possono essere applicate diverse tecniche per ottimizzare il riconoscimento di strutture di query e sotto – query:

- L'invocazione iterativa del tool NLP
- Lo sfruttamento delle annotazioni dei servizi di ricerca o dei domini
- Analisi sintattica / logica dei risultati

Per migliorare la tecnica si aggiunge anche:

- La coerenza sui domini synset
- Il feedback utente

Tornando all'esempio precedente, si possono osservare dettagliatamente i passaggi:

- Identificare i synset delle parole: rainy ha un solo synset (ADJ: (1) showery, rainy); US ha quattro synset (NOUN: (2) uracil, (3) uranium, (4) U, (5) United States), lo stato ha diversi synset.
- Raggruppamento dei synset per dominio: si assume che: il synset (1) è associato al dominio "tempo"; i synset (2), (3), (9) sono associati a "chimica"; i synset (4) e (7) sono associati alla grammatica; i synset (5), (6), (8) sono associati alla "geografia ".
- Selezione di un synset per ogni parola (in termini di probabilità)
- Se la differenza di probabilità tra due opzioni non è così alta, il feedback utente può essere richiesto in termini di domande come: "stai chiedendo informazioni riguardo geografia o chimica?"

Una volta identificato il dominio, tecniche analoghe possono essere applicate per mappare le query sui servizi. L'ultimo obiettivo di questo compito è di far combaciare ognuna delle sub query identificate su uno o più servizi. Questo passo richiede una partecipazione importante da parte dell'utente.

Nel caso dell'esempio trattato precedentemente si otterrebbe:

S1 = ConfSearch(“DB”,Title,Place,StartDate, EndDate);

S2 = PhotoSearch(“Beach”,Stars, PhotoID,Place);

S3 = RoundTripFlight(From,To,FromDate,ToDate,TotalTime,
TotalCost, RounTripID)

2.5.7 Query planner

Una query di basso livello è una query congiuntiva sui servizi. Un query plan è un ben definito scheduling di invocazioni dei servizi, possibilmente in parallelo, che si adatta ai loro modi di accesso ed espone l’ordine di ranking in cui il servizio restituisce i risultati per costruire un risultato globale ranked.

L’originalità del modello risiede nell’introdurre una classificazione dei servizi semplice ma efficace: i servizi esatti hanno un comportamento relazionare e restituiscono sia una riposta singola che un insieme di risposte senza ranking, i servizi di ricerca restituiscono una lista di risposte in ordine di ranking, in relazione a qualche misura di rilevanza.

I piani di query progettano le invocazioni ai web service e la composizione dei loro input e output. Un piano è definito come l’orchestrazione delle invocazioni ai servizi, possibilmente in parallelo, che tiene in conto le caratteristiche più significative del servizio, inclusa la sua abilità di dividere in chunk i risultati. Nell’ambito dei piani, le operazioni principali sono i join tra i risultati dei web service.

Il Query Planner si appoggia sul concetto di access pattern e progressivamente rifinisce le scelte attraverso i seguenti step:

1. Sceglie specifici access patterns per ognuno dei servizi coinvolti nella query
2. Una volta che gli access pattern sono definiti, ci può essere ancora dell’indeterminatezza nell’ordine di esecuzione. Il Query Planner fissa tale ordine.
3. Seleziona una strategia di esecuzione per ogni join.
4. Determina il numero di richieste atteso relativamente ad ogni servizio per ottenere il numero di risultati desiderato, di modo da associare ad ogni piano un costo di esecuzione.

Il Query Planner ricerca un piano ottimo considerando tutte le scelte fattibili nel contesto e riducendo il suo spazio di ricerca attraverso un’esplorazione branch – and – bound che associa i costi attesi con ogni scelta.

Un esempio di query che può essere considerata dall’ottimizzatore è:

BestMatch(ConfTitle, Place, Stars, DateStart, DateEnd,
TotalCost, TripId, PhotoID) :-
ConfSearch(“DB”, ConfTitle ,Place, DateStart, DateEnd),
PhotoSearch(“Beach”, Stars, PhotoID, Place),
RoundTripFlight(“[*InputCity*]”, Place, DateStart, DateEnd,
TotalTime, TotalCost, TripID),
DateStart>[*InputPeriodStart*], DateEnd<[*InputPeriodEnd*]

La query viene presentata con l’input utente parametrico che indica la città e il periodo di interesse.

Il risultato prodotto dal query planner è la selezione dell'access pattern che minimizza il costo di interazione con i servizi, producendo un dato numero di risultati attesi in output; i risultati sono liste di elementi, ordinati in base alla combinazione del costo basso e dell'alto numero di stelle per la spiaggia.

2.5.8 Query engine

Il query engine si occupa della generazione e dell'esecuzione dei piani di esecuzione delle query; questo include operazioni a granularità fine , come le invocazioni dei servizi, e quindi facilita il controllo dell'esecuzione, anche in presenza di parallelismo. L'input di questo step è il piano delle query generato dal pianificatore. A questo livello, il piano può includere l'allocazione esplicita della cache per memorizzare risultati parziali delle sotto – queries e porzioni di join pre – calcolato tra risultati di servizi invocati di frequente con gli input più frequenti, come anche la specifica delle strategie di esplorazione per lo spazio di ricerca dei join che non sono semplicemente espressi dalla combinazione delle semplici iterazioni. Si sta attualmente lavorando sulla selezione di un numero limitato di nodi che rappresentino le operazioni riusabili e su schemi efficienti per il passaggio di parametri tra nodi, di modo da dare al linguaggio, allo stesso tempo, un gran potere espressivo e una buona abilità nell'essere usato per fare generazione e testing delle strategie di join. I risultati generati dai nodi del servizio e le combinazione restituite dai nodi di join sono collezionate nel loro formato "riga" di tuple e valori, e passato al modulo di Trasformazione dei Risultati, per essere processato per essere presentato all'utente.

E' anche responsabilità del query engine, inoltre, di affrontare comportamenti non attesi e applicare politiche di correzione. Riguardo a questo argomento, un gran numero di tecniche sono in fase di studio approfondito.

Parallelamente allo studio di tutti questi argomenti, ovviamente, vi è anche la progettazione di un'interfaccia che consenta la sottomissione di query multi – dominio, come anche la trasformazione e il rendering dei risultati per il consumo da parte dell'utente.

Nei prossimi mesi si proseguirà con lo studio degli argomenti descritti e con l'implementazione dei singoli componenti.

3 ANALISI DEI REQUISITI

3.1Descrizione del problema.

Il profilatore è un software che si propone di fornire all'utente che deve fare una query complessa, risolvibile tramite search computing, un insieme di informazioni prestazionali riguardanti un determinato servizio. Grazie a queste informazioni, l'utente sarà messo nelle condizioni di decidere se effettuare una query utilizzando un servizio piuttosto che un altro.

Il profilatore è uno strumento che, prima di tutto, va istruito; questo significa che per prima cosa sarà compito dell'utente inserire delle informazioni fondamentali per effettuare tutto il testing successivo.

In particolare, è compito dell'utente definire un servizio, i suoi pattern di accesso, ovvero gli insiemi di parametri attraverso cui questo può venire invocato, delle query concrete da effettuare come test.

Una volta in possesso di tutte queste informazioni, il profilatore, ad intervalli regolari o a comando, potrà effettuare i test e aggiornare i propri dati.

Le metriche più importanti relativamente ad un servizio di ricerca sono: il tempo di risposta, il rapporto tra il tempo di risposta e la dimensione dei dati restituiti, la dimensione totale dei dati restituibili, l'affidabilità dei risultati restituiti e la probabilità di failure.

Si è pensato che l'approccio più corretto con il problema fosse quello di, prima di progettare e sviluppare il software, iniziare ad analizzare manualmente le performance di alcuni servizi. A tal proposito, sono stati effettuati un buon insieme di test preliminari per studiare, ad esempio, l'andamento dei tempi di risposta o delle dimensioni dei dati restituiti; scopo di tale studio era individuare quali fossero effettivamente i dati di cui tenere traccia e in che modo farlo.

3.2Test preliminari.

Sono stati presi in considerazione tre dei servizi invocabili attraverso lo SCCE (GoogleTheatres, Yahoo!Restaurants ed IgnyteMovieService) ed effettuato un insieme di test orientati al funzionamento e alle performance.

Tutti i test sono stati effettuati sfruttando le api della libreria open source HttpClient. La libreria mette a disposizione metodi per effettuare chiamate http di vario tipo e per gestire le relative risposte.

Scopo principale dei test è rilevare i tempi di esecuzione, che sono stati prelevati sempre a cavallo del metodo execute () della libreria HttpClient.

l primo test è stato effettuato su dati che si sapevano essere funzionanti per i 3 motori.

Per ogni servizio è stata effettuata una sequenza di 100 query identiche e il test è stato ripetuto cinque volte. Sono state testate due versioni, differenti in un dettaglio relativo al codice. Nella prima variante le 100 chiamate vengono effettuate sempre sullo stesso oggetto HttpGet, mentre nella seconda per ognuna delle 100 chiamate viene creato un nuovo oggetto HttpGet.

In totale, quindi, sono stati rilevati i dati dieci volte.

Per ulteriore verifica, è stato eseguita un'altra volta l'ultima versione del test in un momento in cui la rete che si stava utilizzando per l'esperimento era completamente scarica.

Per il servizio Google Theatres i parametri della query erano: Marina, San Francisco, California, 1.

Per il servizio Yahoo! Restaurants i parametri della query erano: Marina, San Francisco, California, Meat, 1.

Per il servizio Ignite Movie Service i parametri della query erano: 20131, 1000, 1.

L'ultimo parametro, per ogni servizio, è il chunk index, ovvero l'intero che identifica il numero di chunk che è stato richiesto. In questo caso viene sempre richiesto il primo chunk.

Per questo primo test, nel codice si è provveduto a fissare a 0 i tempi di risposta per le query che avessero risposto con status code diverso da 200. Nei test successivi, qualora si incontrasse una situazione del genere, sarà anche studiato il relativo tempo di risposta.

Google Theatres risponde con un chunk da sei elementi, nonostante la sua chunk size sia dieci. Il primo chunk, quindi, in questo caso, è anche l'ultimo. Gli elementi riportano informazioni sintetiche sia sul cinema che sugli spettacoli in programmazione.

Yahoo! Restaurants risponde con un chunk di cinque elementi, che non è l'ultimo. La chunk size di questo servizio, infatti, è proprio cinque. Ogni elemento riporta informazioni abbastanza sintetiche sul locale consigliato.

Ignite Movie Service risponde con un chunk di 50 elementi di dimensioni variabili, in quanto ogni elemento corrisponde ad un film ed al suo interno contiene informazioni sui cinema che hanno in programmazione quel film.

Per quanto riguarda i tempi di esecuzione, l'analisi verrà fatta servizio per servizio e solo successivamente sarà tentato un paragone tra i risultati.

La prima query verso Google Theatres ha sempre un tempo di esecuzione di circa un secondo; tutti i tempi di esecuzioni indicati per esteso, salvo ulteriori specifiche, vanno intesi in nano-secondi.

Si riportano i tempi di esecuzione della prima query nei vari test.

GoogleTheatres - Prima esecuzione	
Variante 1	Variante 2
1.324.356.169	1.259.322.064
1.299.443.575	1.397.165.352
1.258.191.405	1.258.850.287
1.344.854.583	1.293.933.447
1.296.387.949	1.280.447.096

Il test effettuato per ulteriore verifica fa registrare un risultato assolutamente simile ai precedenti, ovvero 1.290.387.116.

Per quanto riguarda le 99 query successive, nonostante esse siano in tutto e per tutto uguali a quella precedente, i tempi di esecuzione sono molto inferiori, anche di diversi ordini di grandezza.

Si riportano le medie delle 99 esecuzioni per ognuno dei dieci test.

GoogleTheatres – Media 99 esecuzioni successive	
Variante 1	Variante 2
3.660.150	5.091.426
3.439.891	5.141.819
3.446.834	5.751.922
3.401.583	5.089.538
3.588.218	4.991.896

Anche in questo caso il test effettuato per ulteriore verifica conferma i risultati, in quanto mostra un valore di 7.058.410; in realtà, date le modalità di svolgimento di questo ultimo test, ci si sarebbe aspettati un valore più basso degli altri, e si può dire che questo risultato, sempre appartenente allo stesso ordine di grandezza degli altri test, confermi i risultati precedenti. Osservando nello specifico i risultati di questo ultimo test, si può notare che, rispetto agli altri casi, si trova un'incidenza maggiore di tempi di esecuzione appartenenti all'ordine di grandezza appena superiore (ovvero 10-20 x 10°6).

Si osserva che nella seconda variante c'è un leggero incremento; questo potrebbe essere dovuto al cambiamento nel codice tra le due varianti (ovvero aver effettuato la chiamata, nel secondo caso, sempre su oggetti HttpGet diversi).

Osservando i risultati si può notare rapidamente che l'incremento non è dovuto a valori alti e isolati ma ad un incremento globale di tutti i tempi di esecuzione. Mentre nella prima variante i risultati oscillavano sempre tra due e tre, in questo secondo caso i valori che si presentano più spesso sono compresi tra 3 e 6.

Passando ad analizzare i risultati relativi a Yahoo! Restaurants si può notare un comportamento assolutamente simile al precedente. La prima query, infatti, ha sempre un tempo di esecuzione di circa un secondo. Per completezza, si riporta la tabella con i valori delle prime esecuzioni della query su Yahoo! Restaurants.

Yahoo!Restaurants - Prima esecuzione

Variante 1	Variante 2
1.306.965.340	1.434.772.576
1.285.429.363	1.270.263.050
1.277.180.759	1.360.906.205
1.289.620.120	1.266.634.516
1.233.814.563	1.317.749.533

Come nel caso precedente, l'ultimo test effettuato per ulteriore verifica conferma i risultati precedenti in quanto restituisce come valore 1.553.425.645.

Per quanto riguarda la media delle 99 query successive, come prima si riportano i risultati in una tabella riassuntiva.

Yahoo!Restaurants – Media 99 esecuzioni successive	
Variante 1	Variante 2
3.412.436	7.306.661
3.411.388	5.271.981
3.505.969	4.947.799
3.652.830	5.195.858
3.276.402	7.379.628

Anche in questo caso il test effettuato per ulteriore verifica conferma i dati precedenti, in quanto ha una media di 7.701.899, perfettamente in linea con il resto dei test dello stesso genere.

Possiamo osservare che, come per il servizio Google Theatres, nella seconda variante c'è un incremento generalizzato dei tempi di esecuzione. In questo caso, oltre ad avere dei tempi globalmente più alti che nella prima versione, ci sono spesso diversi tempi molto più alti della media.

Per quanto riguarda l'ultimo servizio, ovvero Ignite Movie Service, possiamo osservare un certo cambiamento nei tempi di esecuzione. La prima query, infatti, per quasi tutti i test, da' dei risultati intorno ai tre secondi. Come per i servizi precedenti, si riportano i risultati in una tabella.

IgnyteMovieService - Prima esecuzione	
Variante 1	Variante 2
3.719.309.019	3.725.702.428

3.503.859.150	3.323.105.800
3.994.484.215	3.225.174.987
3.469.498.847	3.632.871.846
18.746.164.247	3.836.534.252

Si può dire anche stavolta che il test effettuato come ulteriore verifica confermi i risultati precedenti, in quanto restituisce un valore di 6.090.563.814.

Sorge il dubbio che l'aumento globale nei tempi di esecuzione potrebbe essere dovuto alla grande discrepanza nella dimensione dei risultati tra i due servizi precedenti e quest'ultimo.

Anche le medie dei 99 risultati successivi sono globalmente più alte dei primi due servizi, come mostrano i dati riportati nella tabella.

IgnyteMovieService – Media 99 esecuzioni successive	
Variante 1	Variante 2
4.362.460	5.571.177
4.410.074	6.075.715
4.607.320	5.563.367
4.233.394	6.056.266
4.289.069	5.961.950

L'ultimo test ha media 9.111.495, quindi, per ragionamenti analoghi a quelli già fatti in precedenza per altri motori, si può dire che confermi i risultati.

Giunti a questo punto viene spontaneo domandarsi se vi sia correlazione tra la dimensione dei risultati restituiti ed il tempo di esecuzione della query.

A tal proposito, sempre prendendo in considerazione l'ultimo servizio, sembra sensato andare a testare nuovamente la stessa query modificando solo il raggio di ricerca. Per un primo tentativo si porta il raggio da 1000 miglia a 10 miglia, con lo scopo di avere un numero più basso di risultati e controllare se con essi diminuiscano anche i tempi di esecuzione.

Con tali valori viene restituito un chunk non pieno, quindi anche ultimo. Gli elementi che compongono il chunk sono 30 e i tempi di esecuzione, per quanto riguarda la prima query, sono effettivamente più bassi come si può osservare nella seguente tabella riassuntiva.

IgnyteMovieService – Prima esecuzione – Radius = 10

1.559.440.166
1.345.259.384
2.148.478.717
1.656.495.378
1.599.221.270

Per quanto riguarda, invece, le medie delle altre 99 query, i tempi di esecuzione sono in linea con i precedenti, non c'è nessuna significativa diminuzione nei valori, anzi sono leggermente aumentati, come si può osservare dalla seguente tabella.

IgnyteMovieService – Media 99 esecuzioni successive – Radius = 10
7.782.007
6.109.947
10.496.260
8.453.106
8.719.382

Provando a ripetere lo stesso test ma con raggio 5 vengono restituiti 25 risultati; si è sempre, quindi, nella situazione di avere il primo chunk che è anche l'ultimo e con un numero di risultati non troppo differente dal caso precedente. Si procede, dunque, con l'analisi dei tempi d'esecuzione per capire se ci sia correlazione tra la dimensione del risultato e il tempo di risposta.

Per quanto riguarda la prima query, i tempi d'esecuzione sono assolutamente simili al caso precedente, come si può osservare dalla seguente tabella.

IgnyteMovieService – Prima esecuzione – Radius = 5
1.037.316.894
1.595.725.854
1.277.153.801
1.350.176.349
1.266.581.786

Come ulteriore verifica, allora, si procede all'esecuzione della stessa query assegnando un miglio soltanto come valore del raggio. Purtroppo, vengono restituiti sempre 25 risultati e, per ragioni implementative, non si può scendere ulteriormente con la dimensione del raggio.

3.2.1 Test sulla cardinalità dei risultati

Ignite Movie Service è, per via dei parametri di querying, il servizio più adatto per ricercare una correlazione tra il numero di risultati restituiti e il tempo di esecuzione; questo perché è possibile fissare uno zip code e modificare il raggio di ricerca in modo da cercare di ottenere un numero maggiore o minore di risultati.

Si procede, dunque, con un buon insieme di test su vari zip code e vari raggi di ricerca con lo scopo di avere un numero sufficiente di dati per poter avanzare delle teorie sul rapporto tra il tempo di risposta e la dimensione dei risultati.

I test sono stati effettuati andando a ricercare sia località che presumibilmente potevano fornire un grosso numero di risultati anche in presenza di un raggio ristretto, che località con pochi risultati anche nel caso di ricerche di ampi raggio.

Per prima cosa si va ad esaminare il tempi di risposta relativamente alla prima query nel caso di raggio di un miglio. Per chiarezza si riassumono tutti i dati rilevanti nella seguente tabella.

IgnyteMovieService – Prima esecuzione - Radius = 1			
Zip code	Num risultati	Località	Tempo d'esecuzione
82701	5	Newcastle	863.317.475
59633	6	Canyon Creek	1.592.559.599
70447	16	Madisonville	1.479.545.864
33109	30	Miami Beach	1.638.050.227
94128	35	San Mateo	1.709.526.700
33480	39	Palm beach	1.665.230.427
94102	47	San Francisco	2.077.187.184
60601	Chunk pieno (50)	Chicago	2.227.290.620
10125	Chunk pieno(50)	New York	7.549.393.917
02108	Chunk pieno(50)	Boston	1.860.658.477

Da questo primo test sembrerebbe che, per quanto riguarda la prima interrogazione, i tempi d'esecuzione si modifichino leggermente in relazione al numero di risultati. Per adesso non sarà tenuto conto della quantità di dati internamente al singolo risultato; il chunk, infatti, è composto, in

questo caso, da elementi “film”, i quali contengono informazioni sui cinema che li hanno in programmazione. Per adesso si andrà ad esaminare solo il rapporto tra tempi di risposta e numero di elementi; successivamente si procederà anche con l’analisi della quantità effettiva di dati restituiti.

Si procede con la stessa analisi ma prendendo in considerazione la media delle 99 query successive.

IgnyteMovieService – Media 99 query successive - Radius = 1			
Zip code	Num risultati	Località	Tempo d’esecuzione
82701	5	Newcastle	6.719.286
59633	6	Canyon Creek	5.435.410
70447	16	Madisonville	5.527.231
33109	30	Miami Beach	5.619.199
94128	35	San Mateo	5.876.840
33480	39	Palm beach	6.442.301
94102	47	San Francisco	6.043.485
60601	Chunk pieno (50)	Chicago	6.295.920
10125	Chunk pieno(50)	New York	6.813.841
02108	Chunk pieno(50)	Boston	5.515.636

Quest’ultimo test sembra rivelare che, per quanto riguarda la media di un buon numero di esecuzioni successive della stessa query, il tempo di esecuzione non varia di molto in relazione al numero di risultati restituiti.

Sembra sensato, a questo punto, andare a studiare, per ognuno dei casi sopra elencati, la variazione nei tempi di esecuzione in relazione alla modifica di raggio di ricerca.

Facendo un po’ di tentativi sul primo test, ovvero il caso di Newcastle, si può osservare che il numero di risultati resta sempre cinque, anche quando si imposta il raggio a 1000.

Si riportano, quindi, nella seguente tabella, i primi risultati sperimentali.

Newcaste			
Raggio	Num risultati	Prima esecuzione	Media 99 successive
1	5	863.317.475	6.719.286
1000	5	925.222.968	5.420.626

Si può osservare che, non essendoci state significative modifiche nella dimensione del risultato, anche i tempi di esecuzione non sono variati in maniera evidente.

Il caso successivo, ovvero quello di Canyon Creek, è assolutamente analogo al precedente. Imponendo, infatti, un raggio di 500 miglia, i risultati restituiti sono sempre 6, il che non ci consente di fare uno studio di rapporto tra la mole dei risultati e il tempo di esecuzione, ma serve comunque come studio del comportamento del servizio.

Si riportano i risultati in una tabella come nel caso precedente.

Canyon Creek			
Raggio	Num risultati	Prima esecuzione	Media 99 successive
1	6	1.592.559.599	5.435.410
500	6	735.891.363	5.501.707

Con il caso successivo, ovvero quello di Madisonville, si può iniziare ad osservare un'evoluzione dei risultati relativamente alla modifica del raggio. Ponendo, infatti, il raggio di ricerca a 100 miglia, si passa da 16 a 23 risultati restituiti.

Si riassumono i risultati nella seguente tabella.

Madisonville			
Raggio	Num risultati	Prima esecuzione	Media 99 successive
1	16	1.479.545.864	5.527.231
100	23	1.630.071.978	5.905.582

Un risultato molto interessante è quello relativo alla query su Miami Beach. In questo caso, infatti, se si interroga il servizio con raggio 100, si trova un chunk che non è l'ultimo ma ha una mole di dati molto alta (tanto che non si riescono a visualizzare tutti in consolle).

I risultati riguardo ai tempi di esecuzione sono evidenziati in tabella.

Miami Beach			
Raggio	Num risultati	Prima esecuzione	Media 99 successive
1	30	1.638.050.227	5.619.199
100	molti	4.279.271.039	10.003.272

Il risultato dell'invocazione con raggio 100 fa pensare che ci sia una possibile correlazione tra la mole dei risultati restituiti (quindi non solo il numero ma proprio la quantità di dati) e il tempo di esecuzione della query. Verrà, quindi, successivamente effettuata un'analisi di correlazione tra il numero di caratteri della stringa di risposta e il tempo di esecuzione, per fare un'analisi più mirata a quella che è effettivamente la mole dei risultati restituiti.

Il test su San Mateo ha consentito un'analisi approfondita del rapporto tra cardinalità del risultato e tempo di risposta, in quanto, a differenza dei test precedenti, all'aumentare del raggio varia anche il numero dei risultati. Per prima cosa è stato effettuato un test grossolano per capire se con raggio 100 il numero di risultati, da 35 che erano con raggio 1, si modificava notevolmente. Con raggio 100 il servizio rispondeva con un chunk pieno, quindi si è andato a cercare il raggio critico per cui si passava dall'avere tutti i risultati su un singolo chunk all'avere il primo chunk completamente pieno. Nella ricerca del raggio critico, ovviamente, si è anche andata ad analizzare la variazione dei tempi e del numero dei risultati, come si può osservare dalla seguente tabella.

San Mateo			
Raggio	Numero risultati	Prima esecuzione	Media 99 successive
1	35	1.709.526.700	5.876.840
10	35	1.704.661.835	5.951.095
15	35	1.733.431.694	6.033.507
16	Chunk pieno(50)	2.346.713.828	6.713.554
16 (verifica)	Chunk pieno(50)	2.176.074.683	6.611.475
18	Chunk pieno(50)	2.183.926.379	6.128.403
20	Chunk pieno(50)	2.442.227.364	6.394.593
22	Chunk pieno(50)	2.876.755.565	6.559.592
24	Chunk pieno(50)	6.598.966.209	7.665.234
25	Chunk pieno(50)	4.675.971.457	7.027.341
50	Chunk pieno(50)	8.150.906.749	6.848.627
100	Chunk pieno(50)	7.867.133.812	6.846.184

Anche questo caso porta alla luce una correlazione di qualche tipo tra dimensione dei risultati e tempo d'esecuzione, ma, dal momento che a parità di cardinalità di risultato il tempo di esecuzione cambia, viene spontaneo avanzare due ipotesi:

- Il tempo d'esecuzione dipende in qualche modo dalla mole dei risultati restituiti (nel singolo chunk)

- Il tempo d'esecuzione dipende in qualche modo da tutti i possibili risultati della query, ovvero non solo quelli del chunk richiesto ma tutti quelli che corrispondono a quel particolare zip code e a quel particolare raggio di ricerca

I test sulla mole dei risultati restituiti e restituibili saranno affrontati in seguito, dopo aver finito di esaminare quelli relativi alla cardinalità dei risultati.

Si passa, ora, ad esaminare il test effettuato sulla località di Palm Beach, il quale sembra confermare la teoria del rapporto tra la dimensione dei risultati e il tempo di esecuzione, almeno relativamente alla prima query.

Palm Beach			
Raggio	Numero risultati	Prima esecuzione	Media 99 successive
1	39	1.665.230.427	6.442.301
10	39	1.717.601.539	6.772.238
100	Chunk pieno(50)	2.948.883.028	7.454.546

Questo test, come tutti quelli che hanno evidenziato variazioni significative nel tempo d'esecuzione, sarà rivisitato in seguito, con maggior attenzione alla quantità effettiva di risultati restituiti. In particolare, prima si andrà a vedere di quanti caratteri è composta la stringa del risultato e successivamente di quanti chunk e qual è il totale dei dati che compone il risultato globale.

Per quanto riguarda il test su San Francisco, si è scelto di non continuarlo, in quanto con raggio di un miglio venivano già restituiti 47 risultati, quindi aveva poco senso lavorare sulla cardinalità in un caso del genere. Un discorso analogo è stato fatto nel caso di Chicago, che con raggio un miglio restituiva già un chunk pieno.

Nel caso della città di New York, anche se già con raggio di un miglio si otteneva un chunk pieno, si è comunque proceduto con qualche test, riflettendo sul fatto che il numero di cinema poteva essere tale che, aumentando il raggio, i dati interni al primo chunk potevano cambiare notevolmente.

I dati sperimentali sembrano confermare l'ipotesi formulata, come si può osservare dalla seguente tabella.

New York			
Raggio	Numero risultati	Prima esecuzione	Media 99 successive
1	Chunk pieno(50)	7.549.393.917	6.813.841
100	Chunk pieno(50)	21.062.138.553	5.819.551

10000	Chunk pieno(50)	20.973.071.019	5.921.464

L'ultimo test ha un raggio di ricerca molto alto ma, come hanno evidenziato altri test che, però, non sono stati riportati in questa sede perché fuori contesto, sembra che vengano considerati dei limiti al raggio. Questo comportamento è palese nel caso di ricerche effettuate in zone con poca concentrazione di cinema o semplicemente di abitato. Partendo da un numero ridotto di risultati (ad esempio 5) ed andando progressivamente ad aumentare il raggio di ricerca (anche arrivando a dimensioni assolutamente esagerate) non si possono osservare variazioni nei risultati; questo probabilmente perché viene sempre e comunque tenuto conto che il cinema richiesto deve rispondere ad un determinato zip code.

Il test su Boston, infine, non è stato sviluppato sempre per via del chunk già pieno con raggio di un miglio ed eventualmente lo si considererà nei test successivi.

3.2.2 Test sulla quantità effettiva dei risultati

Per prima cosa si va a riesaminare il test su cui si sono fatte più analisi nel caso della cardinalità, ovvero quello relativo a San Mateo. Si tiene ovviamente traccia del tempo di esecuzione di modo che i dati siano coerenti e si riporta in tabella come d'abitudine una sintesi dei risultati.

San Mateo			
Raggio	Quatità risultati	Prima esecuzione	Media 99 successive
1	30.033	1.952.751.537	11.457.651
10	30.033	1.655.973.594	6.560.847
15	30.033	1.649.870.096	6.066.946
16	52.183	1.921.165.679	7.803.228
18	52.183	2.250.797.518	7.674.690
20	52.183	2.293.986.037	8.268.871
22	61.337	2.700.795.562	7.209.261
24	61.337	2.565.311.094	7.318.783
25	61.337	2.609.252.154	7.773.717
50	68.256	7.427.800.252	8.296.847
100	68.256	7.421.211.355	8.110.131

Questi primi risultati sembrano verificare l'ipotesi di correlazione tra la mole di risultati restituiti e il tempo di esecuzione, ma per ulteriore verifica si procede con l'analisi di altri casi già esaminati in

cui si è presentato un cambiamento radicale nei tempi o, viceversa, nessuna modifica a parità di cardinalità di risultato.

A tal proposito si parte con l'analisi del caso relativo a Newcastle, in cui sia con raggio uno che con raggio 1000 venivano restituiti lo stesso numero di risultati e non vi erano significative modifiche nel tempo di esecuzione. Ci si aspetta che il test riveli che la dimensione dei dati restituiti è effettivamente simile, come anche i tempi di esecuzione.

Newcastle			
Raggio	Quantità risultati	Prima esecuzione	Media 99 successive
1	1.481	827.540.162	5.206.621
1000	1.481	840.447.878	5.809.735

L'evidenza sperimentale, in questo caso, sembra confermare l'ipotesi. A parità di quantità di dati restituiti, il tempo d'esecuzione è assolutamente simile.

Tra gli esempi precedentemente analizzati, ora è interessante andare ad analizzare quello su Miami Beach; la query in questione restituiva 30 risultati con raggio un miglio, e un risultato molto più ampio, anche se inferiore a 50 elementi, con raggio 100.

Si ripete, quindi, il test, con anche l'aggiunta del caso con raggio pari a mille miglia e si riportano i risultati nella seguente tabella.

Miami Beach			
Raggio	Num risultati	Prima esecuzione	Media 99 successive
1	35.065	1.717.864.701	6.256.660
100	101.987	3.615.859.348	9.497.117
1000	101.987	3.596.619.123	8.492.954

Appare evidente che effettivamente, soprattutto per quanto riguarda la prima esecuzione, vi sia una forte correlazione tra la dimensione dei dati restituiti e il tempo di esecuzione. Si decide, quindi, di non analizzare nello specifico ora i casi di Palm Beach e New York, ma di passare all'analisi della correlazione tra il tempo di esecuzione e la quantità totale dei dati che possono essere restituiti variando il chunk index.

3.2.3 Test sulla quantità totale dei dati possibili sulla base della variazione del chunk index

Osservando la tabella relativa al test precedente su San Mateo viene spontaneo domandarsi se ci sia una motivazione per quell'incremento così significativo nel tempo di esecuzione delle ultime due query.

La prima cosa da andare a verificare è, indubbiamente, un eventuale rapporto tra la totalità dei risultati che è possibile restituire (ovviamente sempre divisi in chunk) e il tempo d'esecuzione della query in questione.

Si va a studiare, quindi, l'andamento dei risultati a cavallo della grossa variazione nei tempi di esecuzione, ovvero negli ultimi quattro casi.

San Mateo		
Raggio	Numero di chunk	Dettaglio lunghezze chunk
24	1	61337
25	1	61337
50	2	68256
		72785
100	2	68256
		72785

Questo primo test, tenuti presente i risultati relativi ai tempi di esecuzione indicati in precedenza, sembra confermare la tesi del rapporto tra il numero dei possibili risultati e il tempo di esecuzione della singola query che va a richiedere solamente un chunk.

Come ulteriore verifica si sceglie di analizzare il caso di Palm Beach, considerando, stavolta, per essere più sintetici, il tempo di esecuzione, la dimensione del chunk richiesto e quella totale, al variare del chunk index.

Palm Beach			
Raggio	Tempo di esecuzione	Dimensione primo chunk	Dimensione totale dati disponibili
1	838.521.307	23585	23585
10	840.202.875	23585	23585
100	1.882.189.356	44429	77723

Anche questo test finisce per confermare la teoria della correlazione tra la quantità dei risultati totali e il tempo di risposta.

Visti i risultati dei test effettuati, quindi, si ritiene utile lo sviluppo di un sistema che analizzi le

prestazioni dei diversi servizi, tenendo in qualche modo traccia sia dei tempi di esecuzione che della quantità dei risultati disponibili.

Sembra, quindi, sensato indirizzare l'utente verso la creazione di test composti da insiemi di query di varia natura, per rappresentare al meglio il comportamento globale del servizio di ricerca. L'ideale sarebbe, dunque, avere dei test formati da query per così dire di "lower bound" e di "upper bound" ovvero interrogazioni che si suppone abbiano rispettivamente pochi e molti risultati. Nel set di query di test sarebbe anche ideale avere delle query di feedback, ovvero delle interrogazioni di cui si conosce già il risultato; sarà l'utente ad assegnare un punteggio in base alla bontà dei risultati restituiti. Nell'insieme delle query di test, poi, vi potranno essere un numero arbitrario di query per così dire generali.

Ad ogni invocazione di un servizio registrato, poi, potranno venire aggiornate le statistiche con i dati di quella particolare invocazione.

L'unico attore nel sistema, quindi, è l'utente, che registra il servizio, definisce degli insiemi di query di test, può comandare una nuova esecuzione dei test o semplicemente ispezionare i risultati.

Ci si aspetta, dunque, di poter visionare un riassunto delle caratteristiche di ogni servizio, sia in termini di pattern e test set definiti, che di prestazioni medie. Tali dati riassuntivi, ovviamente, riporteranno anche eventuali punteggi assegnati al servizio tramite l'utilizzo di query di feedback. Sarà possibile definire nuovi pattern, modificare pattern precedentemente definiti, aggiungere insiemi di test, aggiungere query di vario genere a tali insiemi, apportare modifiche a valori già definiti. Sarà, inoltre, disponibile una funzionalità di creazione automatica dell'insieme di test che, dato un servizio, si occuperà di creare, secondo certe regole, un insieme di query di lower bound e di upper bound che definiscano i valori estremi di funzionamento del servizio.

3.3 Scenari e casi d'uso

3.3.1 Scenario 1: registrazione di un nuovo servizio

Use Case	New Service
Pre - conditions	1. L'utente visualizza la home page del sistema
Normal flow	2. L'utente accede alla pagina di registrazione di un nuovo servizio tramite il controllo per l'aggiunta di un nuovo servizio presente nella home page. 3. L'utente inserisce il nome del servizio e clicca sul tasto di aggiunta.
Post - conditions	4. L'utente viene riportato sulla home page e nell'elenco dei servizi disponibili sarà presente anche quello appena registrato.
Situazioni di errore	5. Nel caso in cui sia già presente un servizio con quel nome, oppure non venga inserito alcun

	nome, l'utente viene avvisato tramite una pagina di errore. Attraverso un controllo presente sulla pagina sarà possibile tornare alla home del sistema.

3.3.2 Scenario 2: visualizzazione dei dati riassuntivi relativi ad un servizio

Use case	inspectService
Pre - conditions	1. L'utente visualizza la home page del sistema 2. C'è almeno un servizio registrato nel sistema e, quindi, presente nell'elenco del servizi nella home page.
Normal flow	3. L'utente sceglie un servizio e fa click sul suo nome.
Post - conditions	4. Viene visualizzata la pagina con i dati relativi al servizio selezionato, con la possibilità, attraverso dei comandi, di apportare modifiche agli elementi che lo compongono.
Situazioni di errore	Non vi sono situazioni di errore da segnalare.

3.3.3 Scenario 3: registrazione di un nuovo pattern di accesso al servizio

Use case	NewPattern
Pre - conditions	1. L'utente, tramite il comando di inspect, ha visualizzato i dati riassuntivi relativi ad un servizio.
Normal flow	2. L'utente agisce sul comando di aggiunta di un nuovo pattern presente sotto alla lista di quelli già esistenti e accede alla pagina di creazione del pattern 3. L'utente seleziona il numero di parametri di cui dovrà essere composto il pattern. 4. Vengono create le relative caselle di input. 5. L'utente inserisce i nomi dei parametri che

	compongono il pattern e agisce sul comando di aggiunta.
Post - conditions	6. L'utente viene riportato sulla home page che contiene i dettagli del servizio e nella lista dei pattern troverà anche quello appena inserito.
Situazioni di errore	7. Se il pattern è composto da campi nulli il sistema lo segnala tramite una pagina di errore, attraverso cui si può ritornare alla home del sistema. 8. Se il pattern è composto da parametri con lo stesso nome, il sistema lo segnala tramite una pagina di errore, attraverso cui si può ritornare alla home del sistema. 9. Se il pattern è già stato definito per quel servizio, il sistema lo segnala tramite una pagina di errore, attraverso cui si può ritornare alla home del sistema. 10. Se uno o più parametri del pattern contengono caratteri non validi, il sistema lo segnala tramite una pagina di errore, attraverso cui si può ritornare alla home del sistema. Nel nostro caso, tali caratteri dipendono dal modo in cui si è scelto di restituire i dettagli relativi ad un pattern e sono “,”, “/” e “+”.

3.3.4 Scenario 4: registrazione di un insieme di query di test

Use case	DefineBuiltInTest
Pre - conditions	1. L'utente, tramite il comando di inspect, ha visualizzato i dati riassuntivi relativi ad un servizio.
Normal flow	2. L'utente agisce sul comando di aggiunta di un nuovo test set presente sotto all’elenco di quelli già esistenti e accede alla pagina di creazione del test. 3. L'utente inserisce il nome del nuovo test nel form e agisce sul comando di aggiunta.

Post - conditions	4. L'utente viene riportato alla pagina che riassume i dati del servizio in cui sarà presente il nuovo test set appena inserito.
Situazioni di errore	5. Se non viene fornito nessun nome per il test, viene visualizzata una pagina di errore da cui si può tornare alla home del sistema. 6. Se il nome fornito è già presente tra i nomi dei test per quel servizio, viene visualizzata una pagina di errore da cui si può tornare alla home del sistema. 7. Se il nome contiene un carattere non valido, viene visualizzata una pagina di errore da cui si può tornare alla home del sistema. Nel nostro caso, i caratteri considerati non validi sono “,”, “/” e “+”.

3.3.5 Scenario 5: Aggiunta di query ad un test set

Use case	AddQuery
Pre - conditions	1. L'utente, tramite il comando di inspect, ha visualizzato i dati riassuntivi relativi ad un servizio.
Normal flow	2. Agendo sul comando di inspect relativo ad un particolare BuiltInTest, naviga verso la pagina dei dettagli relativi al test. 3. Nella pagina del test l'utente visualizza le query presenti nel set e i dettagli relativi alll’eventuale ultima esecuzione del test. 4. L’utente agisce sul comando di aggiunta di una nuova query. 5. L'utente accede alla pagina di selezione del pattern che sarà di base per quella particolare query, seleziona il pattern che vuole utilizzare e agisce sul comando per andare avanti ad inserire i valori reali che formeranno la query. 5. L'utente compila il form creato sulla base della scelta pattern con i valori dei parametri

	della query. Nella pagina è anche presente il controllo per la scelta della tipologia della query. 6. L'utente agisce sul comando di salvataggio della query.
Post - conditions	7. L'utente viene riportato alla pagina di riassunto dei dati relativi al servizio a cui ha appena aggiunto una query. Per visualizzare la query appena inserita basterà visualizzare i dati relativi al test set in cui si è scelto di aggiungere la query.
Exceptions	8. Se i valori inseriti per i parametri della query non sono corretti, ovvero contengono dei caratteri non validi, come ",", "/" e "+", viene visualizzata una pagina di errore attraverso cui si può tornare alla home del sistema. 9. Anche in caso di valori nulli o vuoti, all'utente viene segnalato l'errore tramite una pagina apposita da cui è possibile navigare verso la home page del sistema. 10. Nel caso in cui la query che si sta cercando di inserire fosse già presente viene visualizzata una pagina di errore da cui è possibile tornare alla home del sistema. 11. Nel caso in cui non vi siano pattern definiti per quel determinato servizio, al momento della scelta del pattern per la query viene visualizzato un messaggio che invita l'utente a tornare indietro e definirne uno.

Nella pagina dei dettagli del BuiltInTest sarà anche presente, oltre ai dati fondamentali relativi alle query che lo compongono, un riassunto dei risultati dell'ultima esecuzione del test set. A causa delle dimensioni dei dati da memorizzare, si è scelto di salvare i risultati dell'esecuzione solo per le query di tipo feedback e di usare come strumento di memorizzazione la scrittura su file. Per ogni query di feedback, quindi, viene creato un file "idDellaQuery.json" in cui sono salvati i risultati dell'ultima esecuzione. Nel momento in cui si andrà ad ispezionare un test set, sarà possibile visualizzare, quindi, anche i risultati dell'ultima esecuzione di ogni query di feedback dell'insieme, il punteggio assegnato per quel particolare risultato, e il punteggio medio calcolato sulla base dello storico degli score.

3.3.6 Scenario 6: esecuzione di una configurazione di test

Use case	BuiltInTestExecution
Pre – conditions	1. L'utente, tramite il comando di inspect, ha visualizzato i dati riassuntivi relativi ad un servizio.
Normal flow	2. Agendo sul comando di esecuzione relativo ad un test set, l'utente da' il via alla procedura di esecuzione.
Post - conditions	3. Vengono visualizzate le query che verranno eseguite e chiesta all'utente un'ulteriore conferma di esecuzione. 4. L'utente agisce sul comando di esecuzione. 5. Viene visualizzata la pagina dei risultati di esecuzione. Per ogni query vengono visualizzati i dati più importanti relativamente ai tempi di esecuzione e alla dimensione dei risultati restituiti. 6. Nel caso in cui siano presenti delle query di feedback nell'insieme, vengono anche visualizzati i loro risultati e viene chiesto all'utente di inserire lo score, di modo da aggiornare anche quel valore. 7. L'utente salva i punteggi delle query di feedback 8. L'utente viene riportato nella pagina riassuntiva dei dati relativi al test che ha appena eseguito, in cui potrà anche visualizzare i dettagli relativi all'esecuzione appena terminata.
Situazioni di errore	Non vi sono situazioni di errore di rilievo da segnalare.

3.3.7 Scenario 7: rimozione di un servizio

Use case	RemoveService

Pre - conditions	1. L’utente ha già registrato il servizio che vuole eliminare e si trova o nella home del sistema o nella pagina di visualizzazione dei dettagli del servizio.
Normal flow	2. L’utente agisce sul comando di rimozione del servizio presente a fianco al nome. 3. Viene visualizzata la pagina di conferma della rimozione del servizio. 4. L’utente agisce sul comando di conferma della rimozione del servizio.
Post - conditions	5. L’utente viene riportato alla home page del sistema.
Situazioni di errore	Non vi sono situazioni di errore di rilevo da segnalare. Nel caso in cui l’utente non volesse eliminare il servizio e sufficiente tornare alla home del sistema tramite il link apposito. Ogni dato presente nel database relativo a quel servizio viene cancellato, così come i file relativi ad eventuali query di feedback.

3.3.8 Scenario 8: modifica del nome di un servizio.

Use case	ModifyServiceName
Pre - conditions	1. Il servizio di cui si vuole modificare il nome è già registrato e ci si trova nella home page.
Normal flow	2. L’utente agisce sul comando di modifica posto a fianco al nome del servizio. 3. Si arriva alla pagina di modifica del nome del servizio, in cui l’utente può inserire il nuovo nome. La pagina riporta anche l’avviso che in caso modifichi il nome del servizio perderà tutte le informazioni relative alle esecuzioni già effettuate. 4. L’utente agisce sul comando di salvataggio dell’aggiornamento.

Post - conditions	5. Si viene riportati alla home page del sistema, in cui, nell'elenco dei servizi, sarà presente anche quello appena modificato.
Situazioni di errore	6. Se la modifica è in conflitto con dati preesistenti nel database o il nome del servizio presenta caratteri non validi o si tratta di una stringa vuota, viene visualizzata una pagina di errore da cui si può tornare alla home page del sistema.

3.3.9 Scenario 9: modifica dei parametri di un pattern

Use case	ModifyPatternInvocation
Pre - conditions	1. L'utente ha già registrato il pattern che vuole modificare e sta visualizzando la pagina relative ai dettagli del servizio da cui dipende quel particolare pattern.
Normal flow	2. Agendo sul comando di modifica che si trova a fianco al pattern, l'utente raggiunge la pagina di modifica. 3. Nella pagina di modifica sono visualizzate le informazioni relative al pattern che si vuole modificare e viene richiesto l'inserimento del numero dei parametri del nuovo pattern. 4. L'utente inserisce il numero di parametri e da' conferma. 5. Vengono create le caselle richieste in cui l'utente potrà inserire i nomi dei parametri. 6. L'utente da' conferma dei dati immessi.
Post - conditions	7. Si viene riportati nella pagina dei dettagli del servizio, in cui sarà presente, al posto del vecchio pattern, quello nuovo appena registrato.
Situazioni di errore	8. Se il nuovo pattern è composto da campi nulli il sistema lo segnala tramite una pagina di errore, attraverso cui si può ritornare alla home del sistema.

	9. Se il nuovo pattern è composto da parametri con lo stesso nome, il sistema lo segnala tramite una pagina di errore, attraverso cui si può ritornare alla home del sistema. 10. Se c'è già un pattern identico per quel servizio, il sistema lo segnala tramite una pagina di errore, attraverso cui si può ritornare alla home del sistema. 10. Se uno o più parametri del nuovo pattern contengono caratteri non validi, il sistema lo segnala tramite una pagina di errore, attraverso cui si può ritornare alla home del sistema. Nel nostro caso, tali caratteri dipendono dal modo in cui si è scelto di restituire i dettagli relativi ad un pattern e sono ",", "/" e "+".

3.3.10 Scenario 10: modifica del nome di un BuiltInTest

Use case	ModifyBuiltInTestName
Pre – conditions	1. L'utente ha già registrato il test set di cui vuole andare a modificare il nome e si trova nella pagina riassuntiva dei dettagli di un servizio.
Normal flow	2. Agendo sul comando di modifica, viene visualizzata una pagina in cui è evidenziato il vecchio nome del servizio e viene data la possibilità di inserirne uno nuovo. 3. L'utente inserisce il nuovo nome e conferma i dati.
Post – conditions	4. Si viene riportati nella pagina dei dettagli del servizio, in cui sarà presente, al posto del vecchio test, quello nuovo appena registrato.
Situazioni di errore	5. Se al posto del nuovo test viene lasciata la stringa vuota il sistema lo segnala tramite una pagina di errore, attraverso cui si può ritornare alla home del sistema. 6. Se c'è già un test identico per quel servizio, il

	sistema lo segnala tramite una pagina di errore, attraverso cui si può ritornare alla home del sistema. 7. Vengono effettuate le consuete validazioni sui caratteri che possono creare problemi al sistema.

3.3.11 Scenario 11: modifica di una query

Use case	ModifyQuery
Pre – conditions	1. L'utente ha già registrato la query che vuole andare a modificare e sta visualizzando la pagina relativa ai dettagli del test che contiene tale query.
Normal flow	2. Agendo sul comando di modifica si viene portati su una nuova pagina che contiene i vecchi dati della query e permette di inserirne di nuovi. 3. Una volta creata la nuova versione della query l'utente conferma i dati inseriti.
Post – conditions	4. L'utente verrà riportato alla pagina dei dettagli del test in cui sarà presente la nuova query.
Situazioni di errore	5. Come nei casi precedenti, viene effettuato un controllo su valori scorretti o nulli, al fine di evitare situazioni che portino il sistema in stati non validi o non logici. 6. Anche per questa operazione di modifica viene effettuato il controllo che non esista una query identica per quel determinato servizio.

3.4Modello concettuale astratto.

Si vuole fornire una visione di alto livello del dominio applicativo, nei termini dei componenti principali che compongono il Profilatore, e che sono alla base di tutto il meccanismo di analisi.

L’entità centrale, come già detto, è il servizio, che può essere invocato secondo diversi pattern e per cui si possono definire dei test. Un’altra entità fondamentale per il sistema è la Query; in particolare, un test può essere composto da un buon insieme di query di varia natura, a patto che siano tutte diverse tra loro.

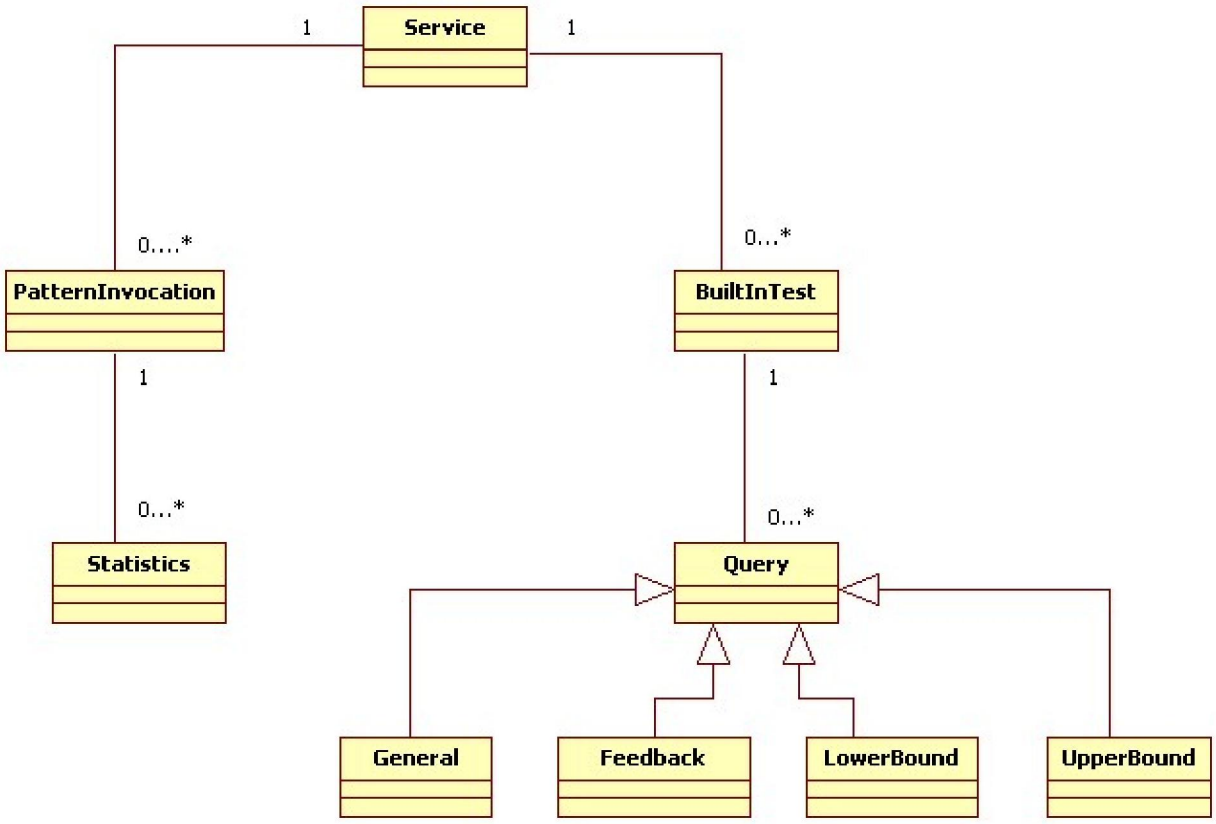


Figura 1: modello concettuale astratto

4 STRUTTURA DEL PROFILATORE

4.1Database design

Il modello del database consente di avere una prima idea un po' più precisa di quelli che sono i dati che vengono gestiti dal profilatore. Nel prototipo architetturale di SeCo, il database è di tipo filebased, ma a causa della natura dei dati che ci si trova a maneggiare per questa applicazione è stato scelto di allontanarsi dal modello di seco ed utilizzare un vero e proprio database.

4.1.1 Modello concettuale

Il modello concettuale mostra il dominio applicativo in termini di concetti principali e relazioni tra di essi.

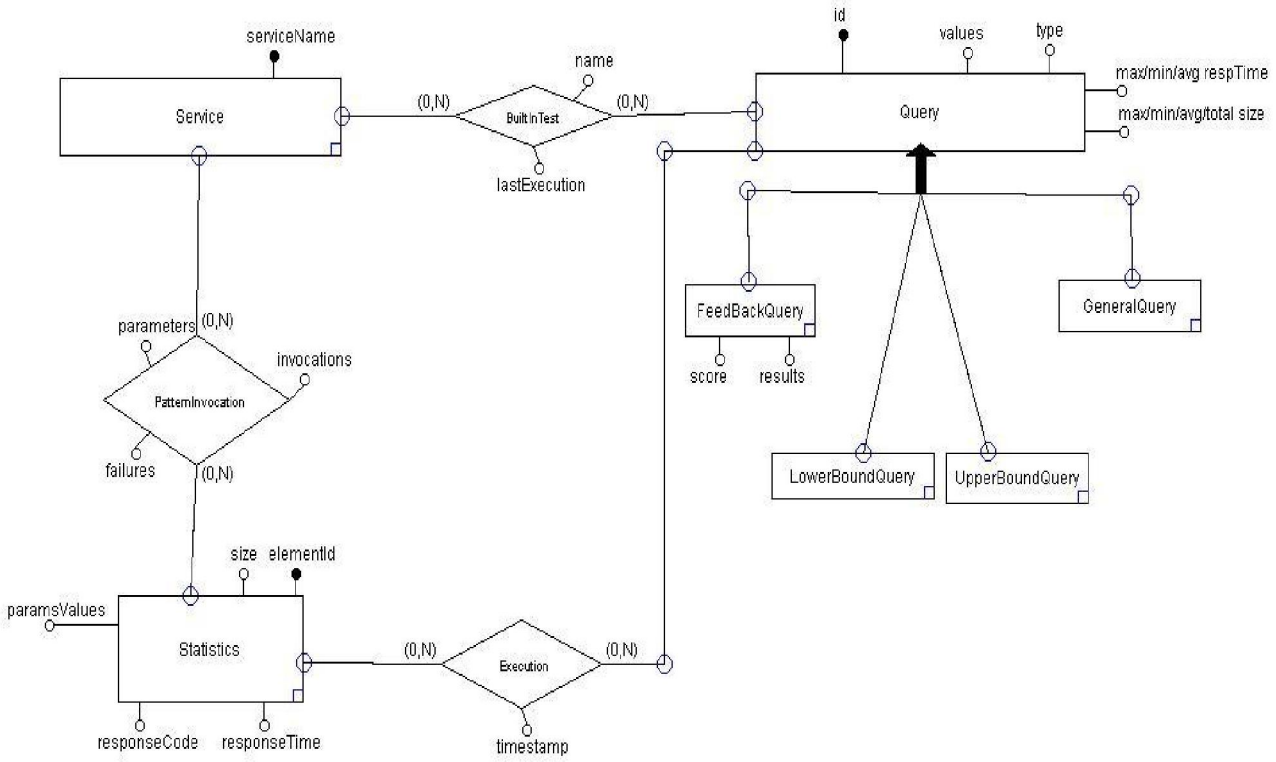


Figura 2: modello ER

Per la memorizzazione di alcuni oggetti si è scelto di utilizzare una particolare funzione di hash coding. Motivo di tale scelta è, soprattutto, la necessità di avere un identificativo univoco per le

risorse, nell'ottica di implementare un servizio Restful. Successivamente si scenderà più nel dettaglio delle motivazioni di questa scelta, e per adesso basti sapere che è importante poter accedere alle risorse tramite soltanto il loro identificativo.

Gli oggetti su cui si applica la funzione di hashing sono PatternInvocation, ProfilerQuery e BuiltInTest.

Dal momento che la funzione di hash code per le stringhe in Java non garantisce l'assenza di conflitti, è stata implementata una semplice variante dell'hashing di base per ridurne la probabilità, comprendendo tutti e soli i parametri di interesse.

Un pattern, ad esempio, è identificato univocamente dal servizio a cui si riferisce e dai parametri da cui è composto. La struttura dati scelta per la memorizzazione dei parametri è quella dell'hashtable, in cui le chiavi sono gli indici di sequenza dei parametri, ovvero gli interi che indicano in che ordine quei parametri vanno a formare il pattern, e come valori i nomi dei parametri. Per calcolare, quindi, l'hash code dell'oggetto PatternInvocation, si concatena l'hash di ogni stringa che rappresenta un parametro, moltiplicato per un numero primo scelto arbitrariamente grande. Nel nostro caso, il numero scelto è 31. Alla fine, ovviamente, all'hashing dei parametri viene anche aggiunto l'hash del nome del servizio. Il numero risultante dalla procedura è l'identificativo del particolare PatternInvocation.

```
for(int count = 0; count < params.size(); count ++){
            patternInvocationHashCode = 31 * patternInvocationHashCode +
            params.get(count).hashCode();
    }

    // Adding also the service name to the hash code
    patternInvocationHashCode = 31*patternInvocationHashCode + serviceName.hashCode();
```

In questo caso si è scelto di non utilizzare iteratori o collezioni per accedere ai valori dei parametri, in quanto per definizione le chiavi sono sequenziali e a partire da zero, e per quello di cui abbiamo bisogno è più che sufficiente.

Per l'oggetto ProfilerQuery il discorso è analogo anche se appena più complesso. Anche in questo caso si ha a che fare con l'hashing di un'hashtable, ma, a differenza della situazione dell'oggetto PatternInvocation, stavolta la sequenza di elementi si riferisce ai valori della query. La politica di hashing è la stessa e, una volta calcolato l'hash delle stringhe dei valori, si passa ad aggiungere, sempre seguendo la stessa tecnica, l'hash di altri campi che concorrono all'identificazione univoca dell'oggetto ProfilerQuery, ovvero il tipo, il nome del servizio, l'hash code del pattern e del BuiltInTest a cui si riferisce.

```
for(int count = 0; count < queryValues.size(); count++){
                String val = new String (queryValues.get(count));
                hc = hc + count;
```

```
            hc = 31 * hc + val.hashCode();
        }
        hc = 31 * hc + type.hashCode();
        hc = 31 * hc + serviceName.hashCode();
        hc = 31 * hc + patternInvocationHashCode;
        hc = 31 * hc + bitHashCode;
```

Per quanto riguarda il BuiltInTest, la procedura è simile ma più semplice, in quanto i campi interessati sono in numero minore. E' necessario, infatti, fare hashing solo del nome del servizio e del nome del test. Vengono, però, ovviamente mantenute quelle precauzioni già adottate per ridurre il rischio di collisioni.

```
// Hash code setting
        int hc = serv.hashCode();
        hc = (31 * hc + bitName.hashCode()) * 31;
```

L'oggetto Service, invece, sarà identificata univocamente solo tramite il suo nome, mentre l'oggetto Statistics da un id generato automaticamente dal sistema.

4.1.2 Modello logico

Il modello logico, derivato dal modello concettuale, mostra le tabelle del database ed i relativi attributi.
Per convenzione, gli attributi sottolineati identificano le chiavi primarie.

Service (serviceName)

Patterninvocation (hashCode,serviceName, params, invocationsNumber, failuresNumber)

Builtintest (hashCode, serviceName, testName, lastExecution)

ProfilerQuery (type, hashCode, values, maxResponseTime, relatedMaxSize, minResponseTime, relatedMinSize, avgResponseTime, relatedAvgSize, results, serviceName, totalDataSize, patternHc, testHc, avgScore, lastScore)

Statistics (elementId, size, responseTime, timestamp, responseCode, serviceName, patternHc, testHc)

4.2 Application design

4.2.1 Panoramica del sistema

Ci si propone di andare a creare un sistema agile, maneggevole e intuitivo per la registrazione dei servizi e la loro analisi prestazionale. Quello che si vuole realizzare, quindi, è un tool che consenta all'utente di effettuare velocemente la registrazione dei servizi e di tutti gli elementi necessari per

effettuare l'analisi prestazionale. Si è pensato, inoltre, di fornire un'implementazione che un domani potesse essere anche solo in parte integrata in SeCo o comunque resa accessibile ad un qualsiasi utente che volesse eseguire qualche tipo di controllo di prestazioni su un servizio. Per tutti questi motivi si è scelto di sviluppare un'applicazione web-oriented, privilegiando così l'intuitività di navigazione.

4.2.2 Descrizione dell'architettura

L'architettura utilizzata si basa sulla piattaforma JEE che consente la suddivisione dei componenti del sistema in più livelli logici che interagiscono tra di loro:

- **livello dati**: l'informazione viene gestita e organizzata in un database a cui accede il livello di controllo;
- **livello controllo**: si occupa della logica applicativa del sistema;
- **livello presentazione**: gestisce l'interazione con l'utente e la visualizzazione dei dati attraverso un browser web.

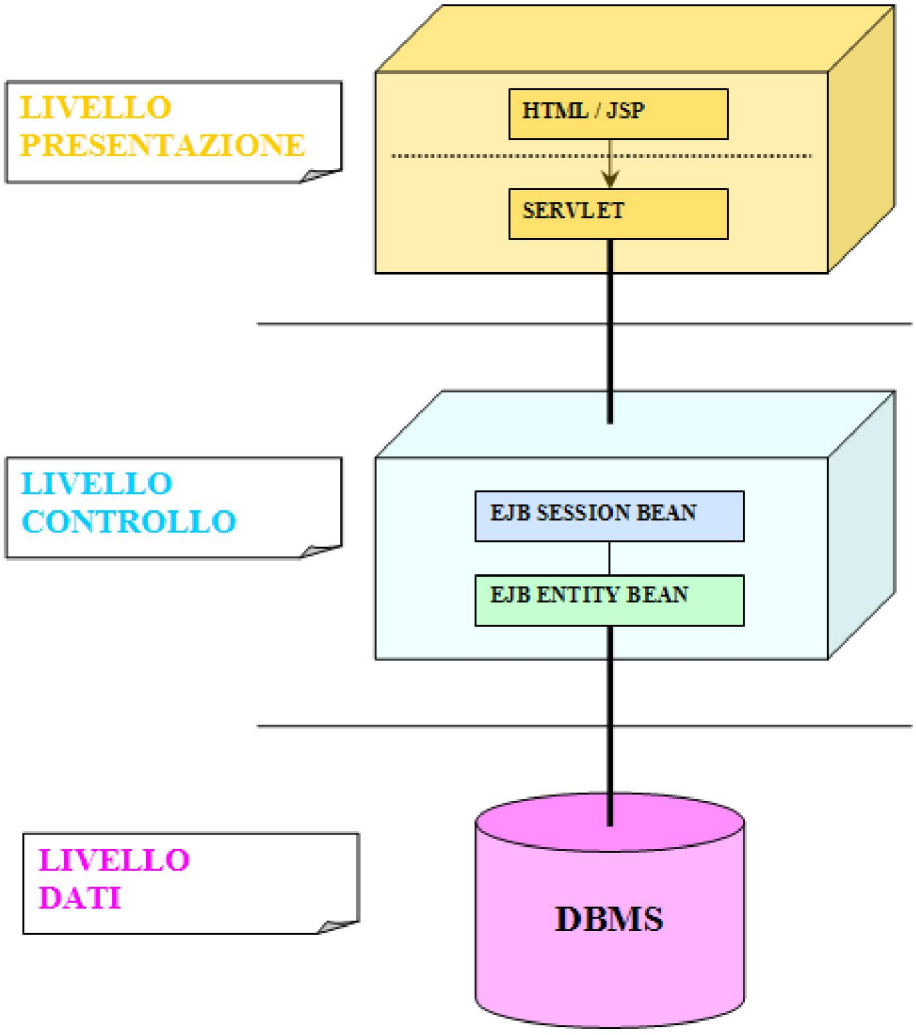


Figura 3: three tier

4.2.3 Scomposizione secondo il pattern MVC

Questo pattern permette una separazione netta tra interfaccia utente, logica di controllo e modello dei dati, che comunicano tra loro attraverso interfacce ben definite.

I componenti sono unità di funzionalità di programma auto-contenute che operano indipendentemente le une dalle altre. Le architetture basate sui componenti generano applicazioni modulari che sono relativamente semplici da mantenere, aggiornare ed estendere.

Nei modelli basati sui componenti, le varie funzioni di una applicazione vengono partizionate in componenti software discreti ed autonomi. Questo permette di assemblare rapidamente le applicazioni con i componenti opportuni a seconda delle funzionalità richieste.

Dal momento che ogni applicazione *enterprise* ha una sua propria struttura, la flessibilità di questo processo fornisce varie tecniche per realizzarle e configurarle a seconda dei loro requisiti; ad esempio, il particolare client che verrà utilizzato dall'utente finale, le considerazioni relative alla sicurezza, i requisiti relativi ai database e alle transazioni, le esigenze di scalabilità, e così via.

La suddivisione in tre livelli si adatta all'utilizzo di uno specifico pattern, il **Model-View-Controller**.

Con il **Model** si identifica il modello dei dati del sistema e le operazioni eseguibili su di essa.

La **View** rappresenta la logica di presentazione dei dati e pertanto rappresenta il mezzo attraverso il quale gli utenti interagiranno con il sistema.

Il **Controller** ha la responsabilità di trasformare le interazioni dell'utente con la View in azioni eseguite sul Model.

A questo punto, dall'architettura a livelli nel pattern MVC, si deriva la seguente struttura:

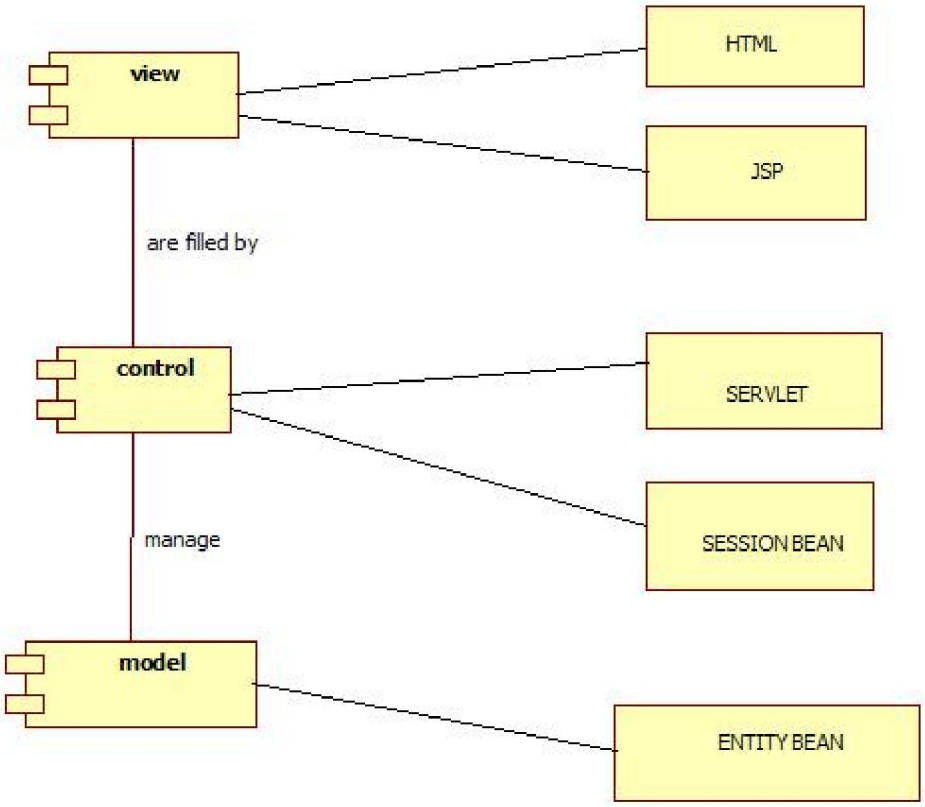


Figura 4: MVC

Le servlet Java, insieme alle pagine JSP, costituiscono l'ossatura del livello Web. Le servlet sono classi Java che vengono utilizzate per estendere il comportamento del server web. Esse si utilizzano solitamente per generare dinamicamente i contenuti in risposta alle richieste dei client. Si utilizzano anche per creare dei controllori che gestiscono vari aspetti dell'interazione dei client web con le componenti dell'applicazione JEE. Mentre le servlet forniscono un approccio da programma alla generazione di contenuti dinamici, le pagine JSP sono documenti testuali che forniscono un'alternativa di tipo markup per creare contenuti statici e dinamici nel livello web.

4.2.4 URI design

Particolare attenzione è stata data alla progettazione delle servlet per accedere ed operare sulle entità del sistema. Si è ritenuto fondamentale strutturare in maniera sensata l'accesso agli oggetti

del sistema, di modo che la base del profilatore rimanesse utilizzabile anche in caso di sviluppi futuri dei livelli superiori dell'implementazione.

Analizzando il problema, si è ritenuto sensato utilizzare un paradigma di tipo REST (*Representional State Transfer*) che prevede che una determinata risorsa, identificata da un URI univoco, sia fruibile tramite i comandi HTTP come POST, GET, PUT e DELETE.

Gli oggetti a cui è necessario poter accedere sono le entità di base del sistema, ovvero Service, PatternInvocation, BuiltInTest, Statistics e ProfilerQuery.

Oltre alla definizione dei metodi di base per le servlet, ovvero GET e POST, si è proceduto anche con la definizione dei metodi di PUT e DELETE. In generale, il metodo GET consente di ottenere informazioni riguardo alla risorsa, POST di aggiungere una nuova istanza di quella tipologia, PUT di modificare un'istanza preesistente e DELETE di cancellarla.

Per ragioni di compatibilità, si è scelto di rendere raggiungibili i metodi di PUT e DELETE anche tramite la chiamata al metodo POST, come sarà evidenziato successivamente mediante delle tabelle di dettaglio.

Si è passati, quindi, al progetto di "nice" URI, ovvero alla definizione prima degli indirizzi di base per le servlet e successivamente alla definizione del rapporto tra metodi e indirizzi in relazione alle funzionalità da esporre.

Omettendo la radice del path, che dipende da dove si può accedere al servizio e che, nel caso di un'installazione locale potrebbe essere http://localhost:8080 , gli indirizzi di base sono:

- /Profiler/ServiceManager : è la servlet che si occupa della gestione dell'entità Service
- /Profiler/PatternInvocationManager: è la servlet che si occupa della gestione dell'entità PatternInvocation.
- /Profiler/BuiltInTestManager: è la servlet che si occupa della gestione dell'entità BuiltInTest.
- /Profiler/QueryManager: è la servlet che si occupa della gestione dell'entità Query.
- /Profiler/StatisticsManager: è la servlet che si occupa della gestione dell'entità Statistics.

Per quanto riguarda le funzionalità e il passaggio dei parametri, ogni entità ha chiaramente bisogno di un discorso a sé, che analizzi le operazioni contemplate e i parametri passati.

Il file web.xml ha dovuto essere modificato per far sì che possa essere contemplato il path servlet/risorsa. In particolare, è necessario che il tag <url-pattern> di ogni servlet abbia come contenuto /nomeDellaServlet/* ; in questo modo si dichiara che quella servlet sarà accessibile tramite qualsiasi path che abbia come radice nomeDellaServlet.

Per l'entità Service le funzionalità sono molto semplici. Sarà necessario poter interrogare il database per ottenere l'elenco di tutti i servizi, oppure il nome del servizio ed è ovviamente fondamentale poter aggiungere e rimuovere un'istanza di Service. Per ragioni di comodità è stata implementata anche la modifica al nome del servizio. In realtà, dal momento che il nome del servizio è chiave primaria per l'entità servizio, a livello di database non è permessa tale modifica ma è stata implementata manualmente per la comodità dell'utente. In questo modo, nel caso di errori nella

scrittura del nome di cui l'utente dovesse accorgersi solo in un secondo tempo, quando magari avrà già finito con la registrazione di tutti gli altri elementi, non sarà costretto a cancellare tutto e ricominciare da capo tutto il progetto.

Nello specifico, quindi, nella servlet di nome "serviceManager" saranno implementati i metodi:

- (GET con parametro): per ottenere dettagli sul particolare servizio.
- GET senza parametri: per ottenere l'elenco di tutti i servizi registrati.
- POST con parametro: per registrare un nuovo servizio.
- DELETE con parametro: per cancellare uno specifico servizio.

Per semplicità, si riporta in forma tabellare il dettaglio sulle operazioni della servlet.

HTTP method	**Risorsa**	**Funzione**	**Dettagli**
GET	/ServiceManager	Restituisce tutti i servizi registrati	PathInfo = null e nessun criterio di selezione aggiuntivo. I servizi vengono restituiti come nome1/nome2/nome3 e così via
	/ServiceManager/ *serviceName*	Restituisce il servizio con nome = *serviceName*	PathInfo = */serviceName*
POST	/ServiceManager	Registra un nuovo servizio	PathInfo = null Nome del nuovo servizio nei parametri della request
	/ServiceManager/ *serviceName*	Cancella il servizio con nome = *serviceName*	PathInfo=/*serviceName* Viene invocata doDelete()

Per quanto riguarda l'entità patternInvocation, invece, deve essere anche disponibile la modifica, quindi sarà implementato anche il metodo PUT della servlet.

HTTP method	**Risorsa**	**Funzione**	**Dettagli**
GET	/PatternInvocation Manager	Se non c'è il nome di un servizio tra i parametri della richiesta restituisce tutti i pattern registrati	PathInfo = null e nessun criterio di selezione aggiuntivo. Viene restituito l'hash code di ogni pattern separato da un "+" dalla sequenza dei parametri del pattern separati tra loro da "/". I pattern sono divisi tra loro da

			“,”.
	/PatternInvocationManager	Se c’è il nome di un servizio tra i parametri della richiesta restituisce tutti i pattern di quel servizio	PathInfo = null e il nome del servizio come criterio di selezione aggiuntivo. Viene restituito l’hash code di ogni pattern separato da un “+” dalla sequenza dei parametri del pattern separati tra loro da “/”. I pattern sono divisi tra loro da “,”.
	/PatternInvocationManager/*patternHC*	Restituisce il pattern con hash code = *patternHC*	PathInfo = */patternHC* Viene sempre restituito l’hash code del pattern e la sequenza dei parametri.
POST	/PatternInvocationManager	Registra un nuovo pattern	PathInfo = null Tutti i dettagli nei parametri della request
	/PatternInvocationManager/*patternHC*	Se il parametro action nella richiesta è pari a “delete”, allora cancella il pattern con hash code = *patternHC*	PathInfo=/*patternHC* Viene invocata doDelete()
	/PatternInvocationManager/*patternHC*	Se il parametro action nella richiesta è pari a “modify”, allora modifica il pattern con hash code = *patternHC*	PathInfo=/*patternHC* Viene invocata doPut()

Un discorso intermedio tra quelli relativi alle entità Service e PatternInvocation va fatto per l’entità BuiltInTest. Le istanze di questo oggetto sono gli insiemi di query che servono per testare i servizi. Come identificativo univoco viene utilizzata una combinazione di hash code del nome del servizio e del nome del test.

HTTP method	**Risorsa**	**Funzione**	**Dettagli**
GET	/BuiltInTestManager	Se non c’è il nome di un servizio tra i parametri della richiesta restituisce	PathInfo = null e nessun criterio di selezione aggiuntivo. Viene restituito l’hash code di ogni BuiltInTest separato da un “+” dal nome

		tutti i test registrati	del test. I test sono divisi tra loro da “,”
	/BuiltInTestManager	Se c’è il nome di un servizio tra i parametri della richiesta restituisce tutti i BuiltInTest relativi a quel servizio	PathInfo = null e il nome del servizio come criterio di selezione aggiuntivo. Viene restituito l’hash code di ogni test separato da un “+” dal nome del test. I test sono divisi tra loro da “,”.
	/BuiltInTestManager/*bitHC*	Restituisce il pattern con hash code = *bitHC*	PathInfo = */bitHC* Viene sempre restituito l’hash code del test e il nome del test.
POST	/BuiltInTestManager	Registra un nuovo test	PathInfo = null Tutti i dettagli nei parametri della request
	/BuiltInTestManager/*bitHC*	Se il parametro action nella richiesta è pari a “delete”, allora cancella il test con hash code = *bitHC*	PathInfo=/*bitHC* Viene invocata doDelete()
	/BuiltInTestManager/*bitHC*	Se il parametro action nella richiesta è pari a “modify”, allora modifica il test con hash code = *bitHC*	PathInfo=/*bitHC* Viene invocata doPut()

L’ultima entità a cui bisogna poter accedere tramite chiamate Rest è Query. Come già accennato, anche in questo caso l’entità è individuata univocamente dal suo hash code, calcolato come combinazione di alcuni parametri in essa contenuti.

Sempre per chiarezza si riporta in forma tabellare il dettaglio delle operazioni disponibili.

HTTP method	**Risorsa**	**Funzione**	**Dettagli**
GET	/QueryManager	Se non c’è l’identificativo di un BuiltInTest tra i parametri della richiesta restituisce tutte le query	PathInfo = null e nessun criterio di selezione aggiuntivo. Viene restituito l’hash code di ogni Query separato da un “+” dall’elenco delle coppie “parametro = valore”. Le query sono divise tra loro da “,”

		registrate	
	/QueryManager	Se c'è l'identificativo di un BuiltInTest tra i parametri della richiesta restituisce tutte le Query relative a quel test	PathInfo = null ed identificativo del servizio come criterio di selezione aggiuntivo. Viene restituito l'hash code di ogni Query separato da un "+" dall'elenco delle coppie "parametro = valore". Le query sono divise tra loro da ",".
	/QueryManager/*queryHC*	Restituisce la query con hash code = *queryHC*	PathInfo = */queryHC* Viene sempre restituito l'hash code della query e l'elenco delle coppie "parametro = valore".
POST	/QueryManager	Registra una nuova query	PathInfo = null Tutti i dettagli nei parametri della request
	/QueryManager/*queryHC*	Se il parametro action nella richiesta è pari a "delete", allora cancella la query con hash code = *queryHC*	PathInfo=/*queryHC* Viene invocata doDelete()
	/QueryManager/*queryHC*	Se il parametro action nella richiesta è pari a "modify", allora modifica la query con hash code = *queryHC*	PathInfo=/*queryHC* Viene invocata doPut()

C'è un'ultima servlet nel sistema, che si comporta in maniera più flessibile rispetto al paradigma, ovvero quella con il nome di StatisticsManager. Si tratta della servlet che si occupa della gestione dell'esecuzione dei test e del salvataggio dei relativi dati.

HTTP method	**Risorsa**	**Funzione**	**Dettagli**
GET	/StatisticsManager	Nei parametri della richiesta si può inserire un campo per effettuare una sorta di proiezione sugli oggetti	PathInfo = null e criteri di selezione aggiuntivi come parametri della richiesta.

		Statistics disponibili. TODO: aggiungere dettagli sulle proiezioni possibili.	
POST	/StatisticsManager /*bitId*	Esegue il BuiltInTest con l'id specificato nel path e va a salvare i risultati nella tabella Statistics.	PathInfo = *bitId*
	/StatisticsManager /*statisticsId*	Se il parametro action nella richiesta è pari a "modify", allora modifica la statistica con id uguale a *statisticsId*. Questa funzionalità viene utilizzata, in particolare, per	PathInfo=/*queryHC* Viene invocata doPut()

4.2.5 Diagrammi di dettaglio

La tecnologia EJB (Enterprice Java Bean) supporta il livello di logica business lato server previsto nella piattaforma JEE; è un'architettura a componenti distribuita, che permette lo sviluppo e il *deployment* delle componenti business delle applicazioni JEE.

Questa tecnologia costituisce una soluzione completa per incapsulare la logica business in componenti riutilizzabili chiamati Bean Enterprise. Un Bean Enterprise rappresenta tipicamente una singola funzione o entità business, per esempio dati contenuti in un database, che può essere assemblata con altri Bean Enterprise e componenti JEE per formare delle applicazioni complete.

Come già precedentemente accennato, la tecnologia EJB contempla tre tipi di bean: Session, Entity e Message Drive Bean.

I *Session Bean* rappresentano le sessioni client con l'applicazione. I client interagiscono con le componenti nel livello di logica business dell'applicazione invocando i metodi dei Session Bean. Mediando l'accesso al livello business, nascondono la complessità ai client: questo consente di semplificare notevolmente lo sviluppo dei client che devono interagire con il livello business.

Gli *Entity Bean* rappresentano i dati business che vengono solitamente memorizzati in una memoria permanente (come un database relazionale) o, meno frequentemente, in altre parti dell'applicazione come in una memoria temporanea. Gli Entity Bean vengono tipicamente utilizzati per presentare i dati presenti in un database relazionale nella forma di oggetti Java, o più specificatamente di oggetti entity. Le altre componenti dell'applicazione possono così accedere ai dati invocando i metodi degli entity bean. In altri termini, forniscono una *view* a oggetti delle sorgenti dei dati in modo che gli altri oggetti possano facilmente accedere e manipolare i dati indipendentemente dal sistema di memorizzazione o dal formato utilizzati.

I *Message Driver Bean* reagiscono ai messaggi asincroni inviati ad un *topic*. Così facendo consentono lo sviluppo di applicazioni JEE le cui componenti comunicano con modalità più

lascamente accoppiata rispetto al caso in cui i client interagiscono con Session ed Entity Bean tramite un meccanismo di invocazione diretta.

Scendendo, quindi, nello specifico dell'implementazione, sulla base del modello relazionale dei dati, si possono definire gli Entity Bean che si trovano alla base di tutto il sistema. Per semplicità di rappresentazione non si scende nello specifico di tutti i metodi setter e getter ma si elencano soltanto gli attributi presenti in ogni Bean.

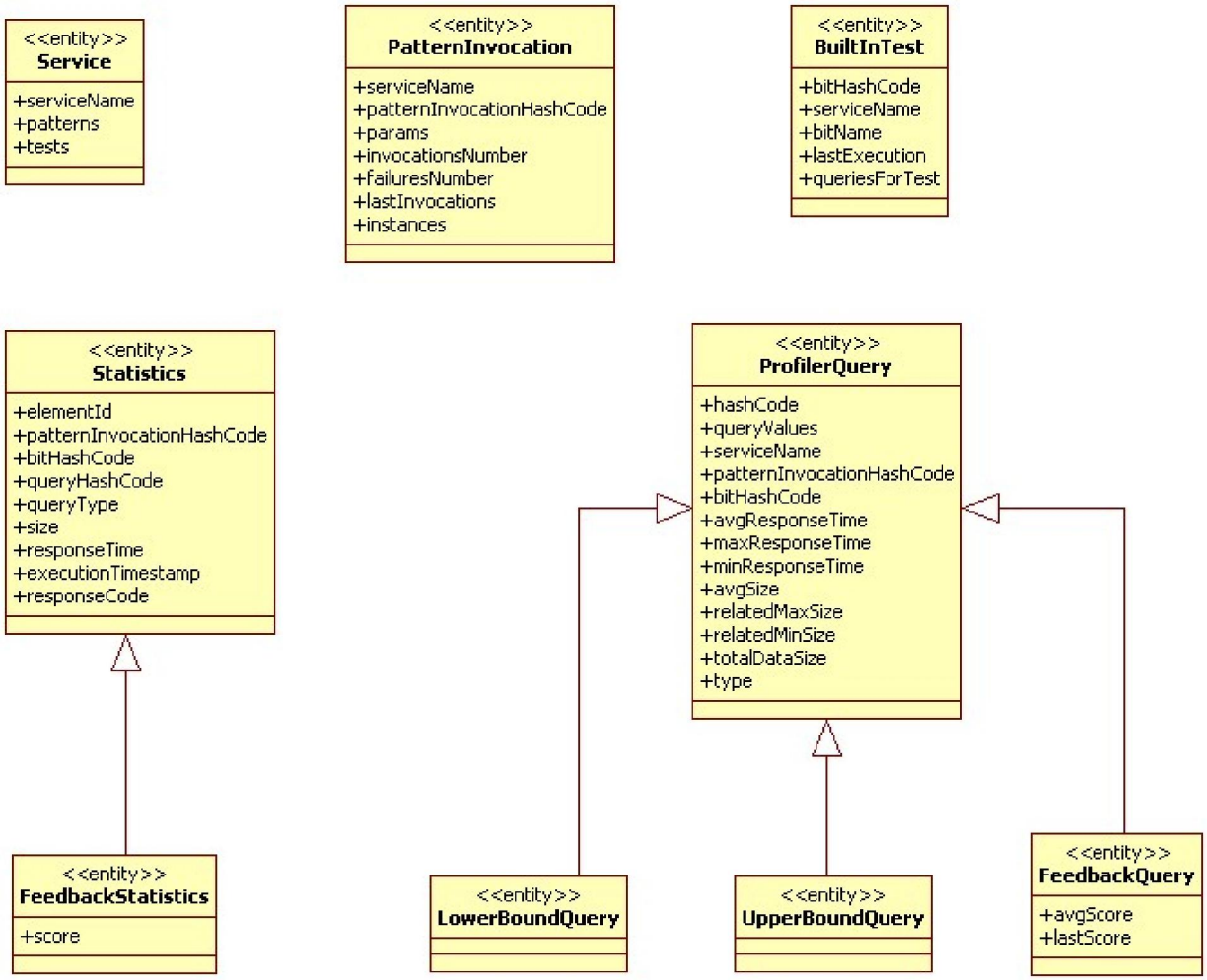


Figura 5: Entity Bean

I Session Bean che sono stati implementati sono tre e consentono le operazioni più importanti sugli oggetti base del sistema. Essi si riferiscono, infatti, alle operazioni sul servizio, sulle query e sugli oggetti Statistics, e si chiamano rispettivamente ManageServiceBean, ManageQueryBean e ManageStatisticsBean, con le rispettive interfacce Remote.

ManageServiceBean consente le operazioni fondamentali sulle entità Service, PatternInvocation e BuiltInTest, ovvero metodi di aggiunta, rimozione, interrogazione secondo vari criteri come ad esempio accedere a tutti i pattern di un dato servizio o sapere se esiste un test con un dato nome.

Il Bean ManageQueryBean, invece, si occupa delle funzioni base di aggiunta, rimozione ed interrogazione sull'oggetto ProfilerQuery (comprese, ovviamente, tutte le classi che ereditano da essa)

5 FUNZIONAMENTO DEL PROFILATORE

5.1 Esempi di utilizzo

Come situazione di studio iniziale, viene considerato il sistema al primo avvio, ovvero senza nessun servizio registrato, e si procede alla registrazione di uno dei servizi che è stato studiato nella fase dei test preliminari. Per comodità, si sceglie di considerare IgniteMovieService, dal momento che è quello con il minor numero di parametri da inserire.

Una volta avviati MySQL e JBoss, si può accedere al sistema per mezzo del proprio browser all'indirizzo http://localhost:8080/Profiler/home.jsp. La pagina che viene caricata è la home di base del sistema, e l'unica operazione possibile è l'aggiunta di un nuovo servizio, effettuabile tramite il link "add".

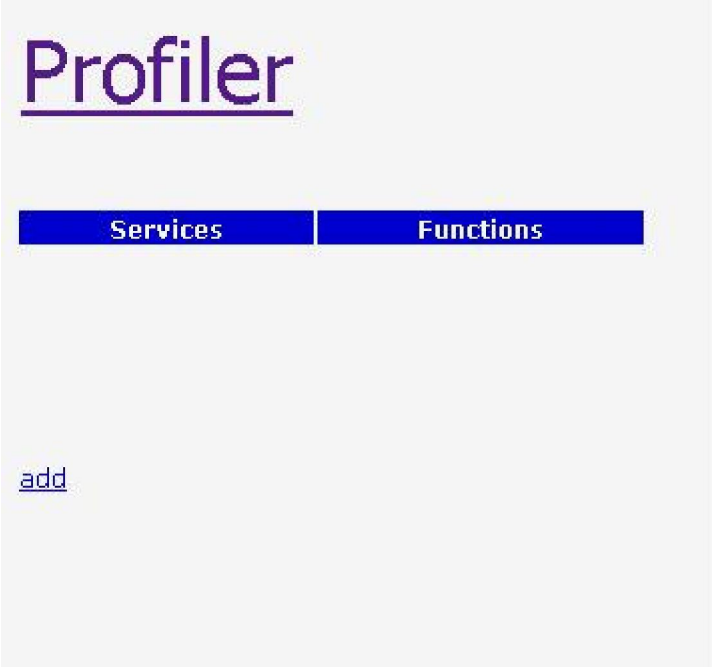


Figura 6: home.jsp

Il click sul link di aggiunta porta l'utente nella pagina di inserimento del nome del nuovo servizio. L'utente, una volta immesso il nome, può cliccare sul pulsante di salvataggio e, in caso di assenza di errori, venire riportato nella home del sistema, in cui, nell'elenco dei servizi, potrà trovare quello appena aggiunto. Nel nostro caso, quindi, inseriamo il nome "ignyteMovieService", diamo la conferma, e osserviamo il comportamento del sistema.

Figura 7: createNewService.jsp

Nel caso ci fosse un qualsiasi tipo di errore , l'utente visualizzerebbe una pagina di errore generica, di segnalazione che qualcosa nei dati immessi non andava bene. Cliccando in alto, sul nome del sistema, ovvero "Profiler" , l'utente potrà tornare alla home e ritentare l'aggiunta del nuovo servizio.

Figura 8: errorPage.jsp

Nel caso in cui, invece, tutto il processo andasse a buon fine, invece, la navigazione procederà verso la home del sistema, che ora conterrà il nuovo servizio.

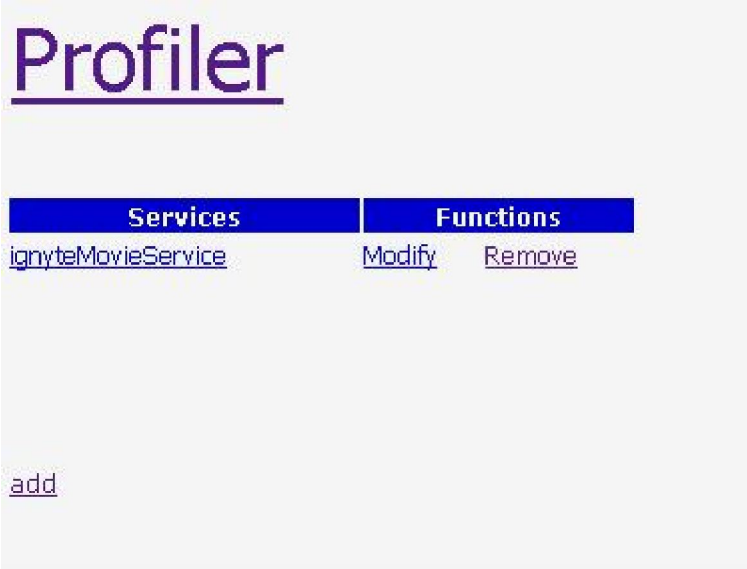


Figura 9: home.jsp

Si può notare che, accanto al nome del servizio sono presenti anche due link di funzionalità disponibili sul servizio, ovvero “Modify” e “Remove”. La prima funzione consente di modificare il nome del servizio ed è pensata per i casi in cui l’utente faccia degli errori di battitura nella creazione del servizio e voglia rimediarvi senza per questo dover ricominciare tutto da capo. La seconda funzione consente di rimuovere il servizio e tutti i dati ad esso associati.

L’utente, a questo punto, può cliccare sul nome del servizio e visualizzare i dettagli più rilevanti di esso, ovvero i pattern con cui è possibile invocarlo e i test set disponibili, ovvero gli insiemi di query che l’utente ha definito per testare le performance di quel determinato servizio.

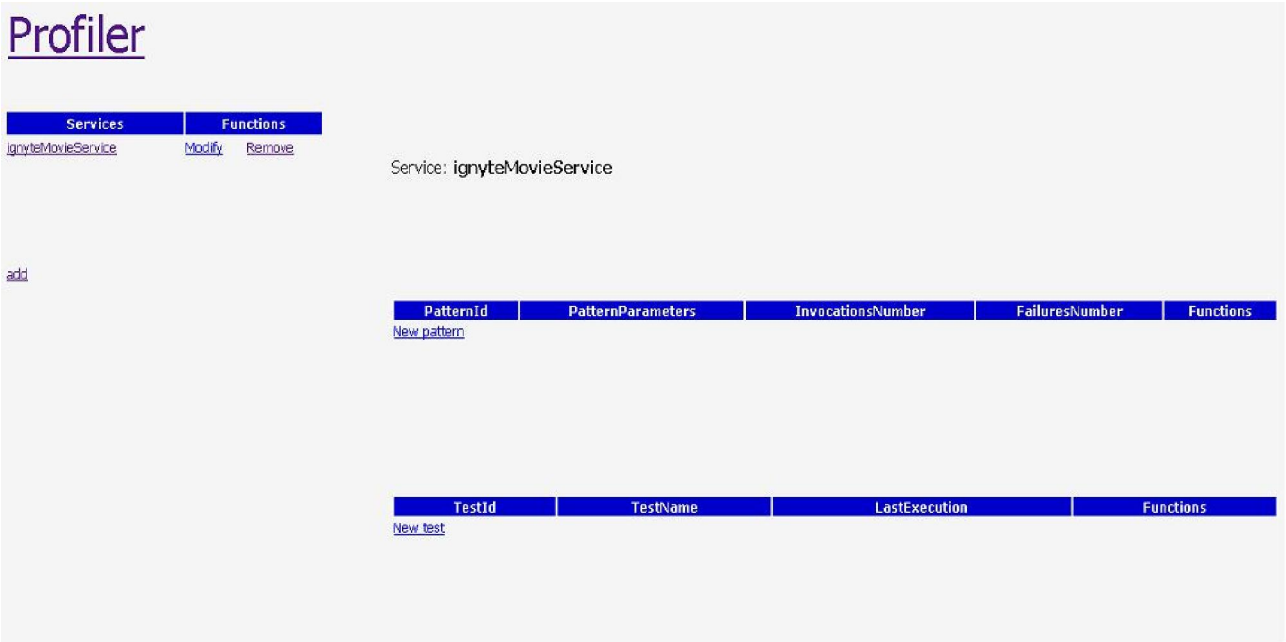


Figura 10: inspectService.jsp

Ovviamente, in questo caso, la pagina presenta soltanto delle tabelle vuote e la possibilità, tramite dei link, di aggiungere nuovi pattern o nuovi test set.

Si esamina, per prima cosa, il caso di aggiunta di un nuovo pattern. In seguito al click sul link di aggiunta, la navigazione prosegue verso una pagina in cui viene richiesto all'utente di inserire il numero di parametri di cui dovrà essere composto il pattern. Una volta inserito e cliccato sul pulsante di conferma, vengono create il numero richiesto di caselle di testo, in cui l'utente potrà inserire i nomi dei parametri che comporranno il pattern. Nel caso in cui l'utente avesse inserito un numero errato o per un qualsiasi motivo cambiasse idea sul numero di parametri, senza cambiare pagina potrà immettere un nuovo numero di parametri e dare ancora il comando di conferma; questo causerà la cancellazione delle caselle precedentemente inserite e l'inserimento di un numero di celle vuote pari alla nuova scelta. Nel nostro caso le caselle da inserire sono tre: "zip code", "radius" e "chunk index", quindi chiediamo al sistema di creare tre caselle di testo e poi andiamo a riempirle con i nomi dei parametri.

Figura 11: createPattern.jsp

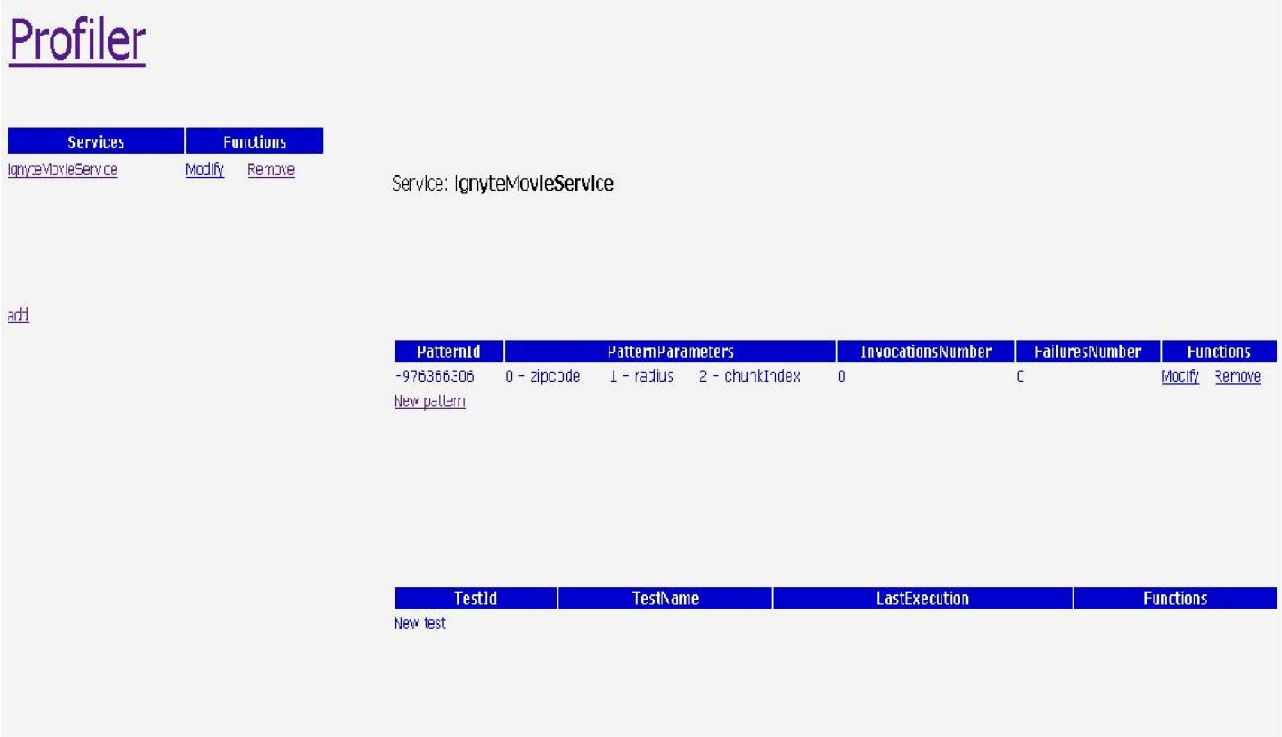


Figura 12: inspectService.jsp

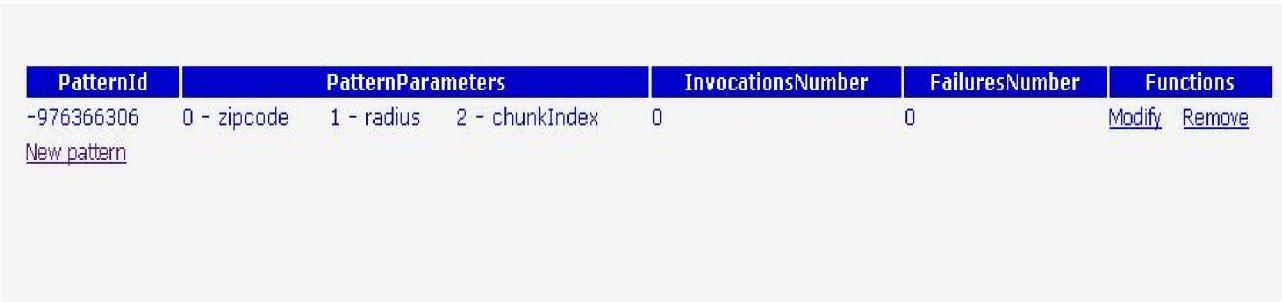


Figura 13: dettaglio inspectService2.jsp

Una volta data la conferma dei dati, se tutto è andato a buon fine si verrà riportati nella pagina relativa ai dettagli del servizio, in cui sarà possibile visualizzare il nuovo pattern inserito. Si può notare che vi sono altri due campi nella tabella dei pattern: quello relativo al numero di invocazioni e quello relativo al numero di failure. Questi parametri vengono aggiornati ad ogni invocazione del pattern e possono essere utili all'utente per avere un'idea di quella che è l'affidabilità del pattern. Accanto ad ogni pattern sono presenti due link, rispettivamente per la rimozione e la modifica del particolare pattern. Anche in questo caso, la modifica può risultare comoda nel caso in cui un utente per qualche motivo abbia sbagliato nell'immissione dei parametri. Nella pagina per modificare il pattern saranno visualizzati i vecchi parametri e verrà richiesto all'utente di inserire i nuovi parametri secondo lo stesso paradigma utilizzato per l'aggiunta di un nuovo pattern. Una volta salvato il pattern modificato, però, l'utente deve tenere presente che saranno andati persi tutti i dati relativi alle vecchie query che sfruttavano quel pattern. Un comportamento del genere è sia logico che necessario, in quanto una query si appoggia in maniera molto profonda sul concetto e sulla

tipologia di un singolo pattern, quindi non avrebbe senso mantenere in un pattern dei dati provenienti da query che non sono state effettuate in realtà seguendo lo stesso pattern.

Si può procedere, a questo punto, con l'aggiunta di un nuovo test set. Il processo è molto semplice e segue lo stesso paradigma utilizzato per l'aggiunta di un nuovo servizio. Per creare un nuovo test set è sufficiente cliccare sul link presente sulla pagina relativa ai dettagli del servizio e la navigazione procederà verso la pagina di immissione del nome del test. Nel nostro caso, lo chiameremo "igniteMovieServiceBit1". Una volta dato il comando di salvataggio, in assenza di errori si verrà riportati alla pagina relativa ai dettagli del servizio, in cui sarà possibile visualizzare anche il servizio appena inserito. Nel nostro esempio, quindi, dopo l'aggiunta del pattern e del test set, nella pagina dei dettagli del servizio appariranno i nuovi elementi.

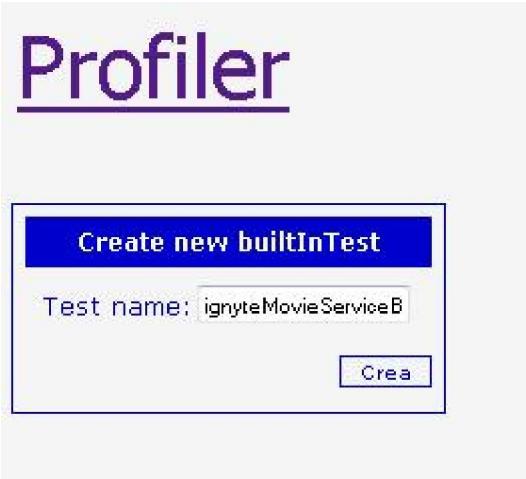


Figura 14: createBit.jsp

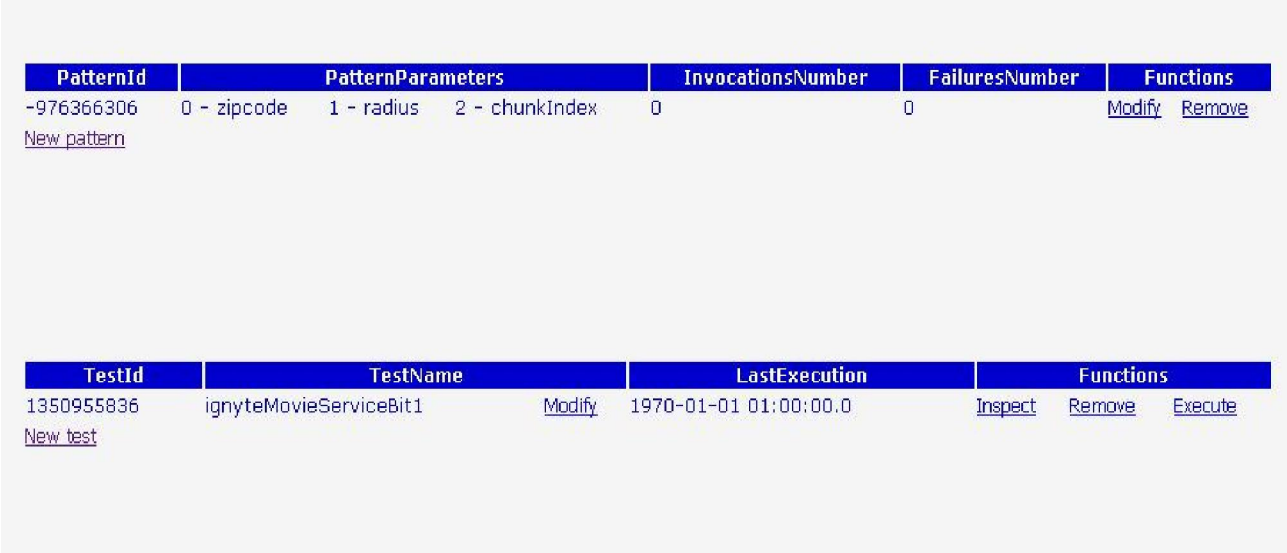


Figura 15: dettaglio inspectService.jsp

Una delle parti più importanti di tutto il sistema è la definizione delle query su cui testare i vari servizi. Per aggiungere una query bisogna prima visualizzare i dettagli di un test tramite il comando

di “inspect” che si può trovare tra le funzionalità disponibili per un test set. Una volta cliccato sul link, la navigazione procede verso una pagina che fornisce una visione generale delle query che compongono l’insieme di test e che riporta dati relativi a precedenti esecuzioni. Nel caso in esame, ci si trova davanti ad una pagina con due tabelle vuote. La prima è quella preposta al contenimento dei dati delle query che compongono il test e presenta un link che consente l’aggiunta di nuovi elementi all’insieme. Quella subito sotto riporterà i risultati dell’ultima esecuzione del test set. Per prima cosa si va ad esaminare l’inserimento di una nuova query. Cliccando sul link di aggiunta di una nuova query, viene visualizzato il primo passo per la definizione di una query, ovvero la scelta del pattern.

Figura 16: createQuery-patternChoice.jsp

Dato che, per il caso in esame, precedentemente era stato definito un solo pattern, allora l’elenco dei pattern disponibili tra cui effettuare la selezione è limitato ad un elemento. Lo si seleziona e si procede facendo click sul pulsante di conferma. La pagina successiva consente la selezione della tipologia di query (a scelta tra General, Feedback, LowerBound ed UpperBound) e l’inserimento dei valori per i campi del pattern.

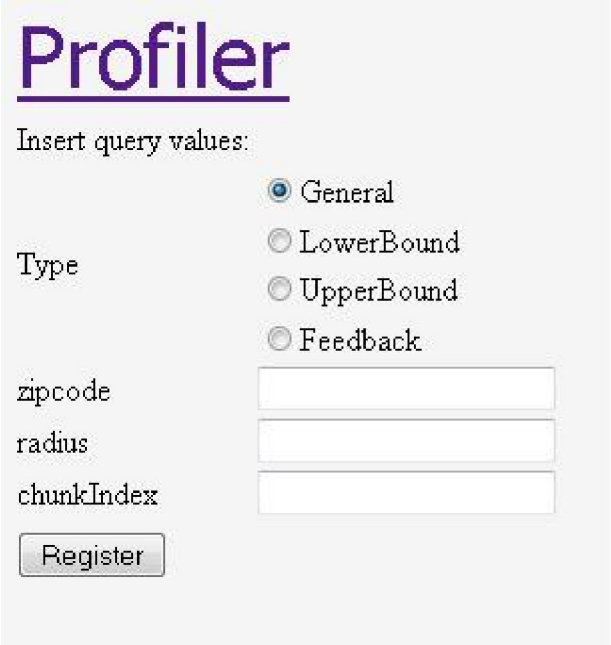
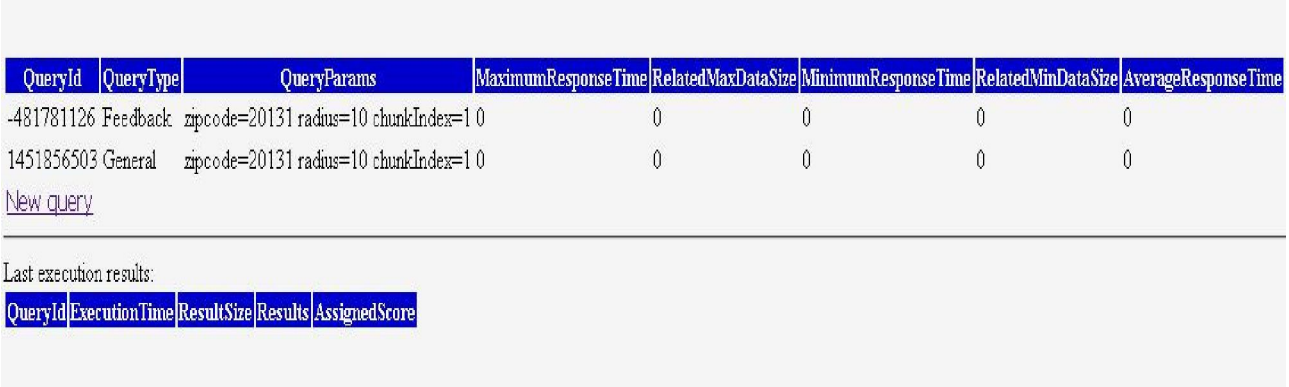


Figura 17: createQuery-paramsInsertion.jsp

Nel nostro caso useremo dei dati su cui è stato testato lo scce, ovvero per il parametro zip code "20131" e per quello radius il valore "10". Decidiamo di definire una query di tipo generale perché ipotizziamo di non sapere che tipo di dati risulteranno dall'invocazione e chiediamo il primo chunk. Una volta premuto il pulsante di salvataggio, se tutto va a buon fine si viene riportati alla pagina dei dettagli del servizio che si sta modificando. Se, invece, qualcosa non va bene nei dati inseriti, viene visualizzata la pagina di errore. Per rendere più completo l'esempio di studio, definiamo anche un'altra query, identica alla precedente ma di tipo Feedback, per poter osservare le differenze nel comportamento del sistema.

Facendo click sul tasto di inspect posto a fianco ad ogni test sarà possibile visualizzare tutte le query che compongono quel test, accompagnate da un notevole numero di statistiche prestazionali ed, eventualmente, anche i risultati dell'ultima esecuzione. Nel nostro caso, ovviamente, non avendo ancora eseguito il test, le statistiche saranno tutte a zero e non saranno presenti dati relativi all'ultima esecuzione. La figura mostra solo una parte della pagina, alcune statistiche, per motivi di visualizzazione, sono state tagliate fuori.



QueryId	QueryType	QueryParams	MaximumResponseTime	RelatedMaxDataSize	MinimumResponseTime	RelatedMinDataSize	AverageResponseTime
-481781126	Feedback	zipcode=20131 radius=10 chunkIndex=1	0	0	0	0	0
1451856503	General	zipcode=20131 radius=10 chunkIndex=1	0	0	0	0	0

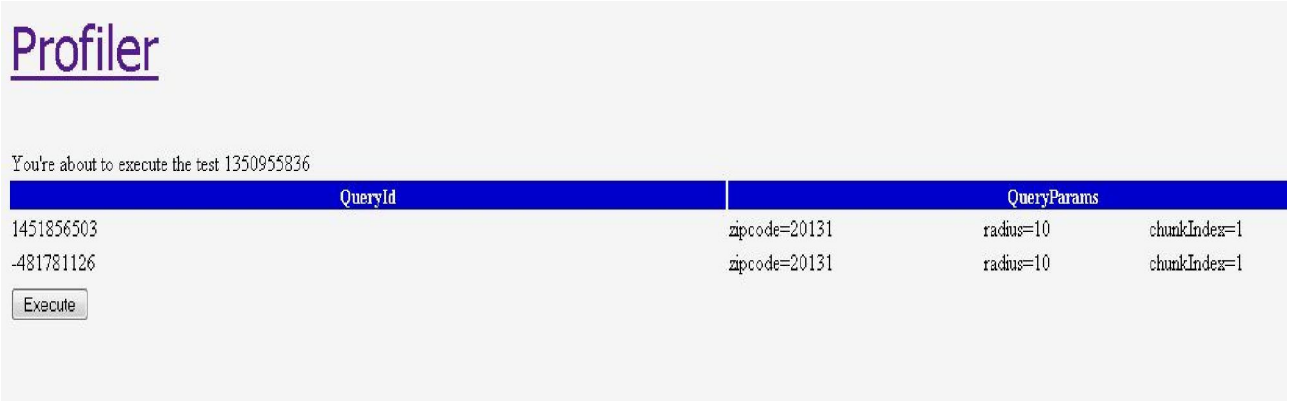
New query

Last execution results:

QueryId	ExecutionTime	ResultSize	Results	AssignedScore

Figura 18: dettaglio inspectBit.jsp

A questo punto si hanno a disposizione tutti gli elementi per poter effettuare una prima esecuzione del test set. Per fare ciò è sufficiente cliccare sul link “Execute” posto a fianco al test e confermare l’esecuzione nella pagina successiva. Il sistema procederà, quindi, all’esecuzione delle query che compongono il test. Nel nostro caso si tratta di una query di tipo General e di una di tipo Feedback. Il sistema fa visualizzare una pagina riassuntiva delle query che si andrà ad eseguire e chiede un’ulteriore conferma di esecuzione.



Profiler

You're about to execute the test 1350955836

QueryId	QueryParams		
1451856503	zipcode=20131	radius=10	chunkIndex=1
-481781126	zipcode=20131	radius=10	chunkIndex=1

Execute

Figura 19: executeBit.jsp

Una volta data la conferma ha il via il processo di esecuzione, che porterà alla visualizzazione di una pagina dei risultati. La pagina dei risultati mostra l’elenco delle query eseguite e per ognuna alcuni dettagli, il tempo di esecuzione e la dimensione dei risultati restituiti. Nel caso in cui fossero presenti delle query di tipo Feedback, il sistema popolerà anche la tabella sottostante. Quest’ultima, infatti, è dedicata alla visualizzazione dei risultati delle query di feedback e all’assegnazione del punteggio. Nel nostro caso, la tabella conterrà un solo elemento, identificato dal suo identificatore univoco; per avere altre informazioni sulla query basta ovviamente osservare la tabella superiore. Accanto ai risultati della query è presente una tendina con numeri da uno a dieci, con cui si può assegnare un punteggio al risultato visualizzato. Si decide di dare un punteggio di sei ai risultati ottenuti e si salvano i dati.

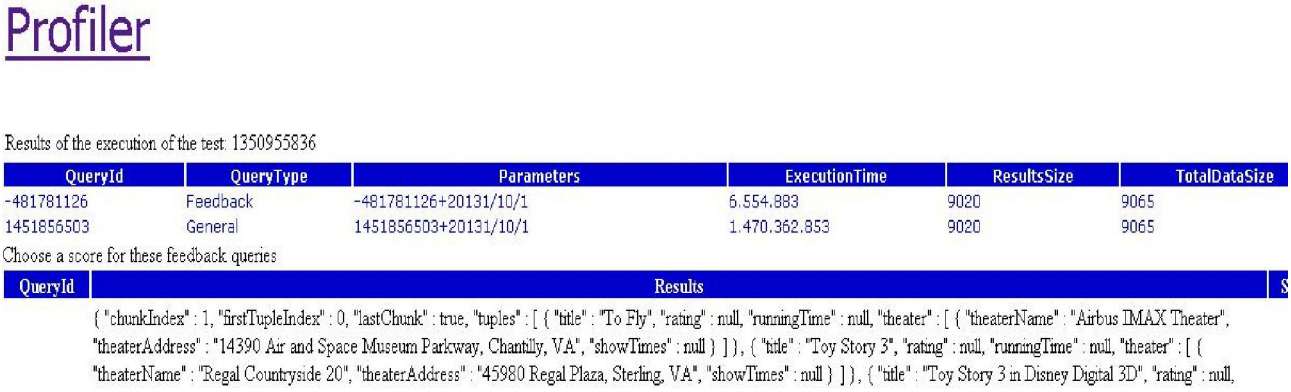


Figura 20: dettaglio showResults.jsp

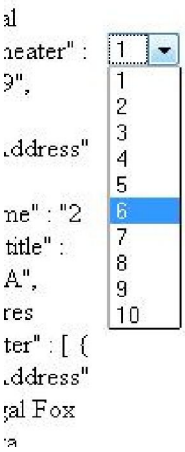


Figura 21: dettaglio showResults.jsp - score selection

Il sistema, dopo il salvataggio dei dati, porta l'utente a visualizzare la pagina dei dettagli del test appena eseguito. Questa pagina è la stessa che troveremmo navigando dalla pagina dei dettagli del servizio, cliccando sul tasto di "inspect" relativo a quel particolare test set. Nella pagina sono riportati i dati relativi alle query che compongono il servizio e i dati relativi all'ultima esecuzione effettuata, ovvero tempi di esecuzione, dimensione dei risultati, eventualmente i risultati veri e propri e il punteggio assegnato alle query di feedback. Si può notare che nei dati relativi alle query sono anche presenti delle statistiche medie sui dati di esecuzione, ovvero tempo di risposta, dimensione dei risultati e, solo per le query di feedback, punteggio medio.

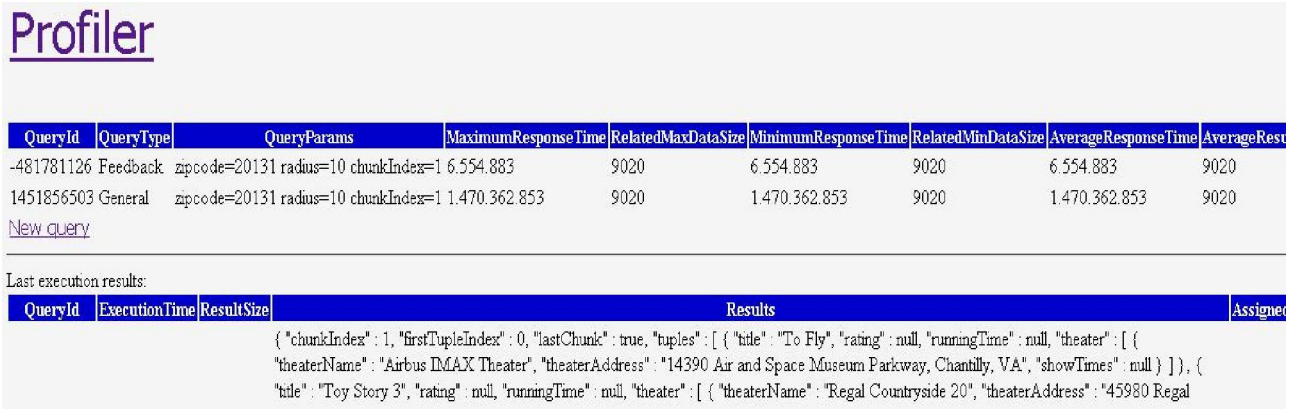


Figura 22: dettaglio inspectBit.jsp

Dati i risultati dei test svolti all'inizio dello studio per la creazione del profilatore, si è ritenuto necessario tenere traccia anche del minore e maggiore tempo d'esecuzione, con la relativa dimensione dei dati, e della dimensione totale dei dati disponibili. Queste informazioni saranno molto utili quando inizieremo ad usare il profilatore per eseguire più o meno lo stesso compito che nelle fasi iniziali è stato svolto in gran parte manualmente.

Tornando alla pagina dei dettagli del servizio, si può notare che ci sono state delle modifiche sia nella tabella dei pattern che in quella dei test. In particolare, il pattern che era stato definito è stato aggiornato con il numero di esecuzioni a cui ha preso parte, ed eventualmente anche delle failure che si sono verificate. Una failure è una situazione in cui il codice di risposta del servizio non è compreso tra 200 e 299. Nel caso in cui si verificasse una failure, ovviamente sarà incrementato sia il contatore delle failure che quello delle esecuzioni, così che l'utente possa avere un'idea più immediata della percentuale di esecuzioni che hanno dato dei problemi. Nella tabella dei test, invece, si può notare che è stato modificato il timestamp dell'ultima esecuzione. Questo valore sarà lo stesso di tutte le query eseguite per quel test l'ultima volta.

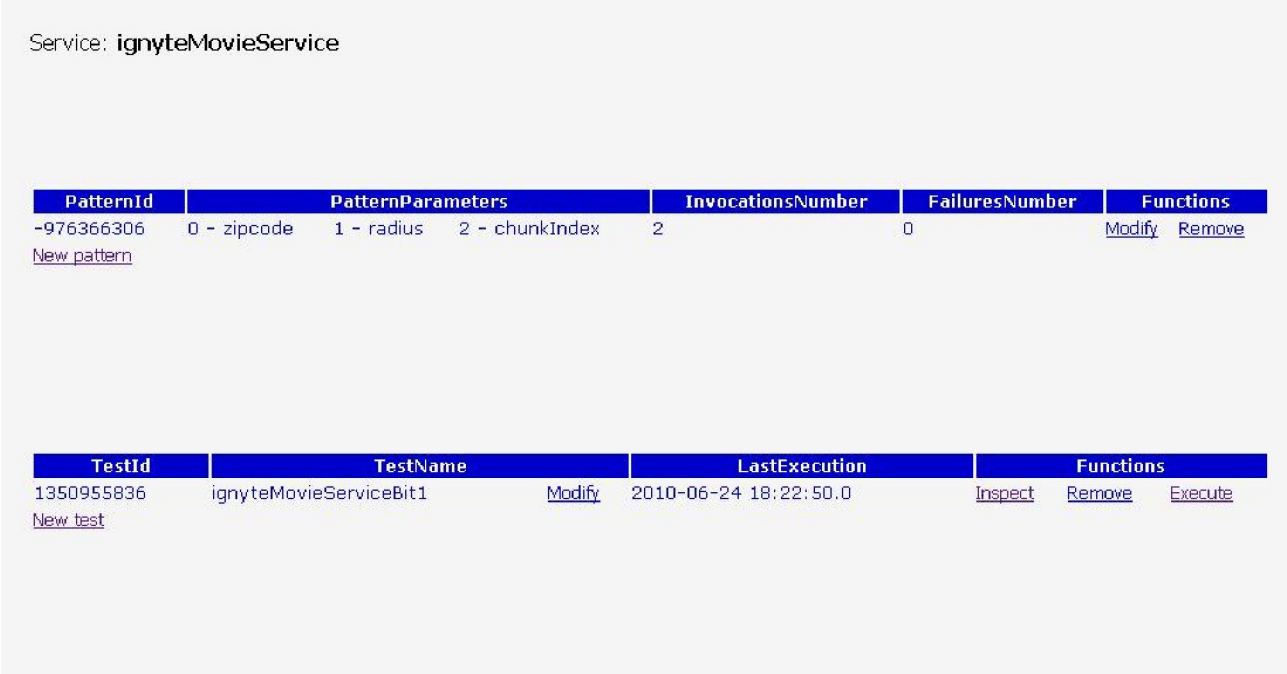


Figura 23: inspectService.jsp

5.2Test finali

L'ultima versione disponibile dello scce consente l'invocazione di diversi servizi:

- PGpoiIT: punti di interesse nell'area diMilano
- wublastById : trova proteine simili
- GoogleTheater: teatri o cinema negli Stati Uniti o in Italia
- GoogleRestaurants: ristoranti negli Stati Uniti o in Italia
- yqlRestaurants: ristoranti negli stati uniti, è un servizio di Yahoo
- ignyteMovieService: teatri o cinema negli Stati Uniti
- pagineGialleRestaurants: ristoranti in Italia

Per comodità sono state create delle classi per caricare nel database delle query per i servizi di maggior rilievo per il testing, di modo che poi si possa studiare il loro comportamento. In particolare, sono state inserite delle query di cui in precedenza era stato testato il buon funzionamento.

Ovviamente, l'inserimento segue tutti gli schemi precedentemente descritti, cioè sono stati creati i servizi, aggiunti i pattern necessari, aggiunti dei test set e delle query a questi test set. Si riportano, quindi, in dettaglio, gli elementi inseriti per ogni servizio.

Nome servizio	PGpoiIT
Pattern inserito	Tipologia, city, country, quartiere, chunkIndex

Nome del test set	pgPoiITBit1
Dettagli delle query	Tipologia = ristorante, city = Milano , country = italia, quartiere = niguarda, chunkIndex = 1 Dal momento che è un servizio che fornisce dati sull'Italia, si è deciso di optare per una query di tipo Feedback, in quanto supponiamo di avere informazioni sufficienti da poter giudicare la bontà dei risultati restituiti.

Nome servizio	GoogleTheater
Pattern inserito	Address, city, country, chunkIndex
Nome del test set	GoogleTheater Bit1
Dettagli delle query	Address = Marina, city = SanFrancisco, country = california, chunkIndex = 1 Anche in questo caso, la tipologia di query scelta è quella General.

Nome servizio	GoogleRestaurants
Pattern inserito	Address, city, country, category, chunkIndex
Nome del test set	GoogleRestaurantsBit1
Dettagli delle query	Address = Marina, city = SanFrancisco , country = california, category = meat, chunkIndex = 1 Anche in questo caso, la tipologia di query scelta è quella General.

Nome servizio	yqlRestaurants
Pattern inserito	Address, city, country, category, chunkIndex
Nome del test set	yqlRestaurants Bit1
Dettagli delle query	Address = Marina, city = SanFrancisco , country = california, category = meat, chunkIndex = 1 Anche in questo caso, la tipologia di query scelta è quella General.

Nome servizio	ignyteMovieService
Pattern inserito	Zipcode, radius, chunkIndex
Nome del test set	ignyteMovieServiceBit1
Dettagli delle query	Zipcode = 20131, radius = 10, chunkIndex = 1 Anche in questo caso, la tipologia di query scelta è quella General.

Nome servizio	pagineGialleRestaurants
Pattern inserito	Address, city, country, category, chunkIndex
Nome del test set	pagineGialleRestaurantsBit1
Dettagli delle query	Address = Via Pacini 1, city = Milano , country = italia, category = carne, chunkIndex = 1 Dal momento che è un servizio che fornisce dati sull'Italia, si è deciso di optare per una query di tipo Feedback, in quanto supponiamo di avere informazioni sufficienti da poter giudicare la bontà dei risultati restituiti.

Per brevità si mostra l'effetto dell'esecuzione di questa classe di caricamento solo per un servizio.

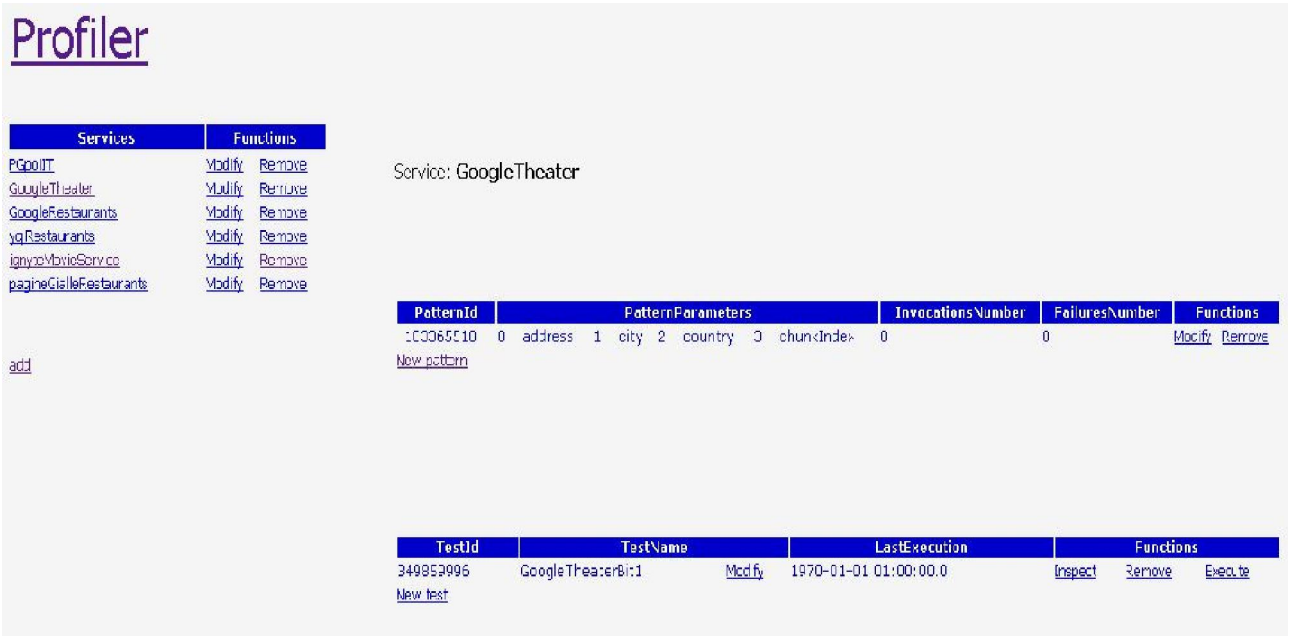


Figura 24: inspectService.jsp

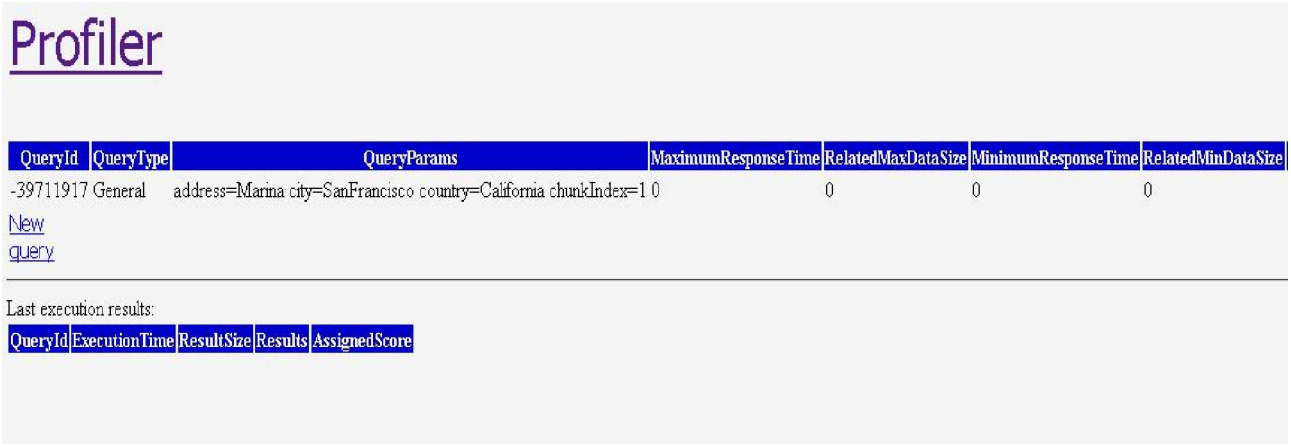


Figura 25: inspectBit.jsp

A questo punto il sistema dispone di tutti gli elementi necessari per poter effettuare un'esecuzione e, più precisamente, per continuare quell'analisi prestazionale che era stata iniziata all'inizio dello studio del problema.

Si riportano i risultati di tutti i test registrati per focalizzare l'analisi verso le prestazioni dei servizi. I test sono stati eseguiti nell'ordine indicato in tabella; più precisamente i servizi sono stati considerati uno per uno nell'ordine della tabella e per ognuno sono state effettuate tre esecuzioni successive.

Nome del servizio	Dimensione dei risultati	1° tempo di esecuzione	2° tempo di esecuzione	3° tempo di esecuzione	Media dei 3 test precedenti
PGpoiIT	93 (vuoto)	1.410.214.915	359.660.515	391.162.914	638.050.314
GoogleTheater	4328	573.775.578	540.217.612	472.886.219	514.941.407
GoogleRestaurants	1720	678.042.880	680.756.283	581.538.575	630.469.078
yqlRestaurants	3031	1.045.531.974	968.308.262	923.052.091	964.986.104
ignyteMovieService	8628	773.482.035	5.869.112	10.846.351	200.260.962
pagineGialleRestaurants	5135	436.579.719	379.457.305	385.786.532	396.902.522

Si può notare che la prima esecuzione in assoluto è quella meno performante e, per ogni servizio, i valori delle esecuzioni successive sono sempre migliori; questa non è una novità, in quanto già dai test preliminari si era potuta osservare una grande discrepanza tra i valori della prima esecuzione e quelli delle 99 successive.

Si va, ora, a richiamare l'esecuzione delle query che erano state effettuate nelle fasi preliminari di studio del problema, sfruttando il Profilatore per effettuare tali test. A questo proposito, si è proceduto con la registrazione di un altro insieme di query, quelle considerate più importanti sulla base dei risultati dei primi test.

Si è partiti con l'esempio di San Mateo relativo al servizio IgnyteMovieService. Tenendo fisso lo zip code, si è fatto variare il raggio. Per facilitare la costruzione e l'esecuzione del test, anche in questo caso è stata creata una classe per creare tante query, diverse tra loro solo per raggio di ricerca.

Si riportano i dati risultanti da tre esecuzioni del test set utilizzando il Profilatore. In questo caso, ovviamente, la prima esecuzione comprende tutte le query indicate in tabella, quindi, globalmente, il test è stato eseguito tre volte, una successiva all'altra.

Raggio	Dimensione dei risultati	1° tempo di esecuzione	2° tempo di esecuzione	3° tempo di esecuzione	Tempo medio dopo le 3 esecuzioni
1	28954	1.141.118.355	4.318.356	4.376.115	288.547.235

10	28954	1.205.121.671	4.614.204	4.720.642	304.794.289
15	28954	1.168.284.307	4.397.696	4.916.477	295.628.739
16	29171	2.045.273.561	9.224.007	7.044.471	517.146.627
18	29171	2.744.540.228	4.279.384	4.449.937	689.429.871
20	29171	1.428.928.258	8.271.931	3.966.775	361.283.434
22	31733	1.327.368.492	9.560.713	4.124.686	336.294.644
24	31733	1.487.192.367	4.530.743	4.212.407	375.036.981
25	31733	1.452.429.498	6.594.204	4.187.055	366.849.453
50	25307	6.105.061.188	5.865.690	4.111.626	1.529.787.532
100	25307	6.337.928.634	41.039.923	4.392.807	1.596.938.542

A questo punto ci si è chiesti il perché di una differenza così grande (anche tre ordini di grandezza) tra la prima esecuzione e quelle successive.

Si fa notare che, in questo caso, lo scce è stato avviato prima di effettuare tutti i test e non è stato fatto ripartire tra un test e l'altro. Dal momento che il test set eseguito è composto ovviamente sempre dalle stesse query, viene spontaneo pensare ad un qualche meccanismo di caching, implementato dal servizio o dallo scce. Per verificare tale ipotesi, si va ad aggiungere un nuovo test set con una sola query, uguale alle precedenti tranne che per il raggio di ricerca, che verrà impostato ad un valore non ancora presente tra quelli già utilizzati; si sceglie di prendere come esempio un raggio di 12.

Si va, quindi, a creare il test "bit2", che contiene un'unica query basata sempre sul pattern definito precedentemente e con valori 94128 come zip code, 12 come raggio e 1 come chunk index. Si procede, ora, a testare la query sempre su 3 esecuzioni. I risultati sono riassunti nella seguente tabella.

Raggio	Dimensione dei risultati	1° tempo di esecuzione	2° tempo di esecuzione	3° tempo di esecuzione	Tempo medio dopo le 3 esecuzioni
12	28954	1.390.532.176	42.850.278	7.021.632	361.856.429

I tre test sono stati effettuati ad una distanza temporale molto ravvicinata ed hanno restituito dei risultati più performanti del test precedente. Appare evidente che sia presente un meccanismo di caching, verificato, poi, attraverso lo studio della documentazione dello scce.

Per verificare se sia solo lo scce o anche il servizio stesso ad implementare tale meccanismo, e che politica di caching venga implementata, si decide di attendere un buon intervallo di tempo e poi ripetere entrambi i test. Si lascia passare più di un'ora e si ripete il test composto da molte tipologie di query.

Raggio	Dimensione dei risultati	1° tempo di esecuzione	2° tempo di esecuzione	3° tempo di esecuzione	Tempo medio dopo le 3 esecuzioni
1	28954	56.531.207	3.779.461	5.411.931	46.785.636
10	28954	3.725.195	3.611.004	3.847.347	41.391.360
15	28954	34.640.296	3.572.102	3.925.848	44.139.578
16	29171	3.742.514	3.573.498	3.872.140	67.940.587
18	29171	73.211.958	3.803.696	4.032.775	98.297.540
20	29171	3.803.835	3.676.026	3.734.763	48.422.296
22	31733	4.571.880	45.653.567	4.312.909	56.178.161
24	31733	32.031.515	100.534.915	33.449.363	92.741.972
25	31733	3.704.242	3.813.962	4.620.629	49.583.016
50	25307	39.919.109	68.387.323	20.629.158	223.624.739
100	25307	3.635.379	52.385.498	4.202.070	215.269.149

Osservando i risultati si può notare che non si è riusciti a replicare il risultato della prima esecuzione (in assoluto), ma i risultati successivi sono allineati e le medie differiscono solo per circa un ordine di grandezza.

Si passa, a questo punto, al secondo test effettuato in precedenza.

Raggio	Dimensione dei risultati	1° tempo di esecuzione	2° tempo di esecuzione	3° tempo di esecuzione	Tempo medio dopo le 3 esecuzioni
12	28954	5.436.026	46.917.415	92.063.028	103.672.424

Anche in questo caso non si è riusciti a replicare i risultati mostrati in precedenza.

Si prova, quindi, a far ripartire lo scce e ripetere il test. Se col riavvio dello scce i tempi sono sempre quelli più alti della prima esecuzione significa che il meccanismo di caching è implementato solo a livello di scce, mentre se i tempi restano più bassi anche col riavvio dello scce è evidente che siamo in presenza di caching implementato anche internamente al servizio.

Esecuzione 1 (dopo riavvio)	Esecuzione 2 (dopo riavvio)	Esecuzione 3 (dopo riavvio)	Esecuzione 4 (dopo riavvio)	Esecuzione 5 (senza riavvio)
8.666.431.069	5.046.593.911	3.278.567.387	3.631.034.950	4.406.007

4.974.931.032	22.904.294.051	15.527.874.352	5.729.000.613	4.811.017
3.943.649.821	13.325.985.565	4.964.466.433	2.179.801.832	6.215.175
22.341.841.421	3.646.071.358	8.231.572.379	6.858.654.452	6.018.223
10.506.493.633	4.198.022.304	4.962.752.040	7.124.994.098	4.544.781
7.989.070.818	7.055.686.686	16.701.446.482	5.982.886.722	4.243.695
6.747.258.660	24.798.344.622	4.789.993.421	6.672.754.498	4.565.175
5.632.217.922	6.259.480.674	9.232.888.886	1.259.938.763	4.923.391
15.766.395.589	8.137.637.675	6.909.742.300	4.502.455.619	5.219.029
3.338.378.272	5.612.879.989	6.566.348.656	1.265.517.964	4.428.496
5.608.737.144	4.997.152.432	3.487.875.065	3.697.119.466	4.539.334

I risultati sperimentali evidenziano la presenza della cache solo a livello dello scce. Il fatto che sia presente un meccanismo di caching non è un problema per il profilatore, il quale è stato progettato apposta per venire incontro a queste situazioni. Tenendo, infatti, traccia per ogni query di un tempo massimo e di un tempo minimo di esecuzione, può avvisare l'utente del range di variazione dei valori. Ogni valore massimo e minimo ha associata la dimensione dei dati restituiti e l'ultima dimensione dei dati totali.

Questo ci permette, ora, di riprendere lo studio di correlazione tra tempi e dimensione dei dati, sia analizzando esempi già studiati, che casi di studio che erano stati temporaneamente accantonati.

Si prova, adesso, ad eseguire nuovamente il test su San Mateo e, ora che si hanno a disposizione in modo più comodo molti più dati, cercare di osservare l'evoluzione dei tempi in base alla mole dei risultati. Più precisamente, si eseguirà il test due volte, sia per avere un campione maggiore che per poter osservare la variazione dei tempi tra la prima e la seconda esecuzione.

Raggio	maxTime	relSize	minTime	relSize	total Data
1	3.483.318.550	27837	4.196.483	27837	34921

10	2.215.718.548	27837	4.283.435	27837	34921
15	2.847.632.031	27837	3.818.991	27837	34921
16	6.101.595.873	24787	3.576.991	24787	53981
18	5.790.363.231	24787	4.355.092	24787	53981
20	10.423.439.362	24787	4.382.889	24787	53981
22	4.759.342.807	28050	3.570.076	28050	61907
24	6.501.168.305	28050	4.550.439	28050	61907
25	7.041.232.545	28050	3.588.096	28050	61907
50	10.802.858.604	25123	5.603.086	25123	135060
100	10.948.106.044	25123	3.637.264	25123	135060

Quello che forse non è stato ancora fatto notare, è che la variazione è sicuramente maggiore nel caso in cui il tempo di esecuzione sia quello del caso peggiore. I dati, tranne che per un singolo valore, ovvero quello relativo al raggio 20, confermano le ipotesi. Si presume che quell'unico valore così alto sia dovuto ad un sovraccarico di qualche genere.

Si prova, a questo punto, a cambiare servizio, e si sceglie di fare qualche test su PGpoiIT.

Si registra un insieme di query che varia solo per la zona di ricerca, e oltre a Niguarda si inseriscono alcune altre zone di Milano.

Zona	maxTime	relSize	minTime	relSize	total Data
Niguarda	545.819.651	92	310.854.871	92	92
Bande nere	583.312.264	3439	335.610.950	3439	11718
Barona	736.735.186	3263	473.004.460	3262	58425
Baggio	287.450.221	3064	283.561.249	3064	40176

Si può notare che i servizi che restituiscono la maggiore quantità di dati sono anche quelli con i tempi di esecuzione maggiori.

Si ripete l'esempio aggiungendo altre zone di Milano, per avere un'ulteriore conferma.

Zona	maxTime	relSize	minTime	relSize	total Data
Niguarda	269.825.145	92	162.051.322	92	92
Bande nere	277.768.334	3439	127.634.448	92	92
Barona	571.288.180	3263	190.576.806	92	92

Baggio	514.647.462	3089	132.127.128	92	92
De Angeli	318.200.847	3433	141.781.079	92	92
San Babila	287.919.624	3313	129.293.457	92	92
Porta Venezia	272.511.939	3470	139.924.907	92	92

In questo caso, l'utente si può rendere conto di un comportamento anomalo del servizio o dello scce. Alla seconda richiesta successiva, il servizio risponde con tutti chunk vuoti. Per curiosità si riprova un'altra esecuzione. Ovviamente non si hanno informazioni sui dati totali in quanto l'ultimo valore salvato è quello relativo al chunk vuoto.

Zona	time	Data size	Total data
Niguarda	384.271.045	92	92
Bande nere	129.150.074	92	92
Barona	127.549.311	92	92
Baggio	331.772.823	3089	39854
De Angeli	315.345.735	3433	30263
San Babila	128.112.302	92	92
Porta Venezia	298.579.149	3470	6894

Anche in questo caso, il comportamento è stato anomalo, ma il profilatore ne tiene comunque traccia e consente all'utente di fare delle valutazioni sui singoli servizi.

Si riprendono, ora, due test che erano stati accantonati precedentemente, ovvero quelli su Palm Beach e su New York utilizzando IgnyteMovieService.

Si inizia prendendo in considerazione il caso di Palm Beach e, a differenza del test effettuato nella fase preliminare di studio del problema, si da' alla variazione del raggio una granularità più fine.

Raggio	Dati restituiti	Dati totali	Tempo di risposta
1	18993	19038	795.613.688
10	18993	19038	821.082.498
25	27302	27347	850.016.273
50	62358	66066	1.716.757.926
75	62358	66066	2.823.792.270
100	62358	66066	1.534.230.957

I dati sembrano confermare la teoria; il tempo di risposta del servizio aumenta con il numero di dati restituiti e restituibili.

Si passa, ora, ad analizzare uno zip code di New York, variando il raggio come nel caso precedente.

Raggio	Dati restituiti	Dati totali	Tempo di risposta
1	21478	110386	3.301.312.737
10	21478	110386	3.314.067.08
25	16328	165652	6.912.247.925
50	17815	251416	10.473.315.393
75	17815	251416	10.367.397.183
100	17815	251416	10.591.093.490

In questo caso, la correlazione sembra essere più forte tra il tempo di risposta e i dati totali che possono essere restituiti.

IgnyteMovieService è il servizio con cui la correlazione tra dimensione dei dati e tempo di risposta sembra più evidente.

Si passa, ora, ad evidenziare un possibile utilizzo del profilatore. Si prendono in considerazione tre servizi che fanno ricerca sui ristoranti in Italia, ovvero PGpoiIT, GoogleRestaurants e pagineGialleRestaurants, e si effettua una ricerca di ristoranti nella zona di piazza Piola. Si presume che l'utente voglia controllare quale dei tre servizi è il più performante per effettuare quella query, quindi proceda alla registrazione dei servizi, dei pattern di accesso, dei test set e delle query. Si riportano i valori di ricerca per ogni servizio.

PGpoiIT: tipologia = ristorante, city = Milano, country = Italia, quartiere = Piola, chunkIndex = 1

GoogleRestaurants : address = Piazza Piola 1, city =Milano , country = Italia, category = ristorante, chunkIndex = 1

pagineGialleRestaurants: address = Piazza Piola 1, city =Milano , country = Italia, category = ristorante, chunkIndex = 1

A questo punto si pensa che, ad esempio, l'utente faccia un paio di esecuzioni per ogni test, così da avere un valore massimo e uno minimo. I risultati dell'esecuzione sono riportati in tabella.

Servizio	Tempo massimo	Quantità di dati	Tempo minimo	Quantità di dati	Dati totali
PGpoiIT	1.172.872.181	3204	363.870.408	3203	41777
GoogleRestaurants	758.534.813	1084	756.011.378	1084	7288
pagineGialleRestaurants	576.175.673	4813	569.271.304	4812	8499

L’utente, a questo punto, venuto in possesso di questi dati e, nel caso in cui avesse scelto di dichiarare delle query di tipo feedback, anche del dettaglio de risultati restituiti, potrà decidere, in base al criterio per lui più rilevante, quale servizio gli conviene scegliere per effettuare il suo lavoro. Se, fosse interessato, ad esempio, ad avere un grande quantitativo totale di dati, si troverebbe subito a preferire PGpoiIT.

Se fosse, invece, interessato ai cinema, dovrebbe lavorare su GoogleTheater ed IgnyteMovieService. GoogleTheater può rispondere a query relative sia all’Italia che agli Stati Uniti, mentre IgnyteMovieService ha informazioni relative soltanto agli Stati Uniti, quindi si opta per effettuare un’interrogazione relativamente ai cinema . Anche in questo caso si è scelto di interrogare i servizi su dati simili e, dal momento che i due pattern non combaciano, si è scelto di richiedere a GoogleTheater informazioni riguardo un indirizzo a cui si sa che c’è almeno un cinema, e ad IgnyteMovieService vengono richiesti cinema presso lo zip code relativo a quell’indirizzo. Per aumentare il campione, la ricerca su IgnyteMovieService è stata effettuata sia con raggio 1 che con raggio 10. L’indirizzo di ricerca è “209 West Houston Street” a New York, che dipende dallo zip code 10014.

I risultati dell’invocazione sono riportati in tabella.

Servizio	Tempo massimo	Quantità di dati	Tempo minimo	Quantità di dati	Dati totali
GoogleTheater	1.874.758.803	5102	621.062.593	5102	28765
IgnyteMovieService1	3.872.268.961	27589	6.560.471	27589	110098
IgnyteMovieService10	4.718.881.380	27589	17.562.637	27589	110098

Anche in questo caso sarà poi l’utente a decidere, sulla base dei dati ottenuti, quale sia il servizio che meglio risponde alle sue esigenze di utilizzo.

Nello studio di tutti questi test, va sempre tenuto presente che soprattutto il tempo minimo di esecuzione sarà un po’ più performante che nel caso in cui la query venisse eseguita “a mano” lo stesso numero di volte, in quanto per il calcolo dei risultati totali il profilatore esegue richieste che l’utente non visualizza.

6 UN ALGORITMO GENETICO PER LA CREAZIONE AUTOMATICA DEI TEST

Si è pensato potesse essere una funzionalità utile per l'utente quella di fornire la possibilità di avere un test creato automaticamente dal profilatore. Il ragionamento si basa sull'idea che un utente, alla registrazione di un nuovo servizio, potrebbe avere subito l'esigenza di sapere quali sono i suoi "valori limite" di funzionamento. Nella terminologia adottata per il profilatore, si tratta delle query di LowerBound e UpperBound; si vuole fornire, quindi, all'utente un insieme di query che restituiscono sia un numero molto basso di risultati che molto alto.

E' stato preso in considerazione il servizio IgnyteMovieService, essendo il più intuitivo per approcciare un problema di questo genere. Il problema, nel caso di questo servizio, diventa, quindi, quello di trovare un insieme di query, che spaziano, quindi, su zip code, raggi e chunk di diverso tipo, che diano risultati molto grandi oppure molto piccoli. Il ragionamento effettuato per questo servizio, però, è applicabile per qualsiasi pattern che comprenda questi campi tra i suoi parametri.

Si vuole, per prima cosa, estrarre degli zip code dal web, con lo scopo di formare delle query sulla base di questi. A tal proposito, è stato utilizzato il sito "http://www.zip-code-database.org" , che fornisce gli zip code di tutte le città degli stati uniti. Fare una scansione di tutte le città di tutti gli stati degli Stati Uniti sarebbe, ovviamente, un'operazione troppo onerosa dal punto di vista computazionale; si sceglie, quindi, di considerare gli zip code di due località. La prima località scelta è New York, con lo scopo che fornisca la base per le query di UpperBound, mentre per le query di LowerBound si sceglie Afton, una delle città principali dello stato meno popolato degli Stati Uniti, il Wyoming. Effettuata questa scelta, non è rimasto che creare un parser che, date le pagine contenenti gli zip code di queste due città, li estraesse per poterli poi utilizzare per formare delle query.

Una volta ottenuta, quindi, una lista di zip code, si è riflettuto su che scelte effettuare per implementare un algoritmo che porti ad avere, alla fine, un insieme di query di LowerBound e UpperBound. Vista la tipologia del problema, è sembrato sensato optare per un'implementazione di un algoritmo genetico. La natura numerica dei dati, e le loro caratteristiche, rendono, infatti, abbastanza intuitiva la formazione del gene, la creazione degli operatori di crossover e mutazione e la definizione della funzione di fitness.

Come gene si prende in considerazione la tripletta zip code, radius e chunkIndex. Una volta estratti gli zip code dalle pagine web, si procede, quindi, alla creazione della popolazione di partenza. Questa sarà formata banalmente da tutte le query che si possono formare con gli zip code trovati, raggio e chunkIndex pari a uno. In tabella si riporta un esempio della popolazione di partenza.

Zip code	radius	chunkIndex
10001	1	1
10002	1	1
10003	1	1
10004	1	1
10005	1	1
…	…	…
83110	1	1

Formata, quindi, una grande popolazione di partenza, il passo successivo è l'esecuzione delle query relative e il salvataggio dei risultati. La componente più importante dei risultati ottenuti è la quantità di dati restituiti e restituibili, ovvero il discriminante tra query di LowerBound e di UpperBound.

Una volta in possesso dei risultati relativi alle query della popolazione di partenza, si procede al calcolo della funzione di fitness. La valutazione della bontà del gene viene effettuata sulla base della quantità dei dati restituiti e restituibili. Si avranno, quindi, due funzioni di fitness differenti. Una per le query di UpperBound e una per le query di LowerBound. La prima rende possibile la selezione delle query che restituiscono maggiori risultati, la seconda di quelle con risultati più piccoli.

Si analizza nello specifico il caso della funzione di fitness per le query di UpperBound. La procedura è quella di, una volta effettuate le esecuzioni, selezionare le dieci query che hanno la maggiore quantità di risultati restituiti e, successivamente, le dieci query con il maggior numero di risultati totali. Si arriva, quindi, ad avere un insieme di 20 potenziali query di UpperBound. Si sottolinea che l'insieme che si vuole ottenere in seguito all'esecuzione dell'algoritmo per il caso di UpperBound è formato da solo dieci query.

Zip code single results	Radius single results	chunkIndex single results
Zip0	1	1
Zip1	1	1
…..	…	…
Zip9	1	1

Zip code total results	Radius total results	chunkIndex total results
Zip0	1	1

Zip1	1	1
…..	…	…
Zip9	1	1

L’insieme di query considerato alla fine dell’algoritmo è solamente il primo; il secondo ha la funzione di orientare la ricerca verso quei dati che hanno più possibilità di restituire un maggior numero di risultati.

Si passa, ora, a definire gli step di crossover e mutazione. Il crossover è un meccanismo per cui parti dei geni migliori vengono mischiate per provare ad ottenere individui ancora migliori. In questo caso, si procede a mescolare elementi del primo insieme con elementi del secondo. La scelta viene effettuata in maniera casuale, di modo che ad ogni iterazione dell’algoritmo di crossover si abbiano delle combinazioni differenti. Si ottiene, così, un nuovo insieme di venti query. Nel caso della prima iterazione, ovviamente, questo procedimento non produrrà modifiche sostanziali nell’insieme, in quanto tutti gli elementi sono differenti tra loro solo per lo zip code. A questo insieme viene, quindi, applicato l’algoritmo di mutazione. Come è intuitivo pensare, la mutazione è numerica e può avere luogo su ciascuno dei tre parametri della query. Si definiscono, quindi, tre funzioni di mutazione; una per lo zip code, una per il raggio e una per il chunkIndex.

La funzione di mutazione sullo zip code incrementa o diminuisce il valore di questo, operazione consentita in quanto si tratta di considerare zone adiacenti, in un certo senso cercando di avvicinarsi alla zona che da’ i migliori risultati in termini di quantità. Data la grande quantità di zip code possibili, questo processo non verrà effettuato fino al ritrovamento dello zip code ideale, ma un numero fissato di volte, scelto in base alla potenza di calcolo della macchina su cui si sta eseguendo l’algoritmo. Ad ogni modo, per adesso si considera la singola iterazione dell’algoritmo di mutazione in cui, quindi, si prendono tutti gli elementi e si modifica il loro zip code.

La funzione di mutazione sul raggio effettua anch’essa delle piccole mutazioni sul valore attuale del raggio di ricerca. Sia nel caso precedente che in quello che si sta trattando, la quantità su cui si basa la modifica è scelta in maniera casuale in un intervallo di valori sensati per quella grandezza. In questo modo non verranno continuamente effettuati passi troppo piccoli, ad esempio consentendo solo modifiche unitarie e si amplierà lo spazio di ricerca.

Per il chunkIndex il discorso è appena diverso, in quanto per esperienza si sa che non conviene richiedere valori troppo alti. Si è scelto, quindi, di effettuare ricerche al massimo fino al ventesimo chunk.

Per avere un’operazione di mutazione davvero efficace, si decide di effettuare tutte e tre le modifiche, una dopo l’altra, su tutti gli elementi ottenuti dal passo precedente.

Giunti a questo punto, si ha sia l’insieme dei valori di partenza che quello dei parametri delle query a cui è stato applicato l’operatore di crossover e mutazione tripla e non resta che mandare in esecuzione il nuovo insieme di dati. Si giunge, così, in possesso di per così dire un nuovo insieme di test, formato da nuove query, con risultati differenti. In gergo tecnico, si può dire che attraverso crossover e mutazione sono stati ottenuti dei nuovi insiemi di genitori. L’algoritmo procede, dunque, con il ricalcolo della funzione di fitness per i nuovi elementi, ma, a differenza degli algoritmi genetici tradizionali, a questo punto c’è ancora bisogno dei vecchi genitori per sapere

quali risultati sono i migliori. La procedura, infatti, prosegue con il confronto tra i valori appena calcolati e quelli già presenti nel primo insieme di risultati. Nello specifico, si considera ogni risultato in termini di quantità di dati e se si trova che è maggiore o uguale ai dati già salvati, e i parametri non sono già presenti nell'insieme, si inserisce. Lo stesso viene effettuato seguendo il criterio dei risultati totali.

L'algoritmo viene ripetuto un buon numero di volte, sempre scelto in maniera casuale ma nei limiti della potenza di calcolo della macchina su cui si sta lavorando. Vengono, quindi, ripetuti gli step di crossover e di mutazione tripla e ricalcolata la funzione di fitness, per modificare, poi, gli insiemi risultato.

Alla fine dell'esecuzione, quindi, ci si trova con due insiemi di individui; uno che comprende quelli con le query che restituiscono il maggior numero di valori, e un altro delle query che possono restituire il maggior numero di valori. L'insieme di query che andrà a comporre il risultato finale è ovviamente solo il primo, in quanto il secondo è servito soltanto come riferimento per ampliare lo spazio di ricerca.

Per quanto riguarda le query di LowerBound, l'algoritmo è identico se non per il fatto che la funzione di fitness va ad ordinare per numero di risultati minore e non maggiore.

Una volta eseguiti i due algoritmi si procede, dunque, all'unione dei due insiemi risultato e si ottiene, così, un insieme di query di test generato automaticamente. Nella creazione di tale insieme, ovviamente, si procederà alla rimozione di eventuali elementi duplicati.

Un algoritmo di questo tipo è applicabile a qualsiasi parametro numerico presente in un pattern, ma va ovviamente customizzato sulla base delle caratteristiche dell'elemento in questione. Dal momento che in ogni pattern è presente il campo chunkIndex, questo algoritmo può avere delle applicazioni, ovviamente con delle performance minori dal momento che le modifiche avranno degli effetti più limitati rispetto al caso precedentemente analizzato. Nel caso di parametri letterali, quali città, indirizzi o tipologie di locali, gli approcci possibili sono molteplici. Si va a descrivere quello più calzante con i casi in esame e che meglio ricalca l'approccio precedentemente adottato per i valori numerici. Si parte sempre da una ricerca web dei campi della query; in questo caso, ad esempio, potremmo avere bisogno di indirizzi a Milano. Online si può sfruttare un sito come www.gjm.it e parsare la pagina relativa alle vie di Milano. Si può considerare tutti gli elementi presenti come anche una parte di essi e considerare gli altri in un secondo tempo. Per la tipologia di locale si può o utilizzare sempre un sito web come quello precedentemente indicato oppure creare un piccolo insieme di valori di test significativi a cui attingere. I valori come il chunkIndex potranno essere inizializzati con valori standard come uno. La funzione di fitness è assolutamente identica a quella del caso completamente numerico precedente. L'operatore di crossover è applicabile senza limitazioni, e può funzionare anche nel caso in cui venissero considerate anche città differenti tra loro; potrebbe capitare, infatti, di avere due query, una relativa a Milano sull'indirizzo X e una relativa a Roma sull'indirizzo Y. Scambiando gli indirizzi si avrebbe ad esempio la ricerca dell'indirizzo Y a Milano, che potrebbe esistere e restituire dei risultati più performanti del caso con l'indirizzo X. Per quanto riguarda gli operatori di mutazione, nei casi di operatori letterali la procedura subisce delle ovvie limitazioni. Il massimo che si può pensare di effettuare è una mutazione di eventuali vocali finali negli indirizzi per passare, ad esempio, da "Via

Milano" a "Via Milani". Accorpando, però, il crossover su indirizzi e categorie, le mutazioni numeriche e letterali ed eventuali successivi campionamenti di indirizzi da pagine web, l'algoritmo genetico può, anche in questo caso, avere delle buone prestazioni.

7 CONCLUSIONI E SVILUPPI FUTURI

L'applicazione sviluppata ha caratteristiche che le consentono di svolgere il proprio lavoro in maniera completamente isolata rispetto al progetto SeCo; tuttavia, si può pensare che in un futuro possa essere integrata, anche se magari non completamente, nel progetto globale. Dovranno, ovviamente, essere riviste tutte le scelte relative all'implementazione e cercato un punto d'incontro tra i concetti propri di SeCo e quelli alla base del Profilatore. Si può pensare ad un'integrazione molto forte con il progetto in cui, ad esempio, dopo la registrazione di un nuovo servizio, il sistema preveda una fase di profilazione automatica, di modo da fornire all'utente immediatamente un'idea dei tempi di esecuzione e dei risultati restituiti.

Una volta che SeCo avrà la possibilità di prendere decisioni dinamicamente, i dati forniti potranno essere fruibili non solo manualmente dall'utente umano, ma vi sarà la possibilità che il sistema riesca a prelevarli autonomamente e si comporti di conseguenza. Il criterio di selezione della grandezza su cui discriminare relativamente alla scelta sarà presumibilmente istruito dall'utente, anche se si potrebbe pensare anche in questo caso di inserire un certo livello di automatizzazione, ad esempio proponendo all'utente un sottoinsieme dei criteri di scelta possibili.

Facendo maggiormente riferimento all'attuale implementazione del Profilatore, le evoluzioni più immediate riguardano l'algoritmo genetico e le view disponibili. Potrebbe essere utile, ad esempio, fornire delle proiezioni sui dati diverse da quelle già esistenti; l'ideale sarebbe poter fornire all'utente un insieme di criteri di proiezione, anche trasversali rispetto ai dati del singolo servizio, e creare in maniera dinamica view anche molto complesse.

Per quanto riguarda l'algoritmo genetico, si può pensare ad un'integrazione progressiva di diverse sorgenti di dati, per poter creare delle query di qualsiasi livello di complessità; si tratta, quindi, di poter fornire dei dati di test per qualunque tipo di parametro di un pattern e migliorare, così, la generazione automatica dei test.

Bibliografia.

[1] D. Braga, A. Campi, S. Ceri, A. Raffio. Joining the results of heterogeneous search engines. Inf. Syst. 33(7-8): 658-680, 2008.

[2] D. Braga, S. Ceri, F. Daniel, D. Martinenghi. Optimization of Mutidomain queries on the Web. VLDB'08, pp. 562-573, 2008.

[3] D. Braga, S. Ceri, F. Daniel, D. Martinenghi. Mashing Up Search Services. IEEE Internet Computing 12(5): 16-23, 2008.

[4] I. Elgedawy, Z. Tari, and M. Winiko. Exact functional context matching for web services. In ICSOC, 2004.

[5] R. Fagin. Combining fuzzy information from multiple systems. J.Comput. Syst. Sci., 58(1):83-99, 1999.

[6] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing partial rankings. SIAM J. D. Math., 20(3):628-648, 2006.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci., 66(4):614-656, 2003.

[8] C. Fellbaum, ed. WordNet: An Electronic Lexical Database (Language, Speech, and Communication). MIT Press, May 1998.

[9] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. VLDB J., 13(3):207-221, 2004.

[10] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top- query processing techniques in relational database systems. ACM Comput.Surv., 40(4), 2008.

[11] D. Kossmann, F. Ramsak, S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In VLDB'02, pp. 275-286.

[12] N. Mamoulis, M. L.Yiu, K. H. Cheng, and D. W. Cheung. Efficient top-k aggregation of ranked inputs. ACM TODS, 32(3), 2007.

[13] C. D. Manning. Probabilistic Syntax. In Rens Bod, Jennifer Hay, and Stefanie Jannedy (eds), Probabilistic Linguistics, pp. 289-341. Cambridge, MA: MIT Press, 2003.

[14] MetaSearch. http://www.lib.berkeley.edu/TeachingLib/Guides/ Internet/MetaSearch.html.

[15] S. Osinski, J. Stefanowski, and D. Weiss. Lingo: Search results clustering algorithm based on singular value decomposition. In Intelligent Information Systems, pp. 359-368, 2004.

[16] D. Papadias, Y. Tao, G-Fu, and B. Seeger. Progressive skyline computation in database systems. ACM TODS, 30(1):41-82, 2005.

[17] M. Papazouglu and K. Pohl eds, Wp 2009-2010 expert group: Longer term research challenges in software & services. 2008.

[18] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma. Meteor-s web service annotation framework. In WWW 2004, pp. 553-562.

[19] S. Ran. A model for web services discovery with QOS. SIGecom Exch., 4(1):1-10, 2003.

[20] Stanford Natural Language Proc. Group. http://nlp.stanford.edu/

[21] M. Stollberg, U. Keller, H. Lausen, and S. Heymans. Two-phase web service discovery based on rich functional descriptions. In ESWC '07: pp. 99-113. Springer-Verlag, 2007.

[22] Wordnet. http://wordnet.princeton.edu/

[23] Wordnet Domains. http://wndomains.itc.it/wordnetdomains.html

[24] A Ankolenkar et al.DAML-Shttp://www.daml.org/services/daml-s/2001/10/daml-s.html.

[25] T. Berners-Lee, J. Handler, O. Lassila, The semantic Web, Scientific American, May 2001.

[26] A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, S. Paraboschi. Designing data marts for data warehouses. ACM Trans. Software Engineering Methodology 10, 4 (Oct. 2001), 452-483

[27] A. Bozzon, M. Brambilla, S. Ceri, P. Fraternali, Liquid query: multi-domain exploratory search on the Web, Proc. WWW-2010 Conference, to appear.

[28] S. Ceri, M. Brambilla eds., Search Computing - Challenges and Directions. Springer LNCS, Vol. 5950.

[29] S. Ceri, M. Matera, F. Rizzo, and V. Demaldè. Designing data-intensive Web applications for content accessibility using Web marts. Commun. ACM, 50(4):55–61, 2007.

[30] Colombo, M., Di Nitto, E., Di Penta, M., Distante, D., Zuccalà, M. Speaking a common language: A conceptual model for describing service-oriented systems. ICSOC 2005 pp.48-60

[31] CORDIS: http://cordis.europa.eu/fp7/ict/ssai/ on Sept. 17, 2009.

[32] D. J. De Witt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I. Hsiao, and R. Rasmussen. The Gamma Database Machine Project. IEEE TKDE, 2(1):44-62, 1990.

[33] A. Doan, P. Domingos, A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In SIGMOD, 2001.

[34] Fensel, D., Bussler, C. 'The Web Service Modeling Framework WSMF', Electronic Commerce Research and Applications, Vol. 1, No. 2, pp.113-137

[35] D. Fensel, M. Musen, Special Issue on Semantic Web Technology, IEEE Intelligent Systems (IEEE IS) 16 (2).

[36] Fusion Tables: http://tables.googlelabs.com/

[37] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The Lixto data extraction project: back and forth between theory and practice. In PODS '04.

[38] JSON: http://JSON.org/

[39] A.Y. Levy, A. Rajaraman, J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In VLDB, 1996.

[40] F.Leymann,WSFL.http://www-4.ibm.com/software/solutions/Webservices/pdf /WSFL.pdf.

[41] Martin, D., Burstein et al.. Bringing Semantics to Web Services with OWL-S. World Wide Web 10, 3 (Sep. 2007), 243-277.

[42] OASIS. Web Services Business Process Execution Language. Technical report, http://www.oasis-open.org/committees/wsbpel/, 2007.

[43] Quartel D.S., Steen M.W., Pokraev S., Sinderen M.J.COSMO:A conceptual framework for service modeling and refinement,Information Systems Frontiers,9(2-3),p.225-244,July 2007

[44] U. Srivastava, K. Munagala, J. Widom, and R. Motwani. Query optimization over Web services. In VLDB'06, pages 355-366. VLDB Endowment, 2006.

[45] W3C. Web Service Choreography Interface (WSCI) 1.0. W3C Note, August 2002.

[46] W3C. Web Services Choreography Description Language Version 1.0. Dec. 2004.

[47] J. Wang, J.R. Wen, F. Lochovsky, W.Y. Ma. Instance-based schema matching for Web databases by domainspecific query probing. In VLDB, 2004.

[48] Web Services Architecture (2004) http://www.w3.org/TR/ws-arch/

[49] W.Wu, C. Yu, A. Doan, W. Meng. An interactive clustering-based approach to integrating source query interfaces on the Deep Web. In SIGMOD, 2004.

[50] Yahoo! Search Boss: http://developer.yahoo.com/search/boss/

[51] A. Campi, S. Ceri, A. Maesani, S. Ronchi: Designing Service Marts for Engineering Search Computing Applications The Tenth International Conference on Web Engineering (ICWE 2010), Vienna, Austria , Monday, July 5, 2010

[52] D. Barbieri, A. Bozzon, D. Braga, M. Brambilla,A. Campi, S. Ceri, E. Della Valle, P. Fraternali, M. Grossniklaus, D. Martinenghi, S. Ronchi, M. Tagliasacchi: Data-driven optimization of search service composition for answering multi-domain queries