UNIVERSITÀ DEGLI STUDI DI TRENTO
Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea (triennale) in Informatica

Elaborato Finale

Aggiornamento dei dati in una rete peer-to-peer di database: il progetto Hyperion

Relatore:
Paolo Giorgini

Laurenado:
Piero Turra

Anno Accademico 2007 - 2008

Indice

Capitolo 1

Introduzione

Negli ultimi anni si stà sempre più utilizzando la tecnologia del peer to peer, ossia reti informatiche che non possiedono client fissi, ma una serie di nodi che fungono sia da client che da server, ogni nodo è chamato peer. Attualmente questo sistema è utilizzato per la diffusione di notizie, per il file sharing e da alcuni client di instant messaging.

Hyperion è un progetto attuato tramite la tecnologia peer-to-peer che ha lo scopo di definire un sistema per l'amministrazione dei dati. In pratica questa tecnologia viene usata per amministrare vari database indipendenti in vari peers, permettendone la comunicazione tramite la traduzione dei dati. In particolare il sistema permette anche la manipolazione dei databases e la propagazione delle modifiche attraverso la rete, effettuando inserimenti e cancellazione di dati.

Con tale sistema però potrebbero verificarsi delle inconsistenze nei dati tra i peers, infatti per aggiornare un campo è necessario eseguire una cancellazione e successivamente un inserimento inviando due messaggi distinti. Le due operazioni potrebbero non venir effettuate nel guisto ordine oppure solo in parte a causa di eventuali ritardi dovuti alla propagazione dei pacchetti attraverso la rete o alla loro perdita. Ciò potrebbe causare notevoli disagi in applicazioni come ad esempio databases di compagnie aeree: se per un volo in comune tra due compagnie fosse necessario aggiornare un dato e ciò non avvenisse correttamente, in uno dei database potrebbero trovarsi più istanze dello stesso volo oppure nessuna.

L'obiettivo di questa tesi è quello di rendere consistente l'aggiornamento dei dati, rendendolo attuabile tramite un'unica operazione che sostituisca le azioni di cancellazione e inserimento consecutive. Ciò permette di propagare tutte le modifiche tramite la rete con un unico messaggio.

Nei seguenti capitoli si illustrerà dapprima lo stato dell'arte per quando riguarda i peer to peer, poi con più attenzione gli obiettivi e la struttura di

Hyperion. Di seguito verranno descritte le tecnologie sfruttate nella comunicazione tra i peers: le mapping table e le ECA rule. Successivamente verrà chiarito il problema da risolvere, la soluzione che è stata adottata e l'esito di alcuni test effettuati per verificarne la correttezza.

Capitolo 2

Il Peer-to-Peer e basi di dati

Il progetto di cui si tratta in questa tesi è parte di un sistema peer-to-peer (P2P) per la gestione dei dati, in questo capitolo andremo dapprima a descrivere cos'è il P2P ed in che modo viene utilizzato, di seguito sarà descritto il progetto Hyperion ed i suoi scopi.

2.1 Sistemi Peer-to-Peer

Generalmente per peer-to-peer (o P2P) si intende una rete di computer o qualsiasi rete informatica che non possiede client o server fissi, ma un numero di nodi equivalenti (peer, appunto) che fungono sia da client che da server verso altri nodi della rete.

Questo modello di rete è l'antitesi dell'architettura client-server. Mediante questa configurazione qualsiasi nodo è in grado di avviare o completare una transazione. I nodi equivalenti possono differire nella configurazione locale, nella velocità di elaborazione, nella ampiezza di banda e nella quantità di dati memorizzati. L'esempio classico di P2P è la rete per la condivisione di file (File sharing).

Alcune reti e canali, come per esempio Napster[1], OpenNap[2] o IRC usano il modello client-server per alcuni compiti (per esempio la ricerca) e il modello peer-to-peer per tutti gli altri. Proprio questa doppia presenza di modelli, fa sì che tali reti siano definite ibride. Reti come Gnutella[3] o Freenet[4], vengono definite come il vero modello di rete peer-to-peer in quanto utilizzano una

[1] www.napster.com
[2] opennap.surceforge.net
[3] www.gnutella.com
[4] freenetproject.org

struttura peer-to-peer per tutti i tipi di transazione, e per questo motivo vengono definite pure.

Utilizzi più innovativi prevedono l'utilizzo delle reti peer-to-peer per la diffusione di elevati flussi di dati generati in tempo reale come per esempio programmi televisivi o film. Questi programmi si basano sull'utilizzo delle banda di trasmissione di cui dispongono i singoli utenti e la banda viene utilizzata per trasmettere agli altri fruitori il flusso dati. Questa tipologia di programmi in linea di principio non richiede server dotati di elevate prestazioni, dato che il server fornisce i flussi video a un numero molto limitato di utenti, che a loro volta li ridistribuiscono ad altri utenti.

2.2 Peer-to-Peer e amministrazione di dati

Intuitivamente gli strumenti per l'amministrazione e l'integrazione di dati sono molto adatti allo scambio di informazioni. Sfortunatamente essi soffrono di due problemi importanti: tipicamente abbisognano di una progettazione completa del loro schema prima di essere utilizzati per immmagazzinare o condividere informazioni e sono difficili da estendere perchè la manipolazione dello schema è un grosso lavoro e può compromettere la retrocompatibilità. Come risultato, molti lavori di condivisione dei dati sono facilitati da dei tool nondatabase-oriented con un piccolo supporto di semantica. L'obiettivo dei peer-to-peer per l'amministrazione dei dati (PDMS) è quello di proporre l'uso di un'architettura per la gestione dei dati distribuita, facile da estendere, nella quale ogni utente possa contribuire con nuovi dati, informazioni sullo schema o sulle relazioni con altri peer. I PDMS rappresentano un passo naturale verso sistemi di integrazione dei dati, che sostituiscano i loro singoli schemi logici con una collezione di mappamenti semantici interconnessi tra i database propri di ogni peer. Il progetto Hyperion rappresenta un esempio di questo tipo di sistemi.

2.3 Il sistema Hyperion

Hyperion [1] è un progetto congiunto tra l'Università di Toronto, che ha iniziato il progetto, e l'Università di Trento.

Il progetto Hyperion è nato come prototipo per la ricerca sull'amministrazione dei dati nel modello P2P e può facilmente essere esteso con nuove funzionalità e servizi. Gli obiettivi principali del progetto sono: la definizione precisa di un'architettura per la gestione dei dati nel peer-to-peer; lo studio di un meccanismo di integrazione, scambio e mappatura per creare un am-

biente dinamico ed autonomo; l'implementazione di algoritmi per un'efficente ricerca, recupero e scambio di dati tra i peers.

Nel progetto Hyperion ogni peer include un database con la propria organizzazione e i propri dati e può connettersi o lasciare la rete a propria discrezione. Esso può formare un collegamento con un altro peer per scambiare dei dati. Per definire le corrispondeze tra i valori e gli schemi tra i diversi databases, i peers amministrano delle mapping table. Il progetto fornisce il livello su cui possono essere supportati dei servizi di livello più alto. Un esempio di tali servizi è la possibilità di tradurre e rispondere a delle queries usando i dati in comune.

Un servizio aggiunto di recente è la propagazione delle modifiche effettuate in un database locale verso dei peers designati. L'abilità di permettere e coordinare le modifiche attraverso la rete P2P è ottenuta tramite l'uso distribuito di Event-Condition-Action (ECA) rule. Il servizio da la possibilità di creare le ECA rule e di immagazzinarle nel database del peer controllando ciclicamente quando propagare le modifiche attraverso la rete. Inoltre tiene conto del carattere dinamico della rete tenendo traccia delle modifiche che devono essere inviate quando i peers entrano o lasciano la rete.

2.4 Architettura di Hyperion

L'architettura di Hyperion può essere schematizzata dalla figura 2.1, che mostra le varie parti del sistema durante il processamento di una richiesta da parte di un utente. Per prima cosa saranno spiegate le componenti principali del sistema e successivamente come esse interagiscono nella ricezione di una richiesta.

2.4.1 Componenti del Sistema

Acquaintance: è un canale di comunicazione astratto tra due Hyperion peers. Per condividere e scambiare datii peers hanno bisogno di essere collegati agli altri tramite un'acquaintance. Ogni collegamento è rappresentato da una diversa acquaintance, ogni peer ne amministra un gruppo.

Dispatcher: quando un Service Message arriva attraverso un'acquaintance il primo a riceverlo è il dispatcher. Esso aspetta i messaggi in entrata da un peer ed è collegato alla sua acquaintance. Perciò per ogni acquaintance locale ci sarà un dispatcher nel peer remoto e viceversa, infatti le acquanintance sono bidirezionali e ad una in entrata corrisponde anche una in uscita.

Hyperion Message: ogni peer scambia dei messaggi con gli altri. Hyperion Message è la classe principale per tutti i messaggi di Hyperion. Ogni

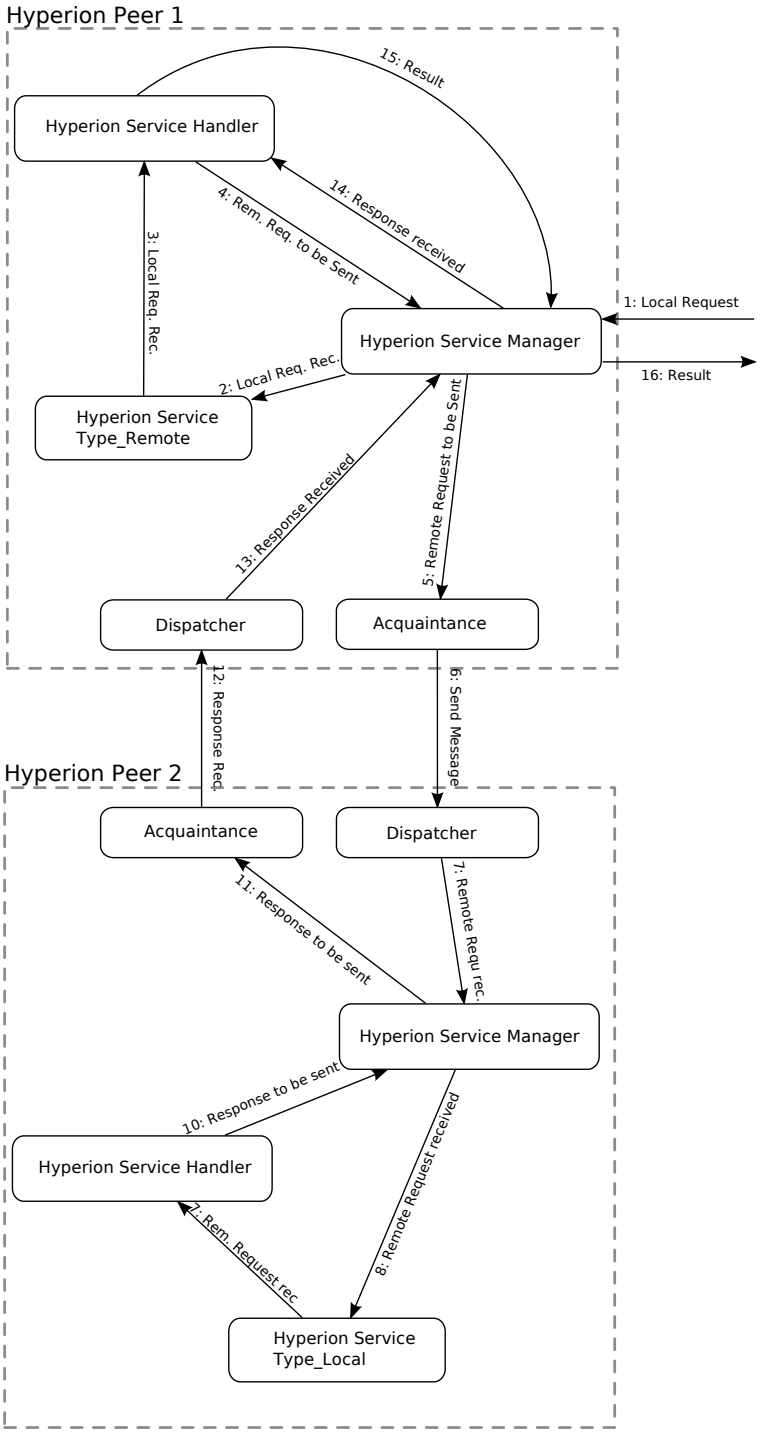

Figura 2.1: Schema dell'architettura di Hyperion durante il processamento di una richiesta.

messaggio è formato da due parti, l'intestazione ed il corpo. L'intestazione contiene l'identità del messaggio, la sua sorgente e la destinazione. Il corpo contiene i dati o altri messaggi più specifici. Tutti i tipi di messaggio all'interno di Hyperion sono un'estensione di Hyperion Message. Un'importante tipo di messaggi sono i Service Message, essi posso essere richieste di servizi, risposte o risultati. Se un messaggio di richiesta arriva da un processo interno allora è impostato come locale, se proviene dall'esterno, come remoto. La differenza tra i due è che il risultato di una richiesta locale viene spedito direttamante al processo richiedente, mentre l'altro deve essere spedito tramite un'acquaintance.

Hyperion Service: ogni peer fornisce all'utente più servizi.Un Hyperion Service è in generale uno di questi servizi. Essi possono operare in due modi: locale e remoto. Un servizio locale processa tutte le richieste localmente, senza l'ausilio di altri peer. Un servizio remoto invece richiede delle risorse remote per finire il proprio lavoro. Un peer può usare un servizio remoto solamente attravesro un acquaintance. Ognuno di essi è identificato dal proprio nome, questo perchè, se un peer locale vuole utilizzare un servizio remoto, il nome di tale servizio dovrebbe essere uguale a quello locale. Più servizi possono essere in esecuzione contemporaneamente in un peer.

Hyperion Service Handler: Un Hyperion Service può gestire più richieste simultaneamente usando gli'Hyperion Service Handler. Quando un servizio riceve una richiesta crea un Service Hanlder che se ne occupi e passa oltre. Questo la processa autonomamente e termina una volta completato il lavoro, mentre il servizio continua ad aspettare nuove istanze.

Hyperion Service Manager: amministra tutti i messaggi di servizio, come richiestte, risposte o risultati, per tutti i servizi offerti dal peer.

Hyperion Peer: ogni peer conosce l'operto dell'Hyperion Service Manager, amministra la ricerca di nuovi peers nella rete, si occupa del guppo di acquaintance dirigendone il collegamento e lo scollegamento e opera tramite le mapping table.

2.4.2 Percorso di una richiesta

Prendendo in considerazione la figura 2.1 si possono descrivere i vari passaggi che seguono una richiesta inviata da un processo del client locale:

1. Un processo nel client invia una richiesta locale al Service Manager per un servizio offerto dal Peer 1 e aspetta una risposta.

2. Il Service Manager cerca il servizio richiesto dal processo, se lo trova gli passa la richiesta.

3. Il servizio crea un Service Handler che processa la richiesta.
4. L'Handler, dopo che la richiesta è stata processata localmente, manda al Service manager una richiesta remota da inviare e lo informa su dove trovare la risorsa in grado di soddisfarla, quindi aspetta una risposta.
5. Il menager invia la richiesta remota attraverso l'acquaintance del peer 1, associata al peer 2.
6. Il Dispatcher collegato con l'acquaintance del peer 2 e associato al peer 1 riceve il Service Message.
7. Il Dispatcher manda il Service Message al Service Manager del proprio peer.
8. Il Service Manager cerca il servizio adatto, se lo trova gli invia la richiesta.
9. Il servizio crea un Service Handler che processa la richiesta.
10. Dopo aver processato localmente la richiesta, l'Handler manda una risposta da inviare al Service Manager.
11. Il manager invia la risposta tramite l'acquaintance del peer 2, associata la peer 1.
12. Il Dispatcher del peer 1, associato con l'acquaintance del peer 2 riceve il Service Message.
13. Il Dispatcher manda la risposta ricevuta al Service Manager del peer 1.
14. Il manager risveglia l'Handler che stava aspettando e gli manda la risposta, aspettando una risposta dall'Handler.
15. L'Handler finische di processare la risposta e da il risultato al Service Manager.
16. Il manager notifica al client che il risultato è pronto.

Capitolo 3

Aggiornamento e propagazione in una rete peer-to-peer

Per effettuare delle modifiche su basi di dati distribuite su una rete di computers ci sono alcuni inconvenienti, come ad esempio la possibilità che i dati siano eterogenei e amministrati in modo differente, perciò sono necessarti degli strumenti che permettano di confrontarle. In questo capitolo si parlerà di questi strumenti, che sono le mapping table, che permettono di mettere in relazione database diversi, e le ECA rule, che permettono ai peers di comunicare tra di loro.

3.1 Interfacciamento dei database: le mapping table

[2] Tradizionalmente, nei sistemi multi-database, lo scambio e l'integrazione dei dati tra fonti etertogenee è attuato attraverso l'uso di view (programmi di traduzione logica). Questi programmi o query dipendono strettamente dalla struttura logica delle sottostanti basi di dati. Per ristrutturare e mappare correttamente i dati, le basi di dati devono essere disponibili a condividere almeno in parte le loro tabelle e devono cooperare nello stabilire ed ammnistrare i programmi e le query.

Per trovare dei dati quando c'è poco o nessun accordo nella loro struttura logica bisogna focalizzare l'attenzione sui contenuti e a cosa essi corrispondono. Se è possibile mapparli, in particolare gli identificativi (nomi o chiavi), si possono anche richiedere e scambiare dati specifici. Questo approccio risulta particolarmente utile nel caso in cui non sia possibile l'uso di nominativi standard. In tali casi peers diversi devono stabilire proprie convenzioni nella nominazione. Gli standard spesso emergono solo dopo che molti elementi

eterogenei hanno sviluppato le proprie convenzioni. Ciò può significare più applicazioni che dipendono dall'uso di convenzioni interne. Così conformarsi agli standards può risultare un processo lento e costoso.

Per trovare dati in questi ambienti sono state inventate le mapping table, che imagazzinano le corrispondenze tra i contenuti. Nel caso più semplice sono delle tabelle contenenti coppie di nomi, provenienti da basi di dati differenti che hanno lo stesso significato. Esse possono essere utilizzate in semplici ricerche di valori dove, per esempio, un peer che cerchi una riga chiamata X, prima consulti una mapping table per trovare il nome a cui corrisponde X nell'altro peer. in generale si potrebbe aver bisogno di mappare dei valori che contengono attributi multipli. Per esempio le coordinate geografiche potrebbero essere indicate con latitudine e longitudine in un primo peer e sottoforma di codice postale in un secondo peer. Anche in questo caso è possibile usare le mapping table per scambiare specifici contenuti. La query 'Prendi tutte le informazioni relative alle coordinate (X,Y)' nel primo peer diventerà 'prendi tutte le informazioni relative al codice postale X' nel secondo peer.

Attualmente la creazione delle mapping table è un processo manuale che richiede molto tempo ed è attuato da degli esperti. Nonostante siano largamente usate, specialmente in campo biologico, non esistono degli strumenti che ne facilitano la creazione e il mantenimento.

3.1.1 Esempi

Si consideri un esempio nel campo dei database biologici. Attualmente c'è un travolgente numero di dati riguardanti la genteica provenienti da tante sorgenti pubbliche, le quali sono specifiche per ogni laoratorio di ricerca. L'integrazione di queste risorse, per fornire un accesso uniforme agli scienziati, sarebbe molto utile, anche se sembrerebbe inottenibile per una miriade di fattori tecnici, politici e finanziari. Tra le ragioni tecniche c'è l'eterogeneità delle basi di dati che spaziano da database relazionali a files formattati a fogli di calcolo. In aggiunta, gli schemi ed i formati evolvono rapidamente in risposta alle nuove tecnologie e requisiti della biologia. Per realizzare un'integrazione, si usano quello che vengono chiamate mapping table. Per esempio nel mondo dei dati biologici , le mapping table vengono utilizzate per accoppiare un dato gene in una base di dati, con una data proteina in un'altra (dove il gene viene utilizzato per codificare la proteina). Da notare che la mapping table non è necessariamente una funzione, ci potrebbero essere più proteine associate ad un gene. Il rapporto tra le tabelle può essere di molti a molti. Questo porta all'esistenza di più sinonimi per lo stesso identificatore. Quando si fa un aggiornamento i vecchi nomi possono dover

essere mantenuti, per esempio, se si riferiscono al contenuto di fonti come articoli di giornale, che non cambiano nel tempo e possono contenere nomi antiquati.

In questo scenario si motivano le principali caratteristiche delle mapping table. In primo luogo si dimostra che possono essere usate per associare valori con disparati domini. in secondo luogo si dimostra che le mapping table sono uno strumento appropriato da usare in sistemi peer-to-peer infatti esse rispettano l'autonomia di ogni peer. In seguito verranno presentati alcuni esempi che motivano le potenzialità di un'applicazione in cui sono utilizzate le mapping table.

Associazioni tra ed attraverso basi di dati

Usando le mapping table, apparentemente, siamo ingrado di associare database sconnessi. In uno scenario integrato tipico, ci si trova spesso in collegamento tramite una parola, per esempio, un insieme di sorgenti che contengono tutte delle informazioni sui geni. Tuttavia, ci sono situazioni in cui sorgenti formate da parole non corrispondenti possono essere associate tramite parole semanticamente vicine.

Per esempio si consideri il Gene database (GDB) (Human Genome Database) e lo SwissProt database (The SWISS-PROT Protein Knowledgebase). Il GDB, a parte le informazioni genetiche, contiene anche una mapping table che contiene le associazioni tra i nomi dei geni in GDB el i nomi delle proteine nello SwissProt. Un'esempio è mostrato nella mapping table (b) della tabella 3.1. Essa associa geni e proteine con nomi di malattie genetiche rappresentate nel MIM database (Online Mendelian Inheritance in Man). La tabella (c) associa direttamente geni e malattie. Ogni tabella può essere costruita da diversi amministratori con differenti conoscenze sui data base sottostanti.

Autonomia dei Peers

L'autonomia è un fattore di massima importanza in qualsiasi sistema peer-to-peer e in molte applicazioni di rete. Le mapping table rispettano l'autonomia delle basi di dati a loro associate. Questo si nota nella mapping table (a) nella tabella 3.1, dove non viene ne specificato a quali geni e proteine ci si riferisce in generale e ne come devono essere rappresentati o salvati nei rispettivi data base. Piuttosto esse codificano, come un esperto ha determinato, che certi geni siano in relazione con altre proteine. Queste informazioni servono per effettuare ricerche attraverso i peers.

Mapping table (a)

GDB_id	SwissProt
g1	p2
g2	p4

Mapping table (b)

GDB_id	SwissProt	MIM_id
g1	p1	m1
g1	p2	m2
g2	p3	m3

Mapping table (c)

GDB_id	MIM_id
g3	m4

Figura 3.1: Un set di mapping table

Ricavo automatico di associazioni valide

In generale, una mapping table è composta da 2 set disgiunti di attributi X e Y (Nella Tabella 3.1 vengono usate delle linee doppie per la loro distinzione). Una riga (x,y) indica che il valore x è associato al valore y. Perciò specifica un'associazione tra valori in 2 peers. Ci sono alcune regole che gli amministratori dovrebbero usare per creare dei set di mapping table. Queste regole permettono di dichiarare l'estensione di tutti i dati dei loro database. Sono stati sviluppati degli strumenti che trovano automaticamente nuove associazioni e identificano inconsistenze nelle tabelle.

Si consideri una singola tabella che associa valori di X a valori di Y. Questa tabella può rappresentare completamente o parzialmente il contenuto di X. In un primo caso i valori di X possono solo essere associati con i valori di Y se sono presenti nella tabella. Perciò valori di X che non appaiono nella mapping table non possono essere associati con valori di Y. Questa viene chiamata semantica *closed-world*. In alternativa un amministratore che ha solo una conoscenza parziale di un dominio potrebbe specificare un semantica *open-world*. Una tabella open-world non costringe i valori di X mancanti ad essere mappati. Perciò potrebbero essere associati con altri valori di Y. Sotto la semantica open-world, nella mapping table (a) della tabella 3.1, un gene che non presente al suo interno può comunque essere associato ad una proteina. Questa semantica tiene conto del fatto che gli amministratori sono spesso esperti solo di una parte di un dominio. Questa semantica è usata spesso in ambienti dove i nuovi dati possono emergere dinamicamente. In domini maggiormente curati, gli amministratori potrebbero voler esprimere una conoscenza completa dei valori di X.

Data un semantica per le mapping table, la più semplice regola per combinarle è quella di trovare le possibili congiunzioni, cercando tutte le asso-

ciazioni che soddisfano tutte le tabelle. Si considerino le mapping table della tabella 3.1. Si riocorda che la tabella (b) indica delle coppie specifiche di geni e proteine che possono essere associate ad una malattia genetica. Si supponga che gli amministratori abbiano costruito queste tabelle usando la semantica open-world. Gli utenti potrebbero usare direttamente la tabella (c) per associare i geni alle malattie. Tuttavia, si potrebbero voler utilizzare le tabelle (a) e (b) (che potrebbero essere state costruite da altri amministratori) per ottenere dei collegamenti addizionali tra geni e malattie. Con una semantica open-world l'associazione (g1,m2) può essere ricavata, infatti si trova una riga *testimone* che coinvolge tutti gli attributi, con g1 come GDB_id e m2 come MIM_id. Questa riga è t=(g1,p2,m2). Si noti che t soddisfa la tabella (a) infatti contiene (g1,p2) e soddisfa la tabella (c) anche se g1 non è menzionato al suo interno. Si osservi che (g1,m1) non è un'associazione valida rispetto alle mapping table, poichè non ci sono righe testimone per quetsi valori (nessun valore di SwissProt soddisfa le condizioni viste precedentemente). Se uno o più tabelle avessero una semantica closed-world, tutte le associazioni tra GDB e MIM cambierebbero. E' stato sviluppato un algoritmo che deduce tutte le associazioni (alias) e determina se un gruppo di mapping table è inconsistente. In conclusione gli amministratori possono costruire le mapping table indipendentemente e, nello stesso tempo, fare uso delle conoscenze di altri amministratori attraverso delle query.

3.2 Coordinazione delle modifiche: le ECA rule

[3] Nel Peer-to-peer (P2P), l'amministrazione dei dati si sviluppa tramite query, aggiornamenti e processi transazionali. Ciò implica che i peers sono basi di dati. L'architettura di tali reti P2P consiste in un insieme di *peer database* posti nei nodi della rete. Ogni peer database è amministrato da un *peer data base managment system* (PDBMS). Solitamente un PDBMS ha un livello P2P che condivide la base di dati con gli altri peers. Prima che questo interfacciamento abbia inizio è necessaria un'*acquaintance* tra i peer databases, cioè un collegamento. Ogni base di dati stabilisce al massimo un'acquaintance con altri peer che fanno parte della rete e in questo modo entra a far parte di essa. I peer collegati sono chaimati *acquaintees*. Gli Acquaintees condividono e coordinano i dati attraverso le acquaintances. Esse non non sono fisse, quindi possono evolvere nel tempo, infatti i peer si collegano o lasciano la rete a loro discrezione. Un collegamento è stabilito tra dei peer che appartengono allo stesso gruppo di interesse. Il fine è una

comunità che descrive un certo campo (per esempio linee aeree, genetica, ospedali, università).

Si assume che il peer database non condivida uno schema globale o un'autorità amministrativa. Inoltre la reperibilità dei loro dati è strettamente locale.

Sintatticamente, il linguaggio dei peers include i normali costrutti per le interrogazioni e l'aggiornamento dei dati. Sono inclusi costrutti che permettono l'utente a stabilire e interropere acquaintances tra i peers. Semanticamente un collegamento uscente è formato da un sottoinsieme di mapping table ed espressioni, oltre ad un insieme di regole di coordinazione. Il primo permette che i dati siano scambiati attraverso le acquaintances, e l'ultimo coordina questi scambi.

In segiuto saranno descritti gli algoritmi usati dai PDBMS per stabilire e eliminare le acquaintances. Saranno dati alcuni esempi di regole di coordinazione, espresse come un'estensione delgli SQL3 triggers standard. Verranno anche considerati i costrutti sintattici usati per stabilire o abolire i collegamenti tra i peers. Infine sarà data un'esecuzione semantica di questi costrutti.

3.2.1 Esempi motivanti: mapping table ed espresioni

Si considerino i database di due peer per la prenotazione dei voli aerei Ontario-Air_DB e Quebec-Air_DB appartenenti a due compagnie aeree fittizie Ontario Air e Quebec Air, rispettivamente. Si assuma che questi database amministrino il prenotamento dei voli in partenza da Ottawa. I loro schemi sono illustrati nella figura 3.3.

Ontario-Air e Quebec-Air possono puntare al coordinamento delle loro attività stabilendo di un'acquantiance. L'obiettivo di tale collegamento può essere quello di scambiare delle informazioni sui voli e di coordinarli verso varie destinazioni. Siccome gli schemi dei due database sono differenti, i dati devono essere resi compatibili prima di essere scambiati. In un contesto più tradizionale , si utilizzano le views per facilitare lo scabio tra fonti eterogenee. Usualmente queste views forzano il contenuto dei peer database a riconoscere il contenuto di un database specifico. Tuttavia due caratteristiche basilari dei PDBMS precludono questo approccio: i peer database devono essere autonomi rispettando i loro contenuti, ed il grande numero di peers conivolti rende un approccio statico inadeguato. Piuttosto ci si focalizza su tecniche di coordinamento dello scambio dei dati tra i peers tramite la ricerca di metodi di vincolazione, non del contenuto dei peers, ma dello scambio di dati in se stesso. Per questo si utilizzano di mapping table espressioni, ed appropriate estensioni degli SQL3 triggers come uno strumento di controllo dello scambio

(a) Ontario-Air database

OA_Passenger

pid	name
1	Lachance
2	Smith
3	Moore

OA_Ticket

pid	fno	meal
1	OA229	Veal
2	OA378	Trout

OA_Flight

fno	date	dest	sold	cap
OA229	01/05	Mont.	120	256
OA341	01/15	Tor.	160	160
OA378	01/21	S.F.	90	124

(b) Quebec-Air database

QA_Passenger

pid	name
1	Michel
2	Lachance

QA_Fleet

aid	type	capacity
B-1	Bombard.27	140
B-2	Embraer L12	130
B-3	Boeing 737	117

QA_Flight

pid	fno
1	QA2132
2	QA1187
2	QA1109

QA_Reserve

fno	date	dest	sold	aid
QA2132	01/07	Dorv.	67	B-3
QA1187	01/15	LBP	118	B-2
QA1109	01/15	MDC	164	B-1

Figura 3.2: Istanze di due basi di dati di compagnie aeree

(a) Schema dell'Ontario-Air database

OA_Passenger (pid,name)
OA_Flight (pid,fno,meal)
OA_Ticket (pid, fno, meal)

(a) Schema del Quebec-Air database

QA_Fleet (aid,type,capacity)
QA_Passenger (pid,name)
QA_Flight (fno,date,to,sold,aid)
QA_Reserve (pid, fno)

Figura 3.3: Schemi dei database Ontario Air e Quebec Air

dei dati tra basi di dati eterogenee. La figura 3.4(a) mostra un esempio del primo costrutto, e le figure 3.4(b)-(c) mostrano esempi del secondo.

Non si scenderà nei particolari per quel che riguarda le mapping table, infatti queste sono già state descritte nella sezione precendente.

Le Mapping expressions mettono in relazione due basi di dati organizzate in modo differente. Esse fanno corrispondere gli schemi dei peer database. L'espressione della figura 3.4 (a) dice che, parlando di scambio di dati, alcuni voli di Quebec-Air sono considerati voli di Ontario-Air, e non necessariamente vice-versa. Così, si assume che le espressioni sono create nel database di Ontario-Air.

Una volta che due database sono collegati, e le mapping table oppure espressioni sono a posto, i peers possono usare i contenuti dell'altro durante le risposte a query e la coordinazione dei dati.

3.2.2 ECA Rules per PDBMS

Il linguaggio

Oltre che tramite le query, i peer sono in grado di coordinare i loro dati tramite le acquaintance. Per alcune applicazioni, una comunicazione a run-time non sarà sufficente e si vorrà sincronizzare i dati non appena saranno aggiornati. In questo esempio, si suppone che le due compagnie vogliano sincronizzare i dati secondo la mapping expression della figura 3.4 (a). Usando questa mapping expression, è possibile ricavare una regola che assicura che i due database rimangono consistenti quando vengono aggiunti dei nuovi

Mapping expression (a)
OA_Flight(fno,date,dest,sold) $\supseteq$ QA_Flight(fno,date,dest,sold,cap)

Mapping table(b)

dest	to
Tor.	LPB
Tor.	ILD
Mont.	Dorv.
Mont.	Mira.
Ottawa.	MDC

Mapping table (c)

OA.fno	QA.fno
OA229	QA2132
OA341	QA1187
OA341	QA1108

Figura 3.4: Mapping table ed espressioni

passeggeri nel peer Quebec-Air. Per rafforzare questa regola, e altre simili, si propone un meccanismo che utilizza ECA rule (regole evento-condizione-azione) con la distintiva caratteristica che eventi, condizioni e azioni nella regola si riferiscono a più peer. Unesempio di tale ECA rule è dato in figura 3.5.

```
create trigger passengerInsertion
after  insert on QA_Passenger
       referencing new as NewPass
for each row
begin
       insert into OA_Passenger values NewPass
       in Ontario-Air_DB;
end
```

Figura 3.5: Esempio di ECA rule per l'inserimento di un passeggero

In accordo con questa regola, un evento riscontrato nel Quebeck-Air database causa un'azione che deve essere eseguita nell'Ontario-Air database. Nello specifico, ogni passeggero inserito nella tabella QA_Passenger genera un evento che lancia la regola soprastante. Essa non ha condizioni e provoca l'inserimento di un passeggero identico nella tabella OA_Passenger.

La regola sopra è espressa in un linguaggio che estende gli SQL3 triggers. Un SQL3 trigger è una ECA rule che porta un nome esplicito. Gli eventi SQL3 sono semplici. Le condizioni sono query SQL, e le azioni sono dichiarazioni SQL che saranno escluse dal database. Ci sono due tipi di trig-

gers, chiamati AFTER e BEFORE triggers. I primi si attivano prima del verificarsi del loro evento scatenante, i secondi dopo l'evento scatenante.

Una regola estende un tipico trigger nominando esplicitamente il database nel quale l'evento, la condizione e l'azione si verificano. Esso contiene un ricco linguaggio per permettere l'espressione dei complessi ambienti attivi riscontrati nella coordinazione dei peer. La regola sopra è un esempio tipico di rafforzamento della consistenza tra peer che coordinano il loro lavoro.

Interazione con le Mapping Tables

La mappatura nelle mapping table è strettamente legata ai valori immagazzinati nell' Ontario-Air e nel Quebec-Air database. Ciononostante è naturale mantenere al loro interno solo le associazioni tra i valori che esistono nelle attuali tabelle di questi database. Questo significa che inserimenti e cancellazioni di valori all'interno di Ontario-Air e Quebec-Air possono portare ad un aggiornamento delle mapping table, per eliminare le discrepanze tra i database di entambi i peers e le loro relative mapping table.

Quando si verifica una cancellazione di una riga in OA_Flight, c'è un ECA rule che rimuove la riga appropriata nella mapping table della figura 3.4 (c). Ovviamente tale regola potrebbe non essere necessaria per molti casi di mapping table. Per esempio, per quel che riguarda la figura 3.4 (b), se avviene una cancellazione di tutti i voli dell'Ontario-Air verso New York, si potrebbe non voler cancellare le rispettive inforamzioni in Figure 3.4 (b), perchè probabilemte saranno utili in futuro. Nel caso della figura 3.4 (c), se un volo cancellato dall'Ontario-Air diventa attivo, potrebbe non corrispondere ancora con lo stesso volo di Quebec-Air. Perciò, eliminare la riga corrispondente dalla Figura 3.4 (c) quando un volo viene cancelato può essere ragionevole. Le regole della figura 3.6 illustrano come la mapping table della figura 3.4 (c) possa essere sincronizzata:

```
create trigger flightDeletion
after   delete on OA_flight
        referencing old as OldFlight
for each row
begin
        delete form MT3.4c where MT3.4c.fnoOA = OldFlight.fno
end
```

Figura 3.6: ECA rule per la cancellazione di un volo.

Qui, *MT3.4c* è il nome della relazione riferito alla mapping table della figura 3.4 (c).

3.2.3 Creazone ed Eliminazione di Acquaintance

Sintatticamente , il linguaggio dei peer include dei costrutti che permettono di stabilire o eliminiare delle acquaintance. Semanticamente , un'acquanintance è un'istanza che interfaccia due peers.

Stabilire Acquaintance

Per connettersi ad una rete , un peer N_i, deve creare un'acquaintance con un peer conosciuto, ciamato N_j, che fa già parte della rete. Si assume, per semplicità, che le acquaintance sono stabilite esplicitamente da un utente, molto probabilmente un amministratore di database. A questo fine il peer language include il costrutto della figura 3.7. Usando questo costrutto, un

```
set acquaintance to <peer database>
  [using mapping tables <lista di mapping table>]
  [using mapping expressions <lista di mapping expression>]
  [belonging to <Gruppo di Interesse>]
```

Figura 3.7: Costrutto per la creazione di un'acquaintance.

amministratore stabilisce delle acquaintance specifiche tra il proprio database ed un altro. Quando un'acquaintance è stata stabilita, viene accoppiata con alcuni vincoli. La cosa più importante di questi vincoli sono le mapping table e le espressioni, che pongono dei vincoli allo scambio di dati tra i peers, e le regole di coordinazione (scritte nel linguaggio ECA descritto sopra), che sono le linee guida per lo scambio.

Il peer N_i emetta il seguente comando per stablire un'acquaintance:
`set acquaintance to` N_j
Dopo di che il seguente algoritmo viene utilizzato per completare la richiesta:
Fase 1. Generazione semi-automatica dei mappamenti:

1. Uso di un algoritmo di abbinamento per avere un accoppiamento iniziale tra le tabelle dei peers N_i e N_j. Questo accoppiamento probabilmente non sarà corretto o completo, e potrebbe dover essere rivisto manualmente dall'amministratore.

2. Creazione di mapping expression e views.

3. Uso dell'abbinamento ottenuto al passo uno per creare e popolare un set iniziale di mapping table.

4. Invio una copia delle mapping table a N_j.

Fase 2. Generazione di regole di consistenza-rafforzamento delle mapping expression ottenute durante la prima fase, e generazione di regole per il mantenimento delle mapping table
Fase 3. Aggiunta di N_j alla lista delle acquaintance di N_j.

L'algoritmo non usa le mapping table iniziali. Per processare una richiesta con una lista iniziale M_1,...,M_n di mapping table è sufficente sostituire il passo 3 della prima fase, evitando l'uso delle mapping table ricevute. L'algoritmo non usa neanche le regole di coordinazione iniziali. Si può assumere che si usi una lista iniziale di regole di coordinazione nello stabilire un'acquaintance. Tuttavia, pensare questo non è precluso in principio, si preferisce pensare che una tale lista sia fornita da un gruppo di interesse, piuttosto che progettata per peer particolari da un progettista di database. La ragione principale per fare questo è che si vuole creare una tecnologia orientata all'utente finale.

Regole generiche per gruppi di interesse

Il semplige algoritmo descritto sopra considera che tutti i peer appartengono ad un singolo gruppo. Però, sarebbe meglio classificare i peer all'interno di gruppi di interesse. Si da per scontato che un gruppo di interesse abbia degli schemi standard conosciuti da tutti i membri. Nel campo delle compagnie aeree, per esempio, si potrebbe avere lo schema S_A per i peer database delle linee aeree, lo schema S_{TA} per le agenzie di viaggio,e lo schema S_{RA} per le compagnie aeree regionali. Le regole sono scritte per un comune modello di scambio dei dati e coordinazione tra S_A, S_{TA} e S_{RA}. Quando un particolare gruppo di peer, ciamato linea aerea a_1, agenzia di viaggi ta_1 e linea regionale ra_1 decide di coordinarsi tramite la regola R_1, essi devono legare/accoppiare i loro rispettivi schemi allo schema principale, secondo la definizione di R_1, così da trovare delle mappature appropriate. Con questi collegamenti, si crea un accordo tra a_1, ta_1, ra_1 che infatti vogliono coordinarsi tramite la regola R_1.

Quando i peers si collegano alla rete, devono registrarsi per essere sicuri di appartenere ad un gruppo di interesse. Si da per scontato che ci sono delle regole generiche create rispettando uno schema standard, esse sono adattate a database specifici cha appartengono al gruppo. infatti, una regola generica potrebbe coinvolgere gli schemi di più gruppi. Per semplicità, si assume che gli schemi standard e le regole generiche sono conosciute da ogni membro di un gruppo di interesse.

Come esempio, sarà illustrato passo passo un meccanismo per impostare un'acquintance in presenza di un gruppo di interesse. Sarà usata la seguente terminologia. Uno schema di mappamento è una corrispondenza tra schemi appartenenti a due peer. Un mappamento viane chaimato "rigoroso" se rappresenta una funzione di identità che mappa ogni valore di un attributo A nel primo peer in se stesso nel seconodo peer. Per esempio i valori dell'attributo "data" nell'Ontario-Air database vengono mappati in se stessi, senza modifiche, nel database di Quebec-Air. Un mappamento viane chiamato - sciolto se rappresenta delle funzioni che mappano valori differenti dello stesso attributo. Infine una view ha lo stesso significato usato solitamente.

Si prendano in considerazione le tabelle di Ontario-Air e Quebec-Air date nella sezione 3.2.1 e che entrambi appartengono al gruppo di interesse *Airlines*.

Passo 1. Il gruppo *Airlines* usa il seguente schema, chiamato S_A:

Ticket(pid,name,fno,meal)
Flight(fno,date,destination)
FlightInfo(fno,sold,cap,aid,type)

Per Onatario-Air , si hanno le seguenti mapping table $M_{S_A \rightarrow OA}$:

M_{OA_1}: OA_Passenger(pid, name) ← Ticket(pid,name,fno,meal)
M_{OA_2}: OA_Ticket(pid, fno, meal) ← Ticket(pid,name,fno,meal)
M_{OA_3}: OA_Flight(fno, date, dest, sold, cap) ← Flight(fno,date,dest),
FlightInfo(fno,sold,cap,aid,type)

Dalle tre mapping table sopra, M_{OA_2} è di tipo sciolto (perchè l'amministratore del database suppone che mappare fnos potrebbe non essere preciso, infatti usualmente le linee aeree usano numeri di voli differenti). Anche M_{OA_3} e di tipo sciolto (perchè l'amministratore sospetta che differenti compagie considerino diverse le destinazioni). Infine M_{OA_1} è un mappamento rigoroso.
Per Quebec-Air, ci sono le seguenti mapping table $M_{S_A \rightarrow QA}$:

M_{QA_1}: QA_Passenger(pid,name) ← Ticket(pid,name,fno,meal)
M_{QA_2}: QA_Reserve(pid,fno) ← Ticket(pid,name,fno,meal)
M_{QA_3}: QA_Flight(fno,date, to, sold, aid) ← Flight(flight,fno,date,to),
FlightInfo(fno, sold, cap, aid, type)
M_{QA_3}: QA_Fleet(aid,type,capacity) ← FlightInfo(fno,sold,cap,aid,type)

M_{QA_2} e M_{QA_3} sono mapping table del tipo sciolto e M_{QA_1} e M_{QA_4} rigorose.

Passo 2.

A. Si provano a definire delle **views** tra le tabelle di Ontario-Air e Quebec-Air. Si osservi che da $M_{S_A \rightarrow OA}$ e $M_{S_A \rightarrow QA}$ i mappamenti M_{OA_1} e M_{QA_1} sono definiti sullo stesso set di attributi. Perciò si crea la view:

V1: OA_Passenger(pid, name) ← QA_Passenger(pid, name)

Anche gli attributi delle mapping table M_{OA_2} e M_{OA_2} si sovrappongono. Perciò si mappa quello con più attributi verso quello con meno attributi usando la seguente view:

V2: QA_Reserve(pid, fno) ← OA_Ticket(pid, fno, meal)

Infine dalle tabelle M_{OA_3}, M_{QA_3}, M_{OA_4}, si ricava la view:

V3: OA_Flight(fno, date, dest, sold, cap) ←
QA_Flight(fno, date, dest. sold, aid), QA_Fleet(aid, type, cap)

dove "to" è stato rinominato in "dest" e "capacity" come "cap".

B. A questo punto si definiscono le **mapping table** tra Ontario-Air e Quebec-Air, usando le views V1-V3. V1 deriva da un mappamento rigoroso. Perciò può essere usato così com'è. V2 invece deriva da due mappamenti di tipo sciolto. In questo caso c'è bisogno di informazioni aggiuntive per poter usare questa views: occorre un mappamento di valori per riparare la mancanza dell'attributo "fno" Per questo si crea un mapping table MT_1(OA.fno, QA.fno). V3 deriva da uno schema sciolto, mancano gli attributi "fno" e "dest". Per "fno" è già stato creata una mapping table. Per "dest" e "to" viene creata la mapping table MT_2(OA.dest, QA.to).

Passo 3. A questo punto vine chiesto all'utente/amministratore se vuole creare delle mapping expression per ciascuna view od alcune nuove da zero. Si supponga che l'utente crei un'espressione da V1:

ME: OA_Passenger(pid, name) $\supseteq$ QA_Passenger(pid,name)

Come visto nella sezione 3.2.1, la mapping expression ME ha un significato esecutivo e non dice nulla sulla struttura della relazione.

Passo 4. L'utente riempie le mapping table com visto in figura 3.1

Passo 5.

A. Usando la mapping expression ME, viene creata la regola della figura 3.8 per assicurare la consistenza dell'acquaintance tra i databse di Ontario-Air e Quebec-Air.

B. E' necessario creare una regola per il mantenimanto delle mapping table MT_1 e MT_2. Generalmanete l'amministratore deve decidere che tipo di regole sono necessarie per la manutenzione di una mapping table, in accordo con il tipo di informazioni che essa contiene. Ogni volta che vine cancella ta una riga in OA_Flight, c'è una regola che rimuove la corrispondente riga in MT_2. La regola flightDeletion data nella figura 3.6 è adatta a questo scopo.

Passo 6. Si prenda come esempio in seguente ambiente con due peer conivolti: DB_A e DB_B sono impostati in modo che quando un volo di DB_A

```
create trigger enforceME
after   insert on QA_Passenger
        referencing new as New in Quebec-Air_db
for each row
begin
        insert into OA_Passenger values (New.pid, New.name)
        in Ontario-Air_DB
end
```

Figura 3.8: Regola per l'inserimento di una passeggero.

```
create trigger AFullFlight
before update of sold on FlightInfo
        referencing new as New
        referencing old as Old in DB_A
when  New.sold = New.cap
for each row
begin
        insert into Flight values (New.fno, date, New.destination);
        insert into FlightInfo values (New.fno,o, New.cap, New.aid,
             New.type)
end
```

Figura 3.9: ECA rule che inserisce un nuovo volo.

è completo, un nuovo volo viene creato da DB_B per accogliere nuovi passeggeri. La regola generica della figura 3.9 ha questa funzione. Quando essa viene personalizzata per i database di Ontario-Air e Quebec-Air, abbiamo la regola della figura 3.10. Si noti che la parte attiva della regola contiene un'in-

```
create trigger OAAFullFlight
before update of sold on OA_Flight
        referencing new as New
        referencing old as Old in Ontario-Air_DB
when  New.sold = New.cap
for each row
begin
        insert into QA_Flight
        values (map(New.fno), date, map(New.destination), 0, null)
        in Quebec-Air_DB
end
```

Figura 3.10: ECA rule che inserisce un nuovo volo usando le mapping table per tradurre i valori.

serimento di una nuova riga, che è una trasformazione di quella modificata nei voli di Ontario-Air. Questa trasformazione è attuata tramite le mapping table.

Passo 7. Quebec-Air ora appartene alla lista delle acquaintance di Ontario-Air e viceversa.

Ora saranno dati i dettagli dell'algoritmo che stabilisce le acquaintance in presenza di un gruppo di interesse. Si supponga che il peer N_i emetta i seguenti comandi:

`set acquaintance to` N_i
`belonging to` SIG

Il seguente algoritmo completa la richiesta:
Se N_j non appartiene al gruppo SIG, allora si segua il semplice algoritmo della sezione 3.2.3, altrimenti si faccia come segue:

1. Per entrambi N_i e N_j, se non sono già presenti, creare i mappamenti tra la tabella standard S_{SIG} di SIG e le tabelle attuali S_{N_i} di N_i e S_{N_j} di N_j.

2. Date le mapping table $M_{i \to SIG}$ da S_{N_i} a S_{N_j}. Usare $M_{i \to j}$ per creare mapping table e views tra N_i eN_j.

3. Creare le mapping expression.

4. Riempire le mapping table, e inviare una copia delle loro istanze a N_j

5. Generare le regole di consistenza-rafforzamento per le mapping expression ottenute al passo 2, e generare le regole per la manutenzione delle mapping table.

6. Personalizzare e attivare le regole generiche di *SIG*, in accordo con le views e le mapping table definitive ottenute al passo 2.

7. Aggiungere N_j alla lista delle acquaintance di N_i.

Eliminazione di Acquaintance

Un peer N_i potebbe voler eliminare una o più aquaintances con altri peer conosciuti, chiamati N_{j_1},...,N_{j_l}. Per semplicità, si assumerà che le acquaintances sono terminate da un utente. A questo scopo il linguaggio dei peer include il seguente costrutto:
`abolish acquaintance to <peer database>`
Quando un'acquaintance viene eliminata, le varie costrizioni che essa implica vengono cancellate, cancellarle può essere semplice com disabilitarle localmente. Tuttavia, se il peer ha solo un'acquaintance, eliminarla può indurre il peer a lasciare la rete. Perciò cancellare una costrizione può essere difficile come riempire il vuoto laciato dal peer mancante. Il costrutto `leave` permette di eliminare un'acquaintance.

In seguito saranno dati i dettagli sull'algoritmo usato da un peer per eliminare un'acquaintance. Il peer N_i lancia il seguente comando:
`abolish acquaintance to` N_j
Che viene eseguito nel seguente modo:
Se N_j non è l'unica acquaintance di N_i, eseguire i seguenti passi

1. Inviare un messaggio a N_j per disabilitare le copie delle mapping table che potrebbero essere state generate da N_i.

2. Disabilitare le mapping table, mapping expression, e coordination rule che si riferiscono all'acquaintance ¡N_i, N_j¿.

Altrimenti eseguire i seguenti passi

1. Inviare un indicatore a N_j.

2. Seguire i passi 1 e 2 descritti sopra.

Il comando `leave` è eseguito per eliminare tutte le acquaintance usando l'algoritmo sopra.

Si noti che l'"indicatore" menzionato nell'algoritmo è un'informazione che l'ultimo peer che è a conoscenza di un'altro in scollegamento tiene per avere traccia della sua appartenenza alla rete. Questo indice può essere usato per decidere cosa fare quando un'interrogazione ha bisogno di una risposta o un evento deve essere eseguito e alcuni peer hanno lasciato la rete.

3.3 Mapping table ed ECA rule in Hyperion

Finora si è parlato di mapping table ed ECA rule in modo abbastanza teorico, ora si vedrà come sono stati usate in pratica queste tecniche nello sviluppo di Hyperion. I database utilizzati si riferiscono a compagnie aeree e sono i seguenti: dairfrance, dalitalia, dklm e dklmcityhopper.

3.3.1 Mapping Table

Le mapping table sono usate per tradurre delle query dal database locale a quello remoto. Come si è già visto le vengono create da un amministratore del sistema, e in questo caso hanno le seguenti caratteristiche. Il loro nome è del tipo *mt_tabellaLocale2tabellaRemota_numero*, che significa:

- *mt_*: è il prefisso che indica che si tratta di una mapping table.
- *tabellaLocale*: è il nome della tabella locale, la quale contiene i valori da tradurre.
- *2*: è una parola chiave che indica la direzione in cui avviene la traduzione, si legge come "to" che in inglese sigifica verso.
- *tabellaRemota*: è il nome della tabella che si trova sul database remoto.
- *_numero*: Ogni attributo o gruppo di attributi di una tabella ha una rispettiva mapping table, con un numero di sequenza diverso.

Ci sono due tipi di attributi in una mapping table, gli attributi locali, che indicano il valore relativo alla tabella locale sono contraddistinti dal prefisso *l_* e gli attributi remoti, che indicano il valore corrispondente nella tabella remota e sono contraddistinti dal prefisso *r_*.

Per ogni peer le corrispondenze tra tabelle locali e remote che possono essere usate nel processo di traduzione devono essere specificate dall'amministratore. A questo scopo c'è una tabella chiamata *mt_corres* che contiene i seguenti attributi:

- *rpeer*: il nome del peer remoto per il quale si vuole specificare una mapping table

- *mapping_table*: Il nome della mapping table usata.

Solamente le mapping table contenute in questa tabella possono essere usate nel processo di traduzione.

mt_afvol2klflight_0

l_volid	**l_voldate**	**r_fid**	**r_fdate**
AF8345	2007-12-11	KL1065	2007-12-11
AF2346	2008-06-13	KL2346	2008-06-13

mt_afvol2klflight_2

l_a	**r_flto**
MXP	Milan
DUS	Dusseldorf
BLQ	Bologna

mt_afvol2klflight_5

l_vendu	**r_sold**
-1	-1

Figura 3.11: Mapping table relative alla tabella dei voli del database dairfrance verso dklm, la prima traduce l'attributo della destinazione, la seconda al numero di biglietti venduti

L'algoritmo di traduzione usa le mapping table nel seguente modo: prendendo come esempio la tabella *mt_afvol2klflight_0* della figura 3.11, *l_volid* e *l_voldate* sono gli attributi locali, che indicano l'ID e la data di un volo, rispettivamente *r_fid* ed r_fdate sono gli attributi della tabella remota. Se la query da tradurre contiene i valori locali *(AF8345,2007-12-11)*, l'algoritmo li tradurrà in *(KL1065,2007-12-11)*. Osservando la mapping table *mt_afvol2klflight_5* si nota che contiene solo i valori *(-1,-1)*, ciò sta a significare che il valore presente nella query da tradurre rimarrà invariato dopo la traduzione, questa viene chiamata la semantica "-1". *(-1,-1)* sarà sempre in fondo alla tabella, infatti gli eventuali accoppiamente presenti sopra saranno valutati prima, successivamente se non sarà trovata nessuna corrispondenza sarà applicata questa regola. Se al posto di *(-1,-1)* ci fosse *(-1,value)*, ciò starebbe a significare che qualsisi valore passato all'algoritmo esso restituirà "value".

Lalgoritmo non funziona se nelle mapping table è contenuto un numero diverso di valori locali e remoti, ad esempio con una tabella di questo tipo l'algoritmo non funzionerebbe correttamente: *MT(l_A,l_B,r_A)*.

3.3.2 Policy Service

Il *policy service* è la parte di Hyperion che si occupa della modifica del database. Le modifiche sono effettuate tramite degli eventi SQL, che sono *insert*, *delete* e *update*. Nel caso di una modifica, ci saranno delle policy (norme) che descrivono le azioni che il peer locale e remoto devono eseguire. Esse sono composte da un insieme di ECA rule. Per esempio, se l'amministratore volesse inserire una riga in una tabella locale ed avvisare gli altri peers di questo, potrebbe definire una policy di inserimento che invii un'ECA rule per ogni peer di un determinato gruppo. Essa specificherà se l'inserimento deve essere propagato sulle mapping table, verso quali peers inviare le modifiche e le mapping table o altre informazioni che potrebbero essere usate nella traduzione della riga locale in quella remota.

Le tabelle *"__policy"*, *"__policymt"* e *"__default"* contengono i dati necessari ad un peer per processare localmente una modifica, tradurre la riga cambiata nella tabella locale in una da inviare ad una tabella remota e propagare le modifiche attraverso la rete.

La tabella *__policy* fornisce i parametri che indicano come deve comportarsi il peer durante una modifica nel database, contiene i seguenti attributi:

- *policyID*: è l'id che identifica uno specifico evento.
- *tableName*: il nome della tabella locale in cui avviene l'evento.
- *eventName*: il numero che identifica il tipo di evento: 0 per INSERT, 1 per DELETE e 2 per UPDATE.
- *MTid*: l'id che identifica il set di mapping table usate dall'algoritmo per questa specifica ECA rule.
- *actionRT*: specifica se l'evento debba o meno essere propagato attraverso la rete: con 0 non viene propagato, con 1 sì.
- *tableRT*: il nome della tabella remota verso la quale l'evento deve essere propagato.
- *peerRT*: il nome del peer remoto verso il quale l'evento deve essere propagato.

La tabella *"__policymt"* contiene i nomi del gruppo di mapping table che servono alle policy definite nella tabella *"__policy"* e la decisione di propagare o meno le modifiche per ogni attributo a cui fanno riferimento. Questa tabella è formata dai seguenti attributi:

- *id*: è uguale al valore contenuto nell'attributo *MTid* della tabella *__policy*.
- *MT*: il nome della mapping table che fa parte del set usato da una data policy.
- *actionMT*: può assumere i valori 0 e 1, il primo indica che una modifica apportata all'attributo a cui fa riferimento la mapping table non viene propagata attraverso la rete, il secondo che viene propagata.

In casi particolari una mapping table potrebbe non essere sufficente a tradurre un valore, perciò sarà necessaria una funzione scritta dall'amministratore del sistema (*mapping expression*) che specifichi come deve essere propagato un valore. Si prenda come esempio l'inserimento di un volo da parte di Airfrance, e che questo inserimento debba essere propagato verso KLM. Il codice del volo sarà diverso da una compagnia ad un'altra. L'amministratore di Airfrance inserirà nel database il volo con un codice a sua scelta, precedentemente avrà scritto una funzione che stabilisce come deve essere creato l'id del volo per KLM, l'algoritmo non consulterà la mapping table, ma eseguirà questa funzione. La tabella *__default* serve a specificare al posto di quali mapping table è necessario chiamare la funzione, i suoi attributi sono i seguenti:

- *MTtable*: indica il nome della mapping table
- *remAtt*: indica il nome dell'attributo remoto della mapping table che deve essere tradotto.
- *fun*: indica il nome della funzione, scritta dall'amministratore, che traduce il valore

Capitolo 4

L'aggiornamento consistente dei dati

4.1 Il problema nell'aggiornamento

Prima del cambiamento, l'algoritmo di propagazione delle modifiche attraverso la rete, svolgeva i propi compiti tramite due istruzioni del linguaggio SQL: INSERT e DELETE. Queste due opearzioni permettono rispettivamente l'insermento di nuove righe nel database e la loro cancellazione. Nel caso in cui si volesse modificare una riga già presente in uno o più campi tramite questi due comandi bisognerebbe prima cancellarla e poi inserirla con i nuovi valori. Questo non presenterebbe problemi, se non eventualmente di prestazioni, se l'operazione avvenisse in un database locale. Nel caso di Hyperion questa operazione deve essere effettuata in una rete di computer, e necessiterebbe di due messaggi separati, uno per ognuna delle due operazioni. Questo potrebbe generare dei problemi nel caso in cui ci fossero dei malfunzionamenti nella rete e l'operazione venisse effettuata solo in parte.

Si prenda come esempio il caso in cui un peer che sarà chiamato P1 volesse aggiornare il numero di passeggeri di un dato volo e debba propagare questa modifica verso P2. Durante l'invio dei messaggi avviene un problema, ed P1 rimane scollegato dopo aver inviato solamante il messaggio di DELETE. P2 riceve il messaggio e elimina il volo indicato da P1. Dopo di che entra in azione un'altro peer , chiamato P3, anche'esso necessita di modificare un dato nello stesso volo, ma una volta inviati i due messaggi non trova la riga prestabilita, e perciò non porta a termine l'operazione. Solo più tardi, quando la connessione sarà ristabilita P1 sarà in grado di completare l'aggiornamento, ma P3 non avrà potuto effettuare l'operazione.

Per ovviare al problema si è pensato di svolgere l'aggiornamento di una

riga del database con un'unica operazione, in modo da evitare i problemi derivanti dall'invio di due messaggi distinti. Questo si è ottenuto usando l'istruzione UPDATE di SQL. Tra l'altro questa estensione permette di eseguire l'operazione in modo più pulito ed efficente.

Considerando l'esempio sopra, si può constatare che con questa estensione l'algoritmo funzionerebbe meglio, infatti l'aggiornamento verrebbe completato oppure non verrebbe effettuato per nulla, permettendo a P3 di svolgere i propri compiti.

4.2 Un algoritmo per l'aggiornamento consistente

In questa sezione si spiega il funzionamento dell'algoritmo di aggiornamento dei dati, in segiuto verrà dato anche un esempio pratico.

Il funzionamento dell'algoritmo può essere diviso in quattro fasi: ricezione della richiesta, traduzione dei dati, aggiornamento del database locale e propagazione dei dati attraverso la rete. L'evento che avvia le operazioni è un messaggio inviato da un utente oppure da un peer remoto. Nel caso di una modifica di una riga nel database esso deve contenere i valori della vecchia riga, quelli che andranno ad aggiornarla, il nome della tabella e un identificatore che specifichi il tipo di evento da esegiure, in questo caso si tratta dell'evento UPDATE. Sotto saranno spiegate in dettaglio le varie fasi con l'ausilio di alcuni spezzoni di pseudocodice.

4.2.1 Ricezione della richiesta

In seguito alla ricezione della richiesta viene creato un servizio, che come visto in precedenza, avvia l'handler che si occupa del suo processamento. I dati ricevuti tramite il messaggio sono salvati un vettore, vengono estratti e salvati separatamente:

- *tablename:* è il nome della tabella su cui si andranno a fare le modifiche.
- *oldtuple:* è un vettore contenente i valori del campo da modificare nella tabella.
- *newtuple:* è un vettore contenente i nuovi valori, che andranno a sostituire quelli presenti nella tabella.
- *tuple_metadata:* è un oggetto che contiene due vettori, uno con gli attributi della tabella interessata e l'altro con i tipi di ogni attributo.

Ossia, il primo indica il nome delle colonne della tabella, il secondo se i valori contenuti per ogni colonna sono numeri, stringhe o altro.

- *arrayOfECARule:* è un vettore, che contiene le eventuali ECA rule associate alla tabella data.

Questa operazione viene svolta in modo differente a seconda del tipo di richiesta, che può essere locale o remota e può trattare un caso di inserimento, cancellazione o aggiornamento di una riga nel database. Infatti per ognuno di questi casi ci sarà bisogno di diverse informazioni. Per esempio nella cancellazione di una riga sarà necessario il vettore *oldtuple* e non il *newtuple*, mentre nel caso dell'aggiornamento ci saranno usati entrambi. Le informazioni sul tipo di richiesta si trovano nelle prime posizioni del vettore, esse non vengono estratte, ma vengono utilizzatoe solo inizialmente durante l'avvio del procedimento.

4.2.2 Traduzione dei dati

La traduzione dei dati avviene se ci sono delle ECA rule che lo specificano, altrimenti l'algoritmo salta subito al passaggio successivo, cioè l'aggiornamento delle tabelle locali.

I dati da tradurre sono quelli presenti in *oldtuple*, *newtuple*, *tablename* e in *tuple_metadata*. I valori vengono tradotti tramite delle query sulle mapping table, che ad ogni valore della tabella locale pongono uno della tabella remota. In questa fase infatti non si fa altro che leggere le mapping table oppure le funzioni predisposte dall'amministratore.

Una volta ultimata la traduzione i dati devono essere preparati ad essere inviati tramite un messaggio, per questo viene creato un vettore simile a quello ricevuto nella fase precedente: ad ogni posizione del vettore viene aggiunta la relativa informazione, in modo ordinato.

4.2.3 Aggiornamento del database locale

Dopo aver preparato i dati, se c'è bisogno di farlo, si aggiornano le mapping table, eseguendo una query sul database locale. L'aggiornamento delle mapping table si fa in casi in cui il valore modificato vada ad interessarle. Prendiamo ad esempio il caso in cui il contenuto di una mapping table sia il codice di un volo assieme alla sua data. Modificando la data nel database dei violi, successivamente bisognerebbe modificarla anche nella mapping table, altrimenti non ci sarebbe più un riscontro di quel determinato volo e non si potrebbe più tradurne il codice.

In seguito si aggiorna la tabella locale nel database. L'aggiornamento viene effettuato tramite il comando UPDATE del linguaggio SQL, esso viene generato dall'algoritmo prendendo i valori da *tablename*, *oldtuple*, *newtuple* e *tuple_metadata*. La query è del tipo `UPDATE nome_tabella SET attributo= valore,... WHERE attributo= valore AND...` , che significa: aggiorna la tabella ponendo per i seguenti attributi il rispettivo valore, modificando le righe che contengono i valori dati. I dati necessari alla parte `SET` vengono presi dal vettore *newtuple* mentre quelli della parte `WHERE` da *oldtuple*. Tutti gli attributi vengono confrontati nella clausola `WHERE`, infatti pensando in modo generale, non sempre una tabella potrebbe avere una chiave primaria per identificare univocamente una riga.

4.2.4 Propagazione attraverso la rete

L'ultimo passaggio rapprsenta la propagazione delle modifiche attraverso la rete, come si è visto ciò avviene solo se ci sono delle ECA rule che indicano di farlo. in questa fase viene preso il vettore creato dopo la traduzione dei dati, e viene predisposto un messaggio per ogni peer remoto verso il quale è stato stabilito di inviare le modifiche. Questi messaggi sono posti in coda in un vettore, se tra i peer esiste un'acquaintance i messaggi vengono spediti subito, in caso contrario il messaggio viene conservato nella coda, in attesa che avvenga un collegamento con il peer remoto.

4.3 Esempio Pratico

Ora sarà descritto il procedimento che segue una richiesta di update della riga *(AF8345,2007-10-11,AMS,CVL,F70,continental,33)* presente nella tabella *airFranceVoli* del database del peer AirFrance nella riga *(AF8345,2007-12-11,AMS,CVL,F70,continental,33)*. Si specifica che nelle varie regole del peer è indicato che un aggiornamento su questa tabella debba essere propagato verso il peer KLM.

airFranceVoli

volid	**voldate**	**da**	**a**	**aid**	**typevol**	**vendu**
AF8345	2007-10-11	AMS	CWL	F70	continental	33

Figura 4.1: Riga di esempio della tabella *airFranceVoli* nel database del peer AirFrance.

I passaggi svolti dall'algoritmo sono i seguenti:

mt_airFranceVoli2KLMflight_0

l_volid	l_voldate	r_flid	r_fldate
AF8345	2007-10-11	KL1065	2007-10-11

mt_airFranceVoli2KLMflight_1 e 2mt_airFranceVoli2KLMflight_3

l_a	r_flfrom
CWL	Cardiff
AMS	Amsterdam

l_aid	r_aid
F70	F70

mt_airFranceVoli2KLMflight_4

l_typevol	r_typefl
continental	regular line

mt_airFranceVoli2KLMflight_5

l_vendu	r_sold
-1	-1

Figura 4.2: Valori delle mapping table utilizzate dall'algoritmo per tradurre una riga della tabella airFranceVoli del database AirFrance (le tabelle 1 e 2 sono uguali, a parte gli attributi che sono *r_flfrom* per la 1 e *r_flto* per la 2, quindi sono state scritte una volta sola).

- Viene lanciato l'Handler, che salva i dati passati dalla richiesta in dei vettori e successivamente inizia il processamento.
- L'algoritmo traduce i dati usando le mapping table della figura 4.2, i vettori degli attributi e della nuova riga diventano rispettivamente *(flid, fladate, flfrom, flto, aid, typefl, sold)* e *(KL1065, 2007-12-11, Amsterdam, Cardiff, F70, continental, 33)* alposto di *(volid, voldate, da, a, aid, typevol, vendu)* e *(AF8345, 2007-12-11, AMS, CVL, F70, continental, 33)*.
- Viene modificata la mapping table *mt_airFranceVoli2KLMflight_0* prendendo gli attributi e valori remoti dai vettori appena tradotti, infatti in questo caso è necessario cambiare la data. Il comando SQL per tale aggiornamento è illustrato nella figura 4.3. si noti che nella clausola

```
UPDATE mt_airFranceVoli2KLMflight_0
SET    l_volid='AF8345', l_voldate='2007-12-11',
       r_flid='KL1065', r_fladate='2007-12-11'
WHERE  l_volid='AF8345' AND l_voldate='2007-10-11'
```

Figura 4.3: Comandi SQL per l'aggiornamento di un campo in una mapping table

where sono considerati solo gli attributi locali, infatti in questa parte

dell'algoritmo si agisce solo localmente, quindi non si ha accesso alle tabelle del peer remoto dove sono contenuti i dati relativi a data e id remoti che andrebbero confrontati nella query. A questo punto viene aggiornata la tabella locale tramite il seguente codice SQL:

```
UPDATE airFranceVoli
SET    volid='AF8345', voldate='2007-12-11', da='AMS',
       a='CWL', aid='F70', typevol='continental', vendu='33'
WHERE  volid='AF8345' AND voldate='2007-10-11' AND da='AMS'
       AND a='CWL' AND aid='F70' AND typevol='continental'
       AND vendu='33'
```

Figura 4.4: Comandi SQL per l'aggiornamento di un campo della tabella *airFranceVoli*

- Finite le operazioni sul database locale l'algoritmo controlla la tabella *__policy* per verificare la necessità di propagare le modifiche: in questo caso è necessario. Il messaggio, contenente i valori tradotti dall'algoritmo, viene messo in coda nel vettore che contiene le policy da inviare, in attesa di un'acquaintance con il peer remoto. Dopo aver verificato che il peer remoto è collegato la richiesta viene spedita subito. Nel caso in cui il collegamento venisse stabilito solo in un secondo momento, all'atto della creazione dell'acquaintance avverrebbe un controllo sulle eventuali richieste da inviare.

Capitolo 5

Test

Finita l'implementazione dell'estensione sono stati fatti dei test per verificarne il funzionamento ed eventuali problemi intrinseci dell'algoritmo. I casi più importanti da verificare sono stati sulla consistenza dei dati dopo delle propagazioni su reti con cicli al loro interno e su aggiornamenti multipli effettuati da peers differenti. In questo capitolo saranno esposti i vari test con realtivi effetti sul database.

5.1 Test 1: propagazione di una richiesta

Il primo test sarà utile a capire come si effettua una simulazione in Hyperion e come vengono preparate le mapping table e le varie tabelle relative alle policy. Questo test riprenderà l'esempio della sezione 4.3. Come si vede nella figura 5.1 la richiesta andrà a modificare la data di un volo su AirFrance, e successivamente propagherà la modifica verso KLM. Per prima cosa si illustrerà come va preparato il test, sucessivamente si verificheranno le modifiche avvenute nei database ed infine si valuterà se queste sono avvenute correttamente.

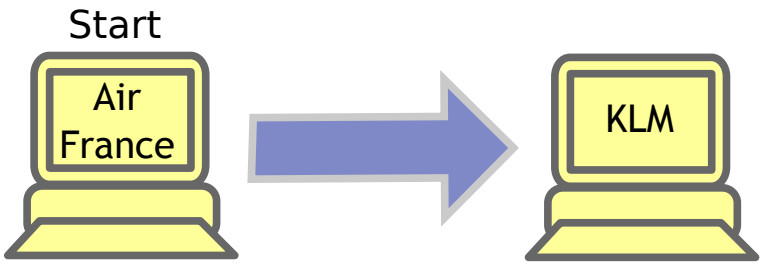

Figura 5.1: Propagazione della richiesta nel Test 1.

L'obiettivo del test è quello di modificare una riga nella tabella *afvol(volid, voldate, de, a, aid, typevol, vendu)* di AirFrance e di propagarlo verso la tabella *klflight(flid, fldate, flfrom, flto, aid, typefl, sold)*.

5.1.1 Preparazione delle mapping table

La prima cosa da fare è quella di preparare le mapping table che permettano di tradurre i dati. si noti che un set di mapping table è unidirezionale, quindi volendo creare un collegamento bidirezionale sono necessari due set di mapping table. Per questo specifico caso si necessita di un solo set, da AirFrance a KLM. le mapping table necessarie a questo test sono quelle presenti nella figura 4.2. Esse andranno riempite aggiungendo i possibili valori che può assumere ogni attributo del database locale, in corrispondenza con quelli relativi al database del peer remoto. Il significato del loro nome e della struttura sono stati spiegati nella sezione 3.3.1. Prendendo come esempio la tabella *mt_airFranceVoli2KLMflight_4* si vede che un volo di tipo *continentale* nel peer AirFrance corrisponde ad un volo *regular line* di KLM, mentre nel caso in cui non ci sia differenza tra i valori dei due peer si agische come nella tabella mt_airFranceVoli2KLMflight _5 utilizzando la -1 semantica.

5.1.2 Preparazione delle mapping expression

Successivamente bisogna verificare non ci siano dei valori che non possono essere bene espressi dalle mapping table, e che quindi necessitino di mapping expression, che a livello di implementazione non sono altro che delle funzioni. In questo caso si vede che i dati contenuti nella tabella *mt_afvol2klflight_0* non sarebbero tradotti correttamente, infatti la traduzione avviene prima dell'aggiornamento sul database, e quindi la mapping table risulterebbe non aggiornata al momento della traduzione. Perciò bisogna stabilire le regole adeguata nella tabella *__default(MTtable, remAtt, fun)* e definire le funzioni. Del significato dei suoi attributi si è parlato nella sezione 3.3.2. Le righe da inserire sono le seguenti: (mt_afvol2klflight_0, r_flid, Fkflightflid) e (mt_afvol2klflight_0, r_fldate, Fkflightfldate), quindi bisogna definire le rispettive funzioni scritte in java. La funzione in figura 5.2 che si occupa della traduzione della data, essa non fa altro che prendere la data dettata dall'utente, contenuta nel vettore *args*, e restituirla all'algoritmo. La funzione contenuta nella figura 5.3 si occupa della traduzine dell'ID del volo, questo non viene modificato rispetto alla mapping table, quindi è sufficente leggerlo da essa, tramite un'interrogazione del database e passarlo all'algoritmo. Perchè le funzioni vengano considerate è necessario che siano invocate dalla classe *MappingExpressioEngine*, rappresentata in figura 5.4, questa classe ha

```
public class Fkflightfldate {
      public static Object apply(Connection conn,
            String nameOfReturnValue, Objects[] args, Object[] types)
            Object value = new Object[];
            String s = (String) args[1];
            return value;
      }
}
```

Figura 5.2: Funzione java che restituisce la data relativa ad un volo.

```
public class Fkflightflid {
      public static Object apply(Connection conn,
            String nameOfReturnValue, Objects[] args, Object[] types)
            String sqlquery = ‘‘select r_flid from mt_afvol2klflight_0
                  where l_volid=’ ’’ +args[0]+‘‘ ’ ’’ ;
            Object value = new Object[];
            ResultSet result = DB.executeQuery(sqlquery, conn);
            while (result.next())
                  value = result.getObject(1);
            result.close;
            return value;
      }
}
```

Figura 5.3: Funzione java che restituisce l’id relativo ad un volo.

```
public class MappingExpressionEngine {
      public static Object run (Connection conn, String fname,
      String nameofReturnValue, Object[] args, Object[] types){
            if(fname.equals(‘‘Fklflightfldate’’))
                  return Fklflightfldate.apply(conn, nameOfReturnValue,
                        args, types);
            if(fname.equals(‘‘Fklflightflid’’))
                  return Fklflightflid.apply(conn, nameOfReturnValue,
                        args, types);
            else throw new RuntimeException(
                  ‘‘cannot run the function ’’+fname);
      }
}
```

Figura 5.4: Classe java che avvia le funzioni chiamate dal programma.

l'unico scopo di avviare la funzione, se non è presente invia un messaggio di errore.

5.1.3 Preparazione delle policy

Per specificare al peer come comportarsi durante una richiesta si è visto che esiste la tabella *__policy(policyID, tableName, eventName, MTid, actionRT, tableRT, peerRT)*. Quindi sarà necessario aggiungere una riga con i parametri relativi ad un update dalla tabella afvol a klflight. La riga da inserire è *1,afvol,2,1,klflight,KLM*. Come visto nella sezione 3.3.2, ciò informa il peer che deve essere propagata una richiesta anche verso KLM.

L'ultima tabella in cui aggiungere informazioni è __policymt. Al suo interno, come visto nella sezione 3.3.2, bisogna aggiungere i nomi di tutte le mapping table create inizialmente e specificare per ognuna se la modifica del proprio attributo deve essere propagata.

5.1.4 Invio della richiesta

Attualmente non esiste un'interfaccia grafica per avviare le richieste di modifica, inserimento o cancellazione, ma è necessario inserire i dati direttamente nel codice della funzione *run_policy_Module()*, che si trova nella classe principale *PeerInterface.java*. Nella figura 5.5 c'è il codice che permette di inviare una richiesta di aggiornamento: I valori vanno inseriti nei vettori in ordine rispetto agli attributi della tabella.

```
private void run_Policy_Module(){
      PeerPolicy peer_policy =
            new PeerPolicy(ambiente.getMainPeer().peer);
      Vector OldTuple = new Vector();
      Vector NewTuple = new Vector();

      String tablename = ‘‘afvol’’;
      int eventname = 2; //2 indica il tipo di evento UPDATE

      NewTuple.add(new String(‘‘AF8345’’));
      NewTuple.add(new String(‘‘2007-12-11’’));
      NewTuple.add(new String(‘‘AMS’’));
      NewTuple.add(new String(‘‘CWL’’));
      NewTuple.add(new String(‘‘F100’’));
      NewTuple.add(new String(‘‘continental’’));
      NewTuple.add(new String(‘‘33’’));

      OldTuple.add(new String(‘‘AF8345’’));
      OldTuple.add(new String(‘‘2007-10-11’’));
      OldTuple.add(new String(‘‘AMS’’));
      OldTuple.add(new String(‘‘CWL’’));
      OldTuple.add(new String(‘‘F100’’));
      OldTuple.add(new String(‘‘continental’’));
      OldTuple.add(new String(‘‘33’’));

      peer_policy.startPeerPolicyModule(tablename, eventname, OldTuple,
            NewTuple);
}
```

Figura 5.5: Sezione di codice che permette l'inserimento dei parametri per la modifica di un volo.

Finita la fase di preparazine non resta che avviare il policy service dall'interfaccia di Hyperion, aprendo il menu policy e avviando "run policy service", e controllare dall'output del programma e nel database se le operazioni sono andate a buon fine. In questo caso l'algoritmo non ha riscontrato problemi, e le tabelle sono state tutte aggiornate. Sono stati effettuati dei tentativi sia con i due peers collegati, sia collegandoli in un secondo momento, ed in entrambi i casi ha funzionato tutto a dovere.

5.2 Test 2: cicli nella rete

Il secondo test proposto è risultato utile a controllare il comportamento dell'algoritmo in presenza di cicli nella rete. Come si vede dalla figura 5.6 la stessa richiesta viene propagata attraverso 4 peers ed infine torna al peer di partenza. Il peers di partenza è AirFrance, e nella figura è contraddistinto dalla scritta *start*. Successivamente la richiesta verrà passata a KLM, KLM-CityHopper, Alitalia ed infine tornerà ad AirFrance. Il procedimento per la

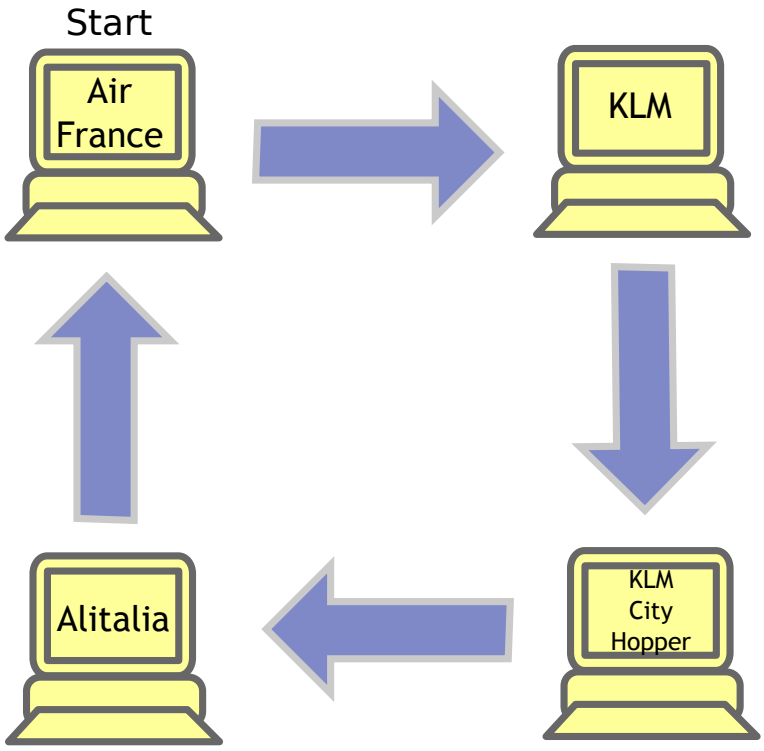

Figura 5.6: Propagazione della richiesta nel Test 2.

preparazione del test è lo stesso del primo, ma deve essere replicato su tutti

e quattro i peers. Anche in questo caso non si sono riscontrati problemi, i dati sono stati aggiornati in tutti i database e la richiesta una volta arrivata ad Alitalia si è fermata, evitando di innescare un ciclo.

5.3 Test 3: richieste simultanee

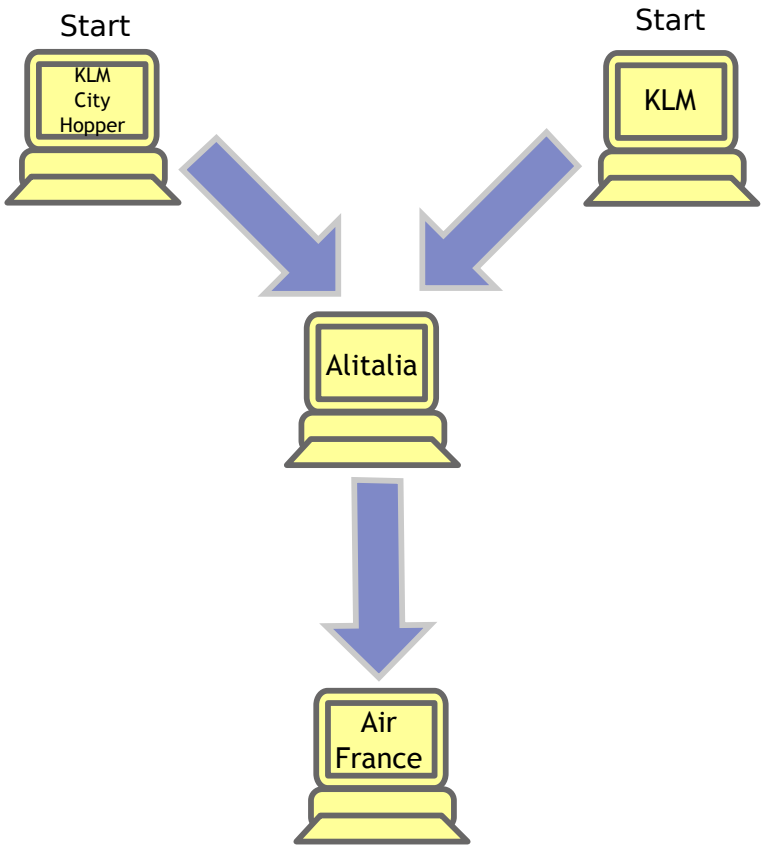

Figura 5.7: Propagazione delle richieste nel Test 3.

Il terzo test è, un po' più complesso, in questo caso si vuole controllare come si comporta l'algoritmo nel caso in cui due peers inoltrino una richiesta contemporaneamente su una stessa riga del database, aggiornando campi differenti. Nella figura 5.7 si vede come sono state propagate le richieste: I peer KLMCityHopper e KLM inviano contemporaneamente una richiesta, entrambe vanno a modificare due valori differenti della stessa riga nel database. Come si vede in figura 5.8 nel peer KLM sarà interessato l'attributo *flto*, che è relativo alla destinazione del volo, e in KLMCityHopper si andrà

a modificare l'attributo *aid*, riferito al tipo di aereo. Le mapping table sono

(a) Query su KLM

```
UPDATE   klflight
SET      flid='KL1065', fldate='2007-12-11', flfrom='Amsterdam',
         flto='Florence', aid='F70', typefl='Europa', sold=33
WHERE    flid='KL1065' AND fldate='2007-12-11' AND
         flfrom='Amsterdam' AND flto='Cardiff' AND
         aid='F70' AND typefl='Europa' AND sold=33
```

(b) Query su KLMCityHopper

```
UPDATE   klcflight
SET      fid='KLc1065', fdate='2007-12-11', ffrom='Amsterdam',
         fto='Cardiff', aid='F70', typef='regular line', sold=33
WHERE    fid='KLc1065' AND fdate='2007-12-11' AND
         ffrom='Amsterdam' AND fto='Cardiff' AND
         aid='F100' AND typef='regular line' AND sold=33
```

Figura 5.8: Query effettuate nel test numero 3

state predisposte in modo che entrambi i peer propaghino la richiesta verso Alitalia, quest'ultima infine invierà le modifiche anche ad AirFrance.

In questa situazione sono stati riscontrati dei problemi. Solo il primo dei due messaggi inviati da KLM e KLMCityHopper aggiorna il database del peer Alitalia, infatti il secondo messaggio non trova una riga nel database corrispondente a quella da modificare, perchè essa sarà già stata cambiata dalla prima richiesta. Si ricorda che l'algoritmo, prima di effettuare l'aggiornamento, confronta tutti i campi per trovare una riga corrispondente a quella presente nel primo peer precedentemente la modifica. Se la riga non viene trovata la modifica non viene effettuata, ma la richiesta viene propagata comunque al peer successivo. Per questo i due messaggi sono inoltrati entrambi verso AirFrance, nel quale avverrà la stessa cosa, considerando però che il primo messaggio arrivato può non essere lo stesso di Alitalia e quindi il campo aggiornato su Alitalia potrebbe non essere lo stesso aggionato su AirFrance.

Il problema deriva dal fatto che i peers KLM e KLMCityHopper non sono collegati tra loro, e possono inviare dei messaggi verso Alitalia, ma non riceverne. In questo scenario è impossibile mantenere la consistenza dei dati infatti i primi due peers sono di fatto isolati ed i loro dati non possono essere accordati.

5.4 Considerazioni

Durante i test si è visto che l'algoritmo in se funziona correttamente. Con alcune impostazioni della rete non è possibile mantenere la consistenza in tutti i database e perciò l'UPDATE può non raggiungere i risultati attesi. Quindi si conclude che per avere una rete con dei database sempre consistenti un buon algoritmo non basta, ma è necessaria una progettazione adeguata della stessa ad opera dell'amministratore. Ad esempio delle possibili tipologie potrebbero essere reti cicliche, come nel secondo test, oppure la creazione di acquaintance bidirezionali anzichè monodirezionali, o l'esistenza di un solo peer che asegua le modifiche con gli altri collegati in modo lineare, in modo che la richiesta si propaghi in una sola direzione. Ma questo aspetto dipende strettamente dal tipo di utilizzo che se ne vuole fare e non esiste una soluzione universale adatta ad ogni caso.

Capitolo 6

Conclusioni

Il fine principale di questa tesi era l'eliminazione delle possibili inconsistenze nei dati, che sarebbero potute verificarsi tramite l'invio prima di un messaggio di cancellazione, poi uno di inserimento per l'aggiornamento dei dati nelle tabelle; cosa che sarebbe potuta avvenire nell'ordine inverso o solo in parte nel caso in cui ci fossero dei malfunzionamenti della rete. Il problema è stato risolto tramite l'aggiunta dell'oparzione di update all'algoritmo, così da poter effettuare tutte le modifiche con un solo passaggio e quindi con un'unico messaggio.

E' risultato evidente che per mantenere la consistenza dei dati su tutta la rete è necessaria anche una buona progettazione della stessa, infatti con determinate configurazioni si è visto che dei peer possono rimanere isolati, con la sola possibilità di inviare richieste agli altri, ma senza la possibilità di riceverne. In questo caso è inevitabile che essi non risultino aggiornati, con la possibilità che a lungo termine le loro richieste non risultino più corrette per il resto della rete.

I vantaggi dati da questa estensione sono appunto una maggior sicurezza durante l'aggiornamento delle tabelle, il minor numero di messaggi inviati tra i peers e le minori operazioni da svolgere da parte del database. Svantaggi rispetto alla situazione precedente non ne ho trovati, e nel caso in cui ne emergessero è comunque possibile effettuare l'operazione con il vecchio metodo.

Elenco delle figure

Bibliografia

[1] The Hyperion Project's site. *http://dit.unitn.it/~hyperion*.

[2] Anastasios Kementsietsidis, Marcelo Arenas, Renée J. Miller. *Managing Data Mappings in the Hyperion Project*. 2003.

[3] Vasiliki Kantere, Iluju Kiringa, John Mylopoulos, Anastasios Kementsietsidis, Marcelo Arenas. *Coordinating Peer Databases Using ECA Rules*. 2003.