UNIVERSITÀ DEGLI STUDI DI MODENA
E REGGIO EMILIA
Facoltà di Ingegneria - Sede di Modena
Corso di Laurea in Ingegneria Informatica

MOMIS:
il wrapper per sorgenti di dati XML

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Francesco Guerra

Correlatore
Ing. Alberto Corni

Controrelatore
Chiar.mo Prof. Letizia Leonardi

Anno Accademico 1999 - 2000

Parole chiave:
Integrazione dati
Dati semistrutturati
XML
ODL_{I^3}
Wrapper

Indice

Elenco delle figure

Introduzione

Lo sviluppo delle tecnologie telematiche, tanto per i sistemi di elaborazione, quanto per le reti di calcolatori, ha portato ad una sempre maggiore presenza di sorgenti informative determinando un vera e propria esplosione nella quantità e nella varietà di dati accessibili.
Parallelamente alla crescita delle sorgenti di informazione si é potuto osservare un degrado per quello che concerne la qualità dell'informazione stessa. Il dato, infatti, spesso si trova collocato in documenti non strutturati, quindi di difficile lettura da parte degli strumenti tradizionali di analisi, inoltre, a causa del fenomeno dell'*information overload*, risulta difficile agli utenti discernere tra le varie sorgenti presenti quelle che contengano dati significativi. Il problema che sta alla base é quindi quello della presenza di una grande varietà di sorgenti, disomogenee le une rispetto alle altre, ma anche a volte di tipo "*semistrutturato*", e quindi *disomogenee* anche nella propria costruzione. Tale eterogeneità dei sistemi può infatti presentarsi in diversi modi sia considerando le differenti piattaforme hardware e software su cui una sorgente è implementata (ad esempio diversi DBMS e linguaggi di interrogazione), sia considerando i modelli di dati utilizzati (relazionale, object-oriented, ecc...), sia facendo riferimento agli schemi usati per la rappresentazione della struttura logica dei dati memorizzati.

In un contesto di questo tipo, nel quale appunto convive una forte eterogeneità sia delle tipologie di sorgenti di informazione, sia delle modellazioni dei dati contenuti nelle singole sorgenti, risulta evidente che, per potere reperire le informazioni desiderate, sarebbe necessario avere una conoscenza specifica sia del contenuto, sia delle strutture, sia dei linguaggi di interrogazione propri delle singole sorgenti. L'utente dovrebbe quindi essere in grado di scomporre la propria interrogazione in una sequenza di sottointerrogazioni rivolte alle singole sorgenti di informazioni provvedendo poi a compiere una operazione di "$trasformazione$" dei risultati parziali, in modo da ottenere una risposta unificata. Tutto ciò dovrebbe essere fatto tenendo presente le possibili trasformazioni che possono subire i dati, le relazioni che

li legano, le proprietà che possono avere in comune e le discrepanze sussistenti tra le diverse rappresentazioni.
Come si é detto, essendo poi il numero delle sorgenti di informazione in continua crescita ed essendo di conseguenza anche l'eterogeneità dei dati costantemente crescente, risulta indispensabile progettare dei meccanismi per automatizzare e rendere pertanto piú agevole per l'utente il processo di interrogazione e di reperimento delle informazioni dalle singole sorgenti.

Questa tesi si inserisce all'interno di un progetto più ampio denominato **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) [1, 2, 3, 4, 5], sviluppato con lo scopo di realizzare un processo semi-automatico di integrazione di sorgenti eterogenee e distribuite.
MOMIS adotta un'architettura a tre livelli con un *Mediatore* che ne occupa la parte centrale ed avente lo scopo di fornire una visione integrata degli schemi locali. Tale visione integrata degli schemi costitutivi le singole sorgenti permette di effettuare delle interrogazioni prescindendo dalla conoscenza della specifica sorgente, ma utilizzando le informazioni che derivano dall'unico schema, sintesi delle diverse eterogeneità. Il *Mediatore* rappresenta pertanto il cuore del sistema ed ha il compito di realizzare l'integrazione degli schemi e di provvedere alla gestione dei risultati delle operazioni di query.

Elementi indubbiamente innovativi di questo progetto sono rappresentati dall'impiego di un approccio *semantico* e dall'uso di logiche descrittive per la rappresentazione degli schemi delle sorgenti. Questi elementi introducono, infatti comportamenti intelligenti che permettono di sfruttare al meglio le conoscenze intensionali, semantiche ed estensionali sia inter-schema sia intra-schema, per generare una vista globale il più possibile espressiva e sulla quale potere effettuare delle interrogazioni significative.
Di fondamentale importanza è quindi l'impiego di ODB-Tools, un ambiente software sviluppato presso l'Università di Modena, in grado di realizzare la validazione di schemi ad oggetti e l'espansione semantica delle interrogazioni.

Se il mediatore rappresenta il cuore del processo di integrazione, il wrapper rappresenta il meccanismo attraverso il quale avviene l'interazione con la singola sorgente. Pertanto dovranno esistere diversi wrapper ognuno ottimizzato per presidiare una certa tipologia di sorgenti. I compiti del wrapper sono sostanzialmente due: in primo luogo deve fornire una rappresentazione, nella logica comune, ODL_{I^3}, dello schema della sorgente alla quale é connesso. Tale descrizione dovrà necessariamente essere il frutto di un compromesso, tra la salvaguardia della struttura della sorgente originaria e la rappresentazione di schemi attraverso il modello dati utilizzato. In secondo luogo deve permettere l'esecuzione delle cosiddette *query locali* e deve fornire i risultati ottenuti al mediatore.

Obiettivo della presente tesi è stata appunto l'analisi e l'implementazione

di un wrapper per sorgenti di dati di tipo XML. Tale linguaggio, prestandosi a modellare sia dati di tipo strutturato sia dati di tipo semistrutturato, permette la rappresentazione di realtà anche complesse. É stato realizzato il traduttore, che consente la trasformazione dello schema rappresentativo della singola sorgente nel corrispettivo nel linguaggio ODL_{I^3}, mentre, non esistendo uno standard di linguaggio di interrogazione per sorgenti di questo tipo, si é preferito non implementare la gestione di query locali, ma analizzare le problematiche connesse all'introduzione di un query language all'interno di sorgenti di tipo XML.

La tesi è organizzata nel modo seguente:

Nel **Capitolo 1** viene dapprima introdotta l'architettura di riferimento I^3 per i sistemi di Integrazione di Informazioni, per poi illustrare le scelte implementative fatte in MOMIS. Viene anche presentato il componente ODB-Tools usato al fine di introdurre, nel sistema, comportamenti intelligenti. Nella parte finale del capitolo vengono illustrate le fasi di interazione con il wrapper da parte del mediatore.

Il **Capitolo 2** descrive le caratteristiche dei dati semistrutturati e introduce le caratteristiche di modellazione dei dati tipiche del linguaggio XML. Vengono presentati due model data, che si differenziano per l'uso del concetto di ordine, e viene delineata la differenza nella modellazione dei dati che intercorre tra la rappresentazione che si ottiene utilizzando ODL_{I^3}e la rappresentazione che si ottiene utilizzando XML.

Nel **Capitolo 3** viene presentato XML-Schema, un linguaggio di nuova concezione non ancora reso standard dall'organismo preposto, che permette una maggiore definizione degli schemi delle sorgenti. La modellazione di dati effettuata con XML-Schema viene confrontata in questo capitolo con quella che si ottiene sia utilizzando XML1.0 sia utilizzando ODL_{I^3}.

Il **Capitolo 4** descrive in un primo momento i desiderata che il W3C, l'organismo preposto all'emanazione degli standard in questo contesto, ha individuato per quello che concerne le funzionalità che un query language deve possedere. Dopo avere brevemente catalogato i query languages proposti alla standardizzazione, vengono presentati due linguaggi, scelti l'uno per le potenzialità implementate, l'altro per la modellazione e la logica di tipo grafico proposta.

Nel **Capitolo 5** é presentato il software realizzato e in particolare il pro-

cesso attraverso il quale viene realizzata la traduzione dal linguaggio che descrive lo schema della sorgente XML in linguaggio ODL_{I^3}. Il codice non è allegato alla tesi ma è comunque reperibile all'indirizzo
http://sparc20.dsi.unimo.it/tesi/index.html.

Sono inoltre presenti sette appendici. In particolare in Appendice A viene riportato un glossario dei termini usati in ambito I^3, in Appendice B sono indicate le regole di traduzione tra XML e ODL_{I^3}, in Appendice C viene mostrato un esempio completo di traduzione tra i due linguaggi, in Appendice D viene mostrata una possibile risoluzione dei problemi connessi con content model innestati, in Appendice E viene mostrata la sintassi di ODL_{I^3}, in Appendice F viene mostrata la sintassi XML e infine in Appendice G viene mostrata la grammatica XML-QL.

Capitolo 1

L'Integrazione delle Informazioni

I principali aspetti teorici da considerare nell'integrazione delle informazioni sono legati alla eterogeneità dei dati, che si manifesta sia nella natura dei dati (testi, immagini, suoni, record, ...) sia nei diversi tipi di sorgenti nei quali tali informazioni sono contenute, che possono comprendere ad esempio pagine HTML, DBMS relazionali o ad oggetti, file system, ecc... . Gli standard esistenti (TCP/IP, ODBC, OLE, CORBA, SQL, etc...) pur risolvendo, anche se solo parzialmente, i problemi legati alle diversità hardware e software dei protocolli di rete e i problemi relativi alle comunicazioni fra i moduli, non trattano le problematiche specifiche connesse con la modellazione delle informazioni. Infatti i modelli di dati e gli schemi si differenziano gli uni dagli altri in modo da dare una struttura logica ai numerosi generi di dati da memorizzare: in questo modo si crea una eterogeneità *semantica* non risolvibile dagli standard. Inoltre l'abbondanza di informazioni, dovuta ad un numero sempre maggiore di fonti, contribuisce a far sorgere ulteriori problematiche: tra queste é possibile evidenziare l'*information overload* (sovraccarico delle informazioni), che consiste nella difficoltà da parte dell'utente di discerne e di isolare i dati significativi. Altri problemi per i quali si devono trovare soluzioni sono la riduzione dei tempi di accesso, la salvaguardia della sicurezza e della consistenza, gli elevati costi di mantenimento da affrontare per modificare, eliminare o introdurre una nuova sorgente.

I problemi elencati sono a testimonianza della complessità e della molteplicità degli aspetti che le architetture dedicate all'integrazione di sorgenti di dati eterogenei devono tenere in considerazione. Per facilitare il processo di progettazione e di realizzazione di tali moduli, che devono necessariamente essere affidabili, flessibili e tali da far fronte efficacemente ad un panorama in continua evoluzione, occorre cercare di sviluppare un'architettura di moduli

che assicuri il riuso e la capacità di interagire con altri sistemi esistenti.

Gli approcci all'integrazione, descritti in letteratura o effettivamente realizzati, presentano diverse metodologie: la reingegnerizzazione delle sorgenti mediante standardizzazione degli schemi e la creazione di un database distribuito; il *repository independence*, e cioé un approccio che prevede di isolare al di sotto di una vista integrata le applicazioni ed i dati integrati dalle sorgenti, consentendo la massima autonomia e nascondendo al contempo le differenze esistenti; i *datawarehouse* che realizzano presso l'utente finale delle viste, ovvero delle porzioni di sorgenti, replicando fisicamente i dati ed affidandosi a complicati algoritmi di 'allineamento' per assicurare la loro consistenza a fronte di modifiche nelle sorgenti originali.
Nelle pagine seguenti verrà descritta la proposta dell'ARPA (Advanced Research Projects Agency)[6] per un'architettura che favorisca da una parte l'autonomia delle sorgenti, ma che dall'altra assicuri flessibilità e riusabilità. Si presenterà inoltre l'approccio seguito nel progetto MOMIS.

1.1 L'Integrazione Intelligente delle Informazioni

L'integrazione delle Informazioni (I^2) si distingue da quella dei dati e dei database in quanto non cerca di collegare semplicemente alcune sorgenti, quanto piuttosto risultati opportunamente selezionati da esse [7]. Per ottenere risultati selezionati è richiesta *conoscenza* ed *intelligenza* finalizzate alla scelta delle sorgenti e dei dati, alla loro fusione e alla conseguente sintesi. L'integrazione comporta perciò grandi difficoltà, sia a livello teorico sia a livello pratico. La dimensione e la quantità delle problematiche che insorgono qualora si desideri fondere informazioni provenienti da sorgenti autonome fanno sì che si senta l'esigenza di ideare una metodologia sia riusabile, per diminuire i costi di sviluppo di tali applicazioni, sia flessibile, per far fronte intelligentemente alle evoluzioni fisiche e logiche delle sorgenti interessate.

1.1.1 Il Programma I^3

Un'ambiziosa ricerca, finalizzata ad indicare un'architettura di riferimento che realizzi l'integrazione di sorgenti eterogenee in maniera automatica, è quella del programma I^3, sviluppato dall'ARPA, l'agenzia che fa capo al Dipartimento di Difesa americano [6]. In quel contesto si é potuto osservare che l'integrazione aumenta il valore dell'informazione ma richiede una forte adattabilità realizzativa: si devono infatti riuscire a gestire i casi di aggiornamento e sostituzione delle sorgenti, dei loro ambienti e/o piattaforme, della loro ontologia e della semantica. Le tecniche sviluppate dall'*Intelligenza Artificiale*,

potendo efficacemente dedurre informazioni utili dagli schemi delle sorgenti, diventano pertanto uno strumento prezioso per la costruzione automatica di soluzioni integrate flessibili e riusabili.

Progettare la costruzione di supersistemi ad hoc che interessino una grande quantità di sorgenti non correlate semanticamente, è faticoso ed il risultato è un sistema scarsamente manutenibile o adattabile, strettamente finalizzato alla risoluzione dei problemi per cui è stato implementato. Secondo il programma I^3, una soluzione a questi problemi può essere trovata mediante l'introduzione di architetture modulari sviluppabili secondo i principi proposti da uno standard. Tale standard deve porre le basi dei servizi che devono essere realizzati attraverso l'integrazione e deve cercare di abbassare i costi di sviluppo e di mantenimento.
Pertanto costruire nuovi sistemi risulta realizzabile con minor difficoltà e minor tempo (e quindi costi) se si riesce a supportare lo sviluppo delle applicazioni riutilizzando la tecnologia già sviluppata. Per la riusabilità è fondamentale l'esistenza di interfacce ed architetture standard. Il paradigma suggerito per la suddivisione dei servizi e delle risorse nei diversi moduli, si articola su due dimensioni:

- *orizzontalmente* in tre livelli: livello utente, livello intermedio che fa uso di tecniche di IA, livello delle sorgenti di dati;
- *verticalmente*: diversi domini in cui raggruppare le sorgenti.

In generale i diversi domini non sono strettamente connessi all'interno di un certo livello ma si scambiano informazioni e dati; la combinazione delle informazioni avviene a livello di utilizzatore riducendo la complessità totale del sistema e consentendo lo sviluppo di numerose applicazioni finalizzate a scopi anche diversi fra loro. Ad esempio, si supponga di dover integrare informazioni sui trasporti mercantili, ferroviari e stradali: ciò permette all'utente di avere un'idea completa su quale mezzo di trasporto sia maggiormente vantaggioso per le proprie necessità. Aggiungendo a questo altri domini, quali le situazioni metereologiche ed i costi di immagazzinamento e trasporto, si possono facilitare le scelte di un dirigente che deve decidere come e quando consegnare delle merci.

Il livello dell'architettura su cui é necessario focalizzare maggiormente l'attenzione per quello che concerne l'uso di tecniche di intelligenza artificiale è quello intermedio: esso costituisce il punto nodale fra le applicazioni sviluppate per gli utenti ed i dati posti nelle sorgenti. Questo livello deve offrire servizi dinamici quali la selezione delle sorgenti, le gestione degli accessi e delle interrogazioni, il ricevimento e la combinazione dei dati, l'analisi e la sintesi degli stessi.

1.2 Architettura di riferimento per sistemi I^3

L'obiettivo del programma I^3é quello di ridurre il tempo necessario alla realizzazione di un integratore di informazioni, fornendo una raccolta e una formalizzazione delle soluzioni prevalenti nel campo della ricerca. In particolare due sono le ipotesi che rappresentano la base del progetto I^3. La prima é connessa con la difficoltà che si ha nel ricercare delle informazione all'interno della molteplicità delle sorgenti di informazione che in questo momento é possibile individuare. Il secondo aspetto é legato al fatto che le fonti di informazione e i sistemi informativi, pur essendo spesso semanticamente correlati tra di loro, non lo sono in una forma semplice né preordinata. Di conseguenza il processo di integrazione delle informazioni puó risultare molto complesso.
Pertanto é necessario proporre una architettura di riferimento che rappresenti alcuni dei servizi che un integratore di informazioni deve contenere e le possibili interconnessioni fra di essi. Tale descrizione non vuole imporre né delle soluzioni implementative, né é da ritenersi esaustiva delle funzionalità che un sistema di integrazione deve includere.

1.2.1 A cosa serve la tecnologia I^3 e quali problemi deve risolvere

Le applicazioni che possono essere interessate da sviluppi della tecnologia I^3coprono un ampio spettro di campi e possono essere, puramente a titolo esemplificativo, suddivise in questo modo:

- pianificazione e supporto della logistica;
- sistemi informativi nel campo sanitario;
- sistemi informativi nel campo manifatturiero;
- sistemi bancari internazionali;
- ricerche di mercato.

Naturalmente, avendo l'architettura proposta la presunzione di esprimere delle caratteristiche generali, ed essendo la casistica dei campi applicativi vasta, é sicuramente necessario individuare, al di fuori di un insieme di servizi di base, funzionalità specifiche di ogni ambiente di sviluppo. Ad esempio, un integratore il cui scopo sia interagire con sistemi di basi di dati "classici", come ad esempio i sistemi basati sui file, i data base relazionali, i data base ad oggetti, avrà la necessità di un pacchetto base di servizi molto differenti

da un sistema cosiddetto "multimediale", il cui obiettivo sia integrare suoni, immagini ...
Così come possono essere individuati differenti obiettivi per un sistema I^3, allo stesso modo possono essere individuati diversi problemi ai quali é necessario dare risposta. Tra questi, in questa sede si propongono i seguenti:

- *la grande differenza tra le fonti di informazione*:
  - le fonti informative sono semanticamente differenti, e si possono individuare dei livelli di differenze semantiche [8];
  - le informazioni possono essere memorizzate utilizzando differenti formati;
  - possono essere diversi gli schemi, i vocabolari usati, le ontologie su cui questi si basano, anche quando le fonti condividono significative relazioni semantiche;
  - può variare la natura dell'informazione, che può essere rappresentata attraverso testi, immagini, audio, media digitali;
  - può variare il modo in cui si accede alle singole sorgenti. Quindi possono essere presenti molteplici interfacce utente, linguaggi di interrogazione, protocolli e meccanismi di transazione;
- *la semantica complessa ed a volte nascosta delle fonti*: i programmi applicativi possono rappresentare, ad esempio per vecchi sistemi, la chiave per l'uso di informazioni. Senza di essi può essere molto difficile dedurre la semantica contenuta nel database, specialmente nel caso in cui occorra interagire con sistemi molto vasti;
- *l'esigenza di creare applicazioni in grado di interfacciarsi con porzioni diverse delle fonti di informazione*: non è sempre possibile avere a disposizione l'intera sorgente di informazione, ma talvolta si dispone unicamente di una sua parte selezionata che può variare nel tempo;
- *il grande numero di fonti da integrare*: con il moltiplicarsi delle informazioni, lo stesso numero di fonti da integrare per ogni singola applicazione può aumentare in maniera considerevole, e può succedere che sia necessario accedere in modo coordinato a diverse fonti;
- *il bisogno di realizzare moduli I^3 riusabili*: benché questo possa essere considerato uno dei compiti più difficili nella realizzazione di un integratore, è importante progettare non un sistema ad-hoc, bensì un'applicazione i cui moduli possano facilmente essere riutilizzati in altre

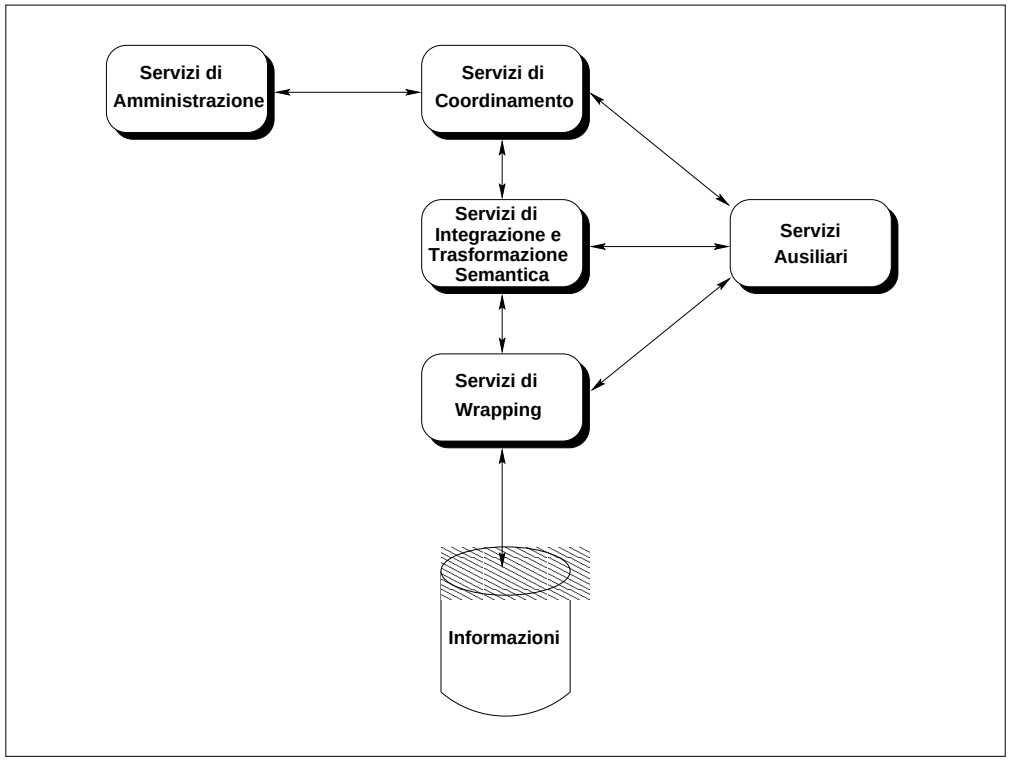

Figura 1.1: Diagramma dei servizi I^3

applicazioni, secondo i moderni principi di riusabilità del software. In questo caso, l'abilità di costruire valide funzioni di libreria può considerevolmente diminuire i tempi e le difficoltà di realizzazione di un sistema informativo che si basa su più fonti differenti.

Si passa ora ad analizzare l'architettura vera e propria di un sistema I^3, riportata in Figura 1.1. L'architettura di riferimento dà grande rilevanza ai Servizi di Coordinamento. Questi servizi giocano infatti due ruoli: come prima cosa, possono localizzare altri servizi I^3 e fonti di informazioni che possono essere utilizzate per costruire il sistema stesso; secondariamente, sono responsabili di individuare ed invocare a run-time gli altri servizi necessari a dare risposta ad una specifica richiesta di dati.
Sono comunque in totale cinque le famiglie di servizi che sono state rappresentate in questa architettura. I servizi individuati, così come rappresentati nell'architettura proposta, possono essere soggetti a due differenti chiavi di lettura: la lettura seguendo l'asse orizzontale e quella seguendo l'asse verticale mettono in evidenza diversi aspetti e diversi compiti del sistema.
Se si percorre l'asse verticale, si può intuire come avviene lo scambio di in-

formazioni nel sistema: in particolare, i servizi di *wrapping* provvedono ad estrarre le informazioni dalle singole sorgenti. Tali informazioni vengono poi impacchettate ed integrate dai Servizi di Integrazione e Trasformazione Semantica, per poi essere passate ai servizi di Coordinamento che ne avevano fatto richiesta. L'asse orizzontale mette invece in risalto il rapporto tra i servizi di Coordinamento e quelli di Amministrazione, ai quali spetta infatti il compito di mantenere informazioni sulle capacità delle varie sorgenti (che tipo di dati possono fornire ed in quale modo devono essere interrogate). Funzionalità di supporto, che verranno descritte successivamente, sono invece fornite dai Servizi Ausiliari, responsabili dei servizi di arricchimento semantico delle sorgenti.
Vengono ora analizzate in dettaglio le funzionalità specifiche di ogni servizio e le problematiche che devono essere affrontate.

1.2.2 Servizi di Coordinamento

I servizi di Coordinamento sono quei servizi di alto livello che permettono l'individuazione delle sorgenti di dati *interessanti*, ovvero che probabilmente possono dare risposta ad una determinata richiesta dell'utente. A seconda delle possibilità dell'integratore che si vuole realizzare, possono essere rappresentati da meccanismi che includono dalla selezione dinamica delle sorgenti (o brokering, per Integratori Intelligenti) fino al semplice *Matchmaking*, in cui il mappaggio tra informazioni integrate e locali è realizzato manualmente ed una volta per tutte. Vediamo alcuni esempi.

1. **Facilitation e Brokering Services**: l'utente manda una richiesta al sistema e questo usa un deposito di metadati per ritrovare il modulo che può trattare la richiesta direttamente. I moduli interessati da questa richiesta possono essere o uno solo alla volta (nel qual caso si parla di Brokering) oppure più di uno (e in questo secondo caso si tratta di facilitatori e mediatori, attraverso i quali a partire da una richiesta ne viene generata più di una da inviare singolarmente a differenti moduli che gestiscono sorgenti distinte, e reintegrando poi le risposte in modo da presentarle all'utente come se fossero state ricavate da un'unica fonte). In questo ultimo caso, in cui una query può essere decomposta in un insieme di sottoquery, si farà uso di servizi di Query Decomposition e di tecniche di Inferenza (mutuate dall'Intelligenza Artificiale) per una determinazione dinamica delle sorgenti da interrogare, a seconda delle condizioni poste nell'interrogazione.
   I vantaggi che questi servizi di Coordinamento portano stanno nel fatto che non è richiesta all'utente del sistema una conoscenza del contenuto

delle diverse sorgenti, ma viene fornita un'unica rappresentazione delle sorgenti e in questo modo un sistema omogeneo che gestisce direttamente la sua richiesta. Pertanto l'utente non é obbligato a conoscere i domini con i quali i vari moduli I^3 si trovano ad interagire. Risulta quindi evidente la considerabile diminuzione di complessità di interazione col sistema che deriva da una architettura di questo tipo.

2. **Matchmaking**: il sistema viene configurato manualmente da un operatore in fase di inizializzazione. Successivamente a quella fase, tutte le richieste verranno trattate allo stesso modo. Sono definiti gli anelli di collegamento tra tutti i moduli del sistema, e nessuna ottimizzazione è fatta a tempo di esecuzione.

1.2.3 Servizi di Amministrazione

Sono servizi usati dai Servizi di Coordinamento per localizzare le sorgenti *utili*, per determinare le loro capacità, e per creare ed interpretare TEMPLATE. I Template sono strutture dati che descrivono i servizi, le fonti ed i moduli da utilizzare per portare a termine un determinato task. Sono quindi utilizzati dai sistemi meno "intelligenti", e consentono all'operatore di predefinire le azioni da eseguire a seguito di una determinata richiesta, limitando al minimo le possibilità di decisione del sistema.
In alternativa all'uso dei Template, possono essere utilizzate le **Yellow Pages**: si tratta di servizi di directory che memorizzano le informazioni sul contenuto delle varie sorgenti e sul loro stato (attiva, inattiva, occupata). Consultando queste Yellow Pages, il mediatore è in grado di spedire alla giusta sorgente la richiesta di informazioni, ed eventualmente di rimpiazzare questa sorgente con una equivalente nel caso non fosse disponibile. Fanno parte di questa categoria di servizi il Browsing: si permette all'utente di "navigare" attraverso le descrizioni degli schemi delle sorgenti, recuperando informazioni su queste. Il servizio si basa sulla premessa che queste descrizioni siano fornite esplicitamente tramite un linguaggio dichiarativo leggibile e comprensibile all'utente. Inoltre tale servizio potrebbe contenere dei servizi Trasformazione del Vocabolario e dell'Ontologia, come pure di Integrazione Semantica. Da citare sono pure i servizi di Iterative Query Formulation: si tratta di sistemi che aiutano l'utente a rilassare o a meglio specificare alcuni vincoli della propria interrogazione per ottenere risposte più precise.

1.2.4 Servizi di Integrazione e Trasformazione Semantica

Questi servizi supportano le manipolazioni semantiche necessarie per l'integrazione e la trasformazione delle informazioni. Il tipico input per questi servizi é rappresentato da una o più sorgenti di dati, e l'output é la "vista" integrata o trasformata di queste informazioni. Tra questi servizi si distinguono quelli relativi alla trasformazione degli schemi (ovvero di tutto ciò che va sotto il nome di *metadati*) e quelli relativi alla trasformazione dei dati. Sono spesso indicati come servizi di Mediazione, essendo tipici dei moduli mediatori.
In particolare é possibile osservare:

1. Servizi di **integrazione degli schemi**. Supportano la trasformazione e l'integrazione degli schemi e delle conoscenze derivanti da fonti di dati eterogenee. Fanno parte di essi i servizi di trasformazione dei vocaboli e dell'ontologia, usati per arrivare alla definizione di un'ontologia unica che combini gli aspetti comuni alle singole ontologie usate nelle diverse fonti. Queste operazioni sono molto utili quando devono essere scambiate informazioni derivanti da ambienti differenti, dove molto probabilmente non si condivide un'unica ontologia. Fondamentale, per la fase di creazione dell'insieme dei vocaboli condivisi, è la fase di individuazione dei concetti presenti in diverse fonti, e la riconciliazione delle diversità presenti sia nelle strutture, sia nei significati dei dati.

2. Servizi di **integrazione delle informazioni**. Provvedono alla traduzione dei termini da un contesto all'altro, ovvero dall'ontologia di partenza a quella di destinazione. Possono inoltre occuparsi di uniformare la "granularità" dei dati (come possono essere le discrepanze nelle unità di misura, o le discrepanze temporali).

3. Servizi di **supporto al processo di integrazione**. Sono utilizzati nel momento in cui una query è scomposta in molte subquery, da inviare a fonti differenti, e nel momento in cui i risultati provenienti dalle singole subquery devono essere integrati. Comprendono inoltre tecniche di *caching*, per supportare la materializzazione delle viste (problematica molto comune nei sistemi che vanno sotto il nome di datawarehouse).

1.2.5 Servizi di Wrapping

Sono utilizzati per fare sì che le fonti di informazioni aderiscano ad uno standard, che può essere interno o proveniente dal mondo esterno con cui

il sistema vuole interfacciarsi. Si comportano come traduttori dai sistemi locali ai servizi di alto livello dell'integratore e viceversa quando si interroga la sorgente di dati. In particolare, sono due gli obiettivi che si prefiggono:

1. permettere ai servizi di coordinamento e di mediazione di manipolare in modo uniforme il numero maggiore di sorgenti locali, anche se queste non sono state esplicitamente pensate come facenti parte del sistema di integrazione;

2. essere il più riusabili possibile. Per fare ciò, dovrebbero fornire interfacce che seguano gli standard più diffusi (e tra questi, si potrebbe citare il linguaggio SQL come linguaggio di interrogazione di basi di dati, e CORBA come protocollo di scambio di oggetti). Questo permetterebbe alle sorgenti estratte da questi wrapper "universali" di essere accedute dal numero maggiore possibile di moduli mediatori.

In pratica, compito di un wrapper è modificare l'interfaccia, i dati ed il comportamento di una sorgente, per facilitarne la comunicazione con il mondo esterno. Il vero obiettivo è quindi standardizzare il processo di wrapping delle sorgenti, permettendo la creazione di una libreria di fonti accessibili; inoltre, il processo stesso di realizzazione di un wrapper dovrebbe essere standardizzato, in modo da poter essere riutilizzato da altre fonti.

1.2.6 Servizi Ausiliari

Aumentano le funzionalità degli altri servizi descritti precedentemente: sono prevalentemente utilizzati dai moduli che agiscono direttamente sulle informazioni. Vanno dai semplici servizi di monitoraggio del sistema (un utente vuole avere un segnale nel momento in cui avviene un determinato evento in un database, e conseguenti azioni devono essere attuate), ai servizi di propagazione degli aggiornamenti e di ottimizzazione.

1.3 Il mediatore

Secondo la definizione proposta da Wiederhold in [9] "un mediatore è un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare informazioni per una applicazione di livello superiore ... Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno, o al più pochi, esperti."
Compiti di un mediatore sono allora:

- assicurare un servizio stabile, anche nel caso di cambiamento delle risorse;
- amministrare e risolvere le eterogeneità delle diverse fonti;
- integrare le informazioni ricavate da più risorse;
- presentare all'utente le informazioni attraverso un modello scelto dall'utente stesso.

Il progetto MOMIS, di cui questa tesi fa parte, ha tra i suoi obiettivi la realizzazione di un Mediatore. In corso di progettazione e realizzazione del sistema, é stato necessario introdurre una ipotesi che restringesse il campo applicativo del sistema stesso (e di conseguenza che restringesse il campo dei problemi a cui dare risposta). Tale ipotesi consiste nel fatto che il mediatore, per il momento, si occupi esclusivamente di sorgenti di dati di testo strutturati e semistrutturati, quali possono essere basi di dati relazionali, ad oggetti, file di testo, pagine html e pagine XML. L'approccio architetturale scelto è quello *classico*, che si sviluppa principalmente su 3 livelli:

1. utente: attraverso un'interfaccia grafica l'utente pone delle query su uno schema globale e riceve un'unica risposta, come se stesse interrogando un'unica sorgente di informazioni;
2. mediatore: il mediatore gestisce l'interrogazione dell'utente, combinando, integrando ed eventualmente arricchendo i dati ricevuti dai wrapper, ma usando un modello (e quindi un linguaggio di interrogazione) comune a tutte le fonti;
3. wrapper: ogni wrapper gestisce una singola sorgente, ed ha una duplice funzione: da un lato converte le richieste del mediatore in una forma comprensibile dalla sorgente, dall'altro traduce informazioni estratte dalla sorgente nel modello usato dal mediatore.

Facendo riferimento ai servizi descritti nelle sezioni precedenti, l'architettura del mediatore che si è progettato è riportata in Figura 1.2. In particolare, in questa tesi, é stato progettato un software che realizzi i servizi di Wrapping per quello che concerne le sorgenti dati di tipo XML. In particolare, come si é osservato in precedenza, il wrapper deve essere una interfaccia fra il nucleo del mediatore, che fornisce le operazioni di integrazione e trasformazione semantica e la singola sorgente. Nella fattispecie il wrapper realizza due funzionalità specifiche:

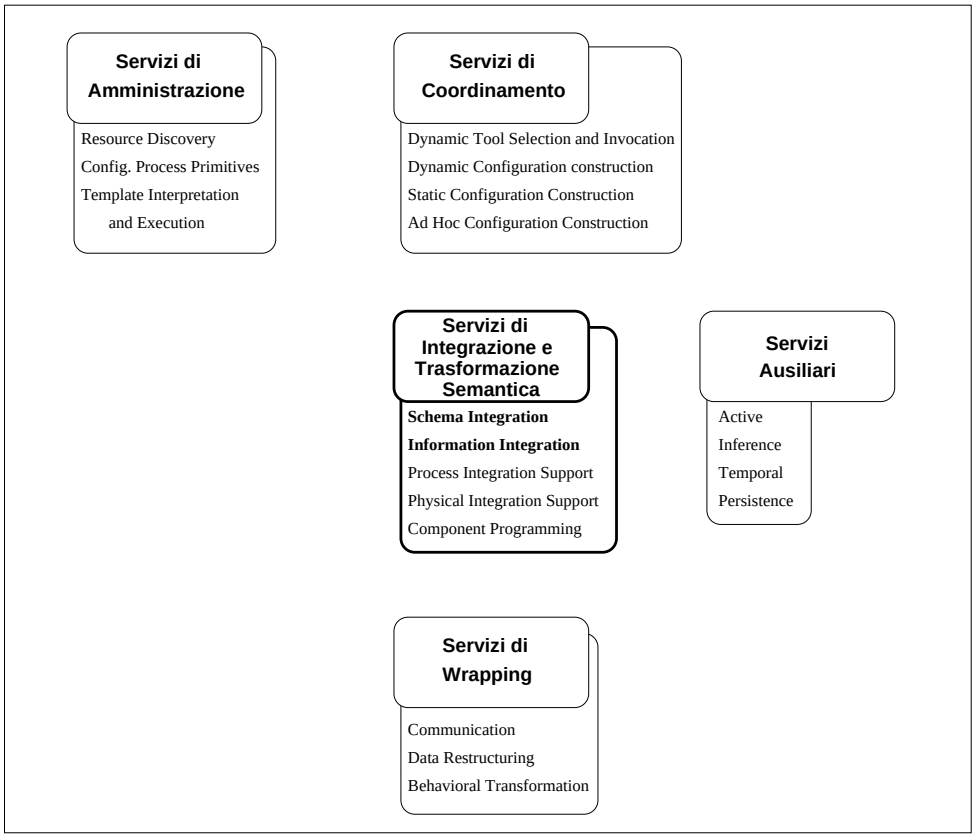

Figura 1.2: Servizi I^3 presenti nel mediatore

1. Restituisce la descrizione in ODL$_{I^3}$della sorgente alla quale é connesso
2. Permette l'esecuzione a livello locale delle query generate dal query manager.

L'impostazione architetturale presentata dimostra che il mediatore in progettazione vuole distaccarsi dall'approccio *strutturale*, cioè sintattico, tuttora dominante tra i sistemi presenti sul mercato. L'approccio strutturale, adottato da sistemi quali TSIMMIS [10, 11, 12, 13], è caratterizzato dal fatto di usare un self-describing model per rappresentare gli oggetti da integrare, limitando l'uso delle informazioni semantiche alle regole predefinite dall'operatore. In pratica, il sistema non conosce a priori la semantica di un oggetto che va a recuperare da una sorgente (e dunque di questa non possiede alcuno schema descrittivo) bensì è l'oggetto stesso che, attraverso delle etichette, si autodescrive, specificando tutte le volte, per ogni suo singolo campo, il significato che ad esso è associato. Questo approccio porta quindi ad un insieme

di vantaggi, tra i quali possiamo identificare:

- la possibilità di integrare in modo completamente trasparente al mediatore basi di dati fortemente eterogenee e magari mutevoli nel tempo: il mediatore non si basa infatti su una descrizione predefinita degli schemi delle sorgenti, bensì sulla descrizione che ogni singolo oggetto fa di sè. Oggetti simili provenienti dalla stessa sorgente possono quindi avere strutture differenti, cosa invece non ammessa in un ambiente tradizionale object-oriented;
- per trattare in modo omogeneo dati che descrivono lo stesso concetto, o che hanno concetti in comune, ci si basa sulla definizione manuale di rule, che permettono di identificare i termini (e dunque i concetti) che devono essere condivisi da più oggetti.

Altri progetti, e tra questi quello proposto, seguono invece un approccio definito *semantico*, che è caratterizzato dai seguenti punti:

- il mediatore deve conoscere, per ogni sorgente, lo schema concettuale (metadati);
- informazioni semantiche sono codificate in questi schemi;
- deve essere disponibile un modello comune per descrivere le informazioni da condividere (e dunque per descrivere anche i metadati);
- deve essere possibile una integrazione (parziale o totale) delle sorgenti di dati.

In questo modo, sfruttando opportunamente le informazioni semantiche che necessariamente ogni schema sottintende, il mediatore può individuare concetti comuni a più sorgenti e relazioni che li legano.

1.3.1 Problematiche da affrontare

Pur avendo a disposizione gli schemi concettuali delle varie sorgenti, non è certamente un compito banale individuare i concetti comuni ad esse, le relazioni che possono legarli, né tantomeno è banale realizzare una loro coerente integrazione. Trascurando per il momento le differenze dei sistemi fisici (alle quali dovrebbero provvedere i moduli wrapper) i problemi che si sono dovuti risolvere, o con i quali occorre giungere a compromessi, sono (a livello di mediazione, ovvero di integrazione delle informazioni) essenzialmente di due tipi:

1. problemi ontologici;

2. problemi semantici.

1.3.2 Problemi ontologici

Come riportato nel glossario A, per ontologia si intende, in questo ambito, "l'insieme dei termini e delle relazioni usate in un dominio, che denotano concetti ed oggetti". Con ontologia quindi ci si riferisce a quell'insieme di termini che, in un particolare dominio applicativo, denotano in modo univoco una particolare conoscenza e fra i quali non esiste ambiguità poiché sono condivisi dall'intera comunità di utenti del dominio applicativo stesso. Non è certamente l'obiettivo né di questo paragrafo, né della tesi in generale, dare una descrizione esaustiva di cosa si intenda per ontologia e dei problemi che essa comporta (ancorché ristretti al campo dell'integrazione delle informazioni), ma ci si limita a riportare una semplice classificazione delle ontologie (mutuata da Guarino [14, 15]), per inquadrare l'ambiente in cui ci si muove. I livelli di ontologia (e dunque le problematiche ad essi associate) sono essenzialmente tre:

1. *top-level ontology*: descrive concetti molto generali come spazio, tempo, evento, azione ... , che sono quindi indipendenti da un particolare problema o dominio: si considera ragionevole, almeno in teoria, che anche comunità separate di utenti condividano la stessa top-level ontology;

2. *domain e task ontology*: descrive, rispettivamente, il vocabolario relativo a un generico dominio (come può essere un dominio medico, o automobilistico) o a un generico obiettivo (come la diagnostica, o le vendite), dando una specializzazione dei termini introdotti nelle top-level ontology;

3. *application ontology*: descrive concetti che dipendono sia da un particolare dominio sia da un particolare obiettivo.

Come ipotesi semplificativa di questo progetto, si è considerato di muoversi all'interno delle domain ontology, ipotizzando quindi che tutte le fonti informative condividano almeno i concetti fondamentali (ed i termini con cui identificarli).

1.3.3 Problemi semantici

Pur ipotizzando che anche sorgenti diverse condividano una visione similare del problema da modellare, e quindi un insieme di concetti comuni, niente testimonia che i diversi sistemi usino esattamente gli stessi vocaboli per rappresentare questi concetti, né tantomeno le stesse strutture dati. Poiché infatti le diverse sorgenti sono state progettate e modellate da persone differenti, è molto improbabile che queste persone condividano la stessa "concettualizzazione" del mondo esterno, ovvero non esiste nella realtà una semantica univoca a cui chiunque possa riferirsi.
Se la persona P1 disegna una fonte di informazioni (per esempio DB1) e un'altra persona P2 disegna la stessa fonte DB2, le due basi di dati avranno sicuramente differenze semantiche: per esempio, le coppie sposate possono essere rappresentate in DB1 usando degli oggetti della classe COPPIE, con attributi MARITO e MOGLIE, mentre in DB2 potrebbe esserci una classe PERSONA con un attributo SPOSA.
Come riportato in [8] la causa principale delle differenze semantiche si può identificare nelle diverse concettualizzazioni del mondo esterno che persone distinte possono avere, ma questa non è l'unica. Le differenze nei sistemi di DBMS possono portare all'uso di differenti modelli per la rappresentazione della porzione di mondo in questione: partendo così dalla stessa concettualizzazione, determinate relazioni tra concetti avranno strutture diverse a seconda che siano realizzate attraverso un modello relazionale, o ad oggetti.
L'obiettivo dell'integratore, che è fornire un accesso integrato ad un insieme di sorgenti, si traduce allora nel non facile compito di identificare i concetti comuni all'interno di queste sorgenti e risolvere le differenze semantiche che possono essere presenti tra di loro. Possiamo classificare queste contraddizioni semantiche in tre gruppi principali:

1. **eterogeneità tra le classi di oggetti**: benché due classi in due differenti sorgenti rappresentino lo stesso concetto nello stesso contesto, possono usare nomi diversi per gli stessi attributi, per i metodi, oppure avere gli stessi attributi con domini di valori diversi o ancora (dove questo è permesso) avere regole differenti su questi valori;

2. **eterogeneità tra le strutture delle classi**: comprendono le differenze nei criteri di specializzazione, nelle strutture per realizzare una aggregazione, ed anche le *discrepanze schematiche*, quando cioè valori di attributi sono invece parte dei metadati in un altro schema (come può essere l'attributo SESSO in uno schema, presente invece nell'al-

tro implicitamente attraverso la divisione della classe PERSONE in MASCHI e FEMMINE);

3. **eterogeneità nelle istanze delle classi**: ad esempio, l'uso di diverse unità di misura per i domini di un attributo, o la presenza/assenza di valori nulli.

È però il caso di sottolineare la possibilità di sfruttare adeguatamente queste differenze semantiche per arricchire il nostro sistema: analizzando a fondo queste differenze e le loro motivazioni si può arrivare al cosiddetto *arricchimento semantico*, ovvero all'aggiungere esplicitamente ai dati tutte quelle informazioni che erano originariamente presenti solo come metadati negli schemi, dunque in un formato non interrogabile.

1.4 Il sistema MOMIS

Considerando le problematiche descritte nel paragrafo precedente, nonché alcuni sistemi preesistenti [12, 16, 17, 18, 19], si è giunti alla progettazione di un sistema intelligente di integrazione di informazioni da sorgenti di dati strutturati e semistrutturati denominato **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources). Il contributo innovativo di questo progetto, rispetto ad altri similari, risiede nella fase di analisi ed integrazione degli schemi sorgenti, realizzata in modo semi-automatico [20, 1, 21]. Un lavoro approfondito è stato svolto anche riguardo alla fase di *query processing* ([22, 2, 23]), cioè per il processo che dalla query posta sullo schema unificato provvede a generare automaticamente le sottoquery da inviare alle sorgenti ed ad integrare i risultati. MOMIS nasce all'interno del progetto MURST 40% INTERDATA dalla collaborazione tra i gruppi operativi dell'Università di Modena e Reggio Emilia e di quella di Milano.

1.4.1 L'approccio adottato

MOMIS adotta un approccio di integrazione delle sorgenti *semantico* e *virtuale* [22]. Il concetto di semantico è stato illustrato nella sezione 1.3. Con virtuale si intende invece che la vista integrata delle sorgenti, rappresentata dallo schema globale, non viene *materializzata*, ma il sistema si basa sulla decomposizione delle query e sull'individuazione delle sorgenti da interrogare per generare delle subquery eseguibili localmente; lo schema globale dovrà inoltre disporre di tutte le informazioni atte alla fusione dei risultati ottenuti localmente per poter ottenere una risposta significativa.

Le motivazioni che hanno portato all'adozione di un approccio come quello descritto sono varie:

- la presenza di uno schema globale permette all'utente di formulare qualsiasi interrogazione che sia con esso consistente;
- le informazioni semantiche che comprende possono contribuire ad una eventuale ottimizzazione delle interrogazioni;
- l'adozione di una semantica *type as a set* per gli schemi permette di controllarne la consistenza facendo riferimento alle loro descrizioni;
- la vista virtuale rende il sistema estremamente flessibile, in grado cioè di sopportare frequenti cambiamenti sia nel numero che nel tipo delle sorgenti, ed anche nei loro contenuti (non occorre prevedere onerose politiche di allineamento);

Inoltre si è deciso di adottare un unico modello dei dati basato sul paradigma ad oggetti, sia per la rappresentazione degli schemi sia per la formulazione delle interrogazioni. Il modello comune dei dati utilizzato nel sistema (ODM_{I^3}) è di alto livello e facilita la comunicazione tra il mediatore ed i wrapper. Per definire questo modello si è cercato di seguire le raccomandazioni relative alla proposta di standardizzazione per i linguaggi di mediazione, nata in ambito I^3: un mediatore deve poter essere in grado di gestire sorgenti dotate di formalismi complessi (ad es. quello ad oggetti) ed altre decisamente più semplici (come i file di strutture), è quindi preferibile l'adozione di un formalismo il più completo possibile.

Per la descrizione degli schemi si è arrivati a definire il linguaggio ODL_{I^3}che si presenta come estensione del linguaggio standard ODL proposto dal gruppo di standardizzazione ODMG-93. Questo permette di cogliere le indicazioni emerse in ambito I^3ed al contempo di discostarsi il meno possibile dalle proposte del gruppo appena citato. Un'applicazione di ODL_{I^3}viene proposta in Appendice C nella stesura dell'esempio di riferimento. ODL_{I^3}sarà spesso utilizzato in questa tesi ma, vista anche la sua natura intuitiva, non sarà ulteriormente descritto in questa sede; una trattazione esauriente può comunque essere trovata in [2, 20, 1, 21].

Per quanto riguarda il linguaggio di interrogazione si è adottato OQL_{I^3}che adotta la sintassi OQL senza discostarsi dallo standard. Questo linguaggio richiede un maggiore sforzo dal punto di vista dello sviluppo di moduli per l'interpretazione e la gestione delle interrogazioni (implementando le funzionalità tipiche di un ODBMS), ma risulta altresì estremamente versatile ed espressivo fornendo la possibiltà di sfruttare le informazioni rappresentate

nello schema globale. Le maggiori difficoltà implementative sono quindi ampliamente giustificate da una maggiore versatilità e da una migliore facilità d'uso per l'utente finale.

Riassumendo: in MOMIS i wrapper si incaricano di tradurre in ODL_{I^3}le descrizioni delle sorgenti, i moduli del mediatore di costruire lo schema globale, mentre tutte le interrogazioni sono poste dall'utente su questo schema utilizzando il linguaggio OQL, o meglio OQL_{I^3}, disinteressandosi dell'effettiva natura ed organizzazione delle sorgenti; sarà il sistema stesso che si incaricherà di tradurre le interrogazioni in un linguaggio comprensibile dalle singole sorgenti e di riorganizzare le risposte ottenute in una unica corretta e completa da fornire all'utilizzatore.

1.4.2 L'architettura generale di MOMIS

In Figura 1.3 è illustrata dettagliatamente l'architettura generale di MOMIS. Lo schema evidenzia l'organizzazione a tre livelli utilizzata.

Livello Mediatore. Il nucleo centrale del sistema è costituito dal **Mediatore** (o *Mediator*) che presiede all'esecuzione di diverse operazioni. Per meglio comprendere i suoi compiti, è opportuno a questo punto illustrare le due fasi ben distinte in cui si articola la sua attività.

La prima funzionalià del Mediatore è quella di generazione dello Schema Globale. In questa fase il modulo del Mediatore denominato **Global Schema Builder** riceve in input le descrizioni degli schemi locali delle sorgenti espressi in ODL_{I^3}e forniti ognuno dal relativo wrapper. A questo punto (utilizzando strumenti di ausilio quali ODB-Tools Engine, WordNet, ARTEMIS) il Global Schema Builder è in grado di costruire la vista virtuale integrata (**Global Schema**) utilizzando tecniche di clustering e di Intelligenza Artificiale. In questa fase è prevista anche l'interazione con il progettista il quale, oltre ad inserire le regole di mapping, interviene nei processi che non possono essere svolti automaticamente dal sistema (come ad es. l'assegnamento dei nomi alle classi globali, la modifica di relazioni lessicali, ...). Oltre allo Schema Globale, altri "prodotti" di questa fase sono le **Mapping Table**, tabelle che descrivono il modo in cui gli attributi globali presenti nello schema generato hanno corrispondenza (*mappano*) nei vari attributi locali presenti negli schemi delle sorgenti.

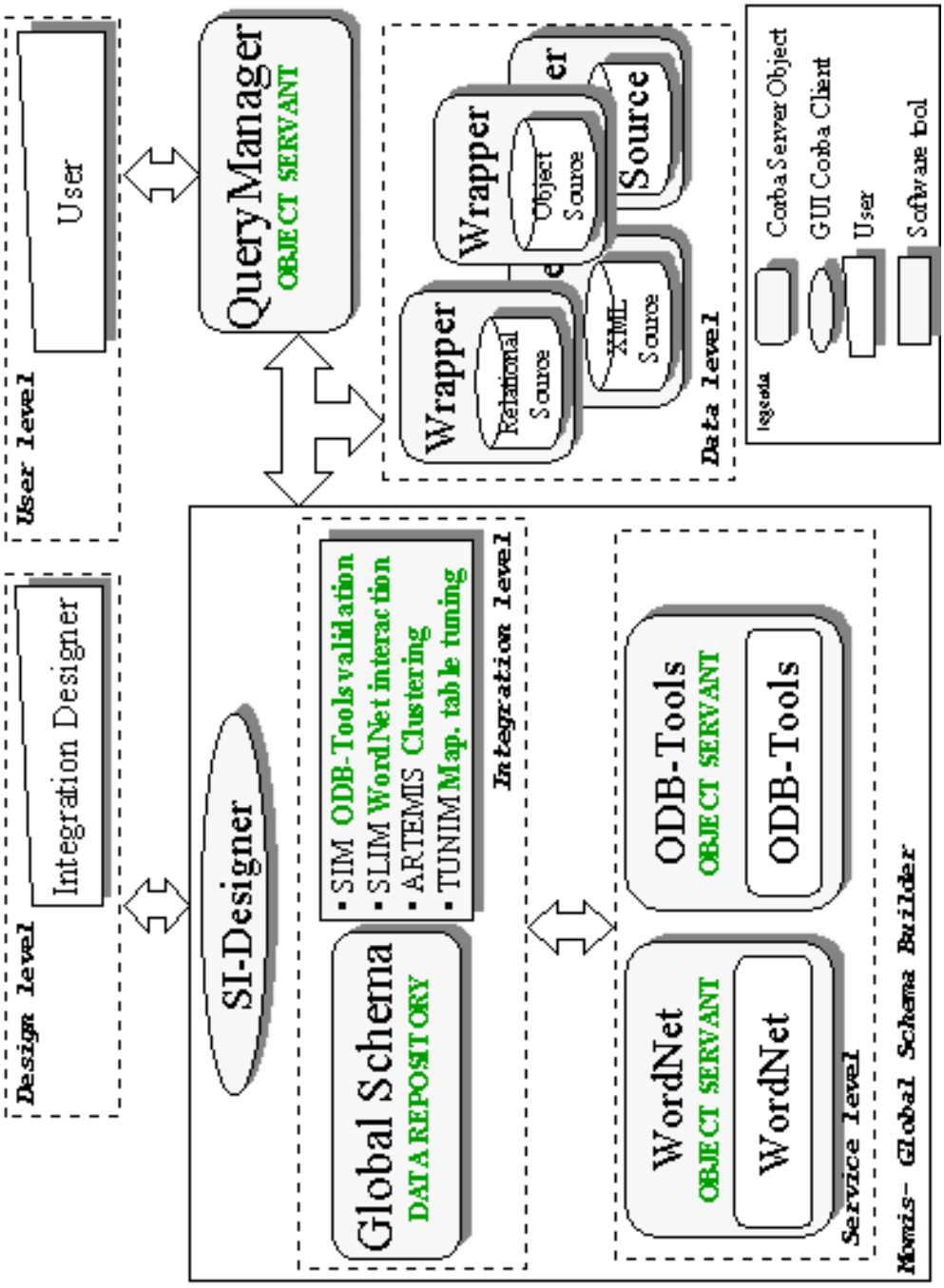

Figura 1.3: Architettura generale di MOMIS

Un secondo importante modulo che compone la struttura del mediatore è il **Query Manager** che presiede alla fase di query processing. In questa fase la singola query posta in OQL_{I^3}dall'utente sullo Schema Globale (che chiameremo *Global Query*) sarà rielaborata in più *Local Query* (anche'esse espresse in OQL_{I^3}) da inviare alle varie sorgenti, o meglio ai wrapper predisposti alla loro traduzione. Questa traduzione avviene in maniera automatica da parte del Query Manager utilizzando tecniche di logica descrittiva.

L'ultimo modulo del Mediatore è rappresentato dall'**Extensional Hierarchy Builder** il quale si occupa della generazione della Conoscenza Estensionale (Gerarchie Estensionali e Base Extension) necessaria per ottimizzare le interrogazioni.

Livello Wrapper. I **Wrapper** costituiscono l'interfaccia tra il mediatore e le sorgenti; ad ogni sorgente corrisponde un determinato wrapper ed ogni wrapper deve essere disegnato esclusivamente per la sorgente (o la tipologia di sorgenti) che sovrintenderà. Ogni wrapper ha due compiti ben precisi:

- in fase di integrazione deve fornire al Global Schema Builder la descrizione della sorgente da integrare in formato ODL_{I^3};
- in fase di query processing deve tradurre la local query (rivolta alla "sua" sorgente) che gli è stata indirizzata dal Query Manager (e che è espressa in OQL_{I^3}) nel linguaggio di interrogazione specifico della sorgente per la quale é stato progettato.

Collegate ai wrapper sono quindi le **Sorgenti**, per questo a volte si parla anche di quattro livelli. Esse sono le fonti da integrare, possono essere DataBase (ad oggetti o relazionali) o parti di essi, file system ed anche sorgenti semistrutturate.
Parte del lavoro compiuto nello svolgere la presente tesi é stato indirizzato alla realizzazione di un wrapper che presiedesse a sorgenti di tipo XML, e quindi sia sorgenti di tipo strutturato sia sorgenti di tipo semistrutturato. Nella fattispecie non é stata possibile implementare la seconda delle funzionalità individuate, in quanto non esiste ancora uno specifico linguaggio di interrogazione per sorgenti XML.

Livello Utente L'utilizzatore del sistema dovrà potere interrogare lo schema globale. L'accesso ai dati si propone del tutto simile a come avvengono le usuali interrogazioni di un DataBase tradizionale: le sorgenti ed il

modo in cui i dati vengono recuperati risultano all'utente del tutto trasparenti, in quanto é il sistema ad occuparsi di tutte le operazioni necessarie per reperire le informazioni e combinare le risposte in un'unica risposta corretta, completa e non ridondante.

In precedenza si sono citati alcuni tool di ausilio per il mediatore, vediamo una loro rapida descrizione:

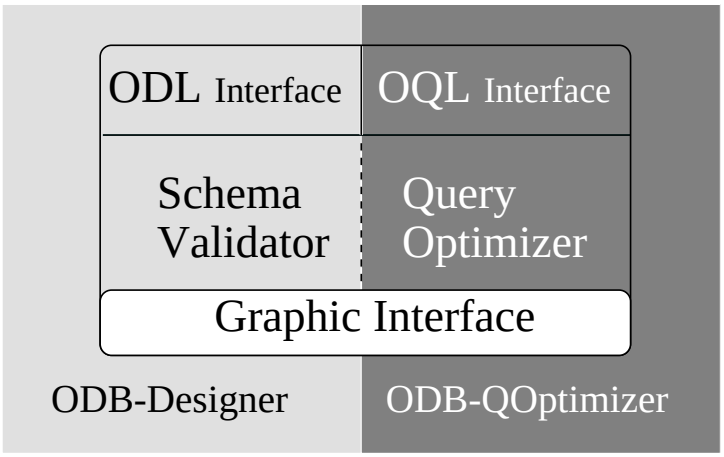

Figura 1.4: Architettura ODB-Tools

- *ODB-Tools* è uno strumento software sviluppato presso il dipartimento di Ingegneria dell'Università di Modena e Reggio Emilia [24, 25]. Esso si occupa della validazione di schemi e dell'ottimizzazione semantica di interrogazioni rivolte a Basi di Dati orientate agli Oggetti (OODB). Facciamo riferimento alla figura 1.4 per una breve spiegazione. ODB-Designer si occupa della validazione di schemi: si può inserire la descrizione di uno schema di DataBase (in ODL) ed il sistema realizzerà automaticamente la sua validazione e la sua riclassificazione; vale a dire che si occuperà di verificare che non esistano classi incoerenti (cioè che non possano essere popolate da nessun oggetto) e di determinare eventuali relazioni di specializzazione non esplicitate dallo schema stesso. ODB-Qoptimizer si occupa invece dell'ottimizzazione semantica delle interrogazioni: se si inserisce una query (in OQL) posta su di un determinato schema, questa viene automaticamente riformulata in una equivalente, ma più efficiente, sfruttando l'espansione semantica ed i vincoli di integrità.

- *WordNet* [26] è un DataBase lessicale on-line in lingua inglese. Esso è capace di individuare relazioni semantiche fra termini; cioè, dato

un insieme di termini, WordNet è in grado di identificare l'insieme di relazioni lessicali che li legano.

- *ARTEMIS* [27] riceve in ingresso il *thesaurus*, cioè l'insieme delle relazioni terminologiche (lessicali e strutturali) precedentemente generate, e sulla base di queste assegna ad ogni classe coinvolta nelle relazioni un coefficiente numerico indicante il suo grado di affinità. Questi coefficienti verranno utilizzati per raggruppare le classi locali in modo tale che ogni gruppo (*cluster*) comprenda solo classi con coefficienti di affinità simili.

1.4.3 Il wrapper XML

Come si é indicato precedentemente, il compito principale di un wrapper é quello di fornire un'interfaccia attraverso la quale il mediatore possa interagire con le sorgenti di dati. In particolare il wrapper deve fornire algoritmi attraverso i quali tradurre lo schema, descritto nel linguaggio della specifica sorgente, nel linguaggio comune ODL_{I^3}. In questo modo i moduli che compongono il mediatore, in parte agendo in maniera automatica e in parte attraverso l'interazione con il progettista, riescono a costruire la vista integrata degli schemi delle sorgenti connesse di cui si é parlato nei paragrafi precedenti.
Inoltre il wrapper deve garantire l' operazione di ricerca delle informazioni, traducendo le interrogazioni nel query language specifico della sorgente che sovrintende e preoccupandosi di gestire l'esito dell'interrogazione. I servizi che deve offrire il wrapper al mediatore possono essere pertanto evidenziati dalla seguente interfaccia, realizzata utilizzando il formato IDL, in quanto il sistema si basa su una tecnologia CORBA, alla quale ogni wrapper del sistema MOMIS deve essere conforme:

```
interface Wrapper {
     /* Get the description of the source in ODLi3. */
     string getDescription();
     /* Executes a Query OQL on the GlobalSchema specified */
     MomisResultSet runQuery( in string oql );
};
```

Quindi, per quello che concerne il primo aspetto, ovvero la traduzione dello schema, si tratta di sviluppare degli algoritmi che consentano la maggiore fedeltà possibile nella descrizione della sorgente attraverso il nuovo linguaggio. Gli aspetti critici e le soluzioni implementate nel software realizzato sono descritte nei capitoli 2 e 5

Per quello che concerne il secondo aspetto, ovvero la gestione delle interrogazioni, non si é ritenuto opportuno realizzare una soluzione implementativa in quanto non essendo ancora stato standardizzato un query language specifico per le sorgenti XML ed essendo strutturalmente semplici le query locali, si é preferito posticipare a un secondo momento l'implementazione di questa seconda parte del modulo. La semplicità strutturale delle interrogazioni locali è stata descritta in [23]. In quella sede si é infatti evidenziato che le query locali, facendo parte della categoria delle cosiddette basic query, non presentano per costruzione elementi complessi. In particolare sono espresse mediante un sottoinsieme del linguaggio OQL, all'interno del quale non sono ammessi:

- join espliciti tra classe diverse (è permessa solamente la navigazione tra aggregazioni ed associazioni per recuperare oggetti complessi),
- subquery (query innestate),
- operatori di ordinamento (order by) o di conversione (listtoset, element, flatten),
- restituzione di strutture complesse (list, array, struct).

Inoltre, un ulteriore elemento di semplificazione deriva dal fatto che il linguaggio XML non consente di esprimere dei "tipi" per quello che riguarda la definizione dei dati. Tutti i dati infatti sono del tipo PCDATA (acronimo che significa "*Parsed Computer Data*" e che può trovare l'equivalente in ODL_{I^3}nel tipo stringa). L'unica problematica, ora ipotizzabile, che sarà necessario affrontare al momento della realizzazione della seconda parte del wrapper concerne la "struttura" nella quale inserire il risultato dell'operazione. Il linguaggio XML é caratterizzato dal fatto che i suoi elementi costitutivi sono facilmente innestabili e questo comporta che spesso gli alberi che descrivono la struttura di un documento siano sviluppati con un grande livello di profondità. Allo stesso modo la risposta a una interrogazione può contenere elementi e strutture innestate a più livelli. Questo fatto però pone delle criticità rispetto alla struttura tabellare, nella quale si é ipotizzato porre i risultati di tutte le query, che é costruita in modo da riproporre per ogni riga il risultato della query che viene posta come titolo della colonna.

La struttura tabellare infatti non permette livelli innestati e quindi sarà necessario realizzare delle soluzioni per permettere le integrazioni dei risultati delle query. Tali soluzioni possono o prevedere particolari interrogazioni la cui risposta non consenta struttura innestate oppure lo sviluppo di algoritmi di "*splitting*" degli elementi risultanti sulla tabella che li memorizza.

Capitolo 2

I Dati Semistrutturati

2.1 Introduzione

La maggior parte dei dati di tipo elettronico esistenti in Internet si trova all'interno di documenti non strutturati (come quelli in HTML o SGML) ed é quindi rappresentata attraverso formati di dati di tipo non standard. La principale caratteristica dei dati di questo tipo è quella di avere una struttura generalmente irregolare, scarsamente definibile a priori e, anche se conosciuta, soggetta a cambiamenti frequenti e senza scadenze preordinate (e.g. i dati nel Web). I dati di questo tipo sono noti in letteratura con il nome di *dati semistrutturati* [28].

La ricerca sui dati semistrutturati ha cercato di estendere le tecniche usualmente utilizzate per la gestione dei database tradizionali ai dati con struttura irregolare, sconosciuta e soggetta a frequenti modifiche. Da una parte, quindi, l'attenzione é stata rivolta all'individuazione di un modello logico per questa nuova tipologia di dati, dall'altra si è arrivati all'elaborazione di linguaggi di query specifici per i dati semistrutturati. Fra questi ultimi hanno assunto particolare rilevanza Lorel, UnQL, MSL, e StruQL.

La ricerca sui dati semistrutturati ha inoltre focalizzato il proprio interesse principalmente su altri tre aspetti: l'integrazione di dati eterogenei, la gestione di strutture dati, la gestione di siti Web. A questi obiettivi che vogliono fondare le basi teoriche all'interno delle quali operare, fa da corollario la risoluzione di altre problematiche, più prettamente operative, relative in primo luogo alla ricerca di una formalizzazione per i dati semistrutturati, e all'uso di tecniche di ottimizzazione e di indexing.

La nascita dell'XML [29](Extensible Mark-up Language) ha suggerito nuove sfide nell'ambito dell'uso dei dati semistrutturati . L'XML é infatti un nuovo standard approvato dal World Wide Web Consortium che si può

ritenere possa diventare di fatto un nuovo formato di scambio dati per il Web. Questo possibile uso del linguaggio è motivato dal fatto che l'XML da una parte è orientato allo scambio di dati fra gli elaboratori (come l'HTML é orientato allo scambio di documenti fra persone), mentre dall'altra l'XML é in grado di esprimere molte caratteristiche dei dati semistrutturati: può rappresentare infatti dati con strutture irregolari e non conosciute in anticipo, e può modellare dati che cambiano frequentemente e senza preavviso. In altre parole é semplice convertire i dati sia struttuturati sia semistrutturati appartenenti a un determinato sorgente nel linguaggio XML, e poi utilizzare questo linguaggio come veicolo di condivisione dei propri dati con altri presenti nel Web. Il linguaggio XML potrebbe perciò rappresentare allo stesso tempo sia sorgenti di dati strutturati sia sorgenti di dati semistrutturati. Se questo è vero, la ricerca sui dati semistrutturati e lo studio delle applicazioni del linguaggio XML devono confluire in un'unica direzione, e l'XML può perciò arrivare a rappresentare un mezzo potente attraverso il quale gestire l'integrazione tra le due diverse tipologie di dati.

Per questo motivo, affinché le applicazioni XML raggiungano la loro piena potenzialità, occorre realizzare gli strumenti adeguati per processare i dati in questo nuovo formato. I Web tools esistenti infatti sono orientati a *operazioni sui documenti*. Per operare in XML è necessario disporre di strumenti orientati a *operazioni sui database*, cioè che siano in grado di effettuare operazioni di estrazione dati, operazioni di integrazione dei dati, operazioni di trasformazione dei dati e operazioni di memorizzazione dei dati. La ricerca fatta finora sui dati semistrutturati può offrire alcune risposte ai problemi che possono manifestarsi nell'uso di database in XML. Può infatti essere vantaggioso estendere all'XML lo studio effettuato per creare un *query language* per i dati semistrutturati. Un'operazione di questo tipo ha portato alla nascita di un nuovo linguaggio, l'XML-QL [30, 31], che recentemente è stato posto all'attenzione del World Wide Web Consortium. L'uso del linguaggio XML comporta però la risoluzione di problemi a cui la ricerca sui dati semistrutturati non ha ancora trovato risposta. Ad esempio non esiste ancora una soluzione completa per quello che riguarda le problematiche inerenti l'inferenza di un modello (problema non ancora considerato), la creazione di database distribuiti (considerato per ora non rilevante) e la memorizzazione dei dati (non esiste ancora una soluzione risultata vantaggiosa rispetto le altre).

Nella sezione successiva si analizzerà il modello OEM quale principale modello proposto in letteratura per rappresentare i dati semistrutturati. A partire dalla concettualizzazione proposta verranno presentati due linguaggi attraverso i quali descrivere tali dati: il linguaggio utilizzato nell'ambito del progetto MOMIS, l'ODL_{I^3}, e il nuovo linguaggio XML il cui modello sotteso,

pur non essendo l'OEM, ben si adatta a rappresentare sorgenti semistrutturate. Verrà analizzata inoltre una estensione dell'XML, XML-Schema, linguaggio non ancora reso standard, ma che al momento permette di esprimere una semantica maggiore rispetto a XML1.0. Si passerà poi nei capitoli successivi ad analizzare le caratteristiche che un query language deve soddisfare per poi presentare in dettaglio XML-QL che in questo momento meglio si adatta alle esigenze individuate. Verrà inoltre presentato il linguaggio XML-GL [32], linguaggio di capacità espressive più limitate rispetto al precedente ma che permette di effettuare query in modo grafico. Lo scopo è quello di estendere le potenzialità di MOMIS da una parte introducendo un wrapper che permetta di interfacciarsi ai dati XML, dall'altra valutando se risulta conveniente produrre uno schema globale, espressione degli schemi rappresentativi delle diverse sorgenti da analizzare, non più in ODL_{I^3}, ma in XML, per poi operare su esso attraverso il query language opportuno.

2.2 Rappresentazione di dati semistrutturati

In questa sezione sarà esaminato in un primo momento il modello di dati che è maggiormente utilizzato per rappresentare i dati semistrutturati. É opportuno sottolineare che il modello OEM si colloca all'interno di un contesto, quello della formalizzazione di modelli di dati, nel quale la ricerca ha già individuato dei paradigmi di riferimento (ad esempio il modello relazionale e quello object oriented). Se nell'ambito della modellizzazione dei dati semistrutturati tutte le proposte possono essere considerate come evoluzioni di questo paradigma, nell'ambito dei linguaggi descrittivi la variabilità è maggiore. Sono stati sviluppati infatti diversi linguaggi descrittivi: ad esempio nell'ambito del progetto MOMIS si é lavorato sulla specializzazione del linguaggio standard ODL per operare nell'ambito dell'integrazione intelligente delle informazioni. Il linguaggio utilizzato in quel contesto, ODL_{I^3}, sarà confrontato nei paragrafi successivi con l'XML, il cui modello strutturale, come si vedrà, si adatta a rappresentare in maniera ottimale i sorgenti semistrutturati.

L'esigenza di racchiudere all'interno di una tipologia a se stante, quella dei dati semistrutturati, i dati aventi determinate caratteristiche viene evidenziata tra gli altri anche da P. Buneman [28]. La ricerca effettuata da Buneman focalizza il proprio obiettivo nell'individuazione di un modello sotteso ai dati. Viene evidenziato infatti che esistono dati che non possono essere rappresentati attraverso i modelli standard (come quelli presentati dal gruppo ODMG), per il fatto che in generale di essi non si ha la conoscenza a priori di una struttura sottesa. Ci sono inoltre alcune forme di dati per le quali avere questa conoscenza è impossibile.

É in questo contesto che il paradigma object oriented, inizialmente pensato come possibile candidato per i dati semistrutturati, va in crisi: la variabilità che la struttura dati potrebbe avere necessita infatti di utilizzare delle rappresentazioni che non siano omogenee e statuarie, ma che possano godere di una certa flessibilità. Tali situazioni possono accadere nel caso in cui si abbia a che fare con istanze di oggetti nelle quali possano non essere presenti tutti gli attributi previsti, oppure viceversa per i quali si abbia una multipla occorrenza di un attributo. Inoltre occorre tenere in considerazione che un attributo potrebbe essere modellato diversamente in oggetti diversi e quindi informazioni collegate semanticamente potrebbero essere rappresentate in maniera diversa nei vari oggetti. In altre parole quando da dati di questo tipo si cerca di inferire una struttura di modellazione comune, ed è questa l'ottica a cui si è indirizzata la ricerca scientifica, è necessario affrontare e risolvere tutte le problematiche di tipo ontologico e semantico che normalmente devono essere affrontate nel caso di integrazioni di dati eterogenei e che sono state anche affrontate nell'ambito delle prime fasi di progettazione di Momis [1]. L'esigenza di individuare un modello unico, in generale pensato avente la struttura di un grafo o di un albero nel quale sia gli archi sia i nodi, intermedi e terminali, assumono significato semantico, rappresenta un elemento unificante e di approdo per i dati semistrutturati. Sono stati individuati, nei modelli proposti in letteratura [28, 33], diversi significati per i nodi, le foglie e gli archi. Il modello che è ormai considerato come base teorica é l'**OEM** che viene in seguito definito.

2.2.1 Object Exchange Model

L'**OEM** (Object Exchange Model) [34, 35] è forse il modello più utilizzato per rappresentare i dati semistrutturati. Introdotto nell'ambito del progetto di ricerca per l'integrazione dei dati Tsimmis [10], l'OEM è stato poi utilizzato in altri progetti. I dati vengono rappresentantati attraverso delle *collezioni di oggetti*, che possono assumere un valore atomico o complesso. Si parla di valore atomico quando esso appartiene ad alcuni modelli base (integer, real, string,...). Il valore di un oggetto complesso è invece rappresentato attraverso un insieme di coppie (attributo, oggetto), dove il termine attributo in questo contesto identifica una stringa tratta dall'insieme $\mathcal{A}$ dei nomi di attributi. Ogni oggetto è poi individuabile in maniera univoca mediante un unico *oid* (object identifier). In questo modo il modello OEM introdotto può essere rappresentato graficamente mediante un grafo ad albero i cui nodi intermedi individuano gli oggetti, i cui archi sono etichettati con gli attributi e le cui foglie hanno associato un valore atomico.

Formalmente un dato semistrutturato puó essere essere definito attraverso

l'insieme *G=(V,E,r,v)*, dove *V* rappresenta l'insieme dei nodi ed è suddiviso in nodi atomici e complessi (V= $V_c \cup V_a$), gli archi sono rappresentati da *E*= $V_c \times \mathcal{A} \times V, r \in V$ rappresenta il nodo radice e *v*: $V_a \longrightarrow \mathcal{D}$ assegna i valori all'oggetto atomico, con D universo dei possibili valori atomici.

Il modello introdotto può essere paragonato al modello **relazionale** e al modello **object oriented**. Per quello che riguarda il confronto con il primo, occorre sottolineare che una **relazione** può essere definita come un *insieme di record*. Attraverso il formalismo introdotto con OEM sia gli insiemi, sia i record possono essere facilmente rappresentati. Infatti gli insiemi sono oggetti OEM nei quali tutti gli attributi sono gli stessi, mentre i record sono oggetti nei quali gli attributi sono distinti.

Name	Code
Peter	o12
Serge	o176
Dan	065

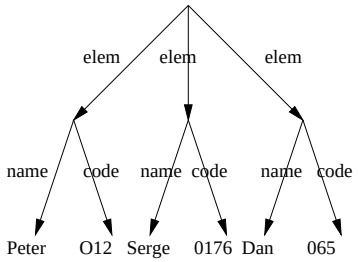

Figura 2.1: Dati relazionali attraverso la modellizzazione OEM

In questo modo è possibile rappresentare i dati relazionali come caso particolare di dati semistrutturati.
OEM differisce dai modelli object oriented per alcuni aspetti sostanziali determinati dallo scopo per il quale i due modelli sono stati sviluppati. Il primo, meno interessato a rappresentare realtà complesse, è molto più semplice. L'OEM supporta unicamente l'innestamento e l'identificazione degli oggetti, mentre altre caratteristiche come le classi, i metodi, e tutte le caratteristiche connesse con l'ereditarietà sono omesse. Un'altra differenza sostanziale fra i due modelli inoltre è manifestata dal fatto che OEM usa il sistema del grafo ad archi etichettati al posto dello schema.

2.2.2 Dati semistrutturati e Object Pattern

Il modello dati utilizzato nell'ambito del progetto MOMIS è anch'esso derivante dalla modellazione introdotta con l'OEM. In quel contesto viene definito il significato di oggetto semistrutturato e a partire da esso viene introdotto il concetto di object pattern.

Definizione 1 (Oggetto Semistrutturato) *Un* oggetto semistrutturato, *indicato con so, è una tripla nella forma* $\langle$`id`, `label`, `value`$\rangle$ *dove:*

- `id` *è l'idenficatore dell'oggetto;*
- `label` *è una stringa che descrive ciò che l'oggetto rappresenta;*
- `value` *è il valore dell'oggetto, che può essere atomico o complesso.*

Per quanto concerne i valori atomici, essi possono essere *integer*, *real*, *string*, *image*, mentre un valore complesso é rappresentato da un insieme di coppie (*id,label*) che individuano ulteriori oggetti semistrutturati. In analogia a questa caratterizzazione, si definisce *oggetto semistrutturato atomico* un oggetto a valore atomico, e *oggetto semistrutturato complesso* un oggetto il cui valore è complesso.

L'oggetto complesso può quindi essere pensato come il padre di tutti gli oggetti che definiscono il suo valore (detti oggetti figli). In generale, un dato oggetto può avere uno o più genitori: nella notazione adottata si indica che un oggetto so' è figlio di un altro oggetto so tramite $so \rightarrow so'$ e indichiamo con $label(so)$ l'etichetta di so. Perciò, intuitivamente, una sorgente semistrutturata S può essere vista come un grafo orientato, in cui i nodi rappresentano gli oggetti semistrutturati e gli archi orientati rappresentano la relazione tra un oggetto complesso e tutti gli oggetti atomici che ne individuano il valore.

Nei modelli di dati semistrutturati, le etichette sono il più possibile esplicative e possono essere utilizzate per raggruppare gli oggetti assegnando la stessa etichetta ad oggetti correlati. Per questa ragione, data una sorgente semistrutturata S, ponendosi come obiettivo quello di scoprire i tipi differenti di oggetti memorizzati, è necessario agire attraverso un approccio basato sui pattern, che tenga conto della semantica di mondo aperto caratteristica delle Description Logics. L'approccio da adottare si può riassumere brevemente nel seguente modo: tutti gli oggetti complessi so di S sono partizionati in base alle loro etichette e valori in insiemi disgiunti, indicati come set_l, in modo tale che tutti gli oggetti che fanno riferimento allo stesso insieme abbiano la stessa etichetta l. Successivamente ad ogni insieme è associato un *object pattern*. Formalmente, un *object pattern* è definito nel seguente modo:

Definizione 2 (Object pattern) *Dato un insieme di oggetti set_l definito sulla sorgente semistrutturata S. L'*object pattern *dell'insieme set_l è una coppia nella forma $\langle l,A \rangle$, dove l è l'etichetta degli oggetti correlati all'insieme set_l, ed $A = \bigcup label(so')$ tale che esiste almeno un oggetto $so \in set_l$ con $so \rightarrow so'$.*

Partendo da questa definizione, è facile capire come un *object pattern* sia rappresentativo di tutti i diversi oggetti che descrivono lo stesso concetto in una data sorgente semistrutturata. Più specificatamente, l indica il concetto e A le proprietà (dette anche attributi) che caratterizzano il concetto all'interno della sorgente. Poiché gli oggetti semistrutturati possono essere eterogenei, a volte le etichette nell'insieme A di un *object pattern* possono essere definite solo per alcuni oggetti dell'insieme set_l e non per tutti: si definiscono queste particolari etichette con il termine di "optional", e si indicano con il simbolo "*".

Come detto la descrizione degli *object pattern* segue la semantica di mondo aperto *open world semantics* tipica dell'approccio seguito in Description Logics [36, 37, 38]. Così facendo, gli oggetti di un pattern condividono una struttura minima rappresentata dalle proprietà non opzionali, ma possono avere altre proprietà addizionali, quelle definite appunto "optional" che caratterizzano il singolo oggetto (o un numero ridotto di oggetti). In questo modo, gli oggetti di una sorgente semistrutturata possono evolversi ed aggiungere nuove proprietà, pur rimanendo istanze valide dell'*object pattern* corrispondente e quindi mantenendo efficaci le query sull'*object pattern*.

L' approccio descritto si pone quindi l'obiettivo di distinguere, all'interno di una sorgente semistrutturata, la parte relativa alla struttura da quella dei dati: il livello intensionale contiene ciascuna descrizione degli *object pattern* secondo la descrizione ODL_{I^3}. Il livello estensionale contiene invece gli oggetti semistrutturati).

2.3 Un modello dati per XML

Come si è evidenziato nelle precedenti sezioni, quando si opera con sorgenti di dati semistrutturati, non si ha una dicotomia netta tra struttura e dati, come avviene nei paradigmi classici, ma si ha una commistione semantica tra quella che è la modellizzazione strutturale e quella che è l'informazione. E' evidente quindi che la distinzione tra modello dati e linguaggio descrittivo assume un significato minore. In quest'ottica si è sviluppato l'**XML**, "Extensible Mark-up Language", uno standard recentemente approvato dal W3C. L'XML nasce come linguaggio di descrizione e per le sue caratteristiche si pensa possa diventare un formato universale di scambio dati nel

Web. E' necessario porre in evidenza che questo linguaggio, nato per operare all'interno del WWW, viene sviluppato con lo scopo di incentivare lo scambio di dati in maniera strutturata attraverso le varie sorgenti di informazioni inserite nella rete Internet. XML nasce quindi in contrapposizione a quello che è il linguaggio di scambio documenti nella rete, l'HTML [39]. Lo scopo è infatti quello di creare un linguaggio che sia orientato allo scambio elettronico di dati "machine-readable", in contrapposizione all'HTML creato principalmente per lo scambio di documenti "human-readable". Questa visione "orientata ai dati" si traduce principalmente in tre sue principali peculiarità [40]:

- L'utente può definire i tag, che rappresentano lo strumento per descrivere la realtà da modellare e che contestualmente racchiudono il valore della qualificazione introdotta, come preferisce (per quello che riguarda i nomi e il numero)
- La struttura del documento può essere dotata di elementi innestati a qualsiasi livello di profondità
- Qualsiasi documento può contenere una rappresentazione opzionale della propria grammatica, chiamata DTD, che può essere utilizzata attraverso specifiche applicazioni.

Anche il dato rappresentabile in XML può essere definito come autodescrivente e mostra forti somiglianze con il dato semistrutturato. Le analogie più facilmente rilevabili possono essere riassunte nell'elenco sottostante dove viene messo in evidenza in che modo il modello OEM può essere trasferito in un modello derivante dall'XML [41]:

- attributo OEM $\longrightarrow$ tag XML
- oggetto OEM $\longrightarrow$ elemento XML
- valore atomico di attributo (string, real, image ...) OEM $\longrightarrow$ insiemi di caratteri in XML (sono ammessi unicamente dati di tipo stringa)

Attraverso lo schema proposto che suggerisce una possibile traduzione degli elementi costitutivi dell'OEM in un corrispettivo linguaggio XML viene messo in evidenza come facilmente si possa impiegare il nuovo standard anche per la rappresentazione dei dati semistrutturati. Infatti, come già sottolineato, XML nasce come linguaggio di mark-up senza avere un data-model associato. Per questo motivo un uso di tale linguaggio nel contesto dei semistrutturati non rappresenta una forzatura, ma una corretta applicazione delle potenzialità introdotte con il linguaggio stesso. In quest'ottica

il possibile parallelismo tra XML (che rappresenta la sintassi) e il modello di dati semistrutturati analizzato (che rappresenta la semantica) contiene però alcuni punti *"critici"* che è necessario analizzare in maniera più dettagliata.

- **Concetto di Ordine**: I documenti XML hanno per definizione un ordine, mentre il modello per i dati semistrutturati analizzato non introduce tale concetto. L'introduzione di un ordine nel modello OEM non rappresenta certamente una sfida dal punto di vista tecnico, e si può anticipare che, il modello di rappresentazione dei dati XML che verrà proposto in seguito, terrà conto della possibilità di operare con dati ordinati. È sicuramente più complesso invece utilizzare il concetto di ordine all'interno di un linguaggio di query, e questo sarà effettuato, seppure in maniera rozza e attraverso una gestione complessa delle operazioni, utilizzando XML-QL.

- **Concetto di Attributo**: Gli elementi XML possono contenere degli attributi, che semanticamente rappresentano un concetto diverso da quello di attributo nel modello OEM. Ad esempio il codice seguente, scritto in XML, utilizza un attributo:

  ```
  <ristorante>
  <nome naz=''ita''>Roma</nome>
  </ristorante>
  ```

  In questo caso naz è un attributo e "ita" è il suo valore. Ogni tag XML può contenere diverse coppie attributo-valore, in cui il valore è rappresentato da una stringa e l'attributo é racchiuso nella definizione del tag. Ci sono due sistemi per fare in modo che l'attributo XML non perda di significato e possa essere quindi trasferito nel modello di dati semistrutturati OEM. Il primo sistema elimina la distinzione tra attibuto e tag portando a considerare il primo allo stesso livello del tag:

  ```
  <ristorante>
  <nome>Roma</nome>
  <naz>ita</naz>
  </ristorante>
  ```

  Il secondo sistema, utilizzabile nel contesto del modello OEM, consiste nell'includere gli attributi XML nel modello associando a ogni nodo un oggetto costituito da un insieme di coppie (attributo-XML,valore).

- **Concetto di Riferimento**: A ogni elemento è possibile associare in modo esplicito un oid ed è possibile quindi a partire da un tag effettuare un riferimento a un altro elemento utilizzando dei particolari attributi. Per definizione l'XML lascia all'applicazione che legge il documento il ruolo di definire come interpretare operativamente i riferimenti. L'unico vincolo richiesto è che tutti gli oid referenziati siano effettivamente presenti. E' attraverso il concetto di riferimento che si può realizzare una struttura ad albero che permetta anche una "navigazione" trasversale fra i nodi che la costituiscono.

- **Concetto di Link**: I documenti XML possono, anche se utilizzando una sintassi per ora non standard, contenere link che indirizzano verso altri documenti XML. Come i tag *anchor* in HTML, i link possono oltrepassare i confini dettati dal singolo server. Ignorare i link nel modello logico potrebbe portare a inefficienze, soprattutto quando si tratterà di problematiche relative alle query nelle pagine XML. In [42] gli autori considerano tre generi di link nel modello dati che propongono: quelli intra-documento, quelli inter-documento ma intra-sito e quelli inter-sito. La gestione delle query proposta in quel contesto arriva a distinguere la posizione dell'informazione da reperire all'interno delle classi individuate. L'XML-QL, come si vedrà, non supporta ancora tale possibilità, che comunque nella modellizzazione della struttura é da tenere in considerazione per eventuali sviluppi futuri.

É necessario sottolineare inoltre che i documenti XML possono fornire alle applicazioni uno strumento, seppure opzionale e non sempre utilizzabile, la **DTD**, *Document Type Definition*, nella quale sono contenute sia le grammatiche regolari sia l'innestamento dei tag. L'uso della DTD permette di superare un impasse all'interno del quale tradizionalmente i ricercatori si sono soffermati. Si é più volte sottolineato il fatto che il dato semistrutturato é per definizione autodescrivente e di conseguenza lo schema fa parte del dato e la struttura é impredicibile a priori. Questa caratteristica genera dei forti inconvenienti sia nella memorizzazione del dato, in quanto la struttura deve essere memorizzata contestualmente alla memorizzazione di ogni singolo item, sia nella gestione delle query. In questo caso infatti da una parte risulta difficile valutare l'efficienza della query in quanto in alcune occasioni si richiede di attraversare tutto il grafo, dall'altra risulta complessa la formulazione della query stessa, in quanto non si ha la completa conoscenza delle informazioni alle quali si può accedere.

La necessità di creare una struttura all'interno del quale racchiudere il documento XML é condizione necessaria per lo sviluppo di applicazioni che

mirino a utilizzare il linguaggio come veicolo di scambio dati. Soltanto attraverso la presenza di una struttura é possibile trovare una soluzione alle domande di data extraction, data integration, data traslation, data storage alle quali un DBMS deve dare risposta. Verranno analizzati in seguito le possibilità offerte dall'XML nell'ambito di tali problematiche.

Si descrivono di seguito due modelli dati per l'XML che sono stati introdotti all'interno di [31]. Tali modelli, pur essendo stati studiati per operare all'interno di uno specifico query language, presentano delle caratteristiche che li rendono universali.

2.3.1 Modello XML ordinato e non ordinato

Si descrivono due modelli di rappresentazione di dati che possono essere descritti utilizzando il linguaggio XML e che possiedono tutte le caratteristiche necessarie per supportare un linguaggio per query. Il modello non ordinato é quello che presenta meno vincoli strutturali e pertanto può essere più facilmente utilizzato. Il secondo modello, che si può considerare come una evoluzione del primo, a fronte di una maggiore rigidità dei vincoli di costruzione, permette la scrittura di interrogazioni più puntuali e quindi maggiormente selettive.

Modello non ordinato. Questo modello si basa sulla definizione di grafo non ordinato.

Definizione 3 (Grafo XML non ordinato) *Un grafo XML non ordinato consiste di:*

- *Un grafo* `G` *nel quale ogni nodo é rappresentato attraverso un'unica stringa chiamata* `OID`, object identifier*;*
- *Ogni arco orientato di* `G` *é etichettato associando a esso il nome degli elementi "tag";*
- *Ogni nodo é etichettato con un insieme di coppie* `(attributo-valore)`
- *Le foglie rappresentano valori di tipo stringa*
- `G` *ha un nodo che é distinto rispetto agli altri chiamato* `root`

Il modello non ordinato considera i dati come valori facenti parte di una struttura ad albero nella quale non ha importanza l'ordine in cui rami e nodi sono posti. In questo contesto si può quindi effettuare una distinzione

tra informazione che é veicolata dal dato e informazione, che non appartiene strettamente al dato, ma che é generata dalla posizione in cui si colloca l'informazione. Nella figura seguente si può osservare una possibile rappresentazione di grafo non ordinato.

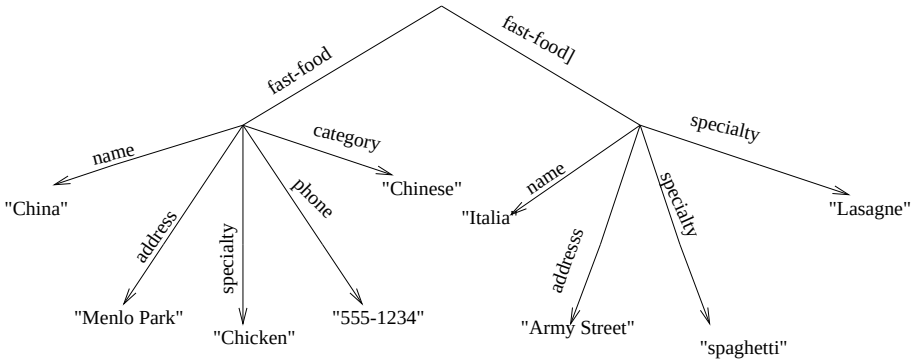

Figura 2.2: Il modello non ordinato

Il modello introdotto deve soddisfare un unico vincolo: ogni nodo non può avere due o più archi uscenti aventi le stesse etichette e gli stessi valori. In queste condizioni risulta evidente che sia tra due nodi ci può essere un solo arco avente un'etichetta data, sia un nodo non può avere due foglie figlie con la stessa etichetta e lo stesso valore.

Modello ordinato. In un grafo XML ordinato esistono due tipologie di ordini, un ordine globale e un ordine locale. L'ordine globale é l'ordinamento che ogni nodo acquisisce in virtù dell'ordine nel quale ogni nodo viene definito nel documento. Rappresentato un ordine globale é possibile definire un ordine locale su ogni arco uscente di ogni nodo. In questo modo é perfettamente lecito che da un nodo escano più archi aventi la stessa etichetta e lo stesso valore, in quanto la differenziazione é definita dall'ordinamento fornito all'arco. In XML i documenti sono sempre di tipo ordinato in quanto la DTD impone un ordine predefinito agli elementi.

Convenzionalmente si indica fra parentesi tonde la numerazione globale, mentre tra parentesi quadre é indicata la numerazione locale. Ovviamente attraverso questa seconda formalizzazione si riesce a esprimere una seman-

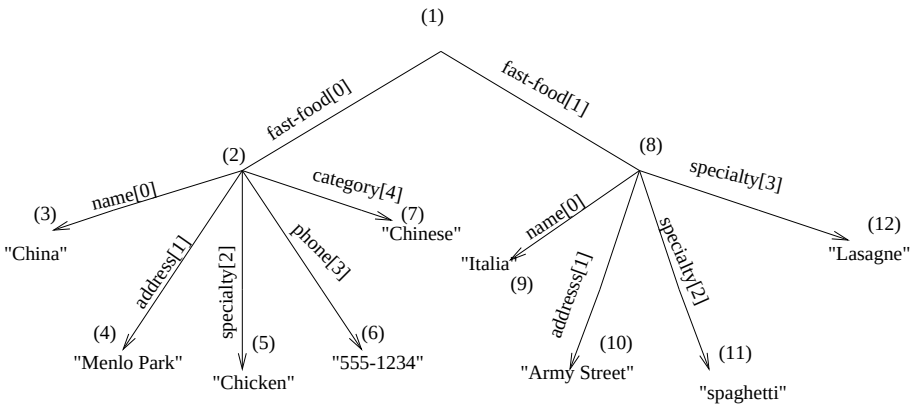

Figura 2.3: Il modello ordinato

tica più ricca di quanto non lo si possa fare con la formalizzazione precedentemente introdotta, questo però a scapito della riconoscibilità di elementi modulari nella struttura introdotta e di conseguenza a scapito della facilità di definizione di un linguaggio di estrazione dei dati. Per questo motivo potendosi considerare il modello ordinato come una estensione di quello non ordinato si potrà utilizzare il primo, più complesso sia in fase di definizione sia in fase di estrazione dei dati, quando é necessario effettuare delle operazioni riguardanti l'ordinamento dei dati, mentre il secondo potrà essere utilizzato in tutti gli altri casi.

Adoperando entrambe le formalizzazioni é necessario disporre di strumenti che consentano di operare tenendo in considerazione i cosiddetti elementi condivisi, ovvero occorre disporre di strumenti che consentano di operare agevolmente nella costruzione di un albero i cui rami possano far riferimento a nodi non appartenenti alla stessa sezione dello stesso albero oppure appartenenti a un albero differente. L'XML fornisce degli elementi di supporto in quanto riserva alcuni attributi per la gestione di tali casi. Infatti gli attributi **ID, IDREF e IDREFS** sono utilizzati rispettivamente per specificare un identificatore unico di un elemento, per effettuare un riferimento ad un altro elemento, per effettuare riferimenti a molteplici elementi. In figura 2.4 viene evidenziato un possibile esempio di uso di attributi ID/IDREF.

Nel modello introdotto infatti si é specificato che ad ogni nodo é associato un unico oggetto identificatore (OID), che corrisponde in pratica all'attributo definito come ID se esso é presente, oppure a un OID generato automaticamente se l'attributo ID non é presente. Graficamente la differenza logica di questa tipologia di attributi dagli altri attributi, nei quali esistono coppie

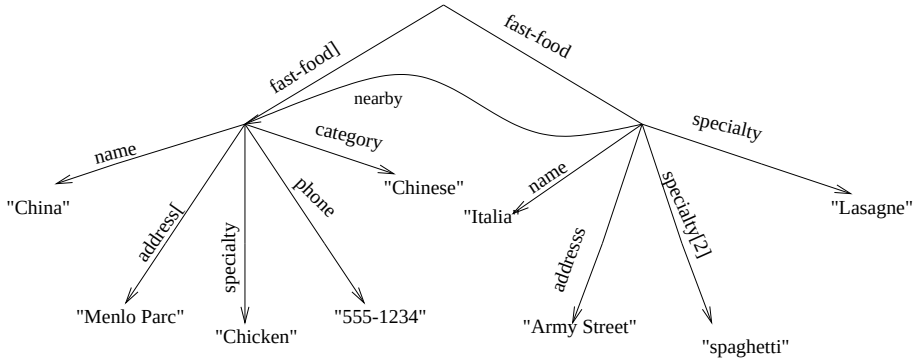

Figura 2.4: Esempio di uso degli attributi ID e IDREF

attributo - nome associate a ogni nodo, non viene evidenziata da un apposito formalismo grafico: un attributo IDREF infatti é rappresentato con un arco orientato dall'elemento referente a quello referenziato, etichettato con il nome dell'attributo. Nonostante questo è sempre possibile, dato un grafo generato da un documento XML, ricostruire il documento originale nel quale le due tipologie di attributi hanno una semantica differente. Infatti, dato un nodo con multipli archi entranti, si riesce sempre a discernere tra archi che identificano elementi e archi che identificano attributi di tipo IDREF che si riferiscono a quel nodo.

Valori Scalari. É necessario risolvere un ultimo problema per quello che riguarda la rappresentazione mediante grafi del linguaggio XML. Si é puntualizzato il fatto che ogni nodo di tipo foglia debba necessariamente contenere un valore. In questo modo però si trovano a esistere dei frammenti XML che non possono essere rappresentati direttamente attraverso un grafo. Ad esempio il codice seguente non ha una sua definizione in grafo.

```
<address>Park Avenue <zipcode>98654</zipcode></address>
```

In tali casi é necessario introdurre degli archi supplementari per rafforzare il fatto che sia possibile associare a ogni etichetta un unico valore.

```
<address><PCDATA>Park Avenue</PCDATA>
<zipcode>98654</zipcode></address>
```

Per concludere occorre ricordare che un grafo XML può essere il risultato di una operazione di parsing di un documento XML nonché essere l'esito

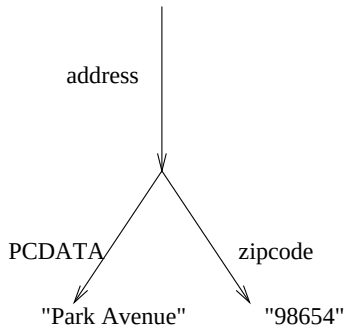

Figura 2.5: Valori scalari

di operazioni di query, o trasformazione di un grafo XML già esistente. In generale un grafo non ha un'unica rappresentazione per il fatto che sia l'ordine degli archi non é specificato (nel modello non ordinato), sia non esiste un'unica modellizzazione dei nodi condivisi.

2.4 Conclusioni

É possibile osservare che le due modellizzazioni proposte, quella cioé inferita dall'analisi dei dati semistrutturati e la seconda proposta per i documenti XML, hanno molti punti di contatto. In particolare é evidente come entrambi i modelli proposti per la rappresentazione di documenti XML possano essere considerati come estensioni dell'OEM. La parte di informazione che il modello XML é in grado di dare e che non è contemplata dall'OEM é infatti composta sia dalla possibilità di inserire degli attributi a livello di nodo, sia dalla possibilità di avere un ordinamento dei nodi.

La possibilità di avere attributi a livello di nodo, che permette all'XML sia di rappresentare attraverso una semantica più ricca l'elemento a cui sono riferiti sia permette a un elemento di referenziare altri elementi di essere referenziato da altri elementi, può essere facilmente estesa al modello OEM. Qualora infatti sia necessario estendere la prima possibilità all'OEM é necessario "elevare logicamente di livello" l'attributo in esame, facendogli assumere il ruolo di nuovo tag il cui valore semantico è puramente attributivo. Per quello che riguarda invece la possibilità di referenziare attraverso attributi nuovi oggetti, il modello OEM prevede una possibilità attraverso l'uso degli identificatori di oggetto di cui ogni oggetto é dotato. La traduzione di tale meccanismo, se non correttamente effettata, potrebbe portare a una perdita di conoscenza da parte dell'elemento figlio dell'identità dell'elemeno

referenziante, cosa spesso irrilevante per quello che riguarda la semantica delle informazioni.

Attraverso l'ordinamento dei nodi e un linguaggio query specifico é possibile estrarre informazioni maggiori e aumentare in questo modo la selezione nella ricerca delle informazioni. Non sempre però in realtà per quello che riguarda il contenuto delle informazioni che i dati semistrutturati racchiudono, la conoscenza dell'ordinamento della struttura porta a reali informazioni utili aggiuntive.

É poi necessario sottolineare come ai fini del progetto MOMIS la definizione stessa di Object pattern introdotta in quel contesto si possa facilmente esportare all'interno di un modello basato sull'XML. É sufficiente infatti sostituire nella definizione stessa, in accordo con le modalità di traduzione introdotte in precedenza, il concetto di object OEM nel concetto di elemento XML, che la definizione così corretta risulta comunque essere significativa e facilmente utilizzabile nel contesto dei dati semistrutturati.

Di grande rilevanza inoltre il fatto che il linguaggio XML rappresenta uno standard di fatto e si sta rivelando uno strumento efficace sulle cui modalità di uso però c'e' forte discussione. In particolare ai vantaggi di utilizzare un linguaggio descrittivo riconosciuto come standard per il web, si associa lo svantaggio della mancanza di specificità che questo linguaggio possiede. Per risolvere questo problema l'attenzione è stata posta su alcune linee principali: da una parte si é cercato di produrre DTD specifiche per i singoli campi applicativi, dall'altra ci sono numerose proposte di standardizzazione di linguaggi XML-like attraverso i quali definire usi specifici dell'XML stesso. In questo momento questo settore di ricerca é notevolmente in crescita e l'ottica é quella di creare linguaggi per rappresentare i documenti (XSL), linguaggi per effettuare riferimenti (Xlink, Xpointer), linguaggi per definire gli schemi senza il contributo della DTD (XML-schema, e XML-Data), e infine linguaggi per effettuare query (XQL, XML-GL, XML-QL).

Nelle sezioni successive si definiranno i "desiderata" che un linguaggio di query per XML dovrebbe soddisfare per poi passare alla descrizione dell'XML-QL che in questo momento sembra da una parte soddisfare meglio i desiderata introdotti. Verrà presentato inoltre il linguaggio XML-GL che ha la peculiarità di permette di scrivere le query attraverso un apposito formalismo grafico.

Per finire, l'uso di modelli basati sul linguaggio XML impone due considerazioni ulteriori riguardanti l'una le difficoltà che è comunque necessario superare per fare in modo che l'XML rappresenti un punto di riferimento nell'ambito della **integrazioni di dati**, l'altra riguardante le **tecniche di misurazione delle performance** qualora si operi su database nei quali è implementata una tecnologia basata su dati XML [43, 44].

2.4.1 Integrazione di dati

Come si è visto XML fornisce una sintassi standard per rappresentare i dati che viene considerata come fulcro sul quale incentrare lo scambio dei dati nel WWW. Come conseguenza di questo fatto, l'interpretazione dei dati e della struttura sottesa ai dati è un compito che evidenzia diverse criticità, rappresentate soprattutto dal fatto che XML, essendo unicamente una sintassi, solo parzialmenete e con strumenti adeguati può essere un ausilio utile per l'integrazione dei dati. Ad esempio, senza la condivisione di una DTD, l'XML non può considerarsi uno strumento efficace per la data integration, soprattutto per il fatto che i nomi e i significati dei tag usati nei documenti XML sono scelti in maniera arbitraria. In particolare, per sfruttare le peculiarità che questo linguaggio può offrire, la ricerca dovrebbe indirizzarsi su queste tematiche:

- **Linguaggio per la descrizione delle sorgenti**: Punto fondamentale per il sistema di integrazione di dati è la realizzazione di un linguaggio che permetta di descrivere i contenuti e le potenzialità di sorgenti di dati diverse. In particolare occorre prestare attenzione al fatto che la semantica è normalmente più ricca che nel modello relazionale, al fatto che è necessario considerare la molteplicità di sorgenti con i quali è possibile trattare, e soprattutto è necessario cercare di sfruttare le informazioni che sono contenute nella DTD.

- **Algoritmi di riformulazione delle Query**: È necessario sviluppare degli algoritmi che possano riformulare le query poste dall'utente sullo schema di mediazione per utilizzarle negli specifici sorgenti. Le tecniche utilizzate nel caso relazionale non sono estendibili facilmente a questo contesto.

- **Traslazioni tra DTD**: È opportuno sviluppare delle tecniche che permettano in maniera agevole di trasferire dei dati conformi a una certa DTD in dati conformi a una DTD diversa.

2.4.2 Misure di complessità

È forse necessario introdurre un nuovo concetto di misura della complessità che tenga in considerazione il contributo apportato dal linguaggio XML all'analisi sui dati semistrutturati. Nello specifico occorre tenere in considerazione due aspetti fondamentali:

- **Numero di sorgenti/file XML**: se è necessario integrare dati da sorgenti XML multipli, gli algoritmi predisposti dovranno tenere in

considerazione tale numero di sorgenti. In generale se ci si propone di operare nell'ambito del WWW l'ipotesi di dovere operare con un largo numero di sorgenti è reale e porta delle riflessioni non contemplate nella ricerca sui database relazionali. In particolare la struttura dei dati XML è diversa da quella classica relazionale e questo comporta, tra l'altro, di dovere operare con un grande numero di file di piccole dimensioni, piuttosto che, come nel caso relazionale, con un piccolo numero di file di notevoli dimensioni.

- **Grado di irregolarità**: trattando con numerosi sorgenti è necessario provvedere a fornire degli strumenti che supportino i dati irregolari. L'irregolarità normalmente è rappresentata da una parte dall'uso di un sottoinsieme di attributi che sono utilizzati per descrivere un largo insieme di attributi (differenza semantica tra gli attributi utilizzati). Dall'altra gli attributi, anche con valore semantico uguale, possono essere strutturati diversamente, e quindi essere posti a livelli diversi nel grafo rappresentativo del sorgente in esame.

Capitolo 3

XML-Schema

3.1 Introduzione

All'interno delle specifiche del linguaggio XML 1.0, parallelamente alla definizione di due tipologie di documenti, gli uni che contengono i dati e gli altri, chiamati DTD, Document Type Definition, che rappresentano il modello strutturale dei dati, vengono introdotti i concetti di well-formedness e di validity. In particolare un documento é detto well-formed nel caso in cui

- Considerato nella sua globalità possa essere rappresentato cosí come viene definito un *document* nelle specifiche XML 1.0. Nella fattispecie tale affermazione implica che il documento XML deve essere costituito da uno o piú elementi e che deve esistere un elemento, detto **root** o **document element**, con la particolare caratteristica di non essere contenuto all'interno di nessun altro elemento.

- Soddisfi tutti i vincoli di well-formedness introdotti nelle specifiche XML1.0.

- Ogni entità che venga referenziata direttamente o indirettamente all'interno del documento sia anch'essa well-formed.

Una istanza viene poi detta di tipo valid se da una parte soddisfa tutti i vincoli di well-formedness e dall'altra rispetta nella sua strutturazione i vincoli e le caratteristiche espresse nella DTD a cui si riferisce.

Per alcune tipologie di applicazioni, tra le quali il W3C ha individuato ad esempio uno scenario che spazia da processi di transazione utilizzati nel commercio elettronico, fino a tool di aiuto nello sviluppo di documenti "tradizionali", i vincoli esprimibili tramite la DTD possono risultare non congruenti rispetto alle necessità (perché possono rappresentare per alcuni

contesti vincoli irrilevanti mentre per altri contesti appaiono essere eccessivamente restrittivi) e di conseguenza risulta irrilevante ai fini pratici l'appartenenza o meno del documento XML all'interno di una delle due tipologie di documenti individuate. Attraverso le regole strutturali che si introducono con XML-Schema si vuole fornire un sistema attraverso il quale aumentare le potenzialità espressive della DTD, operando sulla definizione dei markup e dei vincoli che devono essere rispettati dalle istanze di documento XML, e utilizzando una sintassi che sia XML-based in modo da permettere tra l'altro ai tool, realizzati per i documenti XML, di essere usati anche per queste applicazioni.

In questo momento il Word Wide Web Consortium non ha ancora rilasciato delle specifiche standard per tale linguaggio, anche se sono state fatte delle proposte, ora in corso di revisione, che sembra non debbano discostarsi molto da quanto verrà standardizzato. Il gruppo di lavoro che al momento opera su XML-Schema ha individuato alcuni percorsi di ricerca che possono essere in questo modo schematizzati:

- **Schema strutturale**: é necessario individuare un meccanismo analogo alla DTD per esprimere dei vincoli sulla struttura del documento (soprattutto per quello che riguarda il concetto di ordine, la quantificazione specifica delle occorrenze di un gruppo di elementi o di attributi). In particolare, ponendosi l'obiettivo di indicare delle funzionalità che devono essere raggiunte, si possono indicare i seguenti traguardi:
  - Integrazione con i *namespaces*
  - Definzione di vincoli sul contenuto di un elemento
  - Integrazione strutturale di schemi
  - Introduzione del concetto di ereditarietà
- **Modellazione di dati primitivi**: é necessario introdurre dei tipi di dato primitivi. Una delle piú gravi carenze di XML1.0 é quella di definire con il tipo "PCDATA" ogni istanza di qualsiasi elemento terminale del documento XML
- **Conformità**: Devono essere definite le relazioni che intercorrono tra gli schemi e le istanze di documenti XML. In particolare deve essere gestito il controllo degli errori derivanti dalla non conformità.

La proposta di XML-Schema vuole fornire un aumento della capacità espressiva della DTD, pur mantenendo la conformità con lo standard XML1.0. Affinché tale obiettivo si realizzi é necessario introdurre un nuovo documento

che si affianchi alla DTD e all'istanza, nel quale vengano introdotte le regole per la caratterizzazione del documento XML. Nelle proposte XML-Schema vengono ipotizzati due livelli di concettualizzazione: da una parte viene formalizzato il concetto di schema e quindi viene definita la relativa sintassi XML-based attraverso la quale rappresentare il grafo della base di dati, dall'altra vengono introdotti sia differenti tipi di dato sia un meccanismo per esprimere vincoli che controllino le istanze di ogni elemento.

3.2 XML-Schema vs XML

Essendo, come si é osservato, XML-Schema nato come uno strumento che si affianca alla DTD e che costituisce un mezzo attraverso il quale aumentarne la capacita' espressiva, é facile osservare quelle che sono le innovazione che l'introduzione di tale linguaggio apporta nel contesto dell'XML. Tali innovazioni, come si é evidenziato, sono nell'ordine di:

1. una maggiore definizione della **struttura**, nel senso che é possibile esprimere maggiori vincoli, in merito al numero occorrenze di un elemento in una struttura, in merito alla sequenza con la quale avvengono le occorrenze e in merito alla possibilità di creare degli elementi che ereditino delle caratteristiche di altri elementi già realizzati, ottenendo in questo modo uno schema che sia meglio rappresentativo della relativa istanza

2. una maggiore definizione del **dato**, con l'obiettivo di fornire una tipizzazione specifica per il singolo elemento, partendo da tipi di dato primitivi, arrivando a una ulteriore caratterizzazione dei tipi primitivi stessi

É necessario sottolineare che la sintassi con la quale é possibile esprimere i vincoli introdotti in istanze di documenti XML é completamente basata su XML 1.0. In questo modo si ottengono due effetti positivi correlati: da un lato le applicazioni con le quali attualmente si effettua il parsing sono in grado, senza alcuna modifica, di operare con file XML-Schema. Dall'altro il linguaggio che si introduce, conservando una piena rappresentatività dei documenti conformi a XML 1.0, puó candidarsi a pieno titolo a essere il successore di tale linguaggio.

Per quello che concerne la **struttura**, XML-Schema introduce, rispetto a XML 1.0, una tipizzazione degli elementi. Un elemento infatti viene detto di tipo **complesso**, quando contiene altri sottoelementi o se contiene degli attributi. Negli altri casi, in cui il contenuto dell'elemento é sicuramente terminale, e quindi puó contenere valori, di tipo stringa, numerico, ..., l'elemento

é di tipo **semplice**. Tale caratterizzazione degli elementi, non presente nella versione XML 1.0, sicuramente puø' fornire una utile conoscenza qualora si operi con documenti XML.

Ogni elemento é poi ulteriormente caratterizzabile attraverso il controllo specifico delle occorrenze di istanze della stessa tipologia presenti all'interno dell'elemento padre che lo contiene. XML-Schema permette un controllo molto puntuale mediante la definizione del numero di occorrenze minime e del numero di occorrenze massime per ogni elemento. Sempre nell'ottica di fornire uno schema che aderisca quasi perfettamente alla rispettiva istanza é possibile definire, per gli elementi di tipo complesso, una sintassi che serva a dichiarare in quale ordine debbano comparire nelle istanze sequenze di elementi. Anche in XML 1.0 si puó ottenere tale possibilità, anche se in maniera meno puntuale, utilizzando la sintassi definita per gli elementi di tipo children. Gli elementi children possono infatti consistere di strutture innestate a piú livelli, composte da elementi in sequenza o da elementi in alternativa. Nelle specifiche introdotte in quel contesto non é possibile controllare nulla di piú se non una corretta corresponsione fra le istanze e la DTD. Con le specifiche di XML-Schema é invece possibile caratterizzare in maniera piú puntuale la struttura. In particolare é possibile definire insiemi di elementi che possano sia essere istanziati in alternativa, sia essere istanziati in una qualsiasi sequenza, sia essere istanziati seguendo l'ordine definito nello schema. Tale possibilità di vincolare la struttura dati é realizzabile implementando il concetto di gruppo di elementi e caratterizzando attraverso opportune specifiche tale gruppo. In maniera analoga puó essere esteso il concetto di gruppo anche al contesto degli attributi. In questo modo puó essere definito un attributo di gruppo che sia istanziabile per ogni elemento appartenente al gruppo al quale si riferisce.

Nelle specifiche proposte per XML-Schema, viene inoltre implementata la possibilità di costruire nuovi elementi sulla base di elementi già definiti nello stesso documento o anche in altri documenti. Nella fattispecie é possibile sia estendere le caratteristiche di un elemento definito precedentemente (e quindi aumentarne la capacità espressiva attraverso l'introduzione di ulteriori vincoli o di ulteriori sotto-elementi), sia derivare per restrizione un elemento a partire dalla definizione di un altro elemento. Inoltre é possibile implementare il concetto di nodo equivalente, creando degli elementi analoghi sotto il profilo strutturale, ma aventi differente denominazione. In analogia con la programmazione ad oggetti poi il concetto di equivalenza e di derivazione puó essere arricchito nella propria funzionalità anche attraverso l'introduzione di elementi di tipo astratto, cioé di elementi ai quali non corrisponda una effettiva istanza nella parte dati del documento XML, ma che vengano utilizzati per definire a partire da essi nuovi elementi.

Le specifiche di XML-Schema includono inoltre un meccanismo di definizione di elementi le cui istanze siano uniche all'interno del documento e un meccanismo esplicito di definizione di chiavi. Anche con il linguaggio XML 1.0 é possibile introdurre, unicamente peró per quello che riguarda gli attributi, il tipo ID, che consente di definire valori di attributi che identifichino univocamente gli elementi che li portano. In analogia con la creazione di un attributo di tipo ID, esiste, sempre in XML 1.0, la possibilità di definire un attributo di tipo IDREF, con il quale viene referenziato un attributo di tipo ID. La sintassi XML 1.0 non consente peró di definire in maniera esplicita le coppie ID, IDREF, ma unicamente impone che a ogni attributo ID corrisponda un attributo IDREF. Tale meccanismo, paragonabile al sistema di oid utilizzato nella programmazione ad oggetti, applicato unicamente a livello di attributo di elemento, utilizzato come sostituto del sistema di chiavi che di solito si introduce nella gestione di database appare estremamente riduttivo. Per questo motivo, a fianco di un meccanismo che consente di introdurre l'unicità di qualsiasi elemento, é stato introdotto un sistema per implementare il concetto di key, e di foreign key. Inoltre, a differenza di quanto avviene per XML 1.0, é possibile dichiarare a quale key una foreign key si riferisce.

Infine, la proposta di linguaggio XML-Schema nasce due anni dopo la standardizzazione di XML 1.0. Nel frattempo il W3C ha introdotto altre raccomandazioni che permettono di sfruttare le potenzialità di linguaggi XML-based. La proposta di XML-Schema quindi tiene conto delle specifiche già introdotte e standardizzate dal Consorzio. In particolare viene utilizzato il concetto di namespace (che consente di definire dei domini di elementi e quindi permette di creare delle strutture modulari che non entrano in conflitto) e i concetti derivanti dal linguaggio XPath/Xslt (che consente di indirizzare parti di un documento XML, utilizzato nella fattispecie per caratterizzare, con il concetto di chiave, di unicità, particolari elementi della struttura).

3.3 XML-Schema vs ODL$_{I3}$

Essendo XML-Schema una evoluzione del linguaggio XML1.0 é possibile riproporre in questa sede le considerazioni che sono state fatte in merito alle differenze tra la modelizzazione di una base di dati semistrutturati effettuata utilizzando il modello OEM e descritta attraverso il linguaggio ODL$_{I^3}$e una modellizzazione realizzata in XML1.0.

In particolare si é evidenziato come in XML esista la possibilità di definire il concetto di ordine per quello che riguarda gli elementi, il concetto di attributo a livello di nodo, il concetto di riferimento, attraverso il quale é possibile referenziare altri nodi dello stesso documento, e il concetto di link, per creare

delle connessioni inter e intra schema, e si é visto che tali concetti non vengono supportati in maniera altrettanto puntuale in ODL_{I^3}. Di converso si é evidenziato come la trattazione dei dati sia meno precisa in XML1.0 dove non é possibile la definizione di tipi di dato ma ogni istanza terminale viene assunta come PCDATA (trattabile sostanzialmente come un dato di tipo string).

Si é potuto pertanto osservare in sintesi che se XML permette una migliore rappresentazione della struttura del dato rispetto a ODL_{I^3}, altrettanto non si puó dire per quello che concerne la tipizzazione del dato. Come si é evidenziato nella sezione precedente, le innovazioni introdotte con XML-Schema permettono un superamento di tali limitazioni. Nel caso in cui si vogliano quindi considerare i problemi connessi con la traduzione di rappresentazioni di strutture espresse in ODL_{I^3}in strutture espresse in XML, é necessario effettuare delle distinzioni relative al verso con il quale viene fatta la traduzione. Infatti la semantica rappresentabile in ODL_{I^3}puó essere considerata come un sottoinsieme di quella rappresentabile in XML-Schema. Quindi qualora sia necessario tradurre da ODL_{I^3}in XML-Schema, tale traduzione potrà conservare totalmente i vincoli che sono stati definiti senza l'ausilio di algoritmi di traduzione particolarmente complessi.

Considerazioni analoghe non possono essere effettuate nel caso di una traduzione da XML-Schema a ODL_{I^3}, in quanto, essendo la potenzialità espressiva di ODL_{I^3}meno puntuale, sarà necessario fornire a fianco della traduzione una mapping table nella quale siano evidenziate le perdite di semantica, sostanzialmente di termini di *flatting*, che si sono rese necessarie. Tali perdite di informazioni saranno riguardanti soprattutto la definizione precisa della struttura del dato che non trova analogo potere di espressione in ODL_{I^3}.

Nell'ottica del progetto MOMIS, qualora cioé si vogliano integrare diversi schemi strutturali, si puó osservare che la perdita di informazioni che un wrapper XML-Schema genererebbe nel passaggio tra i due linguaggi non provoca forti problemi. Lo stesso linguaggio ODL_{I^3}viene infatti utilizzato, non per descrivere pedissequamente delle strutture, ma per descrivere delle modellizzazioni di strutture dati (attraverso il concetto di object pattern). Nel momento in cui si trattano documenti XML-Schema, attraverso il quale si puó invece attuare un preciso controllo delle istanze, si rende pertanto forse necessario un ulteriore passaggio, attraverso il quale la struttura che si vuole descrivere venga trasformata in una simile ma con vincoli meno stringenti.

3.4 I tre linguaggi a confronto

Nella prospettiva di avere un quadro sinottico dei tre linguaggi descritti, si propongono le tabelle che seguono, nelle quali vengono rappresentate quelle che sono le principali caratteristiche supportate dai linguaggi. Si é voluto fare una distinzione fra due livelli:

A livello di struttura

Caratteristica	XML-Schema	XML1.0	ODL_{I3}
Distinzione sintattica tra elementi intermedi e terminali	x	x	x
Concetto di attributo a livello di nodo	x	x	
Concetto di riferimento	x	x	
Concetto di link	x	x	
Possibilità di esprimere ordinamenti per elementi figli di uno stesso padre	x	x	x
Possibilità di esprimere elementi figli in alternativa per uno stesso padre	x	x	x
Possibilità di esprimere elementi figli in sequenza per uno stesso padre	x	x	x
Possibilità di creare elementi come estensioni di altri elementi	x		x
Possibilità di creare elementi come restrizioni di altri elementi	x		x
Esistenza di elementi astratti	x		

A livello di nodo

Caratteristica	XML-Schema	XML1.0	ODL_{I^3}
Esistenza di datatype	x		x
Creazione di nuovi tipi	x		x
Gestione di elementi opzionali	x	x	x
Gestione operatore croce	x	x	
Gestione operatore stella	x	x	x
Possibilità di esprime il numero di occorrenze minime e e massime di un elemento figlio	x		

Si propone una traduzione dello stesso documento nei tre linguaggi, in modo da evidenziare le peculiarità di ognuno di essi. In questo caso si é preferito prendere come documento di riferimento una variazione di quello utilizzato nelle proposte di specifiche per XML-Schema, e che vuole rappresentare un ordine commerciale. Lo schema che si tradurrà rappresenta la struttura di una istanza di questo tipo:

```
<?xml version=''1.0''?>
<purchaseOrder orderDate=''1999-10-20''>
        <shipTo country=''US''>
                <name> Alice Smith </name>
                <street>123 Maple Street</street>
                <city>Mill Valley</city>
                <state>CA</state>
                <zip>90952</zip>
        </shipTo>
        <billTo country=''US''>
                <name> Robert Smith </name>
                <street>8 Oak Avenue</street>
                <city>Old Town</city>
                <state>PA</state>
                <zip>95819</zip>
        </billTo>
        <comment></comment>
        <items>
                <item partNum=''872-AA''>
                      <productName>Baby Monitor</productName>
                      <quantity>1</quantity>
                      <price>39.98</price>
                      <shipDate>12-12-1999</shipDate>
                </item>
```

```
        </items>
</purchaseOrder>
```

A tale istanza corrisponde un file XML-Schema del seguente tipo:

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">

   <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
   <xsd:element name="comment" type="xsd:string"/>

   <xsd:complexType name="PurchaseOrderType">
        <xsd:element name="shipTo" type="Address"/>
        <xsd:element name="billTo" type="Address"/>
        <xsd:element ref="comment" minOccurs="0"/>
        <xsd:element name="items"  type="Items"/>
        <xsd:attribute name="orderDate" type="xsd:date"/>
   </xsd:complexType>

   <xsd:complexType name="Address">
        <xsd:element name="name"   type="xsd:string"/>
        <xsd:element name="street" type="xsd:string"/>
        <xsd:element name="city"   type="xsd:string"/>
        <xsd:element name="state"  type="xsd:string"/>
        <xsd:element name="zip"    type="xsd:decimal"/>
        <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
   </xsd:complexType>

   <xsd:complexType name="Items">
        <xsd:element name="item" minOccurs="0" maxOccurs="*">
        <xsd:complexType>
          <xsd:element name="productName" type="xsd:string"/>
          <xsd:element name="quantity">
             <xsd:simpleType base="xsd:positive-integer">
             <xsd:maxExclusive value="100"/>
             </xsd:simpleType>
          </xsd:element>
          <xsd:element name="price"    type="xsd:decimal"/>
          <xsd:element ref="comment"   minOccurs="0"/>
          <xsd:element name="shipDate" type="xsd:date" minOccurs='0'/>
          <xsd:attribute name="partNum" type="Sku"/>
        </xsd:complexType>
```

```
        </xsd:element>
    </xsd:complexType>

    <xsd:simpleType name="Sku" base="xsd:string">
        <xsd:pattern value="/d{3}-[A-Z]{2}"/>
    </xsd:simpleType>

</xsd:schema>
```

Tale struttura rappresenta la modellizzazione di un file XML avente la seguente DTD:

```
<!ELEMENT purchaseOrder (shipTo, billTo, comment?, items)>
<!ATTLIST purchaseOrder orderDate CDATA #IMPLIED>
<!ELEMENT shipTo (name, street, city, state, zip)>
<!ATTLIST shipTo country NMTOKEN #IMPLIED>
<!ELEMENT billTo (name, street, city, state, zip)>
<!ATTLIST billTo country NMTOKEN #IMPLIED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT items (item)*>
<!ELEMENT item (productName, quantity, price, comment?, shipDate?)>
<!ATTLIST item partNum CDATA #IMPLIED>
<!ELEMENT productName (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT shipdate (#PCDATA)>
```

É interessante notare che a partire dalle due rappresentazioni strutturali dello stesso file possono essere creati due differenti documenti ODL_{I^3}. Quello derivato dalla DTD avrà una rappresentazione meno particolareggiata, nella fattispecie non conterrà le informazioni relative ai tipi di dato. Quello ricavato dalla rappresentazione XML-Schema, che qui si propone, avrà una rappresentazione piú dettagliata. Il modello di traduzione tra i linguaggi utilizzato é quello indiviato nell'ambito del gruppo di lavoro al progetto Momis.

```
Interface purchaseOrder
```

```
(source semistructured Order)
{ attribute Address shipTo;
  attribute Address billTo;
  attribute string comment*;
  attribute items items;
  attribute string purchaseOrder_orderDate; };


Interface Address
(source semistructured Order)
{ attribute string name ;
  attribute string street;
  attribute string city;
  attribute string state;
  attribute integer zip;
  attribute string Address_country; };

Interface items
(source semistructured Order)
{ attribute set<item> item; };

Interface item
(source semistructured Order)
{ attribute string productName ;
  attribute integer quantity;
  attribute integer price;
  attribute string comment;
  attribute string shipDate;
  attribute string item_partNum;};
```

Si propone inoltre un ulteriore esempio di file XML-Schema che metta in evidenza le potenzialità espressive ottenibili utilizzando pienamente le caratteristiche di tale linguaggio. In questo caso si vuole rappresentare il riepilogo, raggruppato per regioni, delle componenti vendute attraverso un modulo simile a quello proposto in precedenza:

```
<purchaseReport period="P3M" periodEEnding="1999-12-31">
<regions>
  <zip code="95819">
     <part number="872-AA" quantity="1"/>
```

```
     <part number="926-AA" quantity="1"/>
  </zip>
  <zip code="63143">
     <part number="455-BX" quantity="4"/>
  </zip>
</regions>

<parts>
     <part number="872-AA">Lawnmower</part>
     <part number="872-AA">Baby monitor</part>
     <part number="455-BX">Shelves</part>
</parts>
</purchaseReport>
```

Una delle descrizioni di questa istanza attraverso XML-Schema puó essere la seguente:

```
<schema>
<!-- Sono stati tralasciati elementi connessi con i namespace
non importanti ai fini della rappresentazione strutturale -->

<element name="purchaseReport">
   <complextype>
      <element name="regions" type="RegionsType"/>
      <element name="parts" type="PartsType"/>
      <attribute name="period" type="timeduration"/>
      <attribute name="periodEnding" type="date"/>
   </complextype>
</element>

   <unique>
      <selector>regions/zip</selector>
      <field>@code</field>
   </unique>

   <key name="pNumKey">
      <selector>parts/part</selector>
      <field>@number</field>
   </key>

   <keyref refer="pNumKey">
```

```
      <selector>regions/zip/part</selector>
      <field>@number</field>
   </keyref>

<complexType name="RegionsType">
   <element name="zip" minOccurs="1" maxOccurs="*">
      <complexType>
         <element name="part">
             <complexType content="empty">
                 <attribute name="number" Type="string">
                 <attribute name="quantity" Type="integer">
             </complexType>
         </element>
         <attribute name="code" type="positive-integer">
       </complexType>
    </element>
</complexType>

<complexType name="PartsType">
   <element name="part" minOccurs="1" maxOccurs="*">
      <complexType>
         <element content="textonly">
             <attribute name="number" Type="string">
         </element>
      </complexType>
    </element>
</complexType>
</schema>
```

A questa rappresentazione dello schema corrisponde una DTD piú semplice che non puó definire completamente quanto espresso in precedenza. Ad esempio il concetto di chiave e di elemento unico non sono rappresentabili in maniera analoga nella DTD.

```
<!ELEMENT purchasereport (regions,parts)>
<!ATTLIST purchasereport period CDATA #IMPLIED>
<!ATTLIST purchasereport periodEnding CDATA #IMPLIED>
<!ELEMENT regions (zip*)>
<!ELEMENT zip (part1*)>
<!ATTLIST zip code ID #REQUIRED>
<!ELEMENT part1 EMPTY>
```

```
<!ATTLIST part1 number IDREF #REQUIRED>
<!ATTLIST part1 quantity     #IMPLIED>
<!ELEMENT parts (part2*)>
<!ELEMENT part2 (#PCDATA)>
<!ATTLIST part2 number ID REQUIRED>
```

Come si puó osservare sia il concetto di unicità, sia il concetto di chiave sono rappresentati nella DTD attraverso lo stesso sistema. In questo caso inoltre é impossibile distinguere a quale attributo di tipo ID faccia riferimento l'attributo IDREF. La medesima struttura in ODL_{I^3}potrebbe essere la seguente:

```
interface purchaseReport
(source semistructured Report)
{  attribute RegionsType regions;
   attribute PartsType parts;
   attribute string purchaseReport_period;
   attribute string purchaseReport_periodEnding; };

interface RegionType
(source semistructured Report)
{  attribute set<zip> zip;
   attribute integer zip_code; };

interface zip
(source semistructured Report
foreign_key(part_number) references parts)
{  attribute boolean part;
   attribute string part_number;
   attribute integer part_quantity; };

interface PartsType
(source semistructured Report)
{  attribute set<parts> parts; };

interface parts
(source semistructured Report
key part_number)
{  attribute string part;
   attribute string part_number; };
```

3.5 Considerazioni

Nei precedenti paragrafi si é cercato di illustrare come, attraverso i nuovi aspetti introdotti con XML-Schema, si possa arrivare a rappresentare in maniera precisa e puntuale una struttura di dati. É necessario peró effettuare due importanti considerazioni.

La prima é che, in ottica di rappresentazione di dati semistrutturati e in ottica di integrazione di basi di dati eterogenee, non é sempre particolarmente utile avere una struttura definita in maniera estremamente rigorosa. Nel caso limite, quando sia necessario rappresentare una sorgente di dati fortemente non strutturata, se si utilizza un linguaggio di definizione di strutture estremamente rigido, quella che si ottiene non é la rappresentazione di un modello generale, ma é una struttura che ripropone, in maniera pedissequa, l'istanza alla quale si riferisce.

La seconda considerazione é legata all'estrema libertà sintattica con la quale XML-Schema consente di rappresentare la stessa struttura. Sono consentite infatti diverse tipologie di rappresentazione della stessa base di dati, le une differenti dalle altre solamente per alcuni aspetti sintattici. L'indeterminatezza con la quale si ha la rappresentazione di una struttura puó portare a delle ulteriori difficoltà in fase di analisi della stessa, quindi sia in fase di query sia in fase di integrazione di strutture differenti. Tale limite potrà essere superato unicamente con l'introduzione di una forma "canonica" di linguaggio e attraverso la definizione di opportuni algoritmi che consentano di trasformare un documento XML qualunque nella sua forma canonica. In questa fase il W3C, pur non avendo ancora reso standard il concetto di forma canonica, ha sviluppato un linguaggio, denominato XSLT, che permette la trasformazione sintattica e semantica di documenti XML in altri documenti XML. L'uso combinato di XSLT e della forma canonica di XML potrà permettere un uso piú flessibile delle strutture create con XML-Schema.

Capitolo 4

XML Query Language

4.1 Introduzione

Come si è cercato di mettere in evidenza nelle precedenti sezioni, l'XML nasce con l'obiettivo di diventare un formato fondamentale per lo scambio di dati in Internet. Perchè tale scopo si realizzi è necessario che a fianco dell'implementazione del linguaggio vero e proprio venga definito un linguaggio e una sintassi che consentano di effettuare delle query. Tale sintassi non deve rappresentare semplicemente uno strumento per espandere le potenzialità dei browser che visualizzeranno i file XML, ma deve rendere disponibili tutti gli strumenti e le metodologie necessarie alla gestione di database. In generale per questo motivo devono essere implementati degli strumenti che permettano operazioni di:

- **Data extraction**: estrazione di una selezione di dati ottenuta a partire da una vasta base di dati. Devono essere introdotte delle procedure che consentano di imporre dei criteri di selezione e di ricerca delle informazioni
- **Data integration**: integrazione dei dati contenuti in basi di dati differenti utilizzando dei criteri attraverso i quali effettuare il "merging" delle informazioni
- **Data traslation**: operazione di trasformazione di dati XML confacenti una determinata DTD in dati confacenti una DTD diversa
- **Data storage**: criteri e metodologie di memorizzazione delle informazioni

Essendo le operazioni indicate tra loro interdipendenti, il linguaggio che si andrà a definire dovrà riuscire a dare una risposta organica a tutte le

problematiche proposte da ogni singola operazione. Inoltre, parallelamente all'individuazione di un query-language opportuno, diventa di cruciale importanza prevedere di sfruttare le potenzialità offerte dal meccanismo delle *"viste"*.

Tradizionalmente la vista è un sistema attraverso il quale é consentita la visualizzazione dei dati mediante ottiche di lettura diversificate senza che venga modificata la struttura fisica della base di dati. Si può ipotizzare che i benefici offerti dall'uso delle viste nel contesto dei dati semistrutturati assumano dei caratteri di fondamentale importanza. In questo contesto, infatti, nel quale é necessario avere una visione organica di dati eterogenei, il meccanismo delle viste permette di inserire un'interfaccia strutturata al di sopra dei dati semistrutturati da analizzare preservando l'integrità dei file sorgenti. Un esempio di applicazione delle viste in ambiente XML é stato proposto in [45]. Da notare però che una sintassi appropriata che consenta di utilizzare il sistema dalle viste come metodo di interrogazione di una base di dati, possibilità ancora oggetto di studio e di critica per quello che concerne i DBMS [46], nel contesto dell'XML non é stata ancora inserita in nessun linguaggio proposto come standard al W3C. Nonostante questo é comunque opportuno considerare le ricadute generate dall'implementazione o dalla non implementazione delle viste non appena sia l'XML sia l'opportuno query language si diffonderanno. L'uso delle viste nel contesto dei dati semistrutturati sarà analizzato nel paragrafo 4.4.4.

Il W3C, non avendo ancora scelto un linguaggio per query nell'ambito dell'XML, ha raccolto alcune proposte. Per potere effettuare una valutazione fra di esse è necessario specificare delle caratteristiche peculiari che dovranno essere possedute dal linguaggio [47, 48].

4.2 "Desiderata" per un Query Language

In questa sezione verranno elencate le caratteristiche che si presuppone debbano essere soddisfatte da un linguaggio il cui scopo sia effettuare delle query su un documento XML. Tali caratteristiche derivano da un lato dalla necessità di utilizzare un linguaggio che possa operare con dati non strutturati e dall'altro dall'obiettivo di creare un linguaggio facilmente integrabile nella rete Internet.

1. **Semantica puntuale.** Il linguaggio dovrebbe essere dotato di una semantica definita in modo formale. La formalizzazione è necessaria per potere definire in maniera univoca alcuni concetti chiave, tra i quali è possibile indicare ad esempio la determinazione della struttura del dato

risultante da un'operazione, il concetto di equivalenza tra le operazioni (indispensabile per implementare degli algoritmi di ottimizzazione), e il concetto di contenimento (utilizzato per determinare se un particolare dato può essere considerato come risultato di una query).

2. **Riscrivibilità, ottimizzabilità.** Il dato XML nell'uso comune potrebbe essere anche generato automaticamente come risultato di una operazione di trasformazione di dati progettati secondo altri paradigmi, come quello relazionale, object-oriented, oppure potrebbe provenire da dati memorizzati in formati specifici. Si può pensare che sia più efficace convertire una query nel linguaggio nativo dei dati ai quali le stessa si riferisce piuttosto che dovere convertire il dato nativo in XML e poi applicare su di esso la query. In maniera analoga deve essere possibile riscrivere, utilizzando tecniche di ottimizzazione, una query XML scritta per un documento XML.

3. **Operazioni supportate.** Le operazioni che devono essere necessariamente supportate dal query language sono quella di *selezione* (la evidenziazione di un dato in un documento attraverso una ricerca basata sul contenuto, sulla struttura del documento stesso o sugli attributi del dato in esame), quella di *estrazione* (la selezione di particolari elementi di un documento), quella di *riduzione* (l'eliminazione di specificati sottoelementi di un elemento), quella di *ristrutturazione* (la costruzione di nuovi insiemi di elementi che soddisfino criteri richiesti), e quella di *combinazione* (l'unione di due o più elementi in uno). E' opportuno potere effettuare tutte queste operazioni attraverso la scrittura di un'unica query, e attraverso la sintassi offerta da un unico linguaggio. È necessario mettere in evidenza inoltre che alcune di queste operazioni riducono il volume di dati che il server XML deve gestire: per questo motivo l'uso del query language deve essere esteso al server in modo tale che esso stesso possa ridurre in maniera autonoma il volume di richieste della rete.

4. **Output.** Una query XML dovrebbe generare come risultato un documento XML. L'applicazione di tale proprietà di chiusura ha come risultato diversi aspetti positivi tra i quali la possibilità di definire delle viste attraverso una singola query, la possibilità di operare più agevolmente nella composizione e nella decomposizione delle query (questo rappresenta il punto di partenza sia per la creazione di operazioni di ottimizzazione sia per la possibilità di interrogare con una singola scrittura diversi sorgenti), e soprattutto permette alle applicazioni di utilizzare la stessa interfaccia (e quindi lo stesso sistema di browsing) qualora si

voglia visualizzare un file dati, oppure si voglia analizzare il risultato di una operazione di query.

5. **Semantica della composizione.** Il significato di una espressione utilizzata nel linguaggio dovrebbe essere lo stesso dovunque essa appaia nel documento.

6. **Schema opzionale.** Il query language dovrebbe essere utilizzabile indipendentemente dalla conoscenza di uno schema (DTD) al quale il documento sia conforme. La caratteristica di linguaggio autodescrivente posseduta dall'XML dovrebbe trasferirsi al linguaggio per le query che dovrebbe essere in grado di costruire uno schema "just-in-time" e riferirsi a quello per effettuare le operazioni. Questa capacità mette in evidenza che il linguaggio deve potere operare anche riferendosi unicamente al sorgente XML, nel quale non si ha una precisa conoscenza della struttura.

7. **Uso di uno schema disponibile.** Al contrario, qualora al documento XML sia associatato una DTD, dovrebbe essere possibile sfruttare le conoscenze in esso racchuso e generare una ulteriore DTD per la risposta alla query. La conoscenza di una DTD permette tra l'altro di semplificare l'operazione di generazione del risultato dell'operazione di query.

8. **Conservazione dell'ordine e delle associazioni.** Se necessario, la query dovrebbe essere creata in modo tale da preservare l'ordine e le associazioni degli elementi nel documento XML. In un documento l'ordine potrebbe essere rilevante (vedi 2.3.1) così come il particolare innestamento di sottoelementi in elementi.

9. **Embedding.** Le query dovrebbero essere embedded all'interno di documenti XML. Così facendo una query XML dovrebbe essere costruibile in modo tale da potere contenere dati XML e viceversa all'interno di documenti XML dovrebbe essere possibile inserire delle query. La prima potenzialità permette di potere inserire all'interno di documenti sia dati già memorizzati sia dati che siano il risultato di un'operazione di query. La seconda possibilità permette invece di potere scrivere delle query che siano valutazioni parziali di query XML. Questo meccanismo è utile per effettuare delle valutazioni parziali di dati in un ambiente distribuito laddove si voglia che i dati scelti da un sorgente siano combinati con i dati presenti in un altro sorgente.

10. **Predisposizione per modelli di dato non standard.** Dovrebbe essere permesso un meccanismo di estensioni che, consentendo di discernere tra tipi di file, renda possibile applicare dei criteri specifici a particolari modelli di dato. Un esempio potrebbe essere fornito da operazioni che consentano di discriminare tra diversi sorgenti di tipo multimediale.

11. **Uso dei metadati.** Il linguaggio dovrebbe essere utilizzato per operare anche sugli elementi usati per rappresentare la struttura del documento XML stesso. In questo modo si può esprimere attraverso una precisa sintassi una serie di vincoli nel modello dati e nella struttura del documento in esame.

12. **Operazioni Server-side.** Dovrebbero essere permesse le operazioni sul lato server. In questo modo è possibile ottenere sostanzialmente due vantaggi: da una parte eseguire le query in maniera remota senza che sia cioè necessario comunicare con il punto dal quale ha inizio l'operazione di query, dall'altra il server stesso può fare eseguire degli algoritmi che permettano di ottimizzare una eventuale query non correttamente posta.

13. **Gestione attraverso programmi.** Dovrebbe essere implementata la possibilità di generare le query attraverso del software specifico senza l'intervento diretto dell'utente.

14. **Rappresentazione XML** Una query XML dovrebbe essere rappresentabile utilizzando il linguaggio XML. Questa proprietà evidenzia la necessità di non provvedere a speciali meccanismi per memorizzare e trasportare query XML al di fuori di quelli previsti per l'XML stesso.

L'elenco che qui si é proposto sicuramente non ha carattere di esaustività per quello che riguarda la risoluzione di una problematica, quella della determinazione delle caratteristiche specifiche per un query language, che non può essere affrontata unicamente in base a un elenco. Nonostante questo le caratteristiche che si sono individuate si possono considerare come griglia attraverso la quale potere effettuare un primo raffronto tra i diversi linguaggi che sono stati sviluppati e proposti come standard. Tra di essi stanno assumendo particolare rilevanza i seguenti:

XML-QL. E' il query language che ha caratteri di maggiore completezza, tra quelli proposti al W3C, in quanto sia propone due modelli diversi per la rappresentazione strutturale dei documenti, sia contempla operatori per

estrarre in maniera puntuale le operazioni, sia, infine, permette la definizione di funzioni Skolem per potere ampliare le capacità di selezione. Questo linguaggio ha poi il vantaggio che da un lato rappresenta una versione XML di SQL, e quindi é di facile apprendimento per chi già opera utilizzando il paradigma relazionale, dall'altra estende le potenzialità di SQL, adattandolo alle caratteristiche dei dati semistrutturati. Nella sezione seguente verranno analizzate le caratteristiche e la sintassi di XML-QL.

Il **Lore** (Lightweight Object Repository) è un sistema di database management system, sviluppato a Stanford, orientato alla gestione dei dati semistrutturati. Il query language implementato in quel sistema, chiamato Lorel, è stato progettato estendendo le caratteristiche dell'OQL ai dati semistrutturati. In un secondo momento poi il modello dati del Lore è stato adattato con lo scopo di effettuare query sui dati XML [49].

L'**XSL** (Extensible Stylesheet Language), il linguaggio adibito al controllo del layout dei documenti XML, proposto per la standardizzazione al W3C, può essere considerato anch'esso come un query language. L'espressività di tale linguaggio è però limitata. La sua caratteristica principale è infatti quella di procedere all'interno dell'albero nel quale è riproducibile la struttura dei documenti XML in maniera ricorsiva innestandosi sempre più in profondità negli elementi dell'albero.

L'**XQL** consiste essenzialmente in una estensione delle potenzialità offerte dall'XSL attraverso una sintassi concisa che permette di definire il formato della struttura dati nel quale immettere i risultati delle operazioni.

XML-GL è un linguaggio di tipo grafico che permette di effettuare delle query attraverso un'interfaccia di tipo grafico. In questo momento non stata ancora sviluppato un prototipo, ma la formalizzazione proposta indica che la potenza espressiva di questo linguaggio è pari a quella dell'XML-QL.

WebL è un linguaggio che consente di operare sia in XML sia in HTML. Sviluppa principalmente due caratteristiche: la prima consiste nella definizione di una sintassi per la ricerca delle informazioni, la seconda consiste in un meccanismo che consente di operare parallelamente su più documenti alternando processi di analisi e processi di attesa per il download delle fonti.

La tabella che segue propone un facile raffronto dei linguaggi ora brevemente descritti attraverso la griglia di analisi proposta in precedenza.

	Seman. punt.	Risc. Ott.	Op. sup.	Out-put	Seman. comp.	Sch. opz.	Uso . sch.
XML-QL	y	y	y	y	y	y	n
LOREL	y	y	y	n	n	y	n
XSL	y	y	n	y	y	y	n
XQL	y	y	n	y	y	y	n
XML-GL	y	y	y	y	y	y	y
WEBL	n	y	y	y	y	y	n

	Cons. Ord	Emb.	Pred.	Meta dati	Op. Ser.	Gest. prg.	Rap. XML
XML-QL	y	y	n	y	y	y	n
LOREL	y	y	n	y	y	y	n
XSL	y	y	n	y	y	y	y
XQL	y	y	n	y	y	y	n
XML-GL	y	n	n	y	y	n	n
WEBL	y	y	y	y	n	y	n

4.3 XML Query Requirements: le richieste del W3C

In merito alle caratteristiche che dovrebbero essere possedute da un query language che operi su documenti di tipo XML, il Word Wide Web Consortium ha prodotto in data 31 Gennaio 2000 un working draft [50] nel quale è indicato lo stato di sviluppo della ricerca. Tale documento, pur non avendo carattere di stabilità e di completezza, rappresenta comunque un'indicazione della direzione all'interno della quale il W3C sta effettuando ricerca. E' opportuno sottolineare che nel working draft viene rimarcata in modo particolare l'interdipendenza tra l'attività di ricerca sui query language e l'attività di ricerca di altri gruppi di lavoro. Nella fattispecie vengono tenuti in particolare considerazione gli esiti delle attività legate ai **DOM** (il query language deve essere in grado di ritornare i risultati seguendo una struttura che possa essere utilizzata da un programma DOM), delle attività legate a **XSL e XPointer** (entrambi questi linguaggi utilizzano il linguaggio XPath come sintassi per definire percorsi all'interno di un documento XML), delle attività legate a **XML-Schema** (é forse questa una delle principali novità introdotte da questo draft, cioé la piena compatibilità del query language con XML-Schema), e dell'attività di **XML Information Set** (il modello per rappresentare i dati deve essere completamente compatibile con quello di

XML Infoset).

In particolare gli ambiti che il documento cerca di analizzare sono sostanzialmente due: da una parte vengono presi in considerazione gli ambienti all'interno dei quali verrà introdotto e utilizzato il query language, dall'altra sono introdotte, anche se in maniera ancora poco dettagliata e poco formalizzata, le principali caratteristiche che il nuovo linguaggio deve necessariamente possedere. Tali caratteristiche possono essere suddivise sostanzialmente in tre settori: il primo nel quale sono contenute le specifiche più generali, il secondo nel quale viene indicato il modello dati all'interno del quale il linguaggio deve operare e il terzo indicante le funzionalità da implementare.

4.3.1 Caratteristiche generali

In questa sezione vengono indicate le caratteristiche generali che il linguaggio deve supportare. Nella fattispecie sono imposte delle condizioni in merito a:

1. **Sintassi del query language**: Il linguaggio dal punto di vista sintattico dovrebbe avere almeno due vincoli rappresentativi. In primo luogo deve essere possibile per un umano leggere e scrivere nel linguaggio, in secondo luogo la sintassi deve essere espressa in XML in un modo che corrisponda alla struttura sottesa della query.

2. **Dichiaratività**: il linguaggio deve essere dichiarativo, ma non deve spingere verso una particolare strategia di valutazione

3. **Indipendenza dal protocollo**: il linguaggio deve essere definito in maniera indipendente da tutti i protocolli con i quali é utilizzato.

4. **Gestione degli errori**: Il linguaggio deve sapere gestire condizioni di errore che possano accadere durante l'esecuzione di query. In particolare é necessario prevedere errori causati dalla risoluzione delle query, dall'impossibilità da parte del query processor di reperire funzioni esterne.

5. **Aggiornamenti**: La versione 1.0 del query language non deve precludere la capacità di estendere le potenzialità del linguaggio nelle successive versioni.

6. **Definizione per istanze finite**: Il linguaggio deve essere definito per istanze finite di un modello di dati XML.

L'elenco delle caratteristiche generali richieste pone in evidenza principalmente un aspetto, quello dell'interdipendenza del query language con il

sistema piú complessivo di gestione dei dati introdotto con XML. Il linguaggio deve perciò essere indipendente dal protocollo con il quale viene utilizzato, ma deve esprimere la propria semantica mediante l'XML.

4.3.2 Il modello dati

Il modello dati previsto per il query language rappresenta la sintesi di quelle che sono le direzioni attuali di ricerca attuate dal W3C. In particolare sono al momento oggetto di studio fondamentalmente tre linee di ricerca.

La prima si basa sui risultati del gruppo di lavoro che si occupa del **XML Information Set**. Tale gruppo di lavoro ha come oggetto l'insieme di dati astratti ottenibili da un documento XML well-formed e quindi la visione del documento non come semplice stream di caratteri confacenti a una struttura sintatticamente formalizzata, ma come a strutture logiche dalle quali sia potere dedurre sia potere aggiungere delle informazioni aggiuntive. Il query language che si andrà a formalizzare dovrà essere, a livello logico, confacente alla struttura progettata da questo gruppo di lavoro.

La seconda linea di sviluppo, i cui risultati dovranno essere tenuti in considerazione dal query language, é quella legata ai **namespaces**. Un namespace é una collezione di nomi, identificata attraverso un URI, che serve a definire un vocabolario comune di markup utilizzabile da molteplici moduli software XML. La definizione dei namespace é al momento formalizzata mediante una raccomandazione del W3C del 14 Gennaio 1999.

Il query language, infine, deve rappresentare sia dati conformi allo standard XML1.0, sia deve potere operare con dati conformi alle specifiche di **XML-Schema**. Tali specifiche introducono, per quello che riguarda le influenze sul query language, principalmente un nuovo elemento rappresentato dal potere definire dei modelli di dato. XML 1.0 infatti considera tutti i dati in maniera omogenea, del tipo PCDATA. XML-Schema invece definisce dei tipi di modello darto. Tali tipi possono essere sia semplici (e quindi sono conformi agli usuali tipi di dato: integer, string,...) sia complessi (rappresentati da costrutti particolari di tipi semplici e da unioni di elementi).

4.3.3 Funzionalità del linguaggio

Facendo riferimeno alla nota stessa per una conoscenza esaustiva delle specifiche richieste, si può evidenziare che sostanzialmente le funzionalità che devono essere possedute dal query language sono di due tipi:

1. **Caratteristiche generali del linguaggio**: Le operazioni gestite dal query language devono supportare tutti i tipi di dato rappresentati dal

modello di dati introdotto. Viene introdotta anche la chiusura del linguaggio rispetto al modello di dati adottato. In questo modo sia la query stessa, sia il risultato della query devono essere espressi meramente in termini di modello dati XML. Per quello che riguarda le operazioni viene rimarcata la necessità di prevedere gli operatori di quantificatore universale e quantificatore esistenziale, nonché la necessità di supportare i valori di tipo null.

2. **Operazioni sulla struttura del documento**: Il linguaggio dovrà prevedere meccanismi che consentano di operare sulla struttura del documento. Nella fattispecie é necessario introdurre sistemi sia per preservare la struttura del documento sul quale si effettua la query sia per effettuare delle trasformazioni nella struttura. Inoltre deve essere possibile accedere ai riferimenti, sia interschema, sia intraschema, che eventualmente siano stati introdotti.

4.4 XML-QL: un query language per XML

L'**XML-QL** [31] è una proposta di linguaggio per effettuare query all'interno di documenti XML. Nella sezione precedente si è cercato di mettere in evidenza sia le caratteristiche fondamentali che un query language dovrebbe possedere sia le specifiche richieste dal W3C. In particolare XML-QL pur soddisfacendo buona parte dei punti indicati è stato progettato con le seguenti caratteristiche peculiari:

- È un linguaggio dichiarativo.
- È "chiuso rispetto al modello relazionale": implementa quindi tutte le caratteristiche del modello relazionale, in particolare é implementata l'operazione di join.
- Sono facilmente sviluppabili tecniche di ottimizzazione delle query, metodi per effettuare stime dei costi e sistemi di riscrittura delle query.
- È stato progettato per estrarre dati da documenti XML esistenti e costruire, utilizzando i dati estratti, nuovi documenti XML.
- Può supportare sia il modello ordinato di rappresentazione strutturale dei documenti XML sia il modello dati non ordinato.

In particolare rispetto agli altri linguaggi candidati per la standardizzazione, l'XML-QL propone alcune caratteristiche aggiuntive che lo rendono

particolarmente completo: in primo luogo utilizza il concetto di ordine sia per quello che riguarda l'input (è quindi in grado di effettuare delle selezioni in base a un certo ordinamento) sia per quello che riguarda l'output (é in grado di restituire il risultato ordinato secondo criteri predefiniti). In secondo luogo sono stati implementati sistemi di ricerca e selezione delle informazioni estremamente efficaci, come le query innestate e le funzioni Skolem, attraverso i quali è possibile estrarre le informazioni in maniera puntuale utilizzando una sintassi che combina elementi di XML con elementi della sintassi dei tradizionali query language.

Inoltre occorre tenere in considerazione l'ambiente nel quale presumibilmente si diffonderanno i documenti XML, e quindi l'ambiente nel quale XML-QL dovrà operare prevalentemente. Per questo motivo è necessario considerare le operazioni che dovranno essere effettuate con maggiore frequenza e che saranno quelle di estrazione di dati da documenti XML di grandi dimensioni, e al contrario quelle di integrazione di dati provenienti da numerosi sorgenti XML di piccole dimensioni. Occorrerà prestare attenzione anche alle operazione di trasformazione/conversione dei documenti XML, sia per quello che riguarda la conversione di dati provenienti da modelli non XML in modelli XML, sia per quello che riguarda la trasformazione di dati XML conformi a una certa DTD in dati conformi una DTD differente. Nell'ambito delle operazioni ora individuate XML-QL propone delle soluzioni che permettono attraverso una sintassi specifica di ottenere una risoluzione di queste problematiche di uso comune.

Si propone il seguente esempio di riferimento che rappresenta un frammento di DTD di un documento XML, posto all'ipotetico indirizzo `www.a.b/rest.xml`. Attraverso questo esempio verrà descritto il query language proposto.

```
<!ELEMENT lunch (fast-food | restaurant)*>
<!ELEMENT fast-food (name, address+, phone, specialty)>
<!ATTLIST fast-food year CDATA #REQUIRED>
<!ELEMENT restaurant (name, address+, year?, (creditcard|nocreditcard))>
<!ATTLIST restaurant category CDATA>
<!ELEMENT phone (prefix, number)>
<!ELEMENT address (street+, city)>
```

La DTD proposta indica che un `fast-food` contiene un elemento `name`, uno o più elementi `address`, un elemento `phone` e un elemento `specialty`. Inoltre ogni item fast-food possiede un attributo `year`. Un `restaurant` ha una struttura simile, ma in esso `year` non è un attributo ma un elemento opzionale, non ha l'elemento specialty, ma contiene un elemento che può essere rappresentato da `creditcard` o da `nocreditcard`. Un `restaurant` possiede l'attributo `category`. Un `phone` è rappresentato tramite gli elementi

prefix, number, e address si compone di un elemento street opzionale e di un elemento city obbligatorio. Gli elementi non specificati, come name, specialty, prefix, number,... sono di tipo CDATA e cioè rappresentano dei dati di tipo stringa.

4.4.1 Sintassi base

Si presenta la sintassi base di XML-QL utilizzando come traccia lo schema proposto in [31] e la DTD descritta nel paragrafo precedente. Essendo la grammatica simile a quella di SQL si preferisce mostrare esempi di uso del linguaggio piuttosto che una definizione formale e completa delle operazioni eseguibili. Si sottolinea solamente il fatto che la query più semplice si compone unicamente di due blocchi, l'uno, preceduto dalla clausola WHERE, nel quel sono posti i criteri di selezione e l'altro, preceduto dalla clausola CONSTRUCT, nel quale avviene la costruzione del risultato dell'operazione. Per tutto quello che riguarda in maniera specifica la sintassi del linguaggio, sottolineando il fatto che per ora non esiste nulla di standardizzato, si fa riferimento a quanto scritto in appendice G.

Estrazione dei dati utilizzando i patterns.
XML-QL utilizza gli *element patterns* per immettere dei criteri di selezione dati in un documento XML. Ad esempio per estrarre i nomi dei fast-food il cui prefisso telefonico sia 059 occorre utilizzare il seguente codice:

```
WHERE  <lunch>
          <fast-food>
             <phone>
                <prefix>059</prefix>
             </phone>
             <name> $n </name>
             <address> $a </address>
          </fast-food>
       </lunch> IN ``www.a.b/rest.xml''

CONSTRUCT $n
```

La query procede analizzando il documento XML posto all'URI specificato e seleziona, all'interno di ogni <fast-food>, gli elementi che abbiano almeno un <name> e almeno un <address>, e un <phone> il cui <prefix> coincida con 059. Per ognuno degli elementi selezionati, e per ogni coppia nome-indirizzo individuata, vengono costruite le variabili $n e $a. In questo caso la query ritorna unicamente la variabile $n che rappresenta per ogni

item il nome di un fast-food soddisfacente il criterio introdotto.

Costruzione di dati XML.
Una query può essere utilizzata per creare un nuovo dato XML. La query sottostante serve a produrre un documento XML che contenga un insieme di coppie <street>, <city>, oppure un insieme di elementi <city> appartenenti al documento indicato e sottostanti al criterio di selezione introdotto. La query ritorna sia il campo <name> sia il campo <address> all'interno di un nuovo elemento denominato <result>

```
WHERE <lunch>
         <fast-food>
            <phone><prefix>059</prefix></phone>
            <name> $n </name>
            <address> $a </address>
         </fast-food>
      <lunch> IN ``www.a.b/rest.xml''
CONSTRUCT <result>
             <address> $a </address>
             <name> $n </name>
          </result>
```

La query in esame genera come risultato un nuovo elemento<result> per ogni occorrenza non nulla delle variabili $n $a nel file XML indicato. Viene pertanto creato un nuovo elemento <result> per ogni coppia <name> <address> individuata. In particolare se si considera il seguente frammento di documento XML:

```
<lunch>
   <fast-food year=''1999''>
      <name>China</name>
      <address><city>Los Angeles</city></address>
      <phone><prefix>555</prefix><number>1234</number></phone>
      <specialty>Chinese</specialty>
   </fast-food>
<lunch>
<lunch>
   <fast-food year=''1998''>
      <name>McDonalds</name>
      <address><city>New York</city></address>
      <address><city>Los Angeles</city></address>
```

```
      <phone><prefix>111</prefix><number>98654</number></phone>
      <specialty>Burger</specialty>
   </fast-food>
<lunch>
```

L'applicazione della query descritta in precedenza produce il seguente output:

```
<result>
   <name>China</name>
   <address><city>Los Angeles</city></address>
</result>

<result>
   <name>McDonalds</name>
   <address><city>New York</city></address>
</result>

<result>
   <name>McDonalds</name>
   <address><city>Los Angeles</city></address>
</result>
```

Da notare che viene creato un item <result> per ogni coppia <name> <address> che viene trovata dalla query.

Raggruppamenti utilizzando query innestate.
È possibile eseguire dei raggruppamenti attraverso i quali far comparire assieme a uno specifico elemento altri elementi del documento XML a cui l'elemento stesso é associabile attraverso criteri predefiniti. L'uso di query di questo tipo è ottimale qualora si considerino delle DTD nelle quali sia possibile immettere uno stesso elemento molteplici volte. In quel caso, utilizzando la tipologia di query proposta nel paragrafo precedente, venivano generati molteplici item come risultato, uno per ogni occorrenza nel quale l'elemento si presentava. Utilizzando le query innestate é possibile superare questo problema e ad esempio é possibile associare a ogni fast-food tutti gli <address> a cui un elemento è associato. La query seguente propone infatti come risultato tutti gli <address> associati a ogni <name>.

```
WHERE <lunch>
   <fast-food> $p </fast-food>
```

```
</lunch> IN ‘‘www.a.b/rest.xml’’,

    <name> $n </name>
    <phone>
       <prefix>059</prefix>
    </phone> IN $p

CONSTRUCT <result>
             <name> $n </name>
             WHERE <address> $a </address> IN $p
             CONSTRUCT <address> $a </address>
</result>
```

La prima operazione effettuata dalla query è associare alla variabile `$p` il valore di ogni item <`fast-food`> presente nel documento XML. Una volta che l'associazione tra elementi e variabili è effettuata, la variabile `$p` può essere utilizzata come elemento all'interno del quale operare. In effetti la query che si sta analizzando ricerca all'interno degli elementi identificati dalla variabile `$p` quelli che soddisfino i criteri indicati e partendo da essi costruisce il risultato. Da notare la presenza della clausola `WHERE` anche in sede di definizione dell'output, utilizzata per effettuare il raggruppamento tra gli elementi indicati. Considerata inoltre la frequenza con la quale avviene l'operazione di associazione tra variabile e elemento, il linguaggio prevede un'espressione specifica `CONTENT_AS` da utilizzare in queste circostanze. La query proposta in precedenza può essere quindi riscritta in questo modo:

```
WHERE <lunch>
         <fast-food>
         <name> $n </name>
         <phone><prefix>059</prefix></phone>
         </fast-food>
      </lunch> CONTENTS_AS $p IN ‘‘www.a.b/rest.xml’’
CONSTRUCT <result>
             <name> $n </name>
             WHERE <address> $a </address> IN $p
             CONSTRUCT <address> $a </address>
          </result>
```

L'applicazione della query genera il seguente risultato che deve essere raffrontato con il risultato prodotto dalla query descritta nel punto precedente:

```
<result>
```

```
   <name>China</name>
   <address><city>Los Angeles</city></address>
</result>

<result>
   <name>McDonalds</name>
   <address><city>Los Angeles</city></address>
   <address><city>New York</city></address>
</result>
```

Joining degli elementi per valore.
Attraverso il linguaggio XML-QL è possibile esprimere l'operatore di "join" il cui scopo è quello di unire due o più elementi aventi lo stesso valore. Come esempio si propone la query seguente che ritorna come risultato tutti i <restaurant> nel cui <address> sia presente almeno un <fast-food> aperto dopo il 1995.

```
WHERE <lunch>
         <restaurant>
            <address>
               <street>$s</street>
               <city>$c</city>
            </address>
         </restaurant>
      </lunch>
CONTENT_AS $a IN ``www.a.b/rest.xml''

      <lunch>
         <fast-food year=$y>
            <address>
               <street>$s</street>
               <city>$c</city>
            </address>
         </fast-food>
      </lunch> IN ``www.a.b/rest.xml''

y < 1995

CONSTRUCT <restaurant> $a <restaurant>
```

È possibile, per rendere più agevole la scrittura della query, utilizzare il costrutto `ELEMENT_AS` che consente di associare un elemento a una variabile.

4.4.2 Esempi avanzati in XML-QL

In questa sezione vengono introdotte caratteristiche avanzate di XML-QL. Queste caratteristiche includono la possibilità di utilizzare i *tag variables*, che permettono l'estrazione di informazioni anche all'interno di elementi del grafo che rappresenta la struttura del documento, e le *regular path-expessions* attraverso le quali è possibile specificare un percorso arbitrario del grafo XML nel quale effettuare la ricerca. Attraverso l'uso di queste potenzialità XML-QL viene dotato di strumenti efficaci che permettono di operare in maniera agevole sia per integrare i dati appartenenti a sorgenti diversi, sia per trasformare dati XML conformi a una DTD in dati conformi a un'altra DTD.

Variabili tag.
L'XML-QL supporta l'uso di variabili di tipo tag vale a dire di variabili che possono essere associate ai tag che compongono la struttura del documento e che possono essere quindi utilizzate come criteri di ricerca della query. Attraverso questo sistema é possibile ad esempio estendere un criterio di ricerca a piú tag, in modo da contemplare diverse rappresentazioni strutturali della medesima semantica.

```
WHERE <lunch>
         <$p>
            <address> $a </address>
            <phone><prefix>059</prefix></phone>
            <$e> Chinese</$e>
         </$p>
      </lunch> IN "www.a.b/rest.xml",
      $e IN {name, specialty}
CONSTRUCT <$p>
          <address> $a </address>
          <$e>Chinese<$e>
          </$p>
```

Nell'esempio proposto sia $p sia $e indicano delle variabili di tipo tag e possono quindi rappresentare qualsiasi tipo di elemento. Nella fattispecie $e è vincolato a rappresentare o elementi di tipo <name> o elementi di tipo <specialty>.

Regular-path Expressions.
Il dato XML può essere contenuto in tag piú o meno innestati in una struttura. Nel capitolo precedente si è mostrato come un documento XML possa essere rappresentato graficamente attraverso alberi o attraverso altre tipolo-

gie di grafi con strutture piú arbitrarie. All'interno di un albero, struttura rappresentantiva di un documento XML semplice, si possono innestare gli elementi ID e IDREF che, implementando la possibilità di percorrere l'albero in maniera "trasversale", rendono la rappresentazione strutturale meno regolare e di conseguenza più complesso il raggiungimento dei nodi foglia. Per ricercare informazioni in tali strutture occorre spesso attraversare percorsi impredicibili seguendo il grafo rappresentativo del documento, e inoltre percorsi diversi possono raggiungere lo stesso elemento. Le query in queste particolari strutture vengono agevolate utilizzando XML-QL grazie all'uso di *regular-path expressions*, che permettono di specificare elementi strutturali a una profondità di innestamento arbitraria.
La DTD che ora si introduce come esempio mostra in maniera basilare la struttura logica si un documento XML, nel quale ogni elemento può essere definito in maniera ricorsiva:

```
<!ELEMENT elementoXML(nome, contenuto?, elementoXML*)>
<!ELEMENT nome CDATA>
<!ELEMENT contenuto CDATA>
```

Ogni elemento XML infatti ha un nome che lo identifica, ha un contenuto, e può racchiudere un altro elemento XML. É possibile effettuare delle query utilizzando delle regular-path expressions su un documento la cui DTD sia quella introdotta. Ad esempio il codice seguente produce il nome di ogni `elementoXML` che ha come contenuto fast-food indipendentemente dal livello di profondità nel quale l'elemento é posto.

```
WHERE <elementoXML.*>
         <nome>$n</nome>
         <contenuto>fast-food</contenuto>
      </elementoXML> IN "www.a.b/rest.xml",
CONSTRUCT <result>$n</result>
```

In questo contesto si é introdotto l'operatore `*` che indica un qualsiasi livello di innestamento all'interno dell'elemento `elementoXML`, ovvero, una qualsiasi sequenza di termini `elementoXML` nella struttura che definisce il documento. La sintassi consente anche l'uso di un altro operatore, `$`, che indica un generico tag e può essere inserito ovunque sia inseribile il tag. La query scritta in precedenza può essere anche formulata in questo modo:

```
WHERE <$*>
         <nome> $n </nome>
         <contenuto>fast-food</contenuto>
      </$*> IN "www.a.b/rest.xml"
 CONSTRUCT <result>$r</result>
```

Allo stesso modo esistono altri operatori che possono essere utilizzati nella formulazione di query: l'operatore di concatenazione, indicato con `.`, può essere utilizzato per indicare una determinata tipologia di elementi, o meglio l'innestamento di due o piú elementi: `<*.nome>` ad esempio indica un qualsiasi elemento seguito dall'elemento nome. É infine implementato anche l'operatore di alternanza, rappresentata con `|`, che indica la scelta tra due o piú elementi. Attraverso gli operatori introdotti é possibile effettuare delle ricerche complesse che sfruttino come elemento discriminante e di selezione la struttura del documento.

Trasformazione di dati XML utilizzando funzioni Skolem.
Si presuppone che XML-QL possa venire utilizzato in maniera prevalente nell'ambito dell'integrazione di sorgenti XML. In queste circostanze può essere necessario trasformare dati XML conformi a una determinata DTD in dati XML conformi a una DTD differente. Si supponga di avere una DTD che identifichi l'elemento phonerestaurant:

```
<!ELEMENT phonerestaurant (namerestaurant, prefix, number)*>
```

Si può scrivere una query che trasformi i dati XML conformi alla DTD dei ristoranti in precedenza utilizzato in dati conformi al frammento di DTD sopra indicato.

```
WHERE
<lunch>
   <fast-food>
     <phone>
        <prefix>$p</prefix>
        <number>$n</number>
        <name>$name</name>
     </phone>
   </fast-food> IN "www.a.b/rest.xml",
</lunch>
CONSTRUCT
<phonerestaurant ID=phonerestaurantID($p,$n)>
  <prefix> $p </prefix>
  <number> $n </number>
  <namerestaurant> $name </namerestaurant>
</phonerestaurant>
```

La query utilizza gli identificatori di oggetto (OID) e una funzione Skolem per controllare il modo nel quale il risultato é raggruppato. In questo caso

`phonerestaurantID` è una funzione Skolem che genera un nuovo identificatore ogniqualvolta si individua una coppia distinta di valori `prefix, number`.

Integrazione di dati da sorgenti XML diversi.
Attraverso l'uso di XML-QL è possibile integrare dati provenienti da sorgenti diversi. Tale opportunità é realizzabile utilizzando interrogazioni simultanee su sorgenti diversi in combinazione con l'operazione di join (realizzabile anche attraverso le funzioni Skolem). Un esempio può essere offerto dalla possibilità di integrare le informazioni contenute nella DTD relativa ai fast-food (vedi 4.4) con le informazioni provenienti da un altro sorgente nel quale sono contenuti dati riguardanti generiche persone. Si suppone che in <fast-food> sia inserito l'elemento <owner> campo di tipo CDATA nel quale é indicato il proprietario del fast-food utilizzando il quale viene effettuata l'operazione di join.

```
WHERE
<*.fast-food>
    <phone></phone> ELEMENT_AS \$ph
    <owner> $own </owner>
</fast-food> IN "www.a.b/rest.xml",

<*.person>
   <name>$own</name>
    <address></address> ELEMENT_AS \$add
</person> IN "www.a.b/pers.xml",

CONSTRUCT
<result>
     <owner>$own</owner>
     $ph $add
</result>
```

Estensioni per il modello dati ordinato.
Anche se si opera all'interno del modello dati ordinati, i criteri indicati nella clausola `WHERE` sono ancora interpretati come non ordinati. Per esempio se si indica <a> $x <c> $y </c> , sono rappresentati sia i dati <a> string 1<c>string2</c> sia i dati <a> <c>string2</c>string 1. Sebbene i criteri siano considerati come non ordinati, le variabili contenute all'interno dei criteri sono ordinate. Per questo motivo, considerando l'esempio proposto, e fornendo i dati seguenti, le variabili sono ordinate secondo il prospetto che segue

```
<a>
<b> string1 </b>
<c> string2 </c>
<b> string3 </b>
<c> string4 </c>
</a>
```

$x	$y
string1	string2
string1	string4
string3	string2
string3	string4

Inoltre è possibile utilizzare i seguenti operatori:

- supporto per gli indici: è possibile indicare una variabile che consenta di estrarre gli indici locali degli elementi
- clausola `ORDER-BY`: specifica l'ordine degli elementi nell'output.

Definizioni di funzioni.
XML-QL può implementare delle funzioni che abbiano documenti XML come argomenti. Tali funzioni possono essere utilizzate ad esempio per incrementare il riuso delle query, per il fatto che attraverso di esse una singola query può essere utilizzata per effettuare delle ricerche su differenti documenti. Una funzione generalmente ha come argomento gli URI dei sorgenti XML o delle DTD che si vogliono interrogare. Il risultato di una funzione é un file xml (o un dtd) posto all'URI indicato nell'intestazione della funzione stessa. Il prototipo di una funzione è ad esempio il seguente:

```
FUNCTION findfast-food(‘‘www.a.b/rest.xml’’)
```

e ad esso potrebbe corrispondere il codice seguente:

```
FUNCTION findfast-food(‘‘$restaurant’’):’’www.a.b/result.xml’’
WHERE [...]  IN $restaurant
CONSTRUCT [...]
```

4.4.3 Sviluppi futuri

Sono sostanzialmente due le direzioni nelle quali si concentrerà l'attività di evoluzione dell'XML-QL: da un lato si é valutato come necessario espandere l'espressività del linguaggio, dall'altro dovrà essere realizzato un migliore supporto per gli standard imposti dall'XML.

Per quello che riguarda la prima direzione, gli obiettivi considerati principali sono rappresentati dall'estensione all'XML-QL di una sintassi che permetta di utilizzare le cosiddette funzioni di aggregazione allo stesso modo in cui con esse si opera in SQL. Potrebbero essere particolarmente utili le funzioni di massimo, di minimo, che consentono un ordinamento dei dati. L'uso delle funzioni di aggregazione in combinazione con la clausola `GROUP-BY` permette di effettuare un ordinamento dei dati risultato dell'operazione di query.

Per quello che riguarda la realizzazione di un migliore supporto per gli standard XML, l'ottica é quella di rendere le query XML-QL rappresentabili completamente in XML, in modo che possano essere contenute in maniera corretta all'interno di un documento XML.

Inoltre, una spinta innovativa particolare può essere fornita dai linguaggi che si stanno evolvendo a partire dallo standard XML. Nella fattispecie sono di grande importanza le due proposte di linguaggio, XPointer e XLink, il cui scopo é quello di gestire il collegamento tra documenti differenti fornendo un sistema attraverso il quale collegare dati appartenenti a sorgenti diversi a partire da un unico documento di origine. Utilizzando i meccanismi individuati in queste proposte sono ipotizzabili degli algoritmi che permettano di effettuare delle query su diversi sorgenti e quindi consentano la decomposizione della query principale in query piú specifiche che possano essere poi indirizzate ai singoli server.

La standardizzazione della struttura delle DTD, infine, é un problema che ha rilevanza sull'ottimizzazione delle interrogazioni. Lo schema per ora impone unicamente dei vincoli sintattici e non semantici ai dati XML. É facilmente intuibile che partendo da strutture standardizzate siano costruibili algoritmi di ottimizzazione che rendano le query più efficaci e più veloci.

4.4.4 Uso delle viste

Le viste sono uno strumento in un certo senso già definito in XML. Il linguaggio per la generazione di regole di rappresentazione delle pagine, XSL, proposto al W3C, offre un meccanismo di ristrutturazione/trasformazione dei documenti XML attraverso la definizione di *templates rules*. Il meccanismo delle viste dovrebbe andare oltre alla pura definizione del layout delle

pagine, permettendo la visualizzazione dei dati applicando criteri predefiniti di selezione fra essi.

Il linguaggio XML-QL non presenta esplicitamente dei comandi che permettano la gestione di viste. Questo fatto può essere considerato come una delle principali sue carenze. Infatti quando il dato è rappresentato attraverso il linguaggio XML, nel quale non si ha una forte distinzione tra dato e struttura del dato, l'uso di esse potrebbe diventare essenziale. Infatti in questo contesto sia é spesso necessario integrare dei dati provenienti da sorgenti differenti, sia è utile potere inserire un'interfaccia che filtri i dati rendendoli più strutturati.

Non essendo quindi presenti degli operatori ad hoc, e non essendo pure le viste considerate rilevanti in alcuna delle altre proposte di linguaggio di interrogazione specifico per XML, in questa sede ci si limita a evidenziare i privilegi che con l'introduzione di tale meccanismo vengono indotti nella gestione dei dati semistrutturati. In fase di progettazione, si potrebbero perciò ipotizzare delle viste in XML che non siano delle semplici repliche di quelle proposte nei paradigmi relazionale, o object oriented, ma che superino i limiti che tradizionalmente sono stati attribuiti a tali strutture. In particolare occorre prestare attenzione al fatto che spesso quando si opera con di file di tipo XML si opera con file distribuiti e che quindi necessitano di strumenti particolari, quali, ad esempio, la possibilità di evidenziare cambiamenti nei sorgenti ai quali si riferiscono. In primo luogo allora il sistema delle viste deve consentire mezzi attraverso i quali segnalare che é avvenuto un aggiornamento nei dati sia nella direzione vista database, sia nella direzione opposta. La risoluzione di tali problematiche, che interessa non solo la struttura del singolo documento XML, ma l'architettura complessiva del DBMS, presenta ancora aspetti incerti e non ancora approfonditi dalla ricerca.

4.5 XML-GL: un linguaggio grafico per query

XML-GL [32] é un linguaggio grafico per effettuare query su documenti XML proposto al W3C per la standardizzazione. La caratteristica principale di tale linguaggio é il formalismo grafico attraverso il quale sia viene rappresentato il documento XML (e la DTD corrispondente) sia viene introdotta la possibilità di esprimere delle query in maniera intuitiva. Il modello grafico introdotto allo scopo, denominato XML-GDM (XML Graphical Data Model), verrà utilizzato per rappresentare la struttura dei documenti XML sui quali effettuare le query, per scrivere le stesse query e per rappresentare la DTD dei risultati. Rispetto alle specifiche inizialmente individuate sono state

inserite recentemente ulteriori estensioni che permettono la gestione dei link e di query ricorsive.

Considerando lo strumento grafico un utile approccio alla formulazione di query e essendo XML-GL non solo un modo per rappresentare graficamente le interrogazioni, ma anche, grazie all'apporto della logica descrittiva **G-Log** [51] dalla quale trae fondamento teorico, uno strumento attraverso il quale compiere effettivamente l'operazione di estrazione di informazione, si ritiene utile presentare brevemente in questa sede tale proposta. Lo scopo é soprattutto quello di mostrare le differenze, nella formulazione delle interrogazioni, che intercorrono fra XML-QL e XML-GL.

4.5.1 Il modello dati grafico

Il modello dati XML-GDM consiste sostanzialmente di tre concetti:

- **Oggetto**, rappresentato tramite un rettangolo, indica un elemento astratto senza un valore rappresentabile direttamente
- **Proprietà**, rappresentata mediante un cerchio connesso con l'elemento al quale si riferisce, é caratterizzata da un etichetta che coincide con il suo nome e nel caso della descrizione del documento (e non della DTD) ha un valore associato
- **Relazione**, rappresentata mediante un arco tra due oggetti, indica una associazione semantica. Le relazioni hanno una orientazione, dall'oggetto sorgente all'oggetto destinazione.

Utilizzando i concetti ora introdotti é possibile rappresentare un grafo XML-GL e attraverso di esso documenti XML, le corrispondenti DTD, e le query.

Un grafo XML é un grafo <N,A>, dove:

- N é un insieme finito di nodi che rappresentano gli elementi XML. I nodi sono suddivisi in due insiemi disgiunti: l'insieme dei nodi elemento E, etichettati con il nome dell'elemento che rappresentano e l'insieme dei nodi proprietà P, etichettati con una stringa indicante il nome e opzionalmente il valore. I nodi in P sono ancora suddividibili in due ulteriori insiemi disgiunti, l'insieme dei nodi attributo e l'insieme dei nodi contenuto. Esiste una etichetta particolare ANY che può essere usata per definire un nodo nella maniera piú generica possibile.
- A é un insieme di archi etichettati (n,λ,n'), dove λ é un arco e n,n' appartengono a N. Gli archi appartenenti a A possono essere di due tipi:

gli archi C denominati "containment arc" e gli archi R, i reference arc. Gli archi appartenenti a C hanno tutti la stessa etichetta CONT che può essere omessa, mentre gli archi di tipo R sono etichettati con il nome dell'attributo IDREF al quale si riferiscono. 'E possibile esprimere cardinalità e opzionalità attraverso delle opportune notazioni simili a quelle del modello Entity-Relationship.

- Per ogni nodo n in E é possibile indicare un ordine con il quale valutare i nodi figli direttamente raggiungibili (e cioé collegati attraverso un arco). Il primo elemento é indicato graficamente mediante un tratto sull'arco. Gli elementi che devono essere valutati successivamente sono scelti in base al verso orario.

- É possibile definire la mutua esclusione di un insieme di elementi attraverso un segmento, etichettato con la scritta XOR, che tagli tutti gli elementi in esclusione.

4.5.2 Il query language

La rappresentazione di una query XML-GL su un insieme di documenti XML si compone semplicemente di una coppia di grafi. I grafi posti alla sinistra mettono in evidenza le informazioni che devono essere estratte. L'informazione che si ottiene dal grafo posto a destra é l'insieme di elementi, recuperati nella parte a sinistra, che deve essere mostrato nel risultato.

Una query XML-GL logicamente si compone di quattro parti:

1. **Extract part**: é la parte che identifica l'oggetto della query, nel senso che viene indicato sia il documento dal quale estrarre informazioni, sia gli specifici elementi all'interno di questo documento (facendo un parallelismo con SQL, nella extract part viene indicato quanto scritto nella clausola from).

2. **Match part** (opzionale): in questa parte vengono indicate le condizioni logiche che gli elementi estratti devono soddisfare per essere inseriti nel risultato della query (il parallelismo con SQL porta a fare corrispondere a questa parte la clausola where)

3. **Clip part**: sono specificati i sotto elementi degli elementi estratti che soddisfano la match part, che devono essere inseriti nel risultato (sostanzialmente in questa parte l'informazione fornita corrisponde alla select del linguaggio SQL)

4. **Construct part** (opzionale): serve per specificare nuovi elementi che devono essere inclusi nel risultato e le relazioni degli stessi con gli elementi estratti (Sostanzialmente il corrispondente SQL di questa operazione può essere considerato lo statement create view).

Le query possono essere suddivise sulla base degli archi che collegano grafi posti nella parte sinistra con grafi posti nella parte destra. Tali archi servono a collegare elementi facenti parte della struttura originaria del file con nuovi elementi facenti parte della struttura del risultato dell'interrogazione. Una query é detta semplice se esiste un unico legame tra componenti della parte sinistra e della parte destra. Negli altri casi la query é complessa. É stato dimostrato che le query complesse sono riconducibili a unioni di query semplici [52]. Inoltre é stato individuato un algoritmo che consente una risoluzione delle query così come sono state strutturate. Tale algoritmo sfrutta una tabella, detta tabella dei risultati, nella quale vengono poste le istanze che soddisfano la query. L'algoritmo si basa su tre passi:

1. **Extract-Match step**. Passo che ha come input il grafo di sinistra e nella quale si individuano, confrontando tale grafo con l'istanza del documento, le strutture che sono associabili. Le corrispondenze individuate vengono poste nella tabella dei risultati.

2. **Clip step**. Procedura che ha come ingresso la tabella dei risultati e il grafo di destra. In questa fase vengono costruite le strutture che comporranno il risultato.

3. **Construct step**. Fase nella quale si completa la costruzione della struttura che compone la risposta della query e che in cui la struttura creata viene arricchita dei nodi inventati, cioé dei nodi costituiti nel corso dell'esecuzione della query e non presenti in precedenza nella costruzione della istanza. Input di questa parte dell'algoritmo é la parte destra della query e la tabella dei risultati, popolata dai valori di OID trovati negli step precedenti.

4.5.3 Alcuni esempi di query XML-GL

Si propongono in questa sezione alcuni esempi di query XML-GL. In particolare si ripropongono alcune delle query che sono state scritte al momento della presentazione del linguaggio XML-QL. In questo modo si vuole favorire il confronto fra i due linguaggi e evidenziare le peculiarità grafiche di XML-GL.

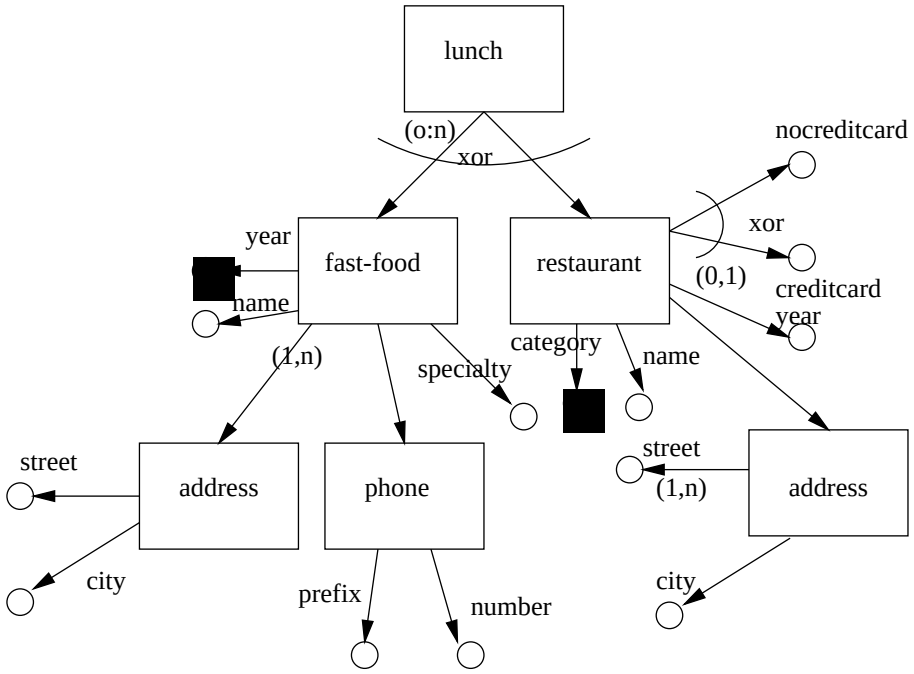

Figura 4.1: La DTD secondo il modello dati XML-GDM

Si ripropone pertanto, mediante l'uso del modello dati XML-GDM, una rappresentazione della DTD già utilizzato nella sezione 4.4

Come si é sottolineato precedentemente attraverso l'analisi del modello grafico rappresentativo del documento, utilizzando un apposito formalismo é possibile formulare delle query. In particolare si vogliono mostrare tre esempi di query, uno relativo alla pura estrazione di informazioni, il secondo nel quale si focalizza l'attenzione sulla costruzione del risultato e il terzo nel quale verrà evidenziata una operazione di join, rimandando a trattazioni piú specifiche [32] la definizione rigorosa della sintassi del linguaggio.

La query che qui si propone estrae tutti i nomi di fast-food dei quali il prefisso telefonico é rappresentato dal numero 059.

Occorre ricordare che il grafo di sinistra esprime la condizione che l'istanza deve soddisfare, mentre il grafo di destra mostra la "ricostruzione" dell'informazione. Nella fattispecie il risultato della query ha la struttura che possedeva nel documento che si é interrogato.

Attraverso il secondo esempio si vuole mostrare come sia possibile, utilizzando il linguaggio XML-GL, costruire attraverso l'operazione di query un nuovo documento XML nel quale siano contenuti unicamente alcuni campi

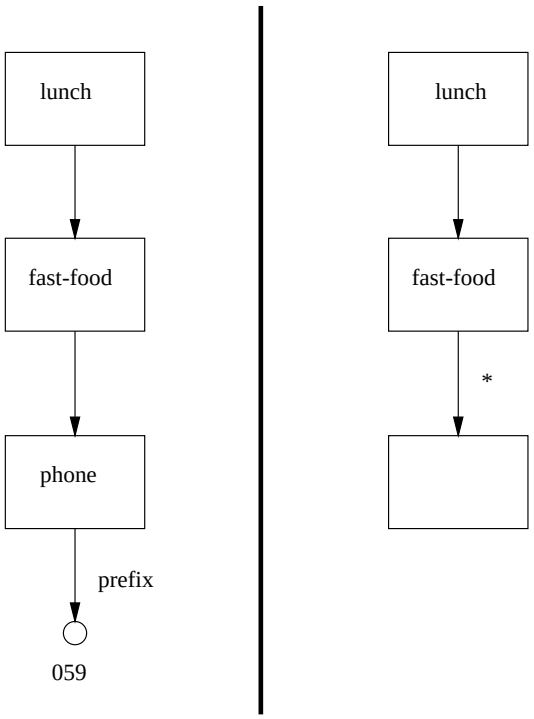

Figura 4.2: Esempio di query di selezione

selezionati. La query che si propone costruisce un nuovo documento XML nel quale all'interno del tag result sono contenuti unicamente gli indirizzi dei fast-food i cui valori soddisfando la condizione indicata.

Infine si vuole mostrare un esempio di query che realizzi il join fra elementi. In particolare l'interrogazione che si propone genera come risultato le istanze di fast-food e di restaurant che hanno sede nella stessa via e nella stessa città.

Le query che si sono mostrate rappresentano unicamente un esempio della metodologia grafica, attraverso la quale vengono effettuate interrogazioni, che viene introdotta con XML-GL. Si rimanda pertanto a trattazioni piú specifiche la presentazione della sintassi completa del linguaggio e l'evidenziazione di tutte le opportunità che XML-GL permette.

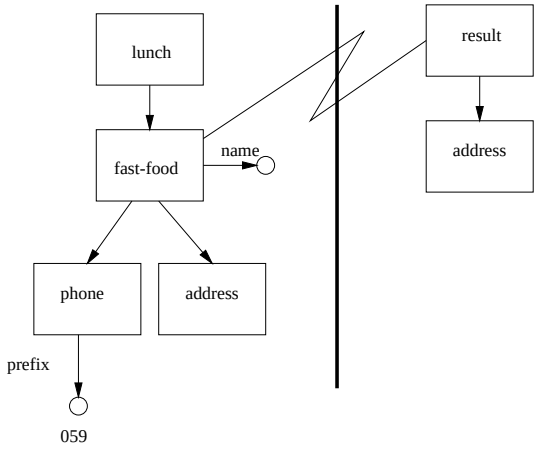

Figura 4.3: Esempio di query di costruzione

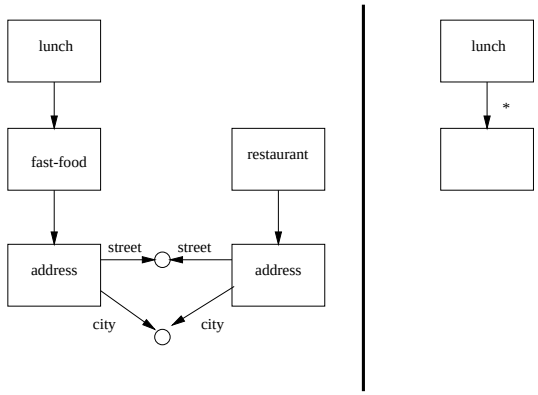

Figura 4.4: Esempio di query di join

4.6 Conclusioni

In questo capitolo sono stati presentati sia le condizioni che il query language dovrà soddisfare, sia due esempi di linguaggi che operano in XML. Si é poi osservato che al momento, pur essendo XML-QL il query language piú completo, non soddisfa completamente le richieste del W3C. In particolare XML-QL non rispetta sostanzialmente due tipologie di richieste del W3C: la prima concerne i linguaggi che devono essere supportati dal query language e la seconda la mancanza di alcuni operatori esplicitamente richiesti invece in [50].

Per quello che concerne il primo aspetto, nelle caratteristiche indicate dal

W3C viene espressamente richiesta l'interdipendenza tra il query language e il risultato dell'attività di altri gruppi di lavoro legati a specifici aspetti di XML. Essendo XML-QL un linguaggio nato nel 1998, anche se in parte successivamente modificato, non tiene in considerazione né gli standard individuati successivamente dal W3C, né le proposte che si sono candidate nei periodi successivi. Nella fattispecie XML-QL non prevede né interoperabilità con XML-Schema, proposta in fase di standardizzazione (vedere il capitolo 3), né supporta gli standard XML-Namespaces e XPath successivamente introdotti. Per quello che concerne gli operatori, occorre evidenziare la mancanza sia del quantificatore universale, sia del quantificatore esistenziale, sia di qualsiasi operatore che generi come risultato una variabile di tipo boolean. In questo momento non é possibile infatti esprimere query che siano basate su delle condizioni (come la restituzione delle istanze unicamente se esiste una istanza che soddisfi particolari vincoli), ma é possibile unicamente generare delle query che diano come risultato istanze di documenti XML.

Per quello che concerne poi XML-GL, occorre sottolineare che la principale peculiarità di tale linguaggio é quello di operare in modo grafico, sia per quello che riguarda l'interfaccia con il progettista (e quindi la scrittura delle interrogazioni) sia per quello che riguarda il recupero delle istanze soddisfacenti i vincoli imposti. Per questo motivo occorre valutare non tanto quanto quel linguaggio si discosti dalle specifiche richieste, ma soprattutto se possa essere piú intuitivo operare in quella maniera, mantenendo in ogni caso tutta la semantica richiesta da un query language. Dalla analisi di XML-GL si può verificare che tale obiettivo é stato raggiunto ed é stato quindi possibile realizzare (almeno in via teorica, non esistendo ad oggi alcuna implementazione) un linguaggio puramente grafico con le stesse caratteristiche di un linguaggio di tipo tradizionale.

Infine occorre sottolineare che, essendo al momento in opera il gruppo di lavoro costituito per realizzare il linguaggio standard, non sono vi sono stati negli ultimi mesi sviluppi in questo settore, ma si attende quanto prodotto dal W3C. In particolare nel mese di Maggio 2000 il gruppo ha prodotto un documento di lavoro concernente il data model sul quale il linguaggio si baserà e ci si aspetta che nel entro il mese di Ottobre 2000 venga pubblicata una prima proposta di linguaggio.

Capitolo 5

Il wrapper XML

5.1 Introduzione

Il wrapper che sovrintende ai documenti di tipo XML, come ogni altro wrapper utilizzato nell'ambito del progetto MOMIS, deve sostanzialmente soddisfare due funzionalità. La prima, consiste nel fornire una descrizione in ODL_{I^3}della sorgente alla quale é connesso, la seconda, nel consentire l'esecuzione di query, generate dal query manager, sulla specifica sorgente. La prima funzionalità indicata viene svolta completamente dal parser DTD e dal traduttore DTD-ODL_{I^3}, utilizzando le tabelle e gli algoritmi di conversione che verranno in seguito illustrati. In particolare é necessario osservare che, oggetto del parser, non é il file dati XML, ma il file (o la sezione del file XML) contenente la rappresentazione della struttura del sorgente. Per questo motivo il wrapper sovrintende unicamente a file XML valid. I file well-formed, non avendo DTD associato, non vengono trattati.

Non é stato ancora possibile implementare la seconda funzionalità, ovvero l'esecuzione di query sulla specifica sorgente. Come si é visto infatti non é ancora disponibile uno standard di interrogazione di sorgenti semistrutturati rappresentati in XML, e ognuno dei linguaggi di interrogazione proposti al World Wide Web Consortium presenta delle carenze rispetto ai "desiderata" espressi dal W3C che fanno presupporre che il query language che verrà standardizzato non possa essere una semplice evoluzione di un linguaggio esistente, ma debba essere qualcosa di completamente differente da quanto finora ipotizzato. Inoltre, in ragione della semplicità strutturale delle query locali generate dal Query Manager, il problema della traduzione nel query language specifico dell'interrogazione non sembra presentare grosse incognite e grandi elementi di problematicità. Si é preferito pertanto non implementare l'esecuzione di query locali, per focalizzare maggiormente l'attenzione su quanto

finora standardizzato.

5.2 La traduzione di elementi critici

Nel primo e nel secondo capitolo si sono messe in luce le differenze tra rappresentazioni di strutture di basi di dati espresse in ODL_{I^3}e strutture espresse in XML. In quel contesto si é evidenziato il fatto che, nonostante alcune diversità strutturali, tutta la semantica esprimibile in XML fosse analogamente rappresentabile in ODL_{I^3}. Si é visto infatti come un elemento XML fosse teoricamente traducibile in attributo ODL_{I^3}. É comunque opportuno ricordare i principali elementi di diversità individuati:

- **Gestione degli attributi**: Il linguaggio ODL_{I^3}modella una struttura ad albero nella quale ogni nodo può contenere un valore, e in quel caso il nodo viene detto atomico, oppure nella quale ogni nodo può referenziare un altro nodo, e in quel caso viene detto complesso. Anche il linguaggio XML permette di modellare analoghe strutture ad albero, con l'unica differenza che il nodo di una struttura XML può veicolare una semantica maggiore. Infatti é possibile associare al nodo degli attributi attraverso i quali fornire una ulteriore qualificazione del nodo al quale sono riferiti, ma la cui rappresentazione strutturale non é analoga a quella di un nodo.

- **Chiavi primarie, esterne, candidate**: Mentre in ODL_{I^3}ogni elemento può avere funzione di chiave primaria, di chiave candidata o di chiave esterna, XML non applica il concetto di chiave, ma il concetto di identificatore e di riferimento a un identificatore. La qualificazione di identificatore o di riferimento non viene poi assegnata all'elemento, ma a uno specifico attributo dell'elemento stesso, per cui non é un elemento ad essere identificatore, ma il valore assunto dallo specifico attributo.

- **Tipi di dato**: il linguaggio XML non supporta i tipi di dato, ma ogni nodo terminale ha un unico tipo di valore denominato PCDATA.

- **Strutture dati**: ODL_{I^3}distingue tra *attribute*, quando si vuole definire il singolo nodo, e *interface*, quando si vogliono definire degli insiemi di nodi. In ODL_{I^3}é poi possibile esprimere delle strutture complesse, definire dei tipi e delle enumerazioni. In XML esistono invece solo due strutture: l'*element* e l'*attribute*. Inoltre non si utilizza una sintassi diversa per distinguere tra la rappresentazione del singlo nodo e la rappresentazione di insiemi di nodi. Tale distinzione viene effettuata sulla base della definizione del contenuto dell'elemento. In questo

modo si vengono a individuare quattro tipologie di nodi: i nodi di tipo *children*, utilizzati per definire gruppi di elementi, i nodi di tipo *mixed*, per definire elementi che possano essere o terminali o rappresentare una struttura dati, i nodi di tipo *empty*, per definire dei nodi a cui non corrisponda alcun valore nella istanza dati, e nodi di tipo *any*, quando non si vuole definire in termini strutturali a che cosa corrisponda il determinato elemento. A queste tipologie di strutture dati non sempre é definito un corrispettivo in ODL_{I^3}. Pertanto si é dovuta progettare un'apposita rappresentazione in ODL_{I^3}di tali strutture, cosí come evidenziato in appendice.

- **Composizione di strutture**: XML permette di creare una struttura dati che sia il risultato dell'unione di strutture precedentemente create. In questo modo la DTD può essere ottenuta come unione di molteplici DTD, e in termini fisici può essere dunque collocata su molteplici file separati.

Una analisi più approfondita degli elementi di criticità sopra elencati e la risoluzione adottata nel wrapper progettato verrà descritta successivamente quando si procederà alla descrizione del software.

5.3 L'algoritmo di traduzione

Il funzionamento del wrapper XML, per quello che concerne la parte di analisi del file sorgente, può essere suddiviso in tre blocchi funzionali che vengono eseguiti in successione temporale.

1. **Parsing della DTD**: in questa fase viene effettuata la lettura della DTD e vengono create le strutture dati corrispondenti agli elementi individuati. La parte più critica é quella relativa alla gestione delle entità, per il trattamento delle quali é necessario operare attraverso una fase di pre-processing per mezzo della quale si sostituisca all'entità il valore che rappresenta.

2. **Determinazione delle chiavi**: é necessario, prima di compiere la traduzione del documento XML, individuare gli elementi e gli attributi che diventeranno nel documento ODL_{I^3}chiavi primarie, chiavi candidate e chiavi esterne. Tale necessità nasce dal fatto che in XML non esiste il concetto di chiave, ma unicamente il concetto di identificatore e di riferimento a identificatore. In questo secondo caso inoltre la sintassi non consente di indicare a quale elemento l'attributo si riferisca, ma solamente il fatto che l'attributo sia un riferimento a un identificatore.

E' necessaria pertanto l'interazione con l'utente attraverso la quale si completi la selezione degli identificatori che "saranno elevati" al rango di chiavi, e attraverso la quale venga esplicitato a quale identificatore un determinato elemento faccia riferimento.

3. **Traduzione del documento**: in questa fase avviene la traduzione del documento XML nel documento ODL_{I^3}, sulla base delle informazioni che si sono ottenute negli step precedenti.

5.3.1 Parsing della DTD

L'operazione di parsing consiste nell'effettuare la lettura della DTD da analizzare e nella creazione delle specifiche strutture dati che verranno successivamente utilizzate dal traduttore. In particolare é possibile distinguere due fasi nelle quali avviene questa operazione. Nella prima fase, propedeutica alla seconda, devono essere effettuate le seguenti operazioni:

- **Individuazione della DTD**: la DTD può essere collocato su specifico file oppure può appartenere a una sezione del documento dati. Nel caso in cui si presenti questa seconda evenienza, é necessario individuare la parte di file XML nella quale é contenuto la DTD e effettuare su di essa le operazioni definite in seguito.

- **Risoluzione delle entità e delle sezioni condizionali**: sostituzione nella DTD delle entità con il contenuto che rappresentano, e scrittura della parte vera della sezione condizionale (la parte falsa non deve essere analizzata).

- **Risoluzione di contenuti innestati**: la sintassi della DTD permette di potere esprimere attraverso un unica espressione elementi innestati a più livelli di profondità. In merito a questo, é opportuno considerare il seguente frammento di DTD:

```
<!ELEMENT persona (((nome, cognome)|codice_fiscale),indirizzo,telefono)>
```

L'esempio mostra che nel file dati associato una persona puø' essere rappresentata attraverso un nome, un cognome, un indirizzo e un telefono, oppure attraverso un codice fiscale, un indirizzo e un telefono. Se si effettuasse la traduzione letterale in ODL_{I^3}di un frammento di codice di tale tipo, sarebbe necessaria la creazione di nuove *interface*, una per ogni livello di elemento innestato, nelle quali rappresentare i termini dichiarati nella specifica di contenuto. Pertanto la traduzione in ODL_{I^3}dell'esempio proposto diventerebbe la seguente:

```
Interface persona
{
  attribute primo_elemento primo_elemento;
  attribute string indirizzo;
  attribute string telefono ;
};

Interface primo_elemento
{
  attribute string nome;
  attribute string cognome;
};
union primo_elemento1
{
  attribute string codice_fiscale;
};
```

Per mantenere una corretta corrispondenza tra la struttura espressa nel linguaggio ODL$_{I^3}$e quella espressa nella DTD e non inserire nuovi elementi non presenti nel file originale, si propone di trasformare, in maniera automatica attraverso opportuni algoritmi, la dichiarazione dell'elemento in una analoga "normalizzata". Applicata all'esempio, la specifica di contenuto normalizzata é la seguente:

```
<!ELEMENT persona ((nome, cognome, indirizzo, telefono)|
                   (codice_fiscale, indirizzo, telefono))>
```

Sostanzialmente l'operazione di normalizzazione consiste, dopo avere associato al termine "," il valore di AND e al termine "I" il valore di OR, nell'ottenere la forma connettiva disgiuntiva dell'espressione. Tale rappresentazione trova una facile traduzione: nel caso considerato, l'esempio proposto diventa

```
Interface persona
{
  attribute string nome;
  attribute string cognome;
  attribute string indirizzo;
  attribute string telefono;
};
```

```
union persona1
{
  attribute string codice_fiscale;
  attribute string indirizzo;
  attribute string telefono;

};
```

Terminate le operazioni di pre-processing ora descritte, é possibile effettuare il parsing della DTD. Tale operazione, che consiste appunto nel trasformare la semantica del file in apposite strutture, é stata realizzata utilizzando le classi del package java javax.swing.text.html.parser. Tali classi, pur essendo state implementate per rappresentare le DTD di file HTML, possono essere utilizzate, operando delle opportune differenziazioni, anche in questo contesto. In particolare nell'ottica della rappresentazione ottenuta con l'uso di tale package, una DTD viene considerata come un insieme di istanze di classi di tipo Element, alle quali é possibile associare un ContentModel, ovvero la rappresentazione del contenuto dell'elemento stesso, e un AttList, ovvero l'elenco degli attributi associati a ogni elemento.

I principali problemi che si sono riscontrati nell'utilizzare tali classi, non progettate per rappresentare la DTD di un file XML, sono stati causati dall'inadeguatezza della struttura utilizzata per la rappresentazione del contenuto dell'elemento e dal fatto che ogni istanza di classe DTD ha proprietà preesistenti che possono entrare in conflitto con nuovi elementi presenti nella DTD. Per quello che riguarda il primo caso, occorre sottolineare che non sono stati progettati metodi né per navigare all'interno della rappresentazione delle specificazioni di contenuto, né per individuare il tipo di occorrenza di ogni singolo elemento facente parte del Content Model, né per evidenziare il livello di profondità al quale l'elemento di contenuto appartiene. Il secondo problema, ovvero la presenza di proprietà già definite all'interno di una istanza DTD, può causare una situazione critica nel caso in cui il nome di un elemento appartenente al DTD abbia lo stesso nome di una delle proprietà preesistenti del DTD. Questo perché gli elementi vengono mappati all'interno della classe rappresentante il DTD dentro un apposito Vector. Nel caso in cui però l'elemento da mappare abbia nome uguale a una delle proprietà del DTD, l'elemento non viene mappato nel Vector, ma nella proprietà. Questo comportamento fa si che nell'algoritmo per individuare le chiavi primarie e le chiavi esterne gli elementi mappati in quel modo vengano ignorati.

5.3.2 Determinazione delle chiavi

In questa fase vengono analizzati gli attributi e, nel caso in cui

1. siano di tipo ID o IDREF
2. abbiano dichiarazione di default di tipo *REQUIRED*
3. siano riferiti a degli attributi tradotti come *interface*

per ognuno di essi, attraverso una interfaccia grafica, viene chiesto al progettista se l'attributo stesso può considerarsi una chiave primaria (nel caso di ID) o esterna (nel caso di IDREF).

La scelta delle chiavi pone degli elementi di problematicità che é necessario approfondire. In primo luogo occorre evidenziare che, come si é indicato nell'elenco soprastante, non é sufficiente per un attributo soddisfare delle caratteristiche sintattiche per essere chiave candidata, ma occorre che sia riferito a un nodo posto a un preciso livello della struttura del file DTD. Tale particolarietà é un elemento di differenziazione fra i due linguaggi: infatti per quello che concerne la sintassi della DTD, qualsiasi elemento può avere un attributo con la qualifica di identificatore. Nel linguaggio ODL_{I^3}solamente le interface possono possedere chiavi. Di conseguenza, potranno essere chiavi unicamente gli attributi riferiti a elementi che saranno tradotti in ODL_{I^3}come *interface*, mentre quelli riferiti a elementi che saranno tradotti come *attribute* potranno unicamente essere tradotti come attributi obbligatori. Il problema della trattazione delle chiavi primarie e delle chiavi esterne é pertanto strettamente connesso con il problema della gestione della traduzione degli attributi al quale si rimanda.

La seconda problematica individuata verte sulla trattazione degli attributi IDREF/IDREFS. Quando un attributo é di tipo IDREF il suo valore rappresenta un identificatore di un elemento e quindi un elemento di tipo ID. Analogamente il valore di un attributo di tipo IDREFS rappresenta molteplici identificatori e quindi diverse istanze di elementi con attributo di tipo ID (istanze che possono essere istanze diverse dello stesso elemento oppure istanze di elementi diversi). Per quello che concerne gli attributi di tipo IDREF quindi sussiste la stessa problematica individuata per gli attributi di tipo ID, per cui non tutti gli attributi IDREF saranno interpretati dal wrapper come foreign key candidate. Inoltre non indicando la sintassi della DTD a quale identificatore l'attributo IDREF faccia rifermento non é possibile per il wrapper né controllare l'operare del progettista nella selezione della chiave primaria referenziata dalla chiave esterna né individuare possibili suggerimenti all'interno dell'elenco delle chiavi primarie. Per quello che concerne poi gli attributi di tipo IDREFS, é necessario osservare che non é possibile

individuare un costrutto ODL_{I^3}che permetta di esprimere la medesima semantica qualora si stia rappresentando in XML un riferimento a istanze di elementi diversi. In quel caso l'attributo dovrà essere tradotto come attributo di tipo stringa obbligatorio.

Infine, per aumentare le capacità espressive del documento ODL_{I^3}ottenuto dalla traduzione, si é pensato che, sempre attraverso l'interazione con il progettista, si potessero individuare degli elementi della DTD traducibili nel corrispettivo ODL_{I^3}come chiavi candidate. Pertanto, in analogia con la definizione delle chiavi primarie, si é pensato di potere ulteriormente qualificare gli attributi di tipo CDATA che abbiano il vincolo dell'obbligatorietà. Il progettista, mediante l'interfaccia grafica, dovrà autorizzare la qualificazione dei suddetti attributi in chiave nel caso in cui sia possibile verificare che le istanze rispettino i vincoli di unicità. É necessario sottolineare inoltre che questa fase di individuazione delle chiavi candidate, seppure critica, in quanto il progettista deve essere completamente certo che l'attributo in oggetto rispetti tutti i vincoli delle chiavi per poterlo qualificare, é quella maggiormente significativa dal punto di vista dell'integrazione dei sorgenti. Essendo infatti l'attributo ID un identificatore nel linguaggio XML, sia dal punto di vista intensionale, sia dal punto di vista estensionale, non sarà particolarmente significativo, in quanto dovrebbe sostanzialmente rappresentare il codice univoco identificativo di ogni singola istanza dell'elemento. La chiave candidata, selezionata tra gli attributi di tipo CDATA, i quali invece veicolano una informazione semantica, rappresenta in questo contesto una delle principali informazioni attraverso le quali determinare la relazioni iter-schema.

5.3.3 Traduzione del documento

In questa fase viene effettuata la traduzione del documento XML nel corrispettivo ODL_{I^3}secondo le regole di conversione descritte in appendice. Il traduttore implementato, per funzionare in maniera corretta, ha la necessità che il progettista espliciti l'elemento radice dell'albero strutturale (denominato *document element* nelle specifiche dell'XML). Per questo motivo nel caso in cui il file oggetto della traduzione sia una DTD, é necessario indicare in maniera esplicita l'elemento di root. Nel caso in cui si indichi come oggetto del wrapper un file XML (che come si é detto deve essere necessariamente *valid*), la sintassi XML consente di ricavare in maniera automatica il file DTD e il document element associato.

L'algoritmo di traduzione necessita per operare di due strutture dati. La prima é la sintesi dell'interazione con il progettista e consiste nell'insieme delle chiavi primarie e esterne che sono state individuate. La secon-

da struttura consiste semplicemente in una lista di elementi, denominata *vectorElement*, utilizzata per scandire gli elementi da tradurre. Tale lista, se si escludono casi particolari che verranno analizzati in seguito e che comunque vengono trattati dal software, per costruzione include i soli elementi che assumono in ODL_{I^3}valore di interface. In fase di inizializzazione l'unico elemento contenuto in vectorElement é il document element.

Il software funziona attraverso la seguente procedura:

1. **Lettura di un elemento di vectorElement.** Viene letto il valore di vectorElement corrispondente a uno specifico puntatore. In fase di inizializzazione il puntatore fa riferimento al document element, unico valore contenuto nella struttura. Se la lista é terminata l'algoritmo termina.

2. **Controllo del tipo dell'elemento selezionato.** In particolare viene verificato se l'elemento analizzato é di tipo MODEL. Un elemento viene detto di tipo model se conforme alle specifiche di un elemento children o di un elemento mixed. Ad esempio:

   ```
   <!ELEMENT persona (identificatore, indirizzo, telefono)>
   <!ELEMENT identificatore (#PCDATA|cognome|Codice_fiscale)>
   ```

   essendo elementi rispettivamente di tipo children e di tipo mixed sono entrambi di tipo MODEL. Se l'elemento in analisi é del tipo indicato, come avviene solitamente al di fuori di casi particolari trattatati successivamente, l'algoritmo procede al punto successivo. In caso contrario l'elemento viene tradotto seguendo le regole di conversione indicate in appendice e l'algoritmo termina.

3. **Creazione di una interface ODL_{I^3}.** Se l'elemento é di tipo model, viene creata una nuova *interface* nel documento ODL_{I^3}.

4. **Individuazione delle chiavi.** Attraverso la lettura della struttura dati contenente le chiavi primarie e esterne selezionate dal progettista é possibile verificare se l'elemento di cui si sta scrivendo la traduzione possiede delle chiavi e in caso affermativo é necessario effettuare la loro traduzione.

5. **Analisi delle specifiche di contenuto.** Per ogni elemento appartenente alla specifica di contenuto dell'elemento in esame occorre:

   (a) Individuare il tipo (MODEL, PCDATA, EMPTY e ANY) e la quantificazione delle occorrenze (*,+,?, nessuna quantificazione).

(b) Nel caso in cui l'elemento sia di tipo MODEL, e nel caso non sia già presente in vectorElement, occorre aggiungerlo in coda.

(c) Individuare gli eventuali attributi associati.

(d) Tradurre, sulla base delle caratteristiche esaminate e in maniera conforme alle regole poste in appendice, l'elemento in analisi e gli attributi associati

6. **Traduzione degli attributi dell'interface.** Devono essere tradotti gli attributi riferiti all'elemento tradotto come interface.

7. **Individuazione dell'interface successiva.** Occorre incrementare di una unità il valore del puntatore che scandisce vectorElement

5.4 Casi particolari

É necessario mettere in evidenza alcuni particolari elementi di criticità che vengono trattati dal traduttore. Il primo é relativo alla gestione degli attributi, che, come si é detto, non essendo presenti nel modello dati supportato da ODL_{I^3}, prima di essere tradotti devono essere opportunamente interpretati. Si sono pertanto individuati due diversi scenari. Nel primo caso si può ipotizzare di "elevare" al rango di elemento ogni attributo, introducendo un opportuno cambiamento nel nome con il quale l'attributo viene tradotto in ODL_{I^3}in modo da rendere evidente questa operazione, anche senza un meccanismo di tabelle di memorizzazione delle conversioni effettuate. Tale scelta determina sostanzialmente due conseguenze: nel caso in cui l'attributo si riferisca

1. a un elemento tradotto come interface, nella rappresentazione attraverso l'albero strutturale introdotto dal modello OEM si ha la nascita di un nuovo nodo posto al di sotto dell'elemento che si sta analizzando. Ad esempio:

```
<!ELEMENT persona (nome, cognome)>
<!ATTLIST persona nazionalita CDATA #REQUIRED>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT cognome (#PCDATA)>
```

viene tradotto in ODL_{I^3}come:

```
Interface persona
```

```
{
  attribute string persona_nazionalita;
  attribute string nome;
  attribute string cognome;
}
```

2. a un generico elemento facente parte della specifica di contenuto di un altro elemento (e quindi tradotto nel corpo di una interface), l'attributo non può essere tradotto come un nuovo nodo posto al di sotto dell'elemento in analisi, in quanto l'elemento non rappresenta una interface, e quindi non può avere nodi sottostanti, ma viene deve essere "elevato di livello", e tradotto allo stesso livello dell'elemento al quale si riferisce. In quest'ottica quindi non devono essere tradotti gli attributi di elementi di tipo MODEL in quanto verranno tradotti nel momento in cui verrà creata la rispettiva interface. Ad esempio:

   ```
   <!ELEMENT persona (nome, cognome)>
   <!ELEMENT nome (#PCDATA)>
   <!ELEMENT cognome (#PCDATA)>
   <!ATTLIST cognome nazionalita CDATA #REQUIRED>
   ```

   viene tradotto in ODL_{I^3}come:

   ```
   Interface persona
   {
     attribute string nome;
     attribute string cognome;
     attribute string cognome_nazionalita;
   };
   ```

Tale meccanismo di traduzione degli attributi, che può sembrare poco funzionale, é l'unico sistema che si é individuato con la caratteristica di conservare il più possibile la struttura del file iniziale.

Nonostante questo la scelta che si é implementata é stata differente. Infatti si é evidenziato nel Capitolo 3, che XML-Schema, linguaggio che consente una gestione più articolata delle strutture, ma non ancora reso standard dal W3C, non utilizza una analoga convenzione, ma considera gli elementi dotati di attributo "di tipo complesso" e quindi li rappresenta come nodi non terminali (traducibili pertanto come interface nel modello ODL_{I^3}). Adottando una analoga convenzione, ogni elemento dotato di attributo viene tradotto come interface e pertanto l'esempio soprastante genera il seguente codice ODL_{I^3}:

```
Interface persona
{
  attribute string nome;
  attribute cognome cognome;
};

Interface cognome
{
  attribute string cognome_nazionalita;
  attribute string PCDATA_NODE;
};
```

Come si evince dall'esempio, adottando una soluzione di questo tipo, é necessario inserire un nodo fittizio, PCDATA_NODE, nel quale memorizzare l'informazione contenuta nel campo PCDATA dell'elemento che si sta traducendo. Nonostante questo, si é pensato di implementare tale codifica degli attributi per rendere più agevole la trattazione delle chiavi (sia quelle esterne sia quelle candidate), in quanto in questo modo, pur snaturando maggiormente il sorgente della DTD, si trattano in maniera omogenea tutti gli attributi, senza considerare la posizione degli stessi nel file ODL_{I^3}tradotto. Inoltre, ritenendo che XML-Schema possa essere a breve tempo considerato uno standard dal W3C, implementando una soluzione adottata in quel contesto, si é pensato di generare in questo modo una soluzione maggiormente standard.

Un secondo elemento di criticità é causato dalla gestione del document element. Si può ipotizzare che normalmente quel particolare elemento sia di tipo MODEL, in quanto da esso deve partire l'intera struttura. Il caso in cui non lo sia rappresenta l'unica occorrenza nella quale vectorElement non contiene elementi di tipo MODEL. Il wrapper é comunque in grado di gestire tale caso, e quindi consente strutture ad albero molto semplici costituite da un solo nodo.

In terzo luogo é necessario evidenziare come il traduttore gestisca gli elementi XML di tipo mixed. In questo caso si é preferito infatti non tradurre gli elementi in maniera tale da preservare la rappresentazione strutturale originale del file, ma per problemi legati alla gestione da parte del software di tali strutture, si é resa necessaria l'introduzione di un nuovo nodo, denominato PCDATA_NODE. Il valore associato al nodo in questione é il valore terminale che l'elemento mixed può assumere. Volendo esemplificare la situazione si può considerare il seguente frammento di codice:

```
<!ELEMENT persona (#PCDATA| cognome)>
```

```
<!ATTLIST persona  nazionalita CDATA #REQUIRED>
<!ELEMENT cognome (#PCDATA)>
```

Tale codice come si é detto potrebbe essere tradotto in maniera letterale in ODL_{I^3}nel seguente modo:

```
Interface persona
{
 attribute string persona_nazionalita;
 string;
};
union persona1
{
  attribute string persona_nazionalita;
  attribute string cognome;
};
```

La traduzione che, per motivi legati alla gestione da parte del software della struttura, si é preferito adottare comporta la scrittura di un frammento di codice di questo tipo:

```
Interface persona
{
 attribute string persona_nazionalita;
 attribute string PCDATA_NODE;
};
union persona1
{
  attribute string persona_nazionalita;
  attribute string cognome;
};
```

Risulta evidente che la traduzione adottata genera un ulteriore nodo nell'albero rappresentativo della base di dati in oggetto, rendendo omegenea la profondità della struttura di ogni ramo dell'union.

Infine, una ulteriore situazione critica può essere causata dal fatto che il parser e di conseguenza anche il wrapper non effettua alcun controllo sulla correttezza della DTD. Per questo motivo talvolta il parser può non percepire la non correttezza del file DTD e tradurre qualcosa di non consistente. Nonostante questo si é pref erito non effettuare controlli, demandando tale compito a software più specifici di validazione di file XML e presupponendo che il parser operi unicamente con file valid.

5.5 L'interfaccia interattiva

E' stato necessario realizzare una interfaccia grafica attraverso la quale interagire con il progettista. In particolare attraverso questo strumento si ottiene il soddisfacimento di due funzionalità: in primo luogo é necessario mettere a disposizione uno strumento attraverso il quale si possa effettuare la scelta degli elementi da promuovere a primary, candidate e foreign key. É possibile in questo modo aggiungere alla semantica espressa dalla DTD parte della semantica che é possibile esprimere in ODL_{I^3}. Inoltre occorre recepire dal progettista le informazioni attraverso le quali il wrapper può effettuare la connessione al Global Schema. In particolare é necessario inserire un nome che identifichi la sorgente, e fornire l'indirizzo e la porta alla quale effettuare la connessione. Una volta effettuate le operazioni di configurazione é possibile effettuare il collegamento al sistema e l'interfaccia viene distrutta in quanto non più utilizzata nelle fasi successive. É necessario sottolineare che lo stato della configurazione implementata mediante l'interfaccia viene salvato su file al momento della connessione. In questo modo, in caso di mancato collegamento é possibile mantenere traccia delle scelte effettuate nel passaggio precedente.

5.6 Conclusioni

Il wrapper é stato implementato utilizzando il linguaggio java, pertanto non necessita di alcuna ricompilazione nel passaggio tra diverse piattaforme. In particolare é necessario sottolineare nuovamente che essendosi utilizzate le classi java progettate per descrivere DTD di file HTML la rappresentazione interna della DTD non sempre é ottimale e non sempre corrisponde alle esigenze. Al momento, essendo standard unicamente le specifiche DOM di primo livello, non é stato possibile disporre di strumenti più efficaci e neppure si é pensato di progettare una ulteriore nuova struttura che modelli un DTD perché sarebbe stata comunque non standard.

Il wrapper inoltre non tratta file XML well-formed. Nel caso in cui però questo si renda necessario, occorre interfacciare il wrapper con un software che costruisca in maniera automatica la DTD di un file XML. Esistendo al momento software di pubblico dominio che realizzano tale scopo, si é preferito demandare all'utilizzatore la creazione del DTD a partire da un file XML, piuttosto che creare quella funzionalità in modo automatico all'interno del wrapper.

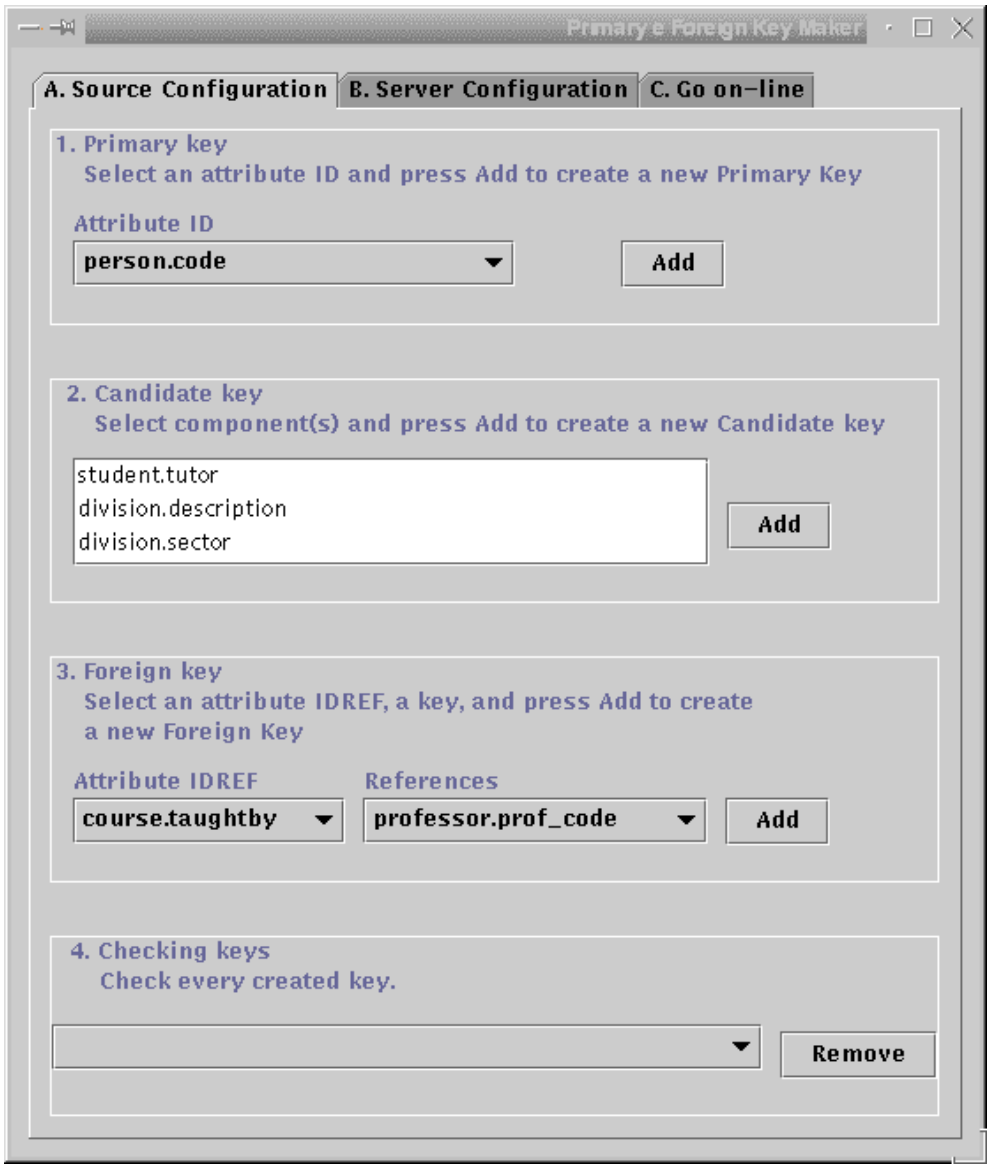

Figura 5.1: Determinazione delle chiavi

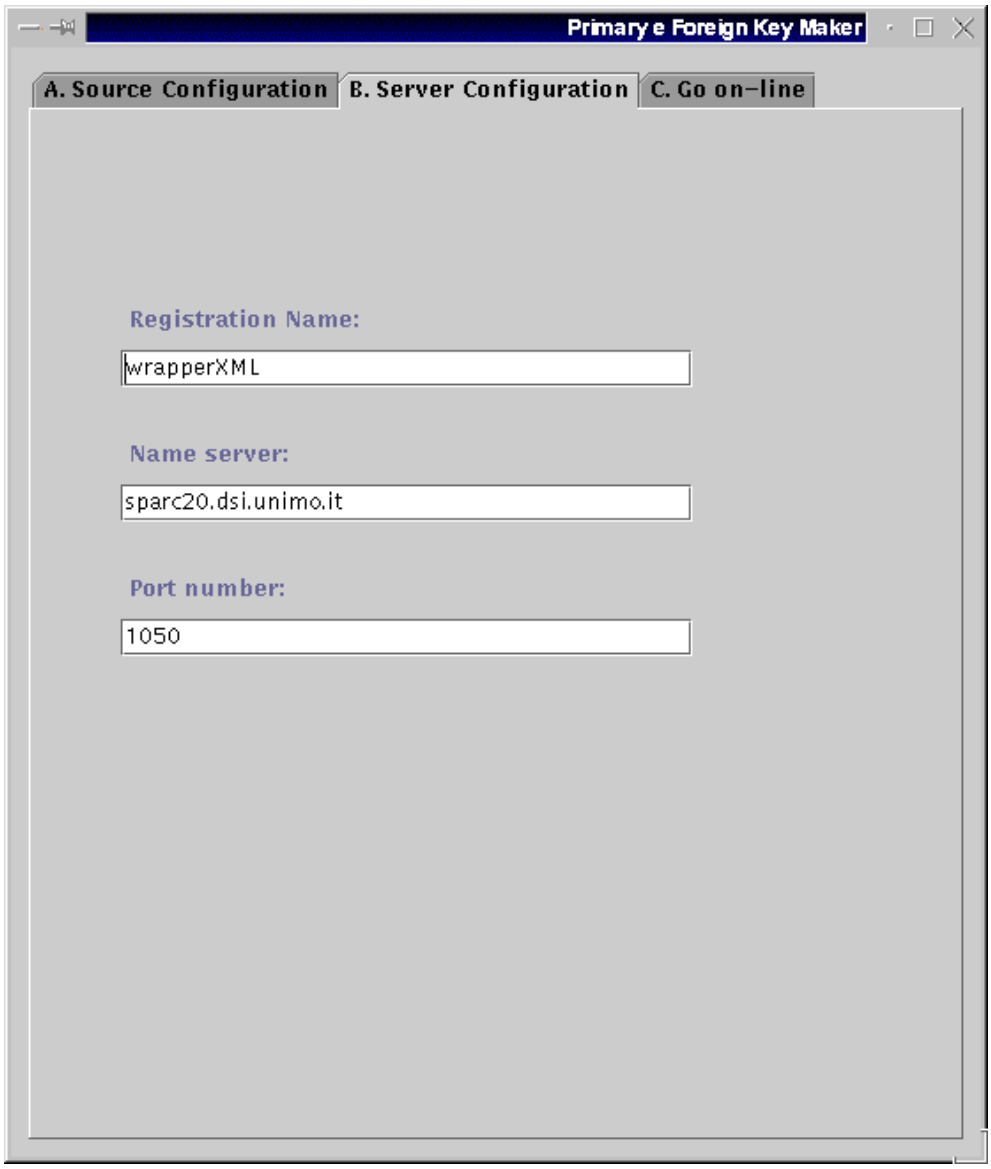

Figura 5.2: Configurazione della connessione

Conclusioni

Il progetto MOMIS si colloca nell'ambito della ricerca sull'integrazione intelligente di informazioni. Lo scopo del progetto é quello di sviluppare uno strumento di estrazione di informazione da sorgenti di dati eterogenee e distribuite.
Si é sottolineato nei capitoli precedenti come il progetto MOMIS si distingua da altri progetti aventi lo stesso obiettivo per l'approccio seguito e per l'introduzione, nell'integrazione delle sorgenti, di comportamenti intelligenti. Tale approccio, che possiamo definire semantico-virtuale, si basa sull'analisi degli schemi forniti dalle sorgenti locali. A partire da quella analisi si costruisce uno schema integrato in modo tale da proporre all'utente le informazioni locali in modo omogeneo e attraverso l'uso di un unico linguaggio descrittivo. La rappresentazione ottenuta costituisce una cosiddetta *vista virtuale*, che non contiene alcun dato, ma che rappresenta la base dell'operazione di integrazione dei dati appartenenti alle singole sorgenti. Il risultato di questa fase consiste, quindi, nella produzione di un insieme di conoscenze intensionali ed estensionali; le prime permettono la risoluzione dei conflitti tra schemi, mentre le seconde indicano il modo in cui le entità del dominio applicativo sono rappresentate sull'insieme di sorgenti.
La principale caratteristica di MOMIS é rappresentata dalla realizzazione della fase di integrazione che avviene mediante un processo semi-automatico. Tale processo, che vede coinvolti il sistema stesso ed il progettista, consente non solo l'unificazione degli schemi delle sorgenti, ma anche la creazione di uno schema globale vero e proprio, direttamente interrogabile dall'utente.

Un aspetto sicuramente innovativo, è poi costituito dalla presenza di componenti intelligenti che portano ad una maggiore efficacia del sistema sia nella fase di integrazione, sia in quella di gestione delle interrogazioni. Nel primo caso, infatti, questi elementi possono essere impiegati per verificare le indicazioni fornite dal progettista e per estenderle mediante nuove relazioni inferite in modo automatico, mentre, nel secondo, può essere sfrut-

tata l'espansione semantica e la sussunzione per riformulare le interrogazioni, generandone altre equivalenti ma più efficienti.
Per garantire una precisa interazione con il progettista é stata posta una notevole cura, in fase di implementazione del software, nella creazione delle interfaccie attraverso le quali avvengono le diverse fasi di acquisizione di informazioni e di implementazione di nuovi vincoli semantici in maniera "manuale".

In questa tesi é stato analizzato il comportamento di un wrapper per sorgenti di dati XML ed é stata realizzata la componente del wrapper che realizza la traduzione dal linguaggio specifico della sorgente in ODL_{I^3}. Sono state evidenziate le difficoltà che tale traduzione comporta, soprattutto nell'ambito della modellazione di sorgenti semistrutturate. Inoltre, durante la fase di traduzione, per sopperire a delle carenze rappresentative del linguaggio XML rispetto alle potenzialità di ODL_{I^3}, il progettista può aggiungere nuove informazioni allo schema. Attraverso una specifica interfaccia é possibile infatti inserire ad esempio l'elenco delle chiavi primarie, delle chiavi esterne e delle chiavi candidate, informazioni che non é possibile inserire nella descrizione XML. In questo modo si ottiene una descrizione della sorgente che non si discosta molto da quella originaria ma che racchiude le informazioni necessarie per l'integrazione.
Una descrizione di tale tipo, molto simile a quella della sorgente di partenza, comporta in primo luogo che, la fase di integrazione e di estrazione delle relazioni semantiche attuate dal mediatore con l'interazione del progettista, avvenga su uno schema integrato che sia veramente la sintesi delle sorgenti. Inoltre agendo in questo modo si agevola la fase di interrogazione della sorgente, permettendo la scrittura di query che possono essere facilmente tradotte nel query language specifico.

É necessario però sottolineare che, nel momento in cui verrà realizzato il modulo che gestisce la fase di interrogazione, si potrà rendere necessario effettuare dei cambiamenti anche sul modulo di traduzione. Questo perché potrebbe essere utile, a fronte di rigidità del query language, modificare parte della rappresentazione della sorgente, utilizzando delle strutture ODL_{I^3}che esprimano una analoga semantica ma che abbiano una struttura differente. Tale evenienza e soprattutto il livello di trasformazione del software realizzato, non essendo ancora stato sviluppato lo standard di query language, non é al momento preventivabile.

Appendice A

Glossario I^3

Questo glossario ed il vocabolario sul quale si basa sono stati originariamente sviluppati durante l'I^3 Architecture Meeting in Boulder CO, 1994, sponsorizzato dall'ARPA, e rifiniti in un secondo incontro presso l'Università di Stanford, nel 1995. Il glossario è strutturato logicamente in diverse sezioni:

- Sezione 1: Architettura
- Sezione 2: Servizi
- Sezione 3: Risorse
- Sezione 4: Ontologie

Nota: poiché la versione originaria del glossario usa una terminologia inglese, in alcuni casi è riportato, a fianco del termine, il corrispettivo inglese, quando la traduzione dal termine originale all'italiano poteva essere ambigua o poco efficace.

A.1 Architettura

- Architettura = insieme di componenti.
- architettura di riferimento = linea guida ed insieme di regole da seguire per l'architettura.
- componente = uno dei blocchi sui quali si basa una applicazione o una configurazione. Incorpora strumenti e conoscenza specifica del dominio.
- applicazione = configurazione persistente o transitoria dei componenti, rivolta a risolvere un problema del cliente, e che può coprire diversi domini.

- configurazione = istanza particolare di una architettura per una applicazione o un cliente.
- collante (glue) = software o regole che servono per per collegare i componenti o per interoperare attraverso i domini.
- strato = grossolana categorizzazione dei componenti e degli strumenti in una configurazione. L'architettura I^3 distingue tre strati, ognuno dei quali fornisce una diversa categoria di servizi:
  1. Servizi di Coordinamento = coprono le fasi di scoperta delle risorse, distribuzione delle risorse, invocazione, scheduling ...
  2. Servizi di Mediazione = coprono la fase di query processing e di trattamento dei risultati, nonché il filtraggio dei dati, la generazione di nuove informazioni, etc.
  3. Servizi di Wrapping = servono per l'utilizzo dei wrappers e degli altri strumenti simili utilizzati per adattarsi a standards di accesso ai dati e alle convenzioni adoperate per la mediazione e per il coordinamento.
- agente = strumento che realizza un servizio, sia per il suo proprietario, sia per un cliente del suo proprietario.
- facilitatore = componente che fornisce i servizi di coordinamento, come pure l'instradamento delle interrogazioni del cliente.
- mediatore = componente che fornisce i servizi di mediazione e che provvede a dare valore aggiunto alle informazioni che sono trasmesse al cliente in risposta ad una interrogazione.
- cliente (customer) = proprietario dell'applicazione che gestisce le interrogazioni, o utente finale, che usufruisce dei servizi.
- risorsa = base di dati accessibile, server ad oggetti, base di conoscenze ...
- contenuto = risultato informativo ricavato da una sorgente.
- servizio = funzione fornita da uno strumento in un componente e diretta ad un cliente, direttamente od indirettamente.
- strumento (tool) = programma software che realizza un servizio, tipicamente indipendentemente dal dominio.

- wrapper = strumento utilizzato per accedere alle risorse conosciute, e per tradurre i suoi oggetti.
- regole limitative (constraint rules) = definizione di regole per l'assegnamento di componenti o di protocolli a determinati strati.
- interoperare = combinare sorgenti e domini multipli.
- informazione = dato utile ad un cliente.
- informazione azionabile = informazione che forza il cliente ad iniziare un evento.
- dato = registrazione di un fatto.
- testo = dato, informazione o conoscenza in un formato relativamente non strutturato, basato sui caratteri.
- conoscenza = metadata, relazione tra termini, paradigmi ... , utili per trasformare i dati in informazioni.
- dominio = area, argomento, caratterizzato da una semantica interna, per esempio la finanza, o i componenti elettronici ...
- metadata = informazione descrittiva relativa ai dati di una risorsa, compresi il dominio, proprietà, le restrizioni, il modello di dati, ...
- metaconoscenza = informazione descrittiva relativa alla conoscenza in una risorsa, includendo l'ontologia, la rappresentazione ...
- metainformazioni = informazione descritiiva sui servizi, sulle capacità, sui costi ...

A.2 Servizi

- Servizio = funzionalità fornita da uno o più componenti, diretta ad un cliente.
- instradamento (routing) = servizio di coordinamento per localizzare ed invocare una risorsa o un servizio di mediazione, o per creare una configurazione. Fa uso di un direttorio.
- scheduling = servizio di coordinamento per determinare l'ordine di invocazione degli accessi e di altri servizi; fa spesso uso dei costi stimati.

- accoppiamento (matchmaking) = servizio che accoppia i sottoscrittori di un servizio ai fornitori.
- intermediazione (brokering) = servizio di coordinamento per localizzare le risorse migliori.
- strumento di configurazione = programma usato nel coordinamento per aiutare a selezionare ed organizzare i componenti in una istanza particolare di una configurazione architetturale.
- servizi di descrizione = metaservizi che informano i clienti sui servizi, risorse ...
- direttorio = servizio per localizzare e contattare le risorse disponibili, come le pagine gialle, pagine bianche ...
- decomposizione dell'interrogazione (query decomposition) = determina le interrogazioni da spedire alle risorse o ai servizi disponibili.
- riformulazione dell'interrogazione (query reformulation) = programma per ottimizzare o rilassare le interrogazioni, tipicamente fa uso dello scheduling.
- contenuto = risultato prodotto da una risorsa in risposta ad interrogazioni.
- trattamento del contenuto (content processing) = servizio di mediazione che manipola i risultati ottenuti, tipicamente per incrementare il valore delle informazioni.
- trattamento del testo = servizio di mediazione che opera sul testo per ricerca, correzione ...
- filtraggio = servizio di mediazione per aumentare la pertinenza delle informazioni ricevute in risposta ad interrogazioni.
- classificazione (ranking) = servizio di mediazione per assegnare dei valori agli oggetti ritrovati.
- spiegazione = servizio di mediazione per presentare i modelli ai clienti.
- amministrazione del modello = servizio di mediazione per permettere al cliente ed al proprietario del mediatore di aggiornare il modello.
- integrazione = servizio di mediazione che combina i contenuti ricevuti da una molteplicità di risorse, spesso eteorgenee.

- accoppiamento temporale = servizio di mediazione per riconoscere e risolvere differenze nelle unità di misura temporali utilizzate dalle risorse.
- accoppiamento spaziale = servizio di mediazione per riconoscere e risolvere differenze nelle unità di misura spaziali utilizzate dalle risorse.
- ragionamento (reasoning) = metodologia usata da alcuni componenti o servizi per realizzare inferenze logiche.
- browsing = servizio per permettere al cliente di spostarsi attraverso le risorse.
- scoperta delle risorse = servizio che ricerca le risorse.
- indicizzazione = creazione di una lista di oggetti (indice) per aumentare la velocità dei servizi di accesso.
- analisi del contenuto = trattamento degli oggetti testuali per creare informazioni.
- accesso = collegamento agli oggetti nelle risorse per realizzare interrogazioni, analisi o aggiornamenti.
- ottimizzazione = processo di manipolazione o di riorganizzazione delle interrogazioni per ridurne il costo o il tempo di risposta.
- rilassamento = servizio che fornisce un insieme di risposta maggiore rispetto a quello che l'interrogazione voleva selezionare.
- astrazione = servizio per ridurre le dimensioni del contenuto portandolo ad un livello superiore.
- pubblicità (advertising) = presentazione del modello di una risorsa o del mediatore ad un componente o ad un cliente.
- sottoscrizione = richiesta di un componente o di un cliente di essere informato su un evento.
- controllo (monitoring) = osservazione delle risorse o dei dati virtuali e creazione di impulsi da azionare ogniqualvolta avvenga un cambiamento di stato.
- aggiornamento = trasmissione dei cambiamenti dei dati alle risorse.
- instanziazione del mediatore = popolamento di uno strumento indipendente dal dominio con conoscenze dipendenti da un dominio.

- attivo (activeness) = abilità di un impulso di reagire ad un evento.
- servizio di transazione = servizio che assicura la consistenza temporale dei contenuti, realizzato attraverso l'amministrazione delle transzioni.
- accertamento dell'impatto = servizio che riporta quali risorse saranno interessate dalle interrogazioni o dagli aggiornamenti.
- stimatore = servizo di basso livello che stima i costi previsti e le prestazioni basandosi su un modello, o su statistiche.
- caching = mantenere le informazioni memorizzate in un livello intermedio per migliorare le prestazioni.
- traduzione = trasformazione dei dati nella forma e nella sintassi richiesta dal ricevente.
- controllo della concorrenza = assicurazione del sincronismo degli aggiornamenti delle risorse, tipicamente assegnato al sistema che amministra le transazioni.

A.3 Risorse

- Risorsa = base di dati accessibile, simulazione, base di conoscenza, ... comprese le risorse "legacy".
- risorse "legacy" = risorse preesistenti o autonome, non disegnate per interoperare con una architettura generale e flessibile.
- evento = ragione per il cambiamento di stato all'interno di un componente o di una risorsa.
- oggetto = istanza particolare appartenente ad una risorsa, al modello del cliente, o ad un certo strumento.
- valore = contenuto metrico presente nel modello del cliente, come qualità, rilevanza, costo.
- proprietario = individuo o organizzazione che ha creato, o ha i diritti di un oggetto, e lo può sfruttare.
- proprietario di un servizio = individuo o organizzzione responsabile di un servizio.

- database = risorsa che comprende un insieme di dati con uno schema descrittivo.
- warehouse = database che contiene o dà accesso a dati selezionati, astratti e integrati da una molteplicità di sorgenti. Tipicamente ridondante rispetto alle sorgenti di dati.
- base di conoscenza = risorsa comprendente un insieme di conoscenze trattabili in modo automatico, spesso nella forma di regole e di metadata; permettono l'accesso alle risorse.
- simulazione = risorsa in grado di fare proiezioni future sui dati e generare nuove informazioni, basata su un modello.
- amministrazione della transazione = assicurare che la consistenza temporale del database non sia compromessa dagli aggiornamenti.
- impatto della transazione = riporta le risorse che sono state coinvolte in un aggiornamento.
- schema = lista dedlle relazioni, degli attributi e, quando possibile, degli oggetti, delle regole, e dei metadata di un database. Costituisce la base dell'ontologia della risorsa.
- dizionario = lista dei termini, fa parte dell'ontologia.
- modello del database = descrizione formalizzata della risorsa database, che include lo schema.
- interoperabilità = capacità di interoperare.
- eterogeneità = incompatibilità trovate tra risorse e servizi sviluppati autonamamente, che vanno dalla paittaforma utilizzata, sistema operativo, modello dei dati, alla semantica, ontologia, ...
- costo = prezzo per fornire un servizio o un accesso ad un oggetto.
- database deduttivo = database in grado di utilizzare regole logiche per trattare i dati.
- regola = affermazione logica, unità della conoscenza trattabile in modo automatico.
- sistema di amministrazione delle regole = software indipendente dal dominio che raccoglie, seleziona ed agisce sulle regole.

- database attivo = database in grado di reagire a determinati eventi.
- dato virtuale = dato rappresentato attraverso referenze e procedure.
- stato = istanza o versione di una base di dati o informazioni.
- cambiamento di stato = stato successivo ad una azione di aggiornamento, inserimento o cancellazione.
- vista = sottoinsieme di un database, sottoposto a limiti, e ristrutturato.
- server di oggetti = fornisce dati oggetto.
- gerarchia = struttura di un modello che assegna ogni oggetto ad un livello, e definisce per ogni oggetto l'oggetto da cui deriva.
- network = struttura di un modello che fa uso di relazioni relativamente libere tra oggetti.
- ristrutturare = dare una struttura diversa ai dati seguendo un modello differente dall'originale.
- livello = categorizzazione concettuale , dove gli oggetti di un livello inferiore dipendono da un antenato di livello superiore.
- antenato (ancestor) = oggetto di livello superiore, dal quale derivano attributi ereditabili.
- oggetto root = oggetto da cui tutti gli altri derivano, all'interno di una gerarchia.
- datawarehouse = deposito di dati integrati provenienti da una molteplicità di risorse.
- deposito di metadata = database che contiene metadata o metainformazioni.

A.4 Ontologia

- Ontologia = descrizione particolareggiata di una concettualizzazione, i.e. l'insieme dei termini e delle relazioni usate in un dominio, per indicare oggetti e concetti, spesso ambigui tra domini diversi.
- concetto = definisce una astrazione o una aggregazione di oggetti per il cliente.

- semantico = che si riferisce al significato di un termine, espresso come un insieme di relazioni.
- sintattico = che si riferisce al formato di un termine, espresso come un insieme di limitazioni.
- classe = definisce metaconoscenze come metodi, attributi, ereditarietà, per gli oggetti in essa istanziati.
- relazione = collegamento tra termini, come *is-a*, *part-of*, ...
- ontologia unita (merged) = ontologia creata combinando diverse ontologie, ottenuta mettendole in relazione tra loro (mapping).
- ontologia condivisa = sottoinsieme di diverse ontologie condiviso da una molteplicità di utenti.
- comparatore di ontologie = strumento per determinare relazioni tra ontologie, utilizzato per determinare le regole necessarie per la loro integrazione.
- mapping tra ontologie = trasformazione dei termini tra le ontologie, attraverso regole di accoppiamento, utilizzato per collegare utenti e risorse.
- regole di accoppiamento (matching rules) = dichiarazioni per definire l'equivalenza tra termini di domini diversi.
- trasformazione dello schema = adattamento dello schema ad un'altra ontologia.
- editing = trattamneto di un testo per assicurarne la conformità ad una ontologia.
- algebra dell'ontologia = insieme delle operazioni per definire relazioni tra ontologie.
- consistenza temporale = è raggiunta se tutti i dati si riferiscono alla stessa istanza temporale ed utilizzano la stessa granularità temporale.
- specifico ad un dominio = relativo ad un singolo dominio, presuppone l'assenza di incompatibilità semantiche.
- indipendente dal dominio = software, strumento o conoscenza globale applicabile ad una molteplicità di domini.

Appendice B

Regole di traduzione da XML a ODL_{I^3}

Sono qui elencate le regole di traduzione nel passaggio dalla sintassi espressa nel DTD di un file XML alla sintassi utilizzata in ODL_{I^3}. Occorre ricordare che non puó essere effettuata una traduzione meccanica del singolo elemento, ma occorre valutare la posizione nell'albero strutturale all'interno della quale l'elemento é collocato e quindi distinguere, a livello di traduzione, i nodi terminali dai nodi intermedi.

Traduzione di elementi

Le specifiche del linguaggio XML1.0 individuano 4 tipologie di nodi: nodi di tipo children, mixed, any, empty. In questa sede si preferisce suddividere la categoria mixed in due sezioni (l'una includente unicamente gli elementi di tipo PCDATA e l'altra le restanti tipologie che identificano il tipo mixed). In questo modo, in analogia con il linguaggio XML-Schema, é possibile suddividere gli elementi in due blocchi: il primo includente gli elementi **semplici**, cioé quelli che rappresentano i nodi terminali della struttura ad albero che raccoglie le tipologie pcdata e empty. Il secondo blocco identifica gli elementi **complessi**, cioé quelli che mappano nodi intermedi, oppure nodi terminali con attributi, e che sono costituiti dagli elementi definiti come mixed e children. Nel caso in cui un nodo possieda degli attributi deve essere considerato come complesso indipendentemente dalla sua posizione nell'albero rappresentativo del dtd. Gli elementi di tipo any possono rappresentare nodi facenti parte di entrambe le categorie.

- Elementi di tipo semplice

DTD	ODL$_{I^3}$

`<!ELEMENT element1 (#PCDATA)>`
`<!ELEMENT element2 EMPTY>`
`<!ELEMENT element3 ANY>`
	`attribute string element1;`
	`const string element2="" ;`
	`attribute string element3;`

- Elementi complessi:

  - Elementi di tipo children con elementi figlio di tipo semplice

DTD	ODL$_{I^3}$

`<!ELEMENT elem1 (cspec1,cspec2)>`
`<!ELEMENT cspec1 (#PCDATA)>`
`<!ELEMENT cspec2 (#PCDATA)>`
	`Interface elem1`
	`( ... )`
	`{`
	`  attribute string cspec1;`
	`  attribute string cspec2;`
	`};`

  - Elementi di tipo mixed con elementi figlio di tipo semplice

DTD	ODL$_{I^3}$

`<!ELEMENT elem1 (PCDATA|cspec1)>`
`<!ELEMENT cspec1 (#PCDATA)>`
	`Interface elem1`
	`( ... )`
	`{`
	`attribute string PCDATA_NODE;`
	`};`
	`union elem 11`
	`{`
	`attribute string cspec1;`
	`};`

  - Elementi di tipo children con elementi figlio di tipo complesso

DTD	ODL$_{I^3}$

`<!ELEMENT elem1 (cspec1)>`
`<!ELEMENT csp1 (child1,child2)>`
	`Interface elem1`
	`( ... )`
	`{`
	`attribute csp1 csp1;`
	`};`

Si ha una traduzione degli elementi figlio di tipo children (o mixed) analoga a quella ora proposta nel caso in cui questi siano contenuti in elementi di tipo mixed.

Gestione delle occorrenze

I termini che sovrintendono al controllo delle occorrenze sono di tre tipi: il termine * (che indica la presenza di 0 - n occorrenze dell'elemento), il termine + (da 1 a n occorrenze), il termine ? (elemento opzionale). L'assenza di termine di specificazione serve a denotare la presenza di una sola istanza dell'elemento.
Per quello che concerne la traduzione di questi ulteriori caratteri di specificazione degli elementi in ODL_{I^3}occorre considerare due aspetti: il primo é il fatto che l'elemento + non trova traduzione in ODL_{I^3}, e quindi il comportamento di elementi contrassegnati con il termine "+" viene considerato analogo a quello degli elementi contrassegnati con il termine "*". Il secondo aspetto é rappresentato dal fatto che il traduttore analizza unicamente il tipo di occorrenza degli elementi contenuti nella specifica di contenuto di un elemento di tipo children o di tipo mixed. Non viene pertanto analizzato il tipo di occorrenza del nodo intermedio nel suo complesso, ma quello di ogni specifico nodo sottostante al nodo considerato. Questo perché quello che si vuole modellare in ODL_{I^3}é un *pattern* della struttura dati rappresentata dal file in esame.

DTD	ODL_{I^3}

`<!ELEMENT elem1 (cs1*,cs2+,cs3?,cs4)>`
`<!ELEMENT cs1 (#PCDATA)>`
`<!ELEMENT cs2 (#PCDATA)>`
`<!ELEMENT cs3 (#PCDATA)>`
`<!ELEMENT cs4 (#PCDATA)>`
	`Interface elem1`
	`( ... )`
	`{`
	`attribute set<string> cs1;`
	`attribute set<string> cs2;`
	`attribute string cs3?;`
	`attribute string cs4;`
	`};`

Traduzione di attributi

Le differenti modalitá di traduzione degli attributi sono determinate principalmente da due aspetti. Il primo é relativo al tipo dell'attributo che si vuole tradurre (CDATA, ID, IDREF,...), mentre il secondo aspetto verte sulla cosiddetta dichiarazione di default, che serve a controllare i valori che possono essere assunti dall'attributo stesso. In questa sede si vogliono rappresentare i casi notevoli individuati nella traduzione degli attributi.

- Traduzione di attributi CDATA.

DTD	ODL$_{I^3}$

<!ELEMENT elem1 (cs1,cs2)>
<!ATTLIST elem1 a1 CDATA #REQUIRED>
<!ELEMENT cs1 (#PCDATA)>
<!ATTLIST cs1 acs1 CDATA #REQUIRED>
<!ELEMENT cs2 (chs1,chs2)>
<!ATTLIST cs2 acs2 CDATA #REQUIRED>
	Interface elem1
	(...)
	{
	attribute cs1 cs1;
	attribute cs2 cs2;
	attribute string elem1_a1;
	};
	
	Interface cs1
	(...)
	{
	attribute string PCDATA_NODE;
	attribute string cs1_acs1;
	};

Non viene tradotto l'attributo relativo all'elemento cs2 in quanto cs2 é di tipo children e quindi l'attributo verrá convertito nel momento in cui verrá costruita l'interface relativa all'elemento stesso.

- Influenza delle dichiarazioni di default nella traduzione degli attributi

DTD	ODL$_{I^3}$

<!ELEMENT elem1 (cs1)>
<!ATTLIST elem1
atl1 CDATA #REQUIRED>
atl2 CDATA #IMPLIED>
atl3 CDATA #FIXED "1000">
<!ELEMENT cs1 (#PCDATA)>
	Interface elem1
	(...)
	{
	attribute string cs1;
	attribute string elem1_atl1;
	attribute string elem1_atl2?;
	const string elem1_atl3="1000";
	};

- Traduzione delle diverse tipologie di attributi

DTD	ODL$_{I^3}$
<!ELEMENT el1 (cs1)> <!ATTLIST el1 atl1 CDATA #REQUIRED> atl2 ID #REQUIRED> atl3 IDREF #REQUIRED> atl4 ENTITY #REQUIRED> atl5 NMTOKEN #REQUIRED> atl6 (one|two) "one"> <!ELEMENT cs1 (#PCDATA)>	Interface el1 (...) { attribute string cs1; attribute string el1_atl1; attribute string el1_atl2; attribute string el1_atl3; attribute string el1_atl4; attribute string el1_atl5; attribute enum {one.two} el1_atl6;};

Appendice C

Un esempio di traduzione da ODL_{I^3}a XML

Si propone un esempio di Document Type Definition che verrá tradotto in linguaggio ODL_{I^3}attraverso il parser realizzato. In particolare é facile evincere quelle che sono state le scelte che il progettista ha effettuato attraverso l'interfaccia grafica in merito alla determinazione delle chiavi.

C.1 La Document Type Definition

Si vuole rappresentare una modellazione di una Universitá. In questo contesto l'Univiersitá viene intesa come insieme di Persone, che a loro volta possono essere Studenti e Professori. Come si puó osservare dall'analisi delle document type declaration, ogni istanza di Studenti e Professori deve essere modellata con maggiore dettaglio mediante la definizione degli specifici elementi e degli specifici attributi.

```
<!ELEMENT University (Person)*>
<!ELEMENT Person (first_name, last_name, email, Status)>
<!ATTLIST Person Code ID #REQUIRED>
<!ELEMENT Status (Student | Professor)>
<!ELEMENT Student (year, Course*, homeaddress, rank)>
<!ATTLIST Student StudentId ID #REQUIRED
                 tutor CDATA #REQUIRED>
<!ELEMENT Professor (ptitle, Division, rank)>
<!ATTLIST Professor Prof_code ID #REQUIRED
                    Office_phone CDATA #IMPLIED>
<!ELEMENT Division (Location, fund, employeenr)>
<!ATTLIST Division description CDATA #REQUIRED
                   sector CDATA #REQUIRED>
<!ELEMENT Location (city, street, number, county)>
```

```
<!ATTLIST Location code_location CDATA #IMPLIED>
<!ELEMENT Course (name,room)>
<!ATTLIST Course taughtby IDREF #REQUIRED>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT rank (#PCDATA)>
<!ELEMENT ptitle (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT fund (#PCDATA)>
<!ELEMENT employeenr (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT county (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT room (#PCDATA)>
<!ELEMENT homeaddress (#PCDATA)>
```

La DTD puó essere rappresentata sia attraverso un grafo che si ispiri ai principi del modello OEM sia attraverso il data model XML-GDM. É pertanto interessante osservare la rappresentazione del medesimo esempio utilizzando entrambi i modelli in modo da evidenziare le differenze che intercorrono tra di essi. In particolare in figura C.1 attraverso la modellizzazione OEM é facilmente evincibile la struttura ad albero e le relazioni padre e figlio di ogni nodo. Il data model utilizzato nel linguaggio XML-GL (come si vede in figura C.2) focalizza la propria attenzione sulla struttura logica del documento e quindi sulla effettiva rappresentazione di ogni nodo.

É necessario sottolineare che la rappresentazione della DTD non puó fornire indicazioni in merito ai nodi referenziati attraverso gli attributi IDREF. Tali informazioni possono essere evidenziate dal grafo se si rappresentano, attraverso la stessa tipologia grafica, le istanze del file dati, oppure se si tiene conto, al momento della creazione del grafo, delle scelte effettuate dal progettista.

C.2 La traduzione in ODL$_{I^3}$

Si propone la traduzione in ODL$_{I^3}$dell'esempio proposto. In particolare attraverso l'interfaccia grafica devono essere state effettuate le scelte di eventuale qualificazione delle ATTLIST di tipo ID/IDREF e di tipo CDATA relative ad ogni elemento. Tali scelte danno come risultato, nella traduzione ODL$_{I^3}$, la generazione di *key*, *candidate_key* e *foreign_key*.

```
interface university
(source semistructured univers_xml
extent university )
{
         attribute person person ;
};


interface person
(source semistructured univers_xml
extent person
key (person_code))
{
         attribute string person_code ;
         attribute string first_name ;
         attribute string last_name ;
         attribute string email ;
         attribute status status ;

};


interface status
(source semistructured univers_xml
extent status )
{
         attribute student student ;

}
 union status1
{
         attribute professor professor ;

};


interface student
(source semistructured univers_xml
extent student
key (student_studentid)
candidate_key cK0 (student_tutor))
{
         attribute string student_studentid ;
```

```
        attribute string student_tutor ;
        attribute string year ;
        attribute set<course> course ;
        attribute string homeaddress ;
        attribute string rank ;

};


interface professor
(source semistructured univers_xml
extent professor
key (professor_prof_code))
{
        attribute string professor_prof_code ;
        attribute string professor_office_phone ?;
        attribute string ptitle ;
        attribute division division ;
        attribute string rank ;
};


interface course
(source semistructured univers_xml
extent course
foreign_key (course_taughtby) references professor (professor_prof_code))
{
        attribute string course_taughtby ;
        attribute string name ;
        attribute string room ;
};


interface division
(source semistructured univers_xml
extent division )
{
        attribute string division_description ;
        attribute string division_sector ;
        attribute location location ;
        attribute string fund ;
        attribute string employeenr ;
};
```

```
interface location
(source semistructured univers_xml
extent location )
{
         attribute string location_code_location ?;
         attribute string city ;
         attribute string street ;
         attribute string number ;
         attribute string county ;

};
```

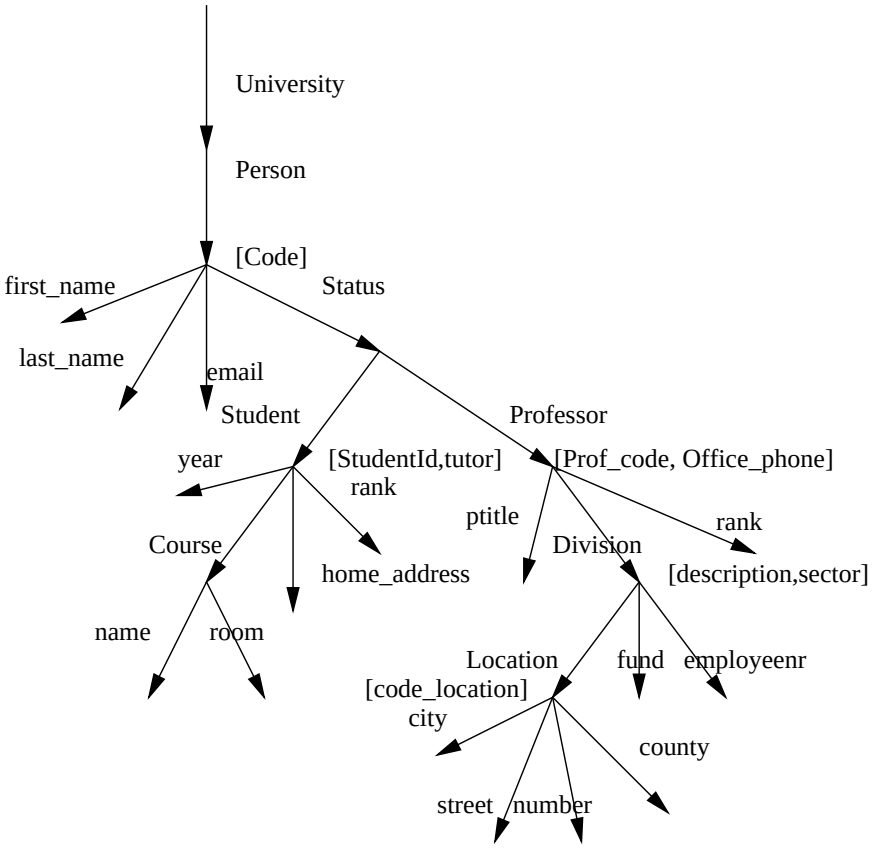

Figura C.1: Modellazione della DTD attraverso un albero OEM

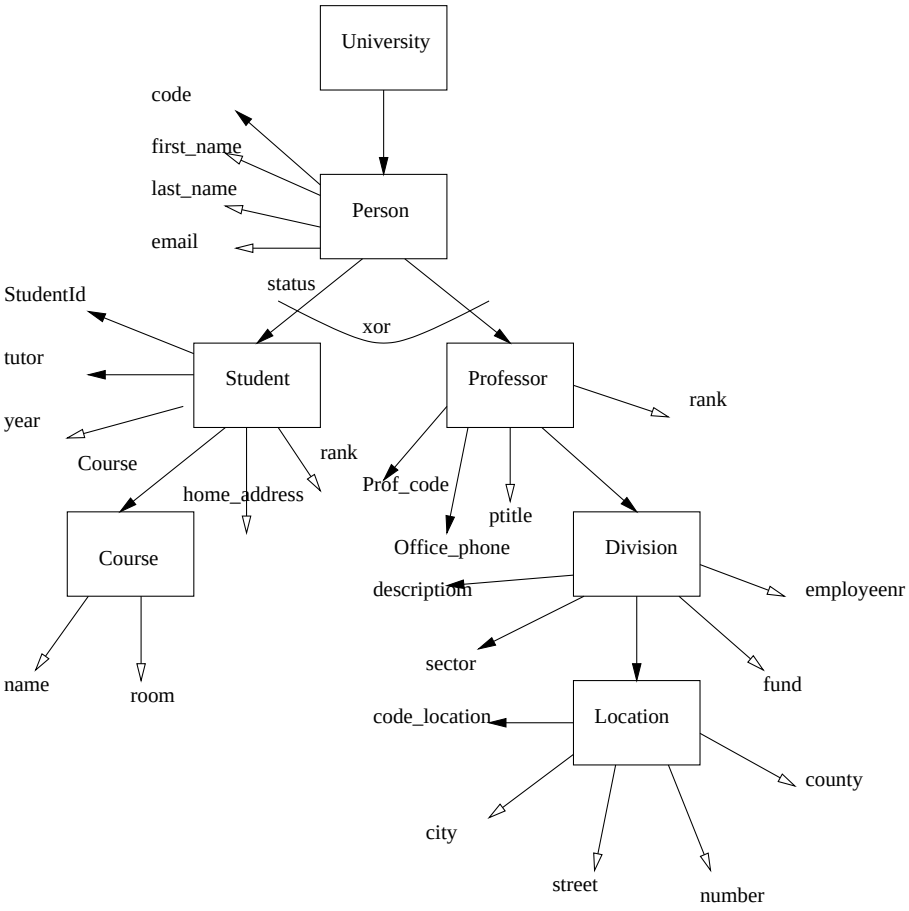

Figura C.2: Modellazione della DTD attraverso il Data Model XML-GDM

Appendice D

La risoluzione di Content Model innestati

Si propone un algoritmo attraverso il quale effettuare la "linearizzazione" di content model di elementi XML innestati a piú livelli di profonditá. Infatti, come si é visto nel capitolo 5, se é lecito descrivere *element* XML il cui contenuto presenti elementi innestati a più livelli di profondità, altrettanto non può essere fatto in ODL_{I^3}dove ogni *interface* ammette la definizione diretta di elementi unicamente a un solo livello di profondità. In particolare si é osservato che nonostante sia lecito in XML un frammento di codice di tale tipo:

```
<!ELEMENT A ((B|C),D)>
```

un costrutto di tale tipo non é tradudicibile in ODL_{I^3}a patto o di introdurre nuovi elementi, oppure di "linearizzare" l'element scrivendolo nella forma, semanticamente equivalente:

```
<!ELEMENT A ((B,D)|(C,D))>
```

In questa sede si vuole pertanto descrivere un algoritmo che consenta la traduzione dalla prima notazione nella seconda, e che permetta quindi, a partire da una qualsiasi definizione di elemento, la scrittura della sua forma normale disgiuntiva. Nel capitolo 5.3.1 é stato approfondito, anche attraverso una ulteriore esemplificazione, il problema in questione.

D.1 L'algoritmo

L'algoritmo che si vuole realizzare genera la forma disgiuntiva normale della specifica di contenuto di un elemento XML (o di parte di esso).

L'algoritmo opera generando un particolare albero. La lettura di ogni singolo ramo dell'albero posta in or logico rispetto alle altre costituisce la forma normale disgiuntiva richiesta. In particolare gli step necessari alla realizzazione della struttura sono i seguenti:

1. Eliminazione delle parentesi più esterne e associazione a ogni blocco di espressione delimitato dalle parentesi ora più esterne di un valore letterale rappresentativo

2. Costruzione dell'albero: se i blocchi definiti allo step precedente sono collegati gli uni agli altri attraverso l'operatore and, i blocchi stessi rappresentano nodi collegati l'uno a un livello inferiore dell'altro, se invece sono collegati attraverso l'operatore di or, l'espressione rappresenta un nodo dal quale partono delle diramazioni rappresentanti i nodi figli.

3. Si analizza, con il medesimo procedimento che si é utilizzato nei primi due step, ognuno dei blocchi generato nel primo step. In particolare si esamina il blocco più a sinistra nella rappresentazione letterale: se é suddivisibile in altri sottoblocchi si procede alla suddivisione degli stessi e all'analisi di quello posto più a sinistra. Se non é suddivisibile in altri sottoblocchi (non vi sono ulteriori parentesi innestate) si costruisce direttamente l'albero dei termini. In questo modo si realizza un albero rappresentativo dell'espressione da mettere in forma normale.

4. Si procede alla lettura di ogni percorso, dal nodo di root, fino alle foglie. Gli elementi componenti il percorso correlati gli uni e gli altri tramite l'operatore and e posti in or rispetto agli altri percorsi individuati rappresentano la forma disgiuntiva normale.

Supponendo ad esempio di dovere generare la forma disgiuntiva normale dell'espressione

```
((A|B|C),(D|(E,F)))
```

che rappresenta il content model di un generico element occorre procedere nella seguente maniera:

- Si elimina la parentesi esterna e si associa un valore di riconoscimento a ogni blocco ottenuto. In questo caso si ottiene l'espressione E_1,E_2
  dove a E_1 corrisponde `(A|B|C)` e a E_2 `(D|(E,F))`
  A questa nuova espressione logica corrisponde la seguente struttura grafica:

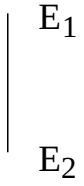

Figura D.1: Prima fase dell'algoritmo

- Si analizza il blocco di sinistra. Non essendo scomponibile in ulteriori sottoblocchi é possibile scrivere direttamente l'albero In figura D.2. é possibile osservare l'albero in questo modo ottenuto.

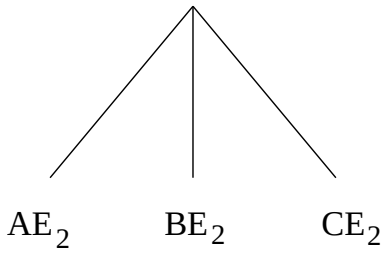

Figura D.2: Seconda fase dell'algoritmo

- Infine si costruisce l'albero finale sviluppando il termine E_2, prima scomponendolo nei due blocchi che lo costituiscono E_{21}, E_{22}, poi sviluppando i due blocchi introdotti, ottenendo l'albero completo

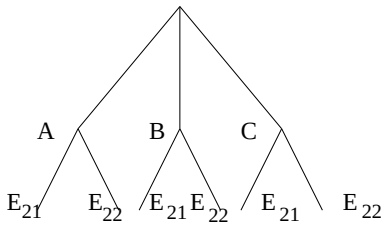

Figura D.3: Terza fase dell'algoritmo

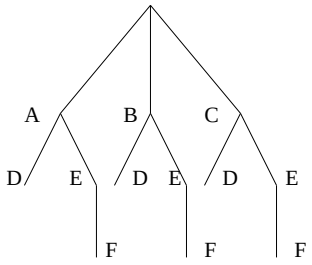

Figura D.4: Albero completo

- A questo punto é sufficiente leggere il valore di ogni ramo per costruire la forma normale disgiuntiva:

```
((A,D)|(A,E,F)|(B,D)|(B,E,F)|(C,D)|(C,E,F))
```

In termini implementativi, la struttura ad albero che qui si introduce piú essere agevolmente realizzata utilizzando uno stack e introducendo una scala di priorità tra i due operatori (e la parentesi) si cui si compone l'espressione da analizzare.

Appendice E

The ODL$_{I^3}$ description language

Si propone la descrizione BNF del linguaggio ODL$_{I^3}$. Sono stati inclusi unicamente gli elementi sintattici che differiscono dalla grammatica originale ODLdello standard ODMG-93.

⟨interface_dcl⟩	::=	⟨interface_header⟩ **{**[⟨ interface_body⟩]**}**; [**union** ⟨identifier⟩ **{** ⟨interface_body⟩ **};**]
⟨interface_header⟩	::=	**interface** ⟨identifier⟩ [⟨inheritance_spec⟩] [⟨type_property_list⟩]
⟨inheritance_spec⟩	::=	: ⟨scoped_name⟩ [,⟨inheritance_spec⟩]

Definizione di modello di schema locale: il wrapper deve potere indicare il tipo e il nome della sorgente per ogni modello.

⟨type_property_list⟩ ::= **(** [⟨source_spec⟩]
[⟨extent_spec⟩]
[⟨key_spec⟩] [⟨f_key_spec⟩] [⟨c_key_spec⟩] **)**
⟨source_spec⟩ ::= **source** ⟨source_type⟩
⟨source_name⟩
⟨source_type⟩ ::= **relational** | **nfrelational**
| **object** | **file**
| **semistructured**
⟨source_name⟩ ::= ⟨identifier⟩
⟨extent_spec⟩ ::= **extent** ⟨extent_list⟩
⟨extent_list⟩ ::= ⟨string⟩ | ⟨string⟩,⟨extent_list⟩
⟨key_spec⟩ ::= **key**[**s**] ⟨key_list⟩
⟨f_key_spec⟩ ::= **foreign_key** (⟨f_key_list⟩)
references ⟨key_list
⟩ [⟨f_key_spec⟩]
⟨c_key_spec⟩ ::= **candidate_key** ⟨identifier⟩
(⟨key_list⟩**)**

Regole di definizione del mapping fra attributi della classe globale dello schema del mediatore e i corrispondenti nelle sorgenti locali.

⟨attr_dcl⟩ ::= **[readonly] attribute**
[⟨domain_type⟩]
⟨attribute_name⟩ **[*]**
[⟨fixed_array_size⟩]
[⟨mapping_rule_dcl⟩]
⟨mapping_rule_dcl⟩ ::= **mapping_rule** ⟨rule_list⟩
⟨rule_list⟩ ::= ⟨rule⟩ | ⟨rule⟩,⟨rule_list⟩
⟨rule⟩ ::= ⟨local_attr_name⟩ |
'⟨identifier⟩'
⟨and_expression⟩ |
⟨union_expression⟩
⟨and_expression⟩ ::= **(** ⟨local_attr_name⟩ **and**
⟨and_list⟩ **)**
⟨and_list⟩ ::= ⟨local_attr_name⟩
| ⟨local_attr_name⟩ **and**
⟨and_list⟩
⟨union_expression⟩ ::= **(** ⟨local_attr_name⟩ **union**
⟨union_list⟩ **on** ⟨identifier⟩ **)**
⟨union_list⟩ ::= ⟨local_attr_name⟩
| ⟨local_attr_name⟩ **union**
⟨union_list⟩
⟨local_attr_name⟩ ::= ⟨source_name⟩.⟨class_name⟩.
⟨attribute_name⟩

...

Relazioni terminologiche utilizzate per definire il Common Thesaurus.

⟨relationships_list⟩ ::= ⟨relationship_dcl⟩; |
⟨relationship_dcl⟩;
⟨relationships_list⟩
⟨relationships_dcl⟩ ::= ⟨local_name⟩
⟨relationship_type⟩
⟨local_name⟩
⟨local_name⟩ ::= ⟨source_name⟩.
⟨local_class_name⟩
[.⟨local_attr_name⟩]
⟨relationship_type⟩ ::= **SYN** | **BT** | **NT** | **RT**
...

Definizione dei vincoli di integrità **OLCD** dichiarazione delle regole (utilizzando le definizioni *if then*) valide per ogni istanza di dato; specificazione delle mapping rule (specificazione delle regole *or* e *and*).

⟨rule_list⟩ ::= ⟨rule_dcl⟩; | ⟨rule_dcl⟩; ⟨rule_list⟩
⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_spec⟩
⟨rule_spec⟩ ::= ⟨rule_pre⟩ **then** ⟨rule_post⟩ |
{ ⟨case_dcl⟩ **}**
⟨rule_pre⟩ ::= ⟨forall⟩ ⟨identifier⟩ **in** ⟨identifier⟩ **:**
⟨rule_body_list⟩
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨case_dcl⟩ ::= **case of** ⟨identifier⟩ **:** ⟨case_list⟩
⟨case_list⟩ ::= ⟨case_spec⟩ | ⟨case_spec⟩ ⟨case_list⟩
⟨case_spec⟩ ::= ⟨identifier⟩ **:** ⟨identifier⟩ **;**

⟨rule_body_list⟩ ::= **(** ⟨rule_body_list⟩ **)** |
⟨rule_body⟩ |
⟨rule_body_list⟩ **and**
⟨rule_body⟩ |
⟨rule_body_list⟩ **and**
(⟨rule_body_list⟩ **)**

⟨rule_body⟩ ::= ⟨dotted_name⟩
⟨rule_const_op⟩
⟨literal_value⟩ |
⟨dotted_name⟩
⟨rule_const_op⟩
⟨rule_cast⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ **in**
⟨dotted_name⟩ |
⟨forall⟩ ⟨identifier⟩ **in**
⟨dotted_name⟩ **:**
⟨rule_body_list⟩ |
exists ⟨identifier⟩ **in**
⟨dotted_name⟩ **:**
⟨rule_body_list⟩

⟨rule_const_op⟩ ::= $=$ | $\geq$ | $\leq$ | $>$ | $<$

⟨rule_cast⟩ ::= **(**⟨simple_type_spec⟩**)**

⟨dotted_name⟩ ::= ⟨identifier⟩ | ⟨identifier⟩**.**
⟨dotted_name⟩

⟨forall⟩ ::= **for all** | **forall**

Appendice F

Grammatica XML

In questa appendice si propone la sintassi XML attraverso la descrizione fornita dalla BNF con l'aggiunta dei vincoli di well-formdness e di validity. In questo modo si vuole fornire una guida esaustiva, anche se di difficile lettura in quanto priva di commenti, del linguaggio XML.

Document

```
[1] document::= prolog element Misc*
```

Character Range

```
[2] Char::= #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD]
                | [#x10000-#x10FFFF]
```

White Space

```
[3] S::= (#x20 | #x9 | #xD | #xA)+
```

Names and Tokens

```
[4] NameChar::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
[5] Name::= (Letter | '_' | ':') (NameChar)*
[6] Names::= Name (S Name)*
[7] Nmtoken::= (NameChar)+
[8] Nmtokens::= Nmtoken (S Nmtoken)*
```

Literals

```
[9] EntityValue::= '"' ([^%&"] | PEReference | Reference)* '"'
                  |  "'" ([^%&'] | PEReference | Reference)* "'"

[10] AttValue::= '"' ([^<&"] | Reference)* '"'
                |  "'" ([^<&'] | Reference)* "'"
```

```
[11] SystemLiteral::= ('"' [^"]* '"') | ("'" [^']* "'")
[12] PubidLiteral::= '"' PubidChar* '"' | "'" (PubidChar - "'")* "'"
[13] PubidChar::= #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]
```

Character Data

```
[14] CharData::= [^<&]* - ([^<&]* ']]>' [^<&]*)
```

Comments

```
[15] Comment::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

Processing Instructions

```
[16] PI::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17] PITarget::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

DATA Sections

```
[18] CDSect::= CDStart CData CDEnd
[19] CDStart::= '<![CDATA['
[20]CData::=  (Char* - (Char* ']]>' Char*))
[21] CDEnd::= ']]>'
```

Prolog

```
[22] prolog::= XMLDecl? Misc* (doctypedecl Misc*)?
[23] XMLDecl::= '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24] VersionInfo::= S 'version' Eq (' VersionNum ' | " VersionNum ")
[25] Eq::=  S? '=' S?
[26] VersionNum::= ([a-zA-Z0-9_.:] | '-')+
[27] Misc::= Comment | PI |  S
```

Document Type Definition

```
[28] doctypedecl::= '<!DOCTYPE' S Name (S ExternalID)? S? ('[' (markupdecl
                    | PEReference | S)* ']' S?)? '>'
                    [VC: Root Element Type ]
[29] markupdecl::= elementdecl | AttlistDecl | EntityDecl | NotationDecl
                   | PI | Comment
                    [VC: Proper Declaration/PE Nesting ]
                    [WFC: PEs in Internal Subset ]
```

Validity Constraint: Root Element Type
Il Name nel "document type declaration" deve coincidere con "element type" dell' elemento radice.
Validity Constraint: Proper Declaration/PE Nesting
Il testo di sostituzione delle entità-parametriche deve essere annidato propriamente

con le dichiarazioni dei markup. Cioé a dire, se il primo carattere o l'ultimo di una dichiarazione di markup (vedi sopra markupdecl) è contenuto nel testo di sostituzione associato ad un riferimento di entità parametrica, entrambi devono essere contenuti nel medesimo testo di sostituzione.

Well-Formedness Constraint: PEs in Internal Subset
Nei sottoinsiemi dei DTD interni, i riferimenti alle entità-parametriche possono capitare solo dove capitano le dichiarazioni dei markup, non all'interno delle dichiarazioni di markup. (Questo non si applica ai riferimenti che capitano nelle entità-parametriche esterne o al sottoinsieme esterno.)

External Subset

```
[30] extSubset::= TextDecl? extSubsetDecl
[31] extSubsetDecl::= ( markupdecl | conditionalSect | PEReference | S )*
```

Standalone Document Declaration

```
[32] SDDecl::= S 'standalone' Eq (("'" ('yes' | 'no') "'") |
               ('"' ('yes' | 'no') '"'))
               [VC: Standalone Document Declaration ]
```

Validity Constraint: Standalone Document Declaration
La dichiarazione di documento "standalone" deve avere il valore "no" se qualsiasi dichiarazione esterna di markup contenga dichiarazioni di:
- attributi con valori di default, se gli elementi a cui questi attributi si applicano sono presenti nel documento senza le specifiche di valore per quegli attributi, o
- entità (oltre che amp, lt, gt, apos, quot), se i riferimenti a quelle entità sono presenti nel documento, o
- attributi con valori soggetti a normalizzazione, dove l'attributo é presente nel documento con un valore che cambierà come risultato della normalizzazione, o
- tipi di elemento con "element content", se spazi bianchi compaiono direttamente all'interno di qualche istanza di quei tipi.

Language Identification

```
[33] LanguageID::= Langcode ('-' Subcode)*
[34] Langcode::= ISO639Code |  IanaCode |  UserCode
[35] ISO639Code::= ([a-z] | [A-Z]) ([a-z] | [A-Z])
[36] IanaCode::= ('i' | 'I') '-' ([a-z] | [A-Z])+
[37] UserCode::= ('x' | 'X') '-' ([a-z] | [A-Z])+
[38] Subcode::= ([a-z] | [A-Z])+
```

Element

```
[39] element::= EmptyElemTag
                | STag content ETag
                [WFC: Element Type Match ]
                [VC: Element Valid ]
```

Well-Formedness Constraint: Element Type Match
Il Name nel tag-di-fine di un elemento deve essere uguale all' "element type" contenuto nel tag-di-inizio.

Validity Constraint: Element Valid
Un elemento é valido se esiste una dichiarazione che rispetta la produzione elementdecl dove il Name corrisponde al tipo di elemento, e si verifica una delle seguenti condizioni:
1.La dichiarazione corrisponde a EMPTY e l'elemento non ha nessun contenuto.
2.La dichiarazione corrisponde a children e la sequenza di elementi figlio appartiene al linguaggio generato dall'espressione regolare che si trova nel modello di contenuto, con gli opzionali spazi bianchi (caratteri che corrispondono al non terminale S) tra ciascuna coppia di elementi figlio.
3.La dichiarazione corrisponde a Mixed e il contenuto consiste di character data e elementi figlio il cui tipo corrisponde ai nomi presenti nel modello di contenuto.
4.La dichiarazione corrisponde a ANY, e sono stati dichiarati i tipi di qualsiasi elemento figlio.

Start-tag

```
[40] STag::='<' Name (S Attribute)* S? '>'
                [WFC: Unique Att Spec ]
[41] Attribute::= Name Eq AttValue
                [VC: Attribute Value Type ]
                [WFC: No External Entity References ]
                [ WFC: No < in Attribute Values ]
```

Well-Formedness Constraint: Unique Att Spec
Nessun nome di attributo può apparire pià di una volta nello stesso tag-di-inizio o nel tag di elemento vuoto

Validity Constraint: Attribute Value Type
L'attributo deve essere stato dichiarato; il valore deve essere del tipo dichiarato. (Per i tipi degli attributi, vedere la sezione "3.3 Dichiarazione lista degli attributi".)

Well-Formedness Constraint: No External Entity References
I valori degli attributi non possono contenere riferimenti diretti o indiretti a entitá

esterne.

Well-Formedness Constraint: No ¡ in Attribute Values
Il testo di sostituzione di qualsiasi entità riferita direttamente o indirettamente in un valore di attributo (a parte "<") non deve contenere il carattere ¡.
End-tag

```
[42] ETag::= '</' Name S? '>'
```

Content of Elements

```
[43] content::= (element | CharData | Reference | CDSect | PI | Comment)*
```

Tags for Empty Elements

```
[44] EmptyElemTag::=  '<' Name (S Attribute)* S? '/>'
                       [WFC: Unique Att Spec ]
```

Element Type Declaration

```
[45] elementdecl::= '<!ELEMENT' S Name S contentspec S? '>'
                     [VC: Unique Element Type Declaration ]
[46] contentspec::= 'EMPTY' | 'ANY' | Mixed | children
```

Validity Constraint: Unique Element Type Declaration Nessun element type puó essere dichiarato piú di una volta.

Element-content Models

```
[47] children::= (choice | seq) ('?' | '*' | '+')?
[48] cp::= (Name | choice | seq) ('?' | '*' | '+')?
[49] choice::= '(' S? cp ( S? '|' S? cp )* S? ')'
               [VC: Proper Group/PE Nesting ]
[50] seq::='(' S? cp ( S? ',' S? cp )* S? ')'
               [VC: Proper Group/PE Nesting ]
```

Validity Constraint: Proper Group/PE Nesting
Il testo di sostituzione di una entitá parametrica deve essere propriamente annidato con gruppi racchiusi tra parentesi. In altre parole, se una delle parentesi di apertura o di chiusura di un costrutto tipo choice, seq, o Mixed é contenuta nel testo di sostituzione di una entitá parametrica, allora entrambi devono essere contenute nel medesimo testo di sostituzione. Per interoperabilitá, se un riferimento ad una entitá-parametrica compare in un costrutto tipo choice, seq, o Mixed, il suo testo di sostituzione non dovrebbe essere vuoto, inoltre nè il primo nè l'ultimo carattere diverso da "blank" del testo di sostituzione dovrebbe essere un connettore (— or ,).

Mixed-content Declaration

```
[51] Mixed::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*'
              | '(' S? '#PCDATA' S? ')'
              [VC: Proper Group/PE Nesting ]
              [VC: No Duplicate Types ]
```

Validity Constraint: No Duplicate Types Lo stesso nome non deve apparire più di una volta in una singola dichiarazione "mixed content".

Attribute-list Declaration

```
[52] AttlistDecl::=  '<!ATTLIST' S Name AttDef* S? '>'
[53] AttDef::= S Name S AttType S DefaultDecl
```

Attribute Types

```
[54] AttType::= StringType | TokenizedType | EnumeratedType
[55] StringType::= 'CDATA'
[56] TokenizedType::= 'ID'
                              [VC: ID ]
                              [VC: One ID per Element Type ]
                              [VC: ID Attribute Default ]
                     | 'IDREF'
                              [VC: IDREF ]
                     | 'IDREFS'
                              [VC: IDREF ]
                     | 'ENTITY'
                              [VC: Entity Name ]
                     | 'ENTITIES'
                              [VC: Entity Name ]
                     | 'NMTOKEN'
                              [VC: Name Token ]
                     | 'NMTOKENS'
                              [VC: Name Token ]
```

Validity Constraint: ID
I valori di tipo ID devono rispettare la produzione Name. Un nome non deve apparire piú di una volta in un documento XML come un valore di questo tipo: cioé i valori ID devono identificare univocamente gli elementi che li portano.

Validity Constraint: One ID per Element Type Nessun "element type" può avere specificato piú di un attributo di tipo ID.

Validity Constraint: ID Attribute Default
Un attributo ID deve avere un default che sia dichiarato #IMPLIED o #REQUIRED.

Validity Constraint: IDREF
Valori di tipo IDREF devono rispettare la produzione Name, e i valori di tipo IDREFS devono rispettare la produzione Names; a ciascun Name deve corrispondere il valore di un attributo ID di qualche elemento del documento XML; cioé i valori IDREF devono essere uguali al valore di qualche attributo ID.

Validity Constraint: Entity Name
Valori di tipo ENTITY devono rispettare la produzione Name, valori di tipo ENTITIES devono rispettare la produzione Names; a ciascun Name deve corrispondere il nome di una entità unparsed dichiarata nel DTD.

Validity Constraint: Name Token
Valori di tipo NMTOKEN devono rispettare la produzione Nmtoken, valori di tipo NMTOKENS devono rispettare la produzione Nmtokens.
Enumerated Attribute Types

```
[57] EnumeratedType::= NotationType | Enumeration
[58] NotationType::=  'NOTATION' S '(' S? Name (S? '|' S? Name)* S? ')'
                     [VC: Notation Attributes ]
[59] Enumeration::= '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'
                     [VC: Enumeration ]
```

Validity Constraint: Notation Attributes
Valori di questo tipo devono essere uguali a uno dei nomi di notazione inclusi nella dichiarazione; tutti i nomi di notazione presenti nella dichiarazione devono essere dichiarati.

Validity Constraint: Enumeration Valori di questo tipo devono essere uguali a uno dei token Nmtoken presenti nella dichiarazione.

Attribute Defaults

```
[60] DefaultDecl::=  '#REQUIRED' | '#IMPLIED'
                      | (('#FIXED' S)? AttValue)
                        [VC: Required Attribute]
                        [VC: Attribute Default Legal]
                        [WFC: No < in Attribute Values]
                        [VC: Fixed Attribute Default ]
```

Validity Constraint: Required Attribute
Se la dichiarazione di default é la parola chiave #REQUIRED, allora l'attributo deve essere specificato per tutti gli elementi di tipo specificato nella dichiarazione "attribute-list".

Validity Constraint: Attribute Default Legal
Il valore di default dichiarato deve rispettare i vincoli lessicali del tipo di attributo lessicale.

Validity Constraint: Fixed Attribute Default Se un attributo ha un valore di default dichiarato con la parola chiave #FIXED, allora tutte le istanze di quell' attributo devono contenere il valore di default.

Conditional Section

```
[61] conditionalSect::= includeSect | ignoreSect
[62] includeSect::= '<![' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'
[63] ignoreSect::= '<![' S? 'IGNORE' S? '[' ignoreSectContents* ']]>'
[64] ignoreSectContents::=  Ignore ('<![' ignoreSectContents ']]>' Ignore)*
[65] Ignore::= Char* - (Char* ('<![' | ']]>') Char*)
```

Character Reference

```
[66] CharRef::= '&#' [0-9]+ ';'
                | '&#x' [0-9a-fA-F]+ ';'
                  [WFC: Legal Character ]
```

Entity Reference

```
[67] Reference::= EntityRef | CharRef
[68] EntityRef::= '&' Name ';'
                   [WFC: Entity Declared ]
                   [VC: Entity Declared ]
                   [WFC: Parsed Entity ]
                   [WFC: No Recursion ]

[69] PEReference::= '%' Name ';'
                   [VC: Entity Declared ]
                   [WFC: No Recursion ]
                   [WFC: In DTD ]
```

Well-Formedness Constraint: Entity Declared
In un documento senza alcun DTD, in un documento con soltanto il sottoinsieme

interno del DTD che non contenga riferimenti ad entità parametrica, o in un documento con "standalone='yes'", il Name dato nel riferimento all'entità deve essere uguale a quello presente in una dichiarazione di entità, ad eccezione di quanto detto i documenti well-formed non hanno bisogno di dichiarare nessuna delle seguenti entità: amp, lt, gt, apos, quot. La dichiarazione di una entità parametrica deve precedere qualsiasi riferimento ad essa. Similmente, la dichiarazione di una entità generale deve precedere qualsiasi riferimento ad essa che appaia in un valore di default all'interno di una dichiarazione di attribute-list. Notare che se le entità sono dichiarate nel sottoinsieme esterno o in una entità parametrica esterna, un processore non-validante non é obbligato a leggere e ad elaborare le loro dichiarazioni; per tali documenti, la regola che un'entità deve essere dichiarata é un vincolo well-formedness solo se standalone='yes'.

Validity Constraint: Entity Declared
In un documento con un sottoinsieme esterno o un'entità parametrica esterna con "standalone='no'", il Name dato nel riferimento di entità deve essere uguale a quella in una dichiarazione di entità. Per interoperabilità, i documenti validi dovrebbero dichiarare le entità amp, lt,entità gt, apos, quot, nella forma specificata nella sezione "4.6 EntitÀ predefinite". La dichiarazione di un'entità parametrica deve precedere qualsiasi riferimento ad essa. Similmente, la dichiarazione di una entità generale deve precedere qualsiasi riferimento ad essa che appaia in un valore di default all'interno di una dichiarazione di attribute-list.

Well-Formedness Constraint: Parsed Entity
Qualsiasi riferimento a entità non deve contenere il nome di una entità unparsed. EntitÀ unparsed possono essere riferite solo nei valori di attributo che sono stati dichiarati di tipo ENTITY o ENTITIES.

Well-Formedness Constraint: No Recursion
Una entità parsed non deve contenere un riferimento ricorsivo su se stessa, sia direttamente che indirettamente.

Well-Formedness Constraint: In DTD
Riferimenti ad entità parametrica possono apparire solamente nel DTD.

Entity Declaration

```
[70] EntityDecl::= GEDecl | PEDecl
[71] GEDecl::=  '<!ENTITY' S Name S EntityDef S? '>'
[72] PEDecl::= '<!ENTITY' S '%' S Name S PEDef S? '>'
[73] EntityDef::= EntityValue | (ExternalID NDataDecl?)
[74] PEDef::= EntityValue | ExternalID
```

External Entity Declaration

```
[75]ExternalID::= 'SYSTEM' S SystemLiteral
                      | 'PUBLIC' S PubidLiteral S SystemLiteral
[76] NDataDecl::= S 'NDATA' S Name
                      [VC: Notation Declared ]
```

Text Declaration

```
[77] TextDecl::= '<?xml' VersionInfo? EncodingDecl S? '?>'
```

Well-Formed External Parsed Entity

```
[78] extParsedEnt::=TextDecl? content
[79] extPE::= TextDecl? extSubsetDecl
```

Encoding Declaration

```
[80] EncodingDecl::= S 'encoding' Eq ('"' EncName '"' |  "'" EncName "'" )
[81] EncName::= [A-Za-z] ([A-Za-z0-9._] | '-')*
```

Appendice G

Grammatica XML-QL

In questa sezione viene descritta la grammatica di XML-QL utilizzando la notazione BNF.
Si adotta inoltre la seguente convenzione: i simboli terminali sono rappresentati tra parentesi angolari, mentre la loro struttura lessicale non viene specificata ulteriormente.
Da notare infine che tale linguaggio, in completa evoluzione, é suscettibile di profondi cambiamenti.

XML-QL	::=	(Function | Query) ⟨EOF⟩
Function	::=	'FUNCTION' ⟨FUN-ID⟩
		'(' (⟨VAR⟩(':' ⟨DTD⟩)?)*')'(':' ⟨DTD⟩)?
		'{' Query '}'
Query	::=	Element | Literal | ⟨VAR⟩ | QueryBlock
Element	::=	StartTag Query EndTag
StartTag	::=	'⟨'(⟨ID⟩|⟨VAR⟩) SkolemID? Attribute* '⟩'
SkolemID	::=	⟨ID⟩ '(' ⟨VAR⟩ (',' ⟨VAR⟩)* ')'
Attribute	::=	⟨ID⟩ '=' ('"' ⟨STRING⟩ '"' | ⟨VAR⟩)
EndTag	::=	'⟨' / ⟨ID⟩? '⟩'
Literal	::=	⟨STRING⟩
QueryBlock	::=	Where Construct ('{' QueryBlock '}')*
Where	::=	'WHERE' Condition (',' Condition)*
Construct	::=	OrderedBy? 'CONSTRUCT' Query
Condition	::=	Pattern BindingAs* 'IN' DataSource | Predicate
Pattern	::=	StartTagPattern Pattern* EndTag
StartTagPattern	::=	'⟨' RegExpr Attribute* '⟩'
RegExpr	::=	RegExpr '*' | RegExpr '+' | RegExpr '.' RegExpr |

		RegExpr '|' RegExpr |
		⟨VAR⟩ ('['⟨INDEX-VAR⟩']')? |
		⟨ID⟩('['⟨INDEX-VAR⟩']')?
BindingAs	::=	'ELEMENT_AS' ⟨VAR⟩ | 'CONTENT_AS' ⟨VAR⟩
Predicate	::=	Expression OpRel Expression
Expression	::=	⟨VAR⟩ | ⟨CONSTANT⟩
OpRel	::=	'<'|'<='|'>'|'>='|'='|'! ='
OrderBy	::=	'ORDER-BY' ⟨VAR⟩+ ('DESCENDING')?
DataSource	::=	⟨VAR⟩ | ⟨URI⟩ | ⟨FUN-ID⟩
		'('DataSource (',' DataSource)*')'

Bibliografia

[1] S. Montanari. Un approccio intelligente all'Integrazione di Sorgenti Eterogenee di Informazione. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1998.

[2] A. Rabitti. Architettura di un Mediatore per un Sistema di Sorgenti Distribuite ed Autonome. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1998.

[3] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. An intelligent approach to information integration. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, june 1998.

[4] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. Exploiting schema knowledge for the integration of heterogeneous sources. In *Sesto Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBD98, Ancona*, pages 103–122, 1998.

[5] S.Bergamaschi, S.Castano, D.Beneventano, and M.Vincini. Semanitc integration and query of heterogenous information sources. *Journal of Data and Knowledge Engineering 1*, 1999.

[6] R. Hull and R. King et al. Arpa i^3 reference architecture, 1995. Available at http://www.isse.gmu.edu/I3_Arch/index.html.

[7] Gio Wiederhold et al. *Integrating Artificial Intelligence and Database Technology*, volume 2/3. Journal of Intelligent Information Systems, June 1996.

[8] F. Saltor and E. Rodriguez. On intelligent access to heterogeneous information. In *Proceedings of the 4th KRDB Workshop*, Athens, Greece, August 1997.

[9] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25:38–49, 1992.

[10] S. Chawathe, Garcia Molina, H., J. Hammer, K.Ireland, Y. Papakostantinou, J.Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *IPSJ Conference, Tokyo, Japan*, 1994. ftp://db.stanford.edu/ pub/chawathe/1994/ tsimmis-overview.ps.

[11] H. Garcia-Molina et al. The TSIMMIS approach to mediation: Data models and languages. In *NGITS workshop*, 1995. ftp://db.stanford.edu /pub/garcia/1995/tisimmis-models-languages.ps.

[12] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specification. Technical report, Stanford University, 1995. ftp://db.stanford.edu /pub/papakonstantinou/1995/ medmaker.ps.

[13] Y.Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *VLDB Int. Conf.*, Bombay, India, September 1996.

[14] N.Guarino. Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. Technical report, Summer School on Information Extraction, Frascati, Italy, July 1997.

[15] N.Guarino. Understanding, building, and using ontologies. A commentary to 'Using Explicit Ontologies in KBS Development', by van Heijst, Schreiber, and Wielinga.

[16] M.J.Carey, L.M. Haas, P.M. Schwarz, M. Arya, W.F. Cody, R. Fagin, M. Flickner, A.W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J.H. Williams, and E.L. Wimmers. Object exchange across heterogeneous information sources. Technical report, Stanford University, 1994.

[17] M.T. Roth and P. Scharz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *Proc. of the 23rd Int. Conf. on Very Large Databases*, Athens, Greece, March 1995.

[18] Y. Arens, C.Y. Chee, C. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[19] Y. Arens, C. A. Knoblock, and C. Hsu. Query processing in the sims information mediator. *Advanced Planning Technology*, 1996.

[20] A. Zanoli. Si-designer, un tool di ausilio all'integrazione di sorgenti di dati eterogenee distribuite: progetto e realizzazione. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1998.

[21] G. P. Grifa. Analisi di Affinità Strutturali fra classi ODL_{I^3} nel Sistema MOMIS. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1999.

[22] A. Zaccaria. Momis: Il componente query manager. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1998.

[23] M. Franceschi. Il componente query manager di momis: utilizzo della conoscenza estensionale. Tesi di Laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 2000.

[24] Domenico Beneventano, Sonia Bergamaschi, Claudio Sartori, and Maurizio Vincini. ODB-QOPTIMIZER: A tool for semantic query optimization in oodb. In *Int. Conference on Data Engineering - ICDE97*, 1997. http://sparc20.dsi.unimo.it.

[25] D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. Odb-tools: a description logics based tool for schema validation and semantic query optimization in object oriented databases. In *Sesto Convegno AIIA - Roma*, 1997.

[26] A.G. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[27] S. Castano and V. De Antonellis. Deriving global conceptual views from multiple information sources. In *preProc. of ER'97 Preconference Symposium on Conceptual Modeling, Historical Perspectives and Future Directions*, 1997.

[28] P. Buneman. Semistructured data. In *Proc. of 1997 Symposium on Principles of Database Systems (PODS97)*, pages 117–121, Tucson, Arizona, 1997.

[29] World Wide Web Consortium. Extensible Markup Language (xml) 1.0. http://www.w3.org/TR/REC-xml, 1998.

[30] A.Deutsch, M. Fernandez, D. Florescu, A.Levy, and D. Suciu. XML-QL: A query language for xml. http://www.w3.org/TR/NOTE-xml-ql/, 1998.

[31] A.Deutsch, M. Fernandez, D. Florescu, A.Levy, and D. Suciu. XML-QL: A query language for xml. http://www.research.att.com/ mff/files/final.html, 1998.

[32] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. Xml-gl: a graphical language for querying and restructuring xml documents. http://www2.elet.polimi.it/people/ceri/Xml-gl, 1998.

[33] Dan Suciu. An overview of semistructured data. *SIGACT News*, 29(4):28–38, 1998.

[34] Y.Papakonstantinou, H.Garcia-Molina, and J.Widom. Object exchange across heterogeneous information sources. Technical report, Stanford University, 1994.

[35] Y.Papakonstantinou, H.Garcia-Molina, and J.Widom. Object exchange across heterogeneous information sources. In *Proc. of ICDE95*, Taipei, Taiwan, 1995.

[36] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Trans. on Database Systems*, 17(3):385–422, September 1992.

[37] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A structural data model for objects. In *SIGMOD*, pages 58–67, Portland, Oregon, 1989.

[38] D. Calvanese, G. De Giacomo, and M. Lenzerini. Structured objects: Modeling and reasoning. In *Proc. of Int. Conference on Deductive and Object-Oriented Databases*, 1995.

[39] Jon Bosak. Four myths about xml. *IEEE Computers*, 31(10):120–122, October 1998.

[40] Jon Bosak. Xml, java, and the future of the web. http://metalab.unc.edu/pub/sun-info/standards/xml/why/ xmlapps.html, 1997.

[41] Dan Suciu. Semistructured data and xml. In *In Proceedings of International Conference on Foundations of Data Organization*, 1998. http://www.research.att.com/ suciu/strudel/external /files/F593433959.ps.

[42] A. Mendelzon, G. Mihaila, and T.Milo. Querying the world wide web. In *In Proceedings of the Fourth Conference on Parallel and Distributed Information System*, Miami, Florida, 1996.

[43] Jennifer Widom. Data management for xml. *IEEE Data Engineering Bulletin, Special Issue on XML*, 22(2):44–52, September 1999.

[44] Alon Levy. More on data management for xml. University of Washington, http://www.cs.washington.edu/homes/alon/widom-response.html, May 1999.

[45] S. Abiteboul. Views and xml. In *Proc. ACM Symp. on Principles of Database Systems*, pages 1–9, 1999.

[46] Alon Levy. Answering queries using views: a survey. University of Washington, submitted for publication 1999, http://www.cs.washington.edu/homes/alon/site/view-survey.ps, 1999.

[47] A. Deutsch, M. Fernandez, D. Florescu, A. Levy andD. Maier, and D. Suciu. Querying xml data. *IEEE Data Engineering Bulletin, Special Issue on XML*, 22(3):27–34, October 1999.

[48] David Maier. Database desiderata for an xml query language". Oregon Graduate Institute, http://www.w3.org/TandS/QL/QL98/pp/maier.html, 1998.

[49] J. Widom R. Goldman, J. McHugh. From semistructured data to xml: Migrating the lore data model and query language. In *Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99)*, Philadelphia, Pennsylvania, 1999.

[50] World Wide Web Consortium. Xml query requirements - w3c working draft 31 january 2000. http://www.w3.org/TR/xmlquery-req, 2000.

[51] J.Paredaens, P.Peelman, and L.Tanca. G-log a declarative graph based language. In *IEE Tans. on Knowlewdge and Data Eng*, 1995.

[52] B.Oliboni and L. Tanca. Querying xml specified www sites: links and recursion in xml-gl, 2000.