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

La conoscenza di MOMIS: il ruolo di XML-Schema e RDF

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Manlio Marchica

Controrelatore
Chiar.mo Prof. Letizia Leonardi

Anno Accademico 2000 - 2001

Parole chiave:
Integrazione delle Informazioni
Web Semantico
XML-Schema
RDF
RDF-Schema

RINGRAZIAMENTI

Ringrazio tutti coloro che mi hanno fornito un aiuto alla realizzazione della presente tesi.
Un ringraziamento particolare va alla mia famiglia che ha creato quell'ambiente di serenitá necessario per lo svolgimento della tesi.

Indice

Introduzione **1**

1 L'Integrazione delle Informazioni **5**
1.1 L'Integrazione Intelligente delle Informazioni 6
1.1.1 Il Programma I^3 6
1.2 Architettura di riferimento per sistemi I^3 7
1.2.1 Uso della tecnologia I^3 8
1.2.2 Servizi di Coordinamento 11
1.2.3 Servizi di Amministrazione 11
1.2.4 Servizi di Integrazione e Trasformazione Semantica . . . 12
1.2.5 Servizi di Wrapping 13
1.2.6 Servizi Ausiliari 13
1.3 Il mediatore 14
1.3.1 Problematiche da affrontare 16
1.3.2 Problemi ontologici 17
1.3.3 Problemi semantici 17

2 Il sistema Momis **21**
2.1 MOMIS 21
2.2 L'architettura di MOMIS 21
2.2.1 Il processo d'integrazione 23
2.2.2 Query processing e ottimizzazione 26
2.3 Gli strumenti utilizzati 27
2.3.1 ODB-Tools 27
2.3.2 WordNet 27

3 Web Semantico **29**
3.1 **RDF(S)** 32
3.1.1 RDF 32
3.1.2 RDF-Schema 38
3.2 XML V_s RDF(S) 42

3.3 Approccio Ontologico 44
3.3.1 OIL (*Ontology Inference Layer*) 45
3.3.2 Aumentare l'espressività di RDF 51
3.3.3 DAML+OIL 51
3.3.4 Differenze fra DAML+OIL e OIL 62
3.4 ODL_{I^3} 62
3.4.1 OLCD 65
3.5 MOMIS nell'architettura del Web Semantico 70

4 Esportazione di schemi ODL_{I^3} in XML-Schema 73
4.1 Introduzione a XML-Schema 73
4.1.1 Origini 73
4.1.2 Validazione e Buona Forma 74
4.1.3 XML-Schema e DTD 74
4.1.4 Documento XML-Schema 74
4.1.5 I tipi in XML Schema 75
4.1.6 Esempi 75
4.1.7 Derivazione di tipi 76
4.1.8 Vincoli 76
4.1.9 Elementi e Attributi 76
4.1.10 Gruppi di elementi ed attributi 79
4.1.11 Annotazioni 79
4.1.12 Namespace 79
4.1.13 Inclusioni & Importazioni 81
4.2 Creazione della Schema 82
4.2.1 Definizione delle Interfacce 82
4.2.2 Il problema delle chiavi 84
4.2.3 Vincoli di integrità referenziale 86
4.2.4 Ereditarietà 87
4.2.5 Attributi ODL_{I^3} 88
4.2.6 Tipi definiti 92
4.2.7 Template Type 93
4.2.8 Tipi "costruiti" (ConstrType) 94
4.2.9 Attributi Globali e mapping rule 96
4.2.10 Relationship 96
4.3 XML-Schema & RDF(S) 97
4.3.1 Attributi openAttrs 97
4.3.2 Uso di RDF con ODL_{I^3} 98
4.4 Il traduttore 106
4.4.1 Introduzione API 106
4.4.2 Generare XML da una struttura dati arbitraria 108

5 XQUERY **119**
5.1 Scenari di utilizzo . . . 119
5.2 Database Desiderata per un linguaggio d'interrogazione per XML 120
5.3 Modello dei dati . . . 122
5.3.1 tipi di Nodo . . . 122
5.3.2 Tipi di dato . . . 123
5.3.3 Esempio . . . 123
5.4 Il Linguaggio d'interrogazione per XML . . . 125
5.4.1 Espressioni di percorso . . . 125
5.4.2 Costruttori di elemento . . . 126
5.4.3 Espressioni FLWR (espressioni FLoWer) . . . 127
5.4.4 Espressioni con operatori e funzioni . . . 129
5.4.5 Espressioni condizionali . . . 129
5.4.6 Quantificatori . . . 129
5.4.7 Filtraggio . . . 130
5.4.8 Tipi di dato . . . 130
5.4.9 Funzioni . . . 131
5.4.10 Tipi di dato definiti dall'utente . . . 131
5.4.11 Operazioni su tipi di dato . . . 132
5.5 Interrogare Basi di Dati Relazionali . . . 134
5.5.1 Selezione . . . 134
5.5.2 Raggruppamento . . . 135
5.5.3 Join . . . 135
5.6 Parser XQuery . . . 136
5.7 Conclusioni . . . 137

Conclusioni e Sviluppi Futuri **138**

A Il linguaggio descrittivo ODL_{I^3} **141**

B Esempio di Ontologia OIL **143**

C Esempio di Ontologia in DAML+OIL **145**

D Metadati RDF-Schema di supporto al Traduttore **151**

E La descrizione di XML-Schema con Schema **153**

F Specifiche RDF Schema **175**

G Esempio di Traduzione **179**

Elenco delle figure

1.1 Diagramma dei servizi I^3 10
1.2 Servizi I^3 presenti nel mediatore 15

2.1 Architettura del sistema MOMIS 22
2.2 Le fasi dell'integrazione 24
2.3 Architettura del Global Schema Builder 25

3.1 L'architettura del Web semantico 31
3.2 Gerarchia delle Classi di RDF Schema 38
3.3 corrispondenza OIL-SHIQ 52
3.4 Estendere RDF(S) tratto da [34] 53

4.1 Schema RDF 99
4.2 XSLT APIs 107
4.3 L'object model del linguaggio ODL_{I^3} 109
4.4 Interface Body 110
4.5 I tipi in ODL_{I^3} 112
4.6 I tipi semplici 113
4.7 ConstrType 114

5.1 Modello dei dati XQUERY 124

G.1 Apertura file ODL_{I^3} 179
G.2 documento Schema 180
G.3 documento RDF-Schema 181
G.4 documento RDF 182

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 varietà di dati accessibili.
Poter gestire in modo efficace questa mole di dati è diventato quindi un fattore cruciale per il successo aziendale ma, paradossalmente, l'aumento nell'offerta di informazione fatica a tradursi in un effettivo vantaggio per l'utente. Tale situazione è la conseguenza di una crescita irregolare che ha portato ad avere una grande varietà di sorgenti disomogenee e quindi difficilmente integrabili.

Il problema di base è appunto l'eterogeneità dei sistemi, la quale può presentarsi in diversi modi, a partire dalle piattaforme Hardware e software su cui una sorgente è basata (ad esempio diversi DBMS e linguaggi di interrogazione), fino ad arrivare ai modelli dei dati (relazionale, object-oriented, ecc...) e agli schemi usati per la rappresentazione della struttura logica dei dati memorizzati.
In un contesto di questo tipo, risulta evidente che, per poter reperire le informazioni desiderate, sarebbe necessario avere familiarità con il contenuto, le strutture ed i linguaggi di interrogazione propri delle singole sorgenti. L'utente dovrebbe quindi essere in grado di scomporre la propria interrogazione in una sequenza di sotto-interrogazioni rivolte alle sorgenti di informazioni provvedendo poi a "$trattare$" i 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.
Disponendo di un numero sempre maggiore di sorgenti e di dati da manipolare diviene difficile trovare persone che posseggano tutte le conoscenze necessarie, perciò risulta indispensabile avere un processo che automatizzi l'intera fase di reperimento ed integrazione delle informazioni.

Questa tesi si inserisce, appunto, in un sistema più ampio denominato **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) [1, 2, 3, 4], sviluppato con l'obiettivo di realizzare l'integrazione di sorgenti eterogenee e distri-

buite.
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. Questa vista integrata deve permettere all'utente la formulazione di interrogazioni svincolandolo dal dover conoscere la struttura ed il contenuto delle singole sorgenti. Il *Mediatore* rappresenta dunque il cuore del sistema ed ha il compito di realizzare l'integrazione degli schemi e di provvedere alla gestione delle interrogazioni.

Elementi indubbiamente innovativi di questo progetto sono l'impiego di un approccio *semantico* e di Logiche Descrittive. 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. Oltre ad una migliore integrazione delle sorgenti si ottiene dunque un processo di gestione delle interrogazioni più efficiente e funzionale.

Per far sì che il sistema MOMIS possa essere usufruito dal più ampio bacino d'utenza è necessario che esso si apra agli standard di comunicazione attuali.

Questo problema è ulteriormente complicato dall'approccio *semantico* e dall'uso di Logiche Descrittive fatto nel sistema MOMIS, in particolar modo sulle modalità di esportazione della conoscenza che tali meccanismi innovativi permettono di catturare.

Alcune delle problematiche del sistema MOMIS sono oggetto di discussione da parte di molti esperti del IT ed hanno portato alla constatazione della necessitá di una terza generazione del web il cosidetto *Semantic-Web* che mira a far in modo che i dati presenti nella rete siano non solo *machine-readable* ma anche *machine-understandable*.

In questa tesi si sono indagate alcune delle proposte più interessanti nate nell'ambito dello sviluppo del *semantic-web* e si é cercato di trovare un approccio per l'esportazione della conoscenza espressa attravarso *l'approccio semantico* del sistema MOMIS.

La tesi è organizzata nel seguente modo: Nel Capitolo 1 viene introdotta l'architettura di riferimento I^3 per i sistemi di Integrazione di Informazioni, ed illustrate le scelte implementative fatte in MOMIS.
Nel Capitolo 2 viene descritto il sistema MOMIS, in particolare la sua architettura, conforme alla proposta internazionale introdotta nel capitolo1. Viene inoltre fatto un accenno degli strumenti software utilizzati.

Il capitolo 3 introduce alle motivazioni che hanno portato alla nascita del progetto del *Semantic-Web* e spiega alcune delle proposte più interessanti nasce in questo ambito.

Il Capitolo 4 descrive il progetto del traduttore ODL_{I^3}-XML-Schema e di

come l'uso di RDF e RDF-Schema può essere di ausilio per mantenere un'elevata percentuale di corrispondenza fra concetti ODL_{I^3} e concetti XML-Schema.

Il capitolo 5 descrive XQUERY il query language per documenti XML, l'ultima proposta del W3C, che rappresenta una prima apertura verso XML-Schema.

Sono inoltre presenti sette appendici: Nell'appendice A viene riportato il BNF del linguaggio descrittivo ODL_{I^3}, nelle Appendici B e C vengono riportati due esempi rispettivamente di Ontologia OIL e di Ontologia DAML+OIL (due dei progetti nati nell'ambito del progetto *Semantic-Web*), nell'Appendice D si riporta il contenuto dei files di supporto al software sviluppato nell'ambito di questa tesi, nell'Appendice E è presentata la definizione della sintassi di XML-Schema attraverso una DTD ed un documento schema, nell'appendice F vi é la presentazione della sintassi RDF-Schema, Infine nell Appendice G si riporta un esempio di utilizzo del traduttore.

Capitolo 1

L'Integrazione delle Informazioni

I problemi principali che si presentano nell'integrazione delle informazioni sono legati alla natura dei dati (testi, immagini, suoni, record, ...) ed ai diversi tipi sorgenti, che possono essere pagine HTML, DBMS relazionali o ad oggetti, file system, etc... .

Gli standard attuali (TCP/IP, ODBC, OLE, CORBA, SQL, etc...) non risolvono completamente i problemi legati alle diversità Hw e Sw dei protocolli di rete e delle comunicazioni fra i moduli, rimangono, infatti, irrisolti quelli specifici della modellazione delle informazioni. I modelli dei dati e gli schemi si differenziano tra loro per quanto riguarda la definizione di una struttura logica per i numerosi generi di dati da immagazzinare e questo crea una eterogeneità semantica (o logica) non risolvibile da questi standard. Inoltre l'abbondanza di informazioni dovuta ad un numero sempre maggiore di fonti, contribuisce a determinare problemi quali l'information overload (sovraccarico delle informazioni); altri problemi per i quali si devono trovare soluzioni sono la riduzione dei tempi di accesso, salvaguardia della sicurezza e della consistenza, gli elevati costi di mantenimento da affrontare per modificare, eliminare od introdurre una nuova sorgente.

Tutti i problemi elencati sottolineano la complessità degli aspetti che le architetture dedicate all'integrazione devono comprendere. Per facilitare il processo di progettazione e realizzazione di tali moduli dedicati, che siano affidabili, flessibili e tali da far fronte efficacemente ad un panorama in continua evoluzione, si cerca 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 creazione di un database distribuito; il repository independence, 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 realiz-

zano 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. Nel seguito si descrive la proposta dell ARPA (Advanced Research Projects Agency)[5] per un architettura che favorisca l'autonomia delle sorgenti, assicu-rando flessibilità e riusabilità e si presenta l approccio seguito per il 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[6]. Per ottenere risultati selezionati é 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 che pratico, oltre a concrete differenze realizzative. 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 che garantisca:

- riusabilità, per diminuire i costi di sviluppo di tali applicazioni.
- flessibilità, 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, é quella del programma I^3, sviluppato dall'ARPA, l'agenzia che fà capo al Dipartimento di americano [5]. L'integrazione aumenta il valore dell informazione ma richiede una forte adattabilità realizzativa: si devono poter 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 uno strumento prezioso per la costruzione automatica di soluzioni integrate flessibili e riusabili.

Costruire in modo premeditato supersistemi ad hoc che interessino una gran quantità di sorgenti non correlate semanticamente, é faticoso ed il risultato é un sistema scarsamente manutenibile e adattabile, strettamente finalizzato alla risoluzione dei problemi per cui é stato implementato. Secondo il programma I^3,

una soluzione a questi problemi deriva dall'introduzione di architetture modulari sviluppabili secondo i principi proposti da uno standard il quale ponga le basi dei servizi che devono essere soddisfatti dall'integrazione ed abbassi i costi di sviluppo e di mantenimento.

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 é 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, moduli intermedi che fanno uso di tecniche di IA, risorse di dati.
- *verticalmente* diversi domini in cui raggruppare le sorgenti, il cui numero deve essere inferiore a dieci.

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 dell'utilizzatore riducendo la complessità totale del sistema e consentendo lo sviluppo di numerose applicazioni finalizzate a scopi 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 ai propri fini. 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 si deve focalizzare l attenzione é quello intermedio: esso costituisce il punto nodale fra le applicazioni sviluppate per gli utenti ed i dati nelle sorgenti. Questo livello deve offrire servizi dinamici quali la selezione delle sorgenti, la 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'architettura di riferimento presentata in questo paragrafo é stata tratta dal sito web [5], e rappresenta una sommaria categorizzazione dei principi e dei servizi che possono e devono essere usati nella realizzazione di un integratore intelligente di informazioni derivanti da fonti eterogenee. Alla base del progetto I^3 stanno infatti due ipotesi:

- la cosiddetta autostrada delle informazioni ‘ e oggigiorno incredibilmente vasta e, conseguentemente, sta per diventare una risorsa di informazioni utilizzabile poco efficientemente.

- le fonti di informazioni ed i sistemi informativi sono spesso semanticamente correlati tra di loro, ma non in una forma semplice nè premeditata. Di conseguenza, il processo di integrazione di informazioni può risultare molto complesso.

In questo ambito, l obiettivo del programma I^3 é di ridurre considerevolmente il tempo necessario per la realizzazione di un integratore di informazioni, raccogliendo e "strutturando" le soluzioni fino ad ora prevalenti nel campo della ricerca.

Da sottolineare, prima di passare alla descrizione dell'architettura di riferimento, che questa architettura non implica alcuna soluzione implementativa, bensì vuole rappresentare alcuni dei servizi che deve includere un qualunque integratore di informazioni, e le interconnessioni tra questi servizi. Inoltre, é opportuno rimarcare che non sarà necessario, ed anzi é improbabile, che ciascun sistema che si prefigge di integrare informazioni (o servizi, o applicazioni) comprenda l'intero insieme di funzionalità che descriverò, bensì usufruirà esclusivamente delle funzionalità necessarie ad un determinato compito.

1.2.1 Uso della tecnologia I^3

Sono molte le applicazioni in cui la tecnologia I^3 può essere utilizzata

- pianificazione e supporto della logistica.
- sistemi informativi nel campo sanitario.
- sistemi informativi nel campo manufatturiero.

Naturalmente, essendo questa riportata una architettura che pretende di essere il più generale possibile, ed essendo la casistica dei campi applicativi così vasta, sarà possibile identificare, al di là di un insieme di servizi di base, funzionalità più adatte ad una determinata applicazione e funzionalità specifiche di un altro ambiente.

Ad esempio, un integratore che vuole interagire con sistemi di basi di dati classici, come possono essere considerati i sistemi basati sui file, quelli relazionali, i DB ad oggetti, necessiterà di un pacchetto base di servizi molto differenti da un sistema cosiddetto "multimediale", che vuole integrare suoni, immagini...

Così come possono essere differenti gli obiettivi di un sistema I^3, saranno differenti pure i problemi che si troverà ad affrontare. Tra questi, possono essere identificati:

- *la grande differenza tra le fonti di informazione*

  - le fonti informative sono semanticamente differenti, e si possono individuare dei livelli di differenze semantiche [7].
  - le informazioni possono essere memorizzate utilizzando differenti formati, come possono essere file, DB relazionali, DB ad oggetti.
  - 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 inoltre la natura stessa delle informazioni, includendo testi,immagini, audio, media digitali.
  - infine, può variare il modo in cui si accede a queste sorgenti: interfacce utente, linguaggi di interrogazione, protocolli e meccanismi di transazione.

- *la semantica complessa ed a volte nascosta delle fonti*: molto spesso, la chiave per l'uso delle informazioni di vecchi sistemi sono i programmi applicativi su di essi sviluppati, senza i quali può essere molto difficile dedurre la semantica che si voleva esprimere, specialmente se si ha a che fare con sistemi molto vasti e quasi impossibili da interpretare se visti solo dall esterno.

- *l'esigenza di creare applicazioni in grado di interfacciarsi con porzioni diverse delle fonti di informazione*: molto spesso, non è sempre possibile avere a disposizione l'intera sorgente di informazione, bensì una sua parte selezionata che può variare nel tempo.

- *il grande numero di fonti da integrare*: con il moltiplicarsi delle informazioni, il numero stesso delle fonti da integrare per una applicazione, ad esempio nel campo sanitario, é aumentato considerevolmente, e decine di fonti devono essere accedute in modo coordinato.

- *il bisogno di realizzare moduli I^3 riusabili*: benchè questo possa essere considerato uno dei compiti più difficili nella realizzazione di un integratore, é importante realizzare non un sistema ad-hoc, bensì un'applicazione i cui moduli possano facilmente essere riutilizzati in altre 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.

Passiamo 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

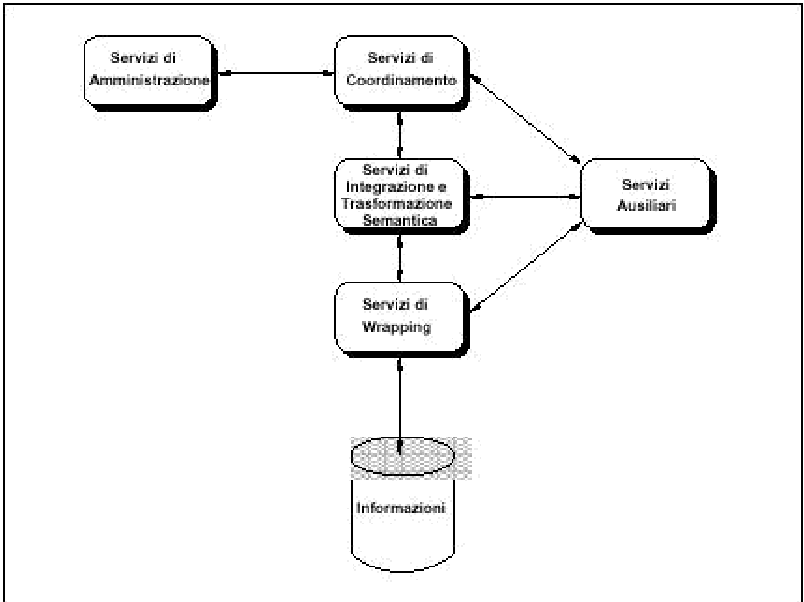


Figura 1.1: Diagramma dei servizi I^3

Coordinamento. Questi servizi giocano infatti due ruoli: come prima cosa, possono localizzare altri servizi I^3 e fonti di informazioni che possono essere utilizzati 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 possono essere identificate in questa architettura: importanti sono i due assi della figura, orizzontale e verticale, che sottolineano i differenti compiti dei servizi I^3.

Se percorriamo l'asse verticale, si può intuire come avviene lo scambio di informazioni nel sistema: in particolare, i serivi di *wrapping* provvedono ad estrarre le informazioni dalle singole sorgenti, che sono poi impacchettate ed integrate dai Servizi di Integrazione e Trasformazioni Semantica, per poi essere passati 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.

Analizziamone in dettaglio funzionalità e problematiche 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, vanno dalla selezione dinamica delle sorgenti (o brokering, per Integratori Intelligenti) al semplice *Matchmaking*, in cui il mappaggio tra informazioni integrate e locali é 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 potranno essere uno solo alla volta (nel qual caso si parla di Brokering) o 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 é richiesta all'utente del sistema una conoscenza del contenuto delle diverse sorgenti, dandogli l'illusione di interagire con un sistema omogeneo che gestisce direttamente la sua richiesta. É quindi esonerato dal conoscere i domini con i quali i vari moduli I^3 hanno a che fare, ottenendone una considerabile diminuzione di complessità di interazione col sistema.

2. Matchmaking: il sistema è configurato manualmente da un operatore all'inizio, e da questo punto in poi tutte le richieste saranno trattate allo stesso modo. Sono definiti gli anelli di collegamento tra tutti i moduli del sistema, e nessuna ottimizzazione é 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 a questi metodi dei Template, sono utilizzate le **Yellow Pages**: servizi di directory che mantengono le informazioni sul contenuto delle varie sorgenti e sul loro stato(attiva, inattiva, occupata). Consultando queste Yellow Pages, il mediatore sarà 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: 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 dall utente. Potrebbe fornirsi a sua volta dei servizi Trasformazione del Vocabolario e dell Ontologia, come pure di Integrazione Semantica. Da citare sono pure i servizi di Iterative Query Formulation: aiutano l'utente a rilassare o 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 saranno una o più sorgenti di dati, e l'output sarà la vista integrata o trasformata di queste informazioni. Tra questi servizi si distinguono quelli relativi alla trasformazione degli schemi (ovvero di tutto ciò che và sotto il nome di metadati) e quelli relativi alla trasformazione dei dati stessi. Sono spesso indicati come servizi di Mediazione, essendo tipici dei moduli mediatori.

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 condivideva un unica ontologia. Fondamentale, per creare questo insieme di vocaboli condivisi, é 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 quel-

la 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 é scomposta in molte subquery, da inviare a fonti differenti, ed i loro risultati 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. 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 erano 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 é 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 per 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 ave-

re 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

Modulo intermedio dell architettura precedentemente descritta, che si pone tra l'utente e le sorgenti di informazioni. Secondo la definizione proposta da Wiederhold in [8] "un mediatore é 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 quando cambiano le 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.

L'approccio architetturale scelto é quello classico, che consta principalmente di 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 mediatore che si é progettato é riportata in 1.2 In particolare, in questa sono state esaminate le seguenti funzionalità:

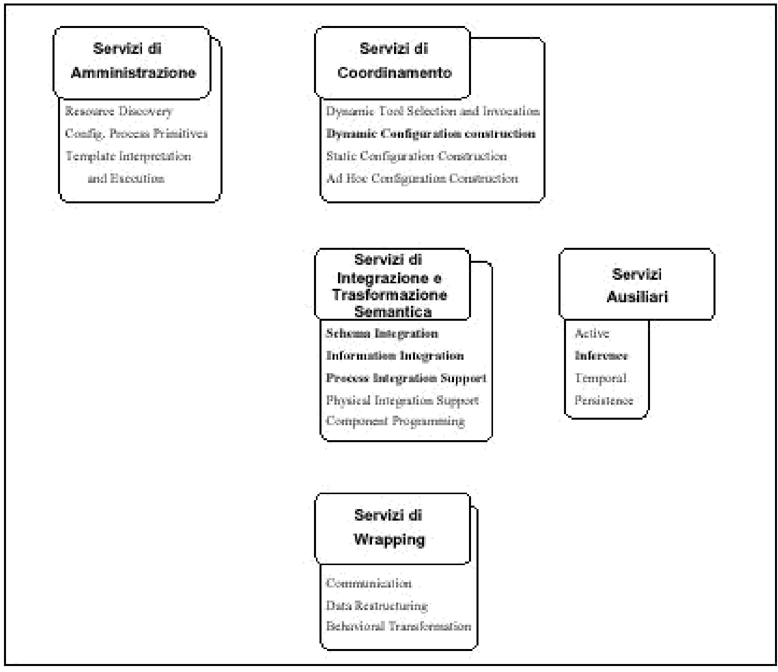


Figura 1.2: Servizi I^3 presenti nel mediatore

- servizi di Coordinamento: sul modello di facilitatori e mediatori, il sistema sarà in grado, in presenza di una interrogazione, di individuare automaticamente tutte le sorgenti che ne saranno interessate, ed eventualmente di scomporre la richiesta in un insieme di sottointerrogazioni diverse da inviare alle differenti fonti di informazione;

- servizi di Integrazione e Trasformazione Semantica: saranno forniti dal mediatore servizi che facilitino l'integrazione sia degli schemi che delle informazioni, nonchè funzionalità di supporto al processo di integrazione (come può essere la Query Decomposition).

- servizi Ausiliari: sono utilizzate tecniche di Inferenza per realizzare, all'interno del mediatore, una fase di ottimizzazione delle interrogazioni.

Parallelamente a questa impostazione architetturale inoltre, il nostro progetto si vuole distaccare dall'approccio strutturale, cioè sintattico, tuttora dominante tra i sistemi presenti sul mercato. L approccio strutturale, adottato da sistemi quali TSIMMIS [9, 10, 11, 12], 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 é l oggetto stesso che, attraverso delle etichette, si autodescrive, specificando tutte le volte, per ogni suo singolo campo, il significato che ad esso é 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 il nostro, seguono invece un approccio definito semantico, che é 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 é certamente un compito banale individuare i concetti comuni ad esse, le relazioni che possono legarli, nè tantomeno é banale realizzare una loro coerente integrazione. Mettendo da parte per un attimo le differenze dei sistemi fisici (alle quali dovrebbero pensare i moduli wrapper) i problemi che si é dovuto 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

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 poichè sono condivisi dall'intera comunità di utenti del dominio applicativo stesso. Non è certamente l'obiettivo nè di questo paragrafo, nè della tesi in generale, dare una descrizione esaustiva di cosa si intenda per ontologia e dei problemi che essa comporta (ancorchè ristretti al campo dell'integrazione delle informazioni), ma mi limito a riportare una semplice classificazione delle ontologie (mutuata da Guarino [13, 14], per inquadrare l ambiente in cui ci si muove. I livelli di ontologia (e dunque le problematiche ad essi associate) sono essenzialmente quattro:

1. top-level ontology: descrivono 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: descrivono, 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: descrivono concetti che dipendono sia da un particolare dominio che da un particolare obiettivo.

Come ipotesi semplificativa di questo progetto, si é 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 ci dice che i diversi sistemi usino esattamente gli stessi vocaboli per rappresentare questi concetti, nè tantomeno le stesse strutture dati. Poichè infatti le diverse sorgenti sono

state progettate e modellate da persone differenti, é 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 [7] la causa principale delle differenze semantiche si può identificare nelle diverse concettualizzazioni del mondo esterno che persone distinte possono avere, ma non é 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 é fornire un accesso integrato ad un insieme di sorgenti, si traduce allora nel non facile compito di identificare i concetti comuni all’interno diqueste 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: benchè 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 é 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’altro 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.

Parallelamente a tutto questo, é 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 cosidetto 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.

Capitolo 2

Il sistema Momis

Tenendo presente tutte le problematiche relative al mediatore esposte nella sezione 1.3, e un insieme di progetti preesistenti quali TSIMMISS [9, 10, 11, 12] GARLIC [15, 16] e SIMS [17, 18], si é giunti alla progettazione di un sistema intelligente di Integrazione delle Informazioni.

2.1 MOMIS

MOMIS(Mediator EnvirOnment for Multiple Information Sources è il progetto di un sistema I^3 per l integrazione di sorgenti di dati strutturati e semistrutturati.

Momis nasce all interno del progetto MURST 40% INTERDATA, come collaborazione fra le unità operative dell Università di Milano e dell Università di Modena.

2.2 L'architettura di MOMIS

MOMIS é stato progettato per fornire un accesso integrato ad informazioni eterogenee memorizzate sia in database di tipo tradizionale (e.g. relazionali, object-oriented) o file system, sia in sorgenti di tipo semistrutturato. Seguendo l'architettura di riferimento I^3 [5] in MOMIS si possono distinguere cinque componenti principali (come si può vedere da 2.1):

1. Wrapper: posti al di sopra di ciascuna sorgente, sono i moduli che rappresentano l'interfaccia tra il mediatore vero e proprio e le sorgenti locali di dati. La loro funzione é duplice:

   - in fase di integrazione, forniscono la descrizione delle informazioni in essa contenute. Questa descrizione viene fornita attraverso il linguaggio ODL_{I^3} (descritto in Sezione 4.2).

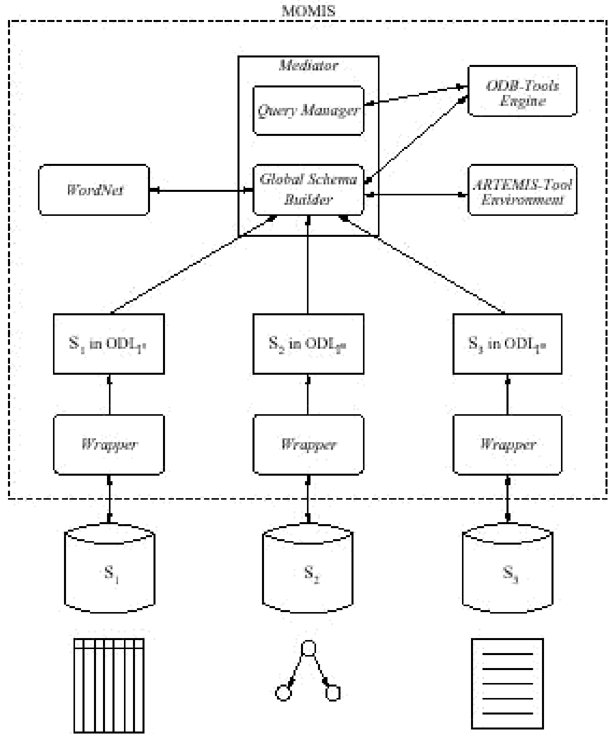


Figura 2.1: Architettura del sistema MOMIS

- in fase di query processing, traducono la query ricevuta dal mediatore (espressa quindi nel linguaggio comune di interrogazione OQL_{I^3}, definito in analogia al linguaggio OQL) in una interrogazione comprensibile (e realizzabile) dalla sorgente stessa. Devono inoltre esportare i dati ricevuti in risposta all interrogazione, presentandoli al mediatore attraverso il modello comune di dati utilizzato dal sistema.

2. mediatore: é il cuore del sistema, ed é composto da due moduli distinti.

   - Global Schema Builder (GSB): é il modulo di integrazione degli schemi locali che, partendo dalle descrizioni delle sorgenti espresse attraverso il linguaggio ODL, genera un unico schema globale da presentare all'utente.
   - Query Manager (QM): é il modulo di gestione delle interrogazioni. In particolare, genera le query in linguaggio OQL_{I^3} da inviare ai wrapper partendo dalla singola query formulata dall'utente sullo schema globale. Servendosi delle tecniche di Logica Descrittiva, il QM genera automaticamente la traduzione della query sottomessa nelle corrispondenti sub-query delle singole sorgenti.

3. ODB-Tools Engine, un tool basato sulla OLCD Description Logics [19, 20] che compie la validazione di schemi e l'ottimizzazione di query [21, 22, 23].

4. ARTEMIS-Tool Environment, un tool che compie analisi e clustering di schemi [24, 25].

5. WordNet, un database lessicale della lingua inglese, capace di individuare relazioni lessicali e semantiche fra termini [26].

Il principale scopo che ci si é preposti con MOMIS é la realizzazione di un sistema di mediazione che, a differenza di molti altri progetti analizzati, contribuisca a realizzare, oltre alla fase di query processing, una reale integrazione delle sorgenti. Entrambe queste fasi sono descritte nei prossimi paragrafi.

2.2.1 Il processo d'integrazione

L'integrazione delle sorgenti informative strutturate e semistrutturate viene compiuta in modo semi-automatico, utilizzando le descrizioni degli schemi locali in linguaggio ODL_{I^3} e combinando le tecniche di Description Logics e di clustering. Come mostrato in figura 2.2, le attività compiute sono le seguenti:

1. Generazione del Thesaurus Comune, grazie al supporto di ODB-Tools e di WordNet. Durante questo passo viene costruito un Thesaurus Comune

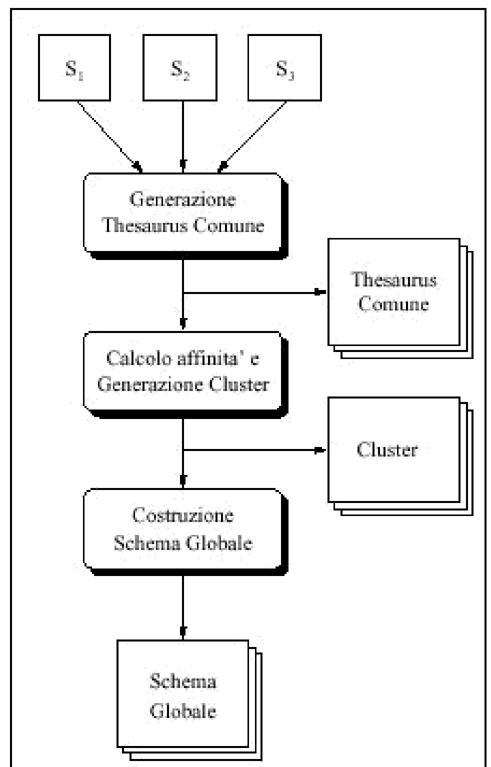


Figura 2.2: Le fasi dell’integrazione

di relazioni terminologiche. Le relazioni terminologiche esprimono la conoscenza inter-schema su sorgenti diverse e corrispondono alle asserzioni intensionali utilizzate in [27]. Le relazioni terminologiche sono derivate in modo semi-automatico a partire dalle descrizioni degli schemi in ODL_{I^3}, attraverso l'analisi strutturale (utilizzando ODB-Tools e le tecniche di Description Logics) e di contesto (attraverso l'uso di WordNet) delle classi coinvolte.

2. Generazione dei cluster di classi ODL_{I^3}, con il supporto dell ambiente ARTEMIS-Tool, le relazioni terminologiche contenute nel Thesaurus vengono utilizzate per valutare il livello di affinità tra le classi ODL_{I^3} in modo da identificare le informazioni che devono essere integrate a livello globale. A tal fine, ARTEMIS calcola i coefficienti che misurano il livello di affinità delle classi ODL_{I^3} basandosi sia sui nomi delle stesse, sia sugli attributi. Le classi ODL_{I^3} con maggiore affinità vengono raggruppate utilizzando le tecniche di clustering [28].

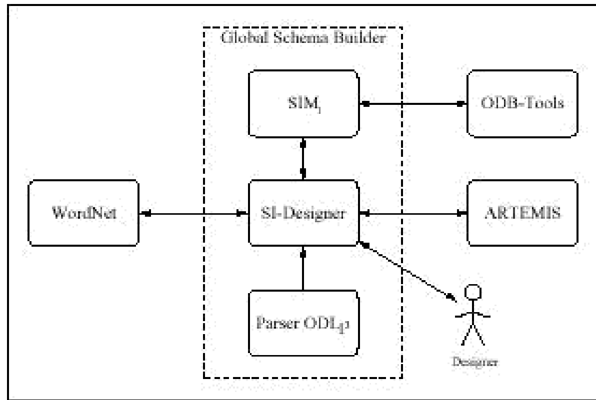


Figura 2.3: Architettura del Global Schema Builder

3. Costruzione dello schema globale del mediatore,i cluster di classi ODL_{I^3} affini sono analizzati per costruire lo schema globale del Mediatore.

   Per ciascun cluster viene definita una classe globale ODL_{I^3} che rappresenta tutte le classi locali che sono riferite al cluster, e che é caratterizzata dall'unione dei loro attributi.

   L'insieme delle classi globali definite costituisce lo schema globale del Mediatore che deve essere usato per porre le query alle sorgenti locali integrate.

Il Global Schema Builder

Il Global Schema Builder é la parte del Mediatore di Momis che si preoccupa della costruzione dello schema integrato, secondo le fasi appena elencate. Come si nota dalla figura 2.3, é composto da tre componenti che interagiscono con strumenti software esterni, descritti a loro volta nella prossima sezione.

I componenti il GSB sono:

1. SIM_1: descritto in [1], si occupa della generazione del Thesaurus comune, in particolare della estrazione delle relazioni dalla struttura degli schemi sorgenti e, con l aiuto di ODB-Tools, della validazione delle relazioni integrate dal progettista nonchè dell'inferenza di nuove relazioni.

2. SI-Designer: ha il duplice scopo di interfacciarsi fra il sistema e il progettista (fornendo a quest'ultimo un'interfaccia amichevole per l'interazione) e di coordinare l'esecuzione dei diversi software che partecipano all integrazione; in particolare:

   - interagisce con WordNet per estrarre automaticamente relazioni lessicali tra i termini usati per dare nome a classi ed attributi.
   - utilizza Artemis per svolgere il calcolo delle affinità.
   - assiste il progettista nelle ultime fasi della costruzione dello schema globale, generando in linguaggio ODL_{I^3} il codice che lo descrive.

3. Parser ODL_{I^3}: effettua l'analisi degli schemi sorgenti, verificandone la consistenza, analizza e archivia, inoltre, tutte le informazioni sullo schema integrato.

2.2.2 Query processing e ottimizzazione

Quando l'utente pone una query sullo schema globale, MOMIS la analizza e produce un insieme di sub-query che saranno inviate a ciascuna sorgente informativa coinvolta. In accordo con altri approcci proposti in quest'ambito [17, 18], il processo consiste di due attività principali:

1. Ottimizzazione Semantica. Il Mediatore agisce sulla query sfruttando la tecnica di ottimizzazione semantica supportata da ODB-Tools in modo da ridurre il costo del piano d accesso. L'ottimizzazione é basata sull inferenza logica a partire dalla conoscenza contenuta nei vincoli di integrità dello schema globale. La stessa procedura di ottimizzazione semantica si realizza in termini locali su ogni query tradotta dal Mediatore nella formulazione del piano d'accesso: in questo caso ci si basa sui vincoli di integrità presenti sui singoli schemi locali.

2. Formulazione del piano di accesso. Dopo aver ottenuto la query ottimizzata, viene generato l'insieme di sub-query relative alle sorgenti coinvolte. A questo scopo, il Mediatore utilizza l'associazione tra classi globali e locali per esprimere la query globale in termini degli schemi locali, nonchè l'eventuale conoscenza di regole interschema definite sulle estensioni delleclassi locali.

2.3 Gli strumenti utilizzati

2.3.1 ODB-Tools

ODB-Tools è un sistema per l'acquisizione e la verifica di consistenza di schemi di basi di dati e per l'ottimizzazione semantica di interrogazioni nelle basi di dati orientate agli oggetti (OODB). É stato sviluppato presso il Dipartimento di Scienze dell'Ingegneria dell Università di Modena [21, 29]. L ambiente teorico su cui é basato ODB-Tools include due elementi fondamentali:

1. OLCD(Object Language with Complements allowing Descriptive cycles), derivante dalla famiglia KL-ONE [30], proposto come formalismo comune per esprimere descrizioni di classi, vincoli di integrità, ed interrogazioni e dotato di tecniche di inferenza basate sul calcolo della sussunzione introdotte per le Logiche Descrittive nell ambito dell Intelligenza Artificiale.

2. espansione semantica di un tipo, realizzata attraverso l'algoritmo di sussunzione.

2.3.2 WordNet

WordNet [26] é un database lessicale elettronico considerato la più importante risorsa disponibile per i ricercatori nei campi della linguistica computazionale, dell'analisi testuale, e in altre aree associate. É stato sviluppato dal Cognitive science Laboratory alla Princeton University, sotto la direzione del Professor George A. Miller. WordNet é un sistema di riferimento, disponibile on-line, la cui architettura é ispirata alle attuali teorie psicolinguistiche legate alla memoria lessicale umana. Sostantivi, verbi, aggettivi e avverbi della lingua inglese vengono organizzati in insiemi di sinonimi (synset), ognuno dei quali rappresenta un determinato concetto lessicale. Vari tipi di relazioni collegano fra loro i synset.

Capitolo 3

Web Semantico

Il Web attualmente é una raccolta d'informazioni utilizzabili dalle persone ma non dagli elaboratori se non dopo una fase di post-elaborazione delle informazioni.

Il Web é arrivato alla sua seconda generazione; la prima é stata quella delle pagine HTML statiche, la seconda è l'attuale con pagine dinamiche generate attraverso l'interrogazione dei dati contenuti in un database; con l'aumentare delle informazioni disponibili e delle aspettative che la gente ripone in Internet é nata l'esigenza di pensare alla *terza generazione del web o web semantico*.

Lo scopo che si prefigge la *terza generazione del web* é quello di rendere le informazioni direttamente utilizzabili dagli elaboratori. Questo é ciò che Tim Berners-Lee chiama *Semantic Web* vedi [31] e [32].

Il raggiungimento di tale obiettivo avrá ripercussioni in molti campi:

- Motori di Ricerca: Attualmente i motori di ricerca restituiscono un'alta percentuale di documenti che non riguardano l'argomento cercato dall'utente; fornendo la capacitá di capire il *significato* delle parole si aumenteranno le capacitá di ricerca.

- E-commerce: Un sito di E-Commerce potrá capire meglio le preferenze di un cliente se capisce il *significato* delle parole che esprimono tali preferenze.

- Diritti di Proprietá: Si potrá esprimere in maniera piú chiara la proprietá intellettuale di una pagina Web, in quanto sará chiaro il significato di parole quali: autore, editore, ecc.. .

Il Web semantico dovrà essere un framework dove agenti software che viaggiano da una pagina ad un'altra possano eseguire compiti sofisticati per gli utenti. Ad esempio un agente che arriva al sito di una clinica ospedaliera potrà non solo leggere le parole chiavi : trattamento, medicine, terapia , ecc.. (come si fà oggi)

ma capirà anche che il Dr. Hartman lavora nella clinica il Lunedi, Mercoledi e Venerdi.

Il Web semantico nell'intento dei suoi promotori non dovrà essere un'entita separata da ciò che é adesso il web ma piuttosto un'estensione, nel quale le informazioni hanno associato un ben definito significato, migliorando così la cooperazione uomo-macchina.

Il segreto del successo di Internet é stato la decentralizzazione delle informazioni, questo ha permesso il raggiungimento di mete all'inizio inaspettate; il web semantico dovrà mantenere tale caratteristica; questo porterà ad accettare una serie di compromessi.

Per far sì che il web semantico rappresenti una svolta é necessario garantire due livelli d'interoperabilità:

1. Sintattica.

2. Semantica.

Vediamo innanzitutto cosa si intende con questi due termini:

Interoperabilità Sintattica Capacità di leggere i dati ed ottenere una rappresentazione utilizzabile da un'applicazione.

Interoperabilità Semantica Capacità di comprendere i dati.

per stabilire questi due livelli d'interoperabilità é necessario definire un linguaggio che permetta di formalizzare una semantica e di fornire un supporto al ragionamento (individuazione di relazioni d'inclusione, equivalenza implicite od inconsistenze).

Queste nuove funzionalità vanno a definire un nuovo livello nell'architettura del web semantico, noto come *"logico"*.

Il primo livello d'interoperabiltà, quella sintattica, é già da tempo raggiunta grazie all'affermarsi dell'XML che permette di definire la struttura dei propri documenti ma non dice nulla sul significato della struttura; mentre il secondo livello d'interoperabilità, quella semantica, ha ricevuto negli ultimi tempi un notevole aiuto dalle ultime proposte del W3C in particolare RDF e RDF-Schema.

L'obiettivo di RDF(S)[1] è quello di definire un meccanismo *generale* di rappresentazione della conoscenza, questo vuole dire non fare alcuna ipotesi sul dominio applicativo nel quale avviene lo scambio, nè definire (a priori) la semantica del dominio applicativo. Il meccanismo fornito da RDF(S) é neutrale per quanto riguarda il dominio perchè deve essere in grado di descrivere le informazioni di qualsiasi natura.

[1] Il termine "RDF(S)" è usato per riferirsi a RDF e RDF-Schema

RDF(S) permette quindi di definire un *vocabolario*, un insieme di termini e di loro definizioni ma non é una logica descrittiva qundi non può essere inserito in quello che abbiamo chiamato *livello logico* perchè non fornisce alcun supporto al ragionamento.

Abbiamo accennato più volte ad un livello logico, in quanto il Web semantico può essere visto come un'architettura a tre livelli.

1° **livello** Data Layer.

2° **livello** Schema Layer.

3° **livello** Logical Layer.

Logical Layer *Semantica Formale e supporto al ragionamento* *- linguaggio di rappresentazione di conoscenza*
Schema Layer *Definizione di un vocabolario / concetti* *- RDF-Schema*
Data Layer *Sintassi per le istanze dei concetti* *- RDF*

Figura 3.1: L'architettura del Web semantico

Il resto del capitolo é organizzato come segue: Descrizione di RDF(S), confronto fra XML e RDF(S), introduzione all'approccio ontologico, descrizione di alcuni linguaggi in grado di occupare il livello logico, descrizione di ODL_{I^3} e OLCD che nell'ambito del progetto MOMIS offrono i serivizi offerti dal livello logico; infine come si puó inquadrare MOMIS nell'architettura del Web Semantico.

3.1 RDF(S)

RDF(S) (Resource Description Framework) é il risultato del lavoro che il W3C stà svolgendo nell'ambito del web semantico; RDF(S) fornisce un modo non ambiguo per la codifica, lo scambio e il riuso di metadati grazie ad un modello simile a quello entità-relazioni. RDF(S) può avere molte applicazioni: supporto ai motori di ricerca nel classificare le informazioni trovate sulla rete, uso da parte di agenti software intelligenti per facilitare la condivisione e lo scambio di conoscenza, indicare la proprietà intellettuale di una pagina web; RDF(S) e la firma digitale potrà essere la chiave di volta del commercio elettronico.

3.1.1 RDF

RDF descrive delle *risorse*, le risorse sono qualsiasi cosa abbia un URI (Uniform Resource Identifier), le caratteristiche di RDF sono:

Indipendenza Ognuno può inventare le proprietà che gli interessano. Noi possiamo inventare la proprietà source, un'altra persona può inventare la proprietà sorgente,ecc..

Interscambio Le proprietà RDF possono essere espresse in XML, quindi sono facilmente interscambiabili.

Scalabilità Le proprietà RDF sono tuple con tre campi (Resource, PropertyType, Value), in questo modo sono facili da gestire e cercare, questa caratteristica in una realtà come Internet in cui la mole di informazione a disposizione é sempre più crescente é molto importante.

Modello dei dati RDF

Il modello dei dati di RDF é molto simile al modello ER, quest'ultimo é più generale.

RDF può essere considerato l'estensione del modello ER al web; perchè ha i concetti di entità e relazione, solamente che quest'ultimo concetto é stato elevato

allo stesso livello delle entità (hanno una loro identità indipendente dalle entità che mettono in relazione) grazie all'utilizzo di URI; in questo modo chiunque può dire tutto su tutto e non é necessario che tutte le informazioni su un'entità siano definite in un unico luogo.

Volendo fare un paragone con un database relazionale potremmo dire che

- Una riga di una tabella é una risorsa RDF.
- Il nome di un campo(una colonna) é il nome di una proprietà RDF.
- Il valore del campo é il valore della proprietà.

Il modello dei dati di RDF si basa su 3 concetti principali:

1. Risorse: Tutto quello che é descritto da espressioni RDF, individuate da un URI.
2. Proprietà: una caratteristica, una relazione usata per descrivere una risorsa; ogni proprietà ha un significato, un insieme di valori che può assumere, le risorse a cui può essere associata ed eventuali relazioni con altre proprietà.
3. Valore della proprietà.

L'insieme delle proprietà e la loro semantica (poca a dir la verità) sono definiti in uno schema RDF, il quale rappresenta una sorta di vocabolario; in questo vocabolario possono essere indicate anche le risorse che verrano descritte.

Sintassi RDF

```
[1] RDF              ::= ['<rdf:RDF>'] description* ['</rdf:RDF>']
[2] description      ::= '<rdf:Description' idAboutAttr? '>'
                            propertyElt*
                         '</rdf:Description>'
[3] idAboutAttr      ::= idAttr | aboutAttr
[4] aboutAttr        ::= 'about="' URI-reference '"'
[5] idAttr           ::= 'ID="' IDsymbol '"'
[6] propertyElt      ::= '<' propName '>' value '</' propName '>'
                         | '<' propName resourceAttr '/>'
[7] propName         ::= Qname
[8] value            ::= description | string
[9] resourceAttr     ::= 'resource="' URI-reference '"'
[10] Qname            ::= [ NSprefix ':' ] name
[11] URI-reference    ::= string, interpreted per [URI]
[12] IDsymbol         ::= (any legal XML name symbol)
[13] name             ::= (any legal XML name symbol)
[14] NSprefix         ::= (any legal XML namespace prefix)
[15] string           ::= (any XML text, with "<", ">",
                           and "&" escaped)
```

L'elemento `RDF` definisce i confini in un documento XML degli statement RDF; l'elemento `Description` indica l'inizio della descrizione di una risorsa, l'attributo `about` o `id` indica la risorsa descritta, se entrambi mancanti si parla di risorsa anonima. All'interno della sezione description vi é l'elenco delle proprietà le quali possono a loro volta essere delle risorse.

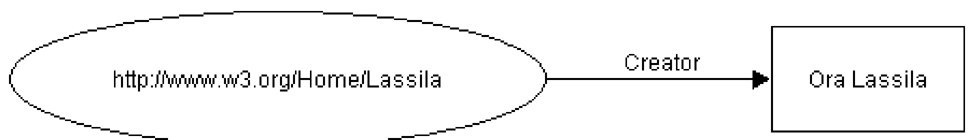
Esempi

Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila

Soggetto(Risorsa)	http://www.w3.org/Home/Lassila
Predicato(Proprietà)	creator
Oggetto(Valore)	Ora Lassila

In RDF/XML:

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:s="http://description.org/schema/">
 <rdf:Description about="http://www.w3.org/Home/Lassila">
  <s:Creator>Ora Lassila</s:Creator>
 </rdf:Description>
</rdf:RDF>
```

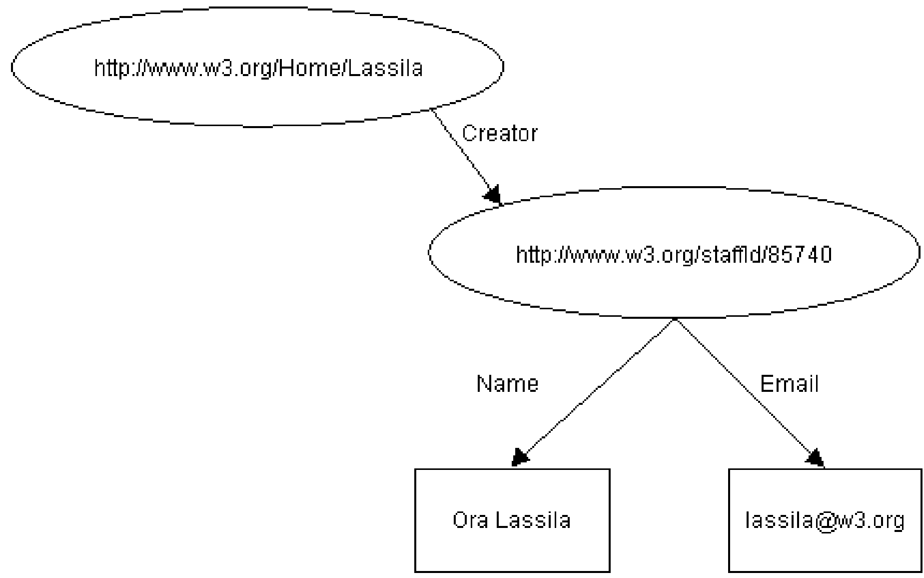


La semantica delle proprietà si ottiene riferendosi ad uno schema RDF; tale riferimento si ottiene attraverso la stessa notazione dei *Namespace* in XML (es: s:Creator).

Come abbiamo detto le proprietà di una risorsa possono essere altre risorse, vediamo un'altro esempio:

The individual referred to by employee id 85740 is named Ora Lassila and has the email address lassila@w3.org. The resource http://www.w3.org/Home/Lassila was created by this individual.

in RDF/XML:




```
<rdf:RDF>
 <rdf:Description about="http://www.w3.org/Home/Lassila">
  <s:Creator rdf:resource="http://www.w3.org/staffId/85740"/>
 </rdf:Description>

 <rdf:Description about="http://www.w3.org/staffId/85740">
  <v:Name>Ora Lassila</v:Name>
  <v:Email>lassila@w3.org</v:Email>
 </rdf:Description>
</rdf:RDF>
```

per rendere più chiaro la relazione fra le due risorse, potremmo scrivere in maniera equivalente:

```
<rdf:RDF>
 <rdf:Description about="http://www.w3.org/Home/Lassila">
  <s:Creator>
   <rdf:Description about="http://www.w3.org/staffId/85740">
    <v:Name>Ora Lassila</v:Name>
    <v:Email>lassila@w3.org</v:Email>
   </rdf:Description>
  </s:Creator>
 </rdf:Description>
</rdf:RDF>
```

Quest’ultimo é un esempio di sintassi RDF abbreviata la quale permette di scrivere con un XML più compatto senza pregiudicare la compresione degli statement da parte di un processore RDF.

Contenitori

A volte ci si deve riferire ad un insieme di risorse, a tale scopo si possono usare i seguenti contenitori:

Bag insieme di risorse in cui l'ordine non é importante.

Sequence insieme di risorse in cui l'ordine é importante.

Alternative insieme di risorse che possono essere scelte come alternative.

Esempio: un documento con due autori indicati in ordine alfabetico, un titolo indicato in due lingue, e con due siti web dove poterlo trovare:

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/metadata/dublin_core#">
 <rdf:Description about="http://www.foo.com/cool.html">
  <dc:Creator>
   <rdf:Seq ID="CreatorsAlphabeticalBySurname">
    <rdf:li>Mary Andrew</rdf:li>
    <rdf:li>Jacky Crystal</rdf:li>
   </rdf:Seq>
  </dc:Creator>

  <dc:Identifier>
   <rdf:Bag ID="MirroredSites">
    <rdf:li rdf:resource="http://www.foo.com.au/cool.html"/>
    <rdf:li rdf:resource="http://www.foo.com.it/cool.html"/>
   </rdf:Bag>
  </dc:Identifier>

  <dc:Title>
   <rdf:Alt>
    <rdf:li xml:lang="en">The Coolest Web Page</rdf:li>
    <rdf:li xml:lang="it">Il Pagio di Web Fuba</rdf:li>
   </rdf:Alt>
  </dc:Title>
 </rdf:Description>
</rdf:RDF>
```

Statement su statement

In RDF uno statement é composto da:

1. Risorsa.
2. Proprietà.

3. Valore della proprietà.

queste tre parti sono chiamate rispettivamente: soggetto, predicato ed oggetto dello statement. Oltre a fare statement su risorse é possibile fare statement su statement, questa possibilità é detta *reification* ed é definita come parte del modello nella recommendation del W3C.

Vediamo un esempio per capirci:

Ralph Swick says that Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.

non diciamo nulla sulla risorsa *http://www.w3.org/Home/Lassila*;
invece esprimiamo un fatto sull'affermazione fatta da Ralph. Per esprimere questo in RDF, poichè gli statement sono fatti su risorse, dobbiamo creare una risorsa dello statement originale (Ora Lassila is the creator of
the resource http://www.w3.org/Home/Lassila) con 4 proprietà: subject, predicate, object e type, questo processo si chiama `reification`. Tale risorsa é il modello dello statement.

Vediamo come é possibile esprimere l'esempio precedente:

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:a="http://description.org/schema/">
 <rdf:Description>
  <rdf:subject resource="http://www.w3.org/Home/Lassila" />
  <rdf:predicate resource=
                "http://description.org/schema/Creator" />
  <rdf:object>Ora Lassila</rdf:object>
  <rdf:type resource="http://www.w3.org/1999/02/22
                     -rdf-syntax-ns#Statement" />
  <a:attributedTo>Ralph Swick</a:attributedTo>
 </rdf:Description>
</rdf:RDF>
```

La reification é usata per esplicitare i vari statement contenuti in una sezione Description:

```
<rdf:RDF>
 <rdf:Description
     about="http://www.w3.org/Home/Lassila" bagID="D_001">
  <s:Creator>Ora Lassila</s:Creator>
  <s:Title>Ora's Home Page</s:Title>
 </rdf:Description>
</rdf:RDF>
```

Aggiungere l'attributo `bagID` ad un blocco `Description` comporta includere nello schema un Bag con gli statement reificati.

```
<rdf:Description aboutEach="#D_001">
 <a:attributedTo>Ralph Swick</a:attributedTo>
</rdf:Description>
```

Il meccanismo della reification é stato introdotto nelle specifiche
di RDF(S) per aumentarne le capacità di descrizione, in quanto rende descrivibile in RDF
qualsiasi elemento definito in RDF.

3.1.2 RDF-Schema

La dichiarazione delle proprietà e la loro semantica sono definite in uno Schema RDF. Uno schema definisce anche i tipi di risorse descritte.

RDF Schema assomiglia ad un linguaggio di programmazione orientato agli oggetti in quanto ha il concetto di classe e proprietà. A differenza di un linguaggio OO, RDF Schema non definisce le classi e le proprietà che le istanze di tali classi hanno ma definisce le proprietà e le classi alle quali si possono applicare (*property oriented*). In questo modo si possono aggiungere nuove proprietà ad una classe senza doverla modificare.

Gerarchia delle classi di RDF Schema

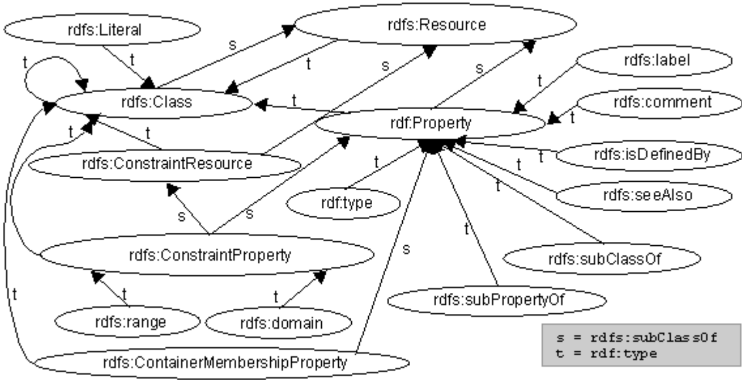


Figura 3.2: Gerarchia delle Classi di RDF Schema

Classi di RDF Schema

Quelle che seguono sono le descrizioni delle classi che costituiscono le fondamenta di RDF Schema.

rdfs:Resource Tutto quello che é descritto attraverso statement RDF é chiamato resource, istanza della classe rdfs:Resource.

rdf:Property rdf:Property rappresenta un sottoinsieme di risorse RDF che sono proprietà.

rdfs:Class rdfs:Class é un concetto simile a quello di classe di un qualsiasi linguaggio OO come Java. Le classi RDF possono essere definite per rappresentare quasi tutto: pagine web, persone, tipi di documento, database o concetti astratti.

rdfs:Literal Questa classe corrisponde all'insieme dei 'Literals' (valori atomici); le stringhe sono esempi di RDF literals.

Proprietà di RDF schema

Le proprietà sono usate per definire delle relazioni fra classi o con superclassi.

rdf:type Questa proprietà introduce una relazione di tipo *is-member of* in quanto indica l'appartenza di una risorsa ad una certa classe, quindi ha tutte le caratteristiche che sono previste per un membro di tale classe; la risorsa é un'istanza della classe. Il valore di rdf:type property deve essere una risorsa di tipo rdfs:Class. Una risorsa può essere istanza di più di una classe.

rdfs:subClassOf Questa proprietà introduce una relazione di inclusione tra classi (*is-a*). La proprietà rdfs:subClassOf é transitiva; se la classe A é sottoclasse della classe B e B é sottoclasse di C, allora A é implicitamente sottoclasse di C. Una classe può avere più di una superclasse. Una classe non può essere sottoclasse di se stessa nè di una sua sottoclasse.

rdfs:subPropertyOf La proprietà rdfs:subPropertyOf è un'istanza di rdf:Property ed indica che una proprietà é la specializzazione di un'altra. se una proprietà P2 é subPropertyOf di una più generale proprietà P1, e se una risorsa A ha la proprietà P2 con valore B, ciò implica che la risorsa A ha anche la proprietà P1 con valore B. Una proprietà non può essere subproperty di se stessa nè di una sua sotto-proprietà.

rdfs:seeAlso La proprietà rdfs:seeAlso indica una risorsa che può fornire ulteriori informazioni sulla risorsa in cui é presente.

rdfs:isDefinedBy La proprietà rdfs:isDefinedBy é una subproperty di rdfs:seeAlso, ed indica la risorsa che definisce la risorsa in cui é usata.

Vincoli

rdfs:ConstraintResource Questa é una sottoclasse di rdfs:Resource le cui istanze sono primitive di RDF Schema coinvolte nell'espressione di vincoli. Questa risorsa rappresenta un meccanismo per permettere l'introduzione di nuovi vincoli nelle future versioni di RDF senza perdere la compatibilità con le vecchie versioni.

rdfs:ConstraintProperty Questa risorsa é una sottoclasse di rdf:Property, le sue istanze sono proprietà che indicano dei vincoli. Questa classe é una sottoclasse di rdfs:ConstraintResource. rdfs:domain e rdfs:range sono istanze di rdfs:ConstraintProperty.

rdfs:range Un istanza di ConstraintProperty usata per indicare la/le classe/i a cui i valori della proprietà devono essere membri. Una proprietà può avere al massimo una proprietà range.

rdfs:domain Un istanza di ConstraintProperty che indica le classi che possono avere tale proprietà. Una proprietà può avere zero una o più domain. Zero domain significa che la proprietà può essere usata in qualsiasi risorsa; un solo domain che la proprietà può essere usata solo nelle istanze di una classe, più di un domain significa che la proprietà può essere usata in una qualunque delle classi indicate nei vari domain. Il range e il domain di rdfs:range e rdfs:domain sono delle istanze di rdfs:Class.

Documentazione

rdfs:comment Fornisce una descrizione di una risorsa.

rdfs:label Fornisce una versione comprensibile dalle persone del nome di una risorsa.

Esempi

Vediamo alcuni esempi:

```
<rdf:RDF xml:lang="en"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<!-- Note: this RDF schema would typically be used in RDF
     instance data by referencing it with an XML namespace
     declaration, for example
     xmlns:xyz="http://www.w3.org/2000/03/example/vehicles#".
     This allows us to use abbreviations such as
```

```
     xyz:MotorVehicle to refer unambiguously to the RDF class
     'MotorVehicle'.
-->
<rdf:Description ID="MotorVehicle">
 <rdf:type
   resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:subClassOf
  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Description>

<rdf:Description ID="PassengerVehicle">
 <rdf:type
      resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description ID="Truck">
 <rdf:type
      resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description ID="Van">
 <rdf:type
      resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description ID="MiniVan">
 <rdf:type
      resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:subClassOf rdf:resource="#Van"/>
 <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>
</rdf:RDF>
```

Questo esempio definisce una gerarchia. Abbiamo una risorsa MotorVehicle la quale ha tre risorse che derivano da essa: PassengerVehicle, Truck e Van. Minivan é una risorsa che deriva da Van e PassengerVehicle.

Le risorse sono definite attraverso il concetto predefinito di `Class`, le relazioni fra classi sono indicate attraversola proprietà predefinita `subClassOf`.

Proprietà

Per definire le proprietà delle risorse si usano i
concetti di `Property` e di `subPropertyOf`, vediamo un esempio:

```
<rdf:RDF xml:lang="en"
```

```
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description ID="biologicalParent">
 <rdf:type
  resource=
  "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdf:Description>

<rdf:Description ID="biologicalFather">
 <rdf:type
  resource=
  "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:subPropertyOf
  rdf:resource="#biologicalParent"/>
</rdf:Description>
</rdf:RDF>
```

abbiamo definito la proprietà:
`biologicalParent` e `biologicalFather` che rappresenta una specializzazione dibiologicalParent.

Vincoli sulle proprietà

Attraverso le proprietà `range` e `domain` si possono stabilire dei vincoli sulle proprietà, rispettivamente i valori che possono assumere e le classi a cui si possono applicare:

```
<rdf:Description ID="registeredTo">
 <rdf:type
  resource=
  "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:domain rdf:resource="#MotorVehicle"/>
 <rdfs:range rdf:resource="#Person"/>
</rdf:Description>
```

la proprietà `registeredTo` si applica alle risorse `MotorVehicle` e assume come valori risorse di tipo `Person`.

3.2 XML V_s RDF(S)

XML permette di definire i propri tag, i quali possono contenere sia testo che altri tag. Sembra che XML sia un ottimo veicolo per lo scambio di metadati; dove XML é carente é nel lato della *scalabilità* (vedi [33]):

- In un documento XML l'ordine in cui appaiono gli elementi é importante, questo però é altamente innaturale quando si tratta di metadati. Che differenza c'è se indichiamo prima la sorgente di un interfaccia o gli extent a cui appartiene ? quello che importa è che siano tutte e due disponibili. Inoltre garantire che tutti i documenti in cui sono presenti questi due metadati rispettino lo stesso ordine rende difficile la loro gestione.

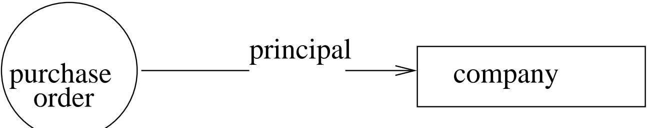
- XML permette una tale definizione:

```
<Description>The value of this property contains some
text, mixed up with child properties such as its temperature
(<Temp>48</Temp>) and longitude
(<Longt>101</Longt>). [&Disclaimer;]</Description>
```

  La rappresentazione nella memoria di un computer di tale sezione dà origine ad una struttura dati strana in cui sono presenti nodi, testo,...

- L'interoperabilità semantica, comporta la necessità di modellare il dominio d'interesse per dare un significato ai dati scambiati. Questo modello può essere definito in termini di oggetti e relazioni fra essi (es: UML) o come schema E-R. Dal modello dobbiamo poi passare ad un DTD o uno Schema XML-Schema. Poichè una DTD descrive semplicemente la struttura di un documento esistono molteplici possibilità per la codifica; in questo modo si perde la corrispondenza fra modello e DTD e non é più possibile ricostruire il modello del dominio d'interesse partendo dalla DTD (vedi [34]).

Vediamo un esempio:



Encoding DTD	Example XML Instance Data
<!ELEMENTPurchaseOrder(principal)> <!ATTLISTPurchaseOrder id ID #REQUIRED> <!ELEMENT principal(Company)> <!ATTLIST Company id ID#IMPLIED>	<PurchaseOrderid=”X”> <principal> <Companyid=″Y″/> </principal> </PurchaseOrder>
<!ELEMENT principal(PurchaseOrder, Company)> <!ELEMENT PurchaseOrder(#CDATA)> <!ELEMENT Company(#CDATA)>	<principal> <PurchaseOrder>X </PurchaseOrder> <Company>Y</Company> </principal>
<!ELEMENT PurchaseOrder(id,principal)> <!ELEMENT id(#CDATA)> <!ELEMENT principal(Company)> <!ELEMENT Company(id)>	<PurchaseOrder> <id>X</id> <principal> <Company> <id>Y</id> </Company> </principal> </PurchaseOrder>
<!ELEMENT relEMPTY> <!ATTLIST rel srcCDATA #REQUIRED typeCDATA #REQUIRED destCDATA #REQUIRED>	<rel src=”X” type=”principal” dest=”Y”/>
<!ELEMENT PurchaseOrderInfo(Company)> <!ATTLIST PurchaseOrderInfo orderIDID #REQUIRED> <!ELEMENT Company(#CDATA)>	<PurchaseOrderInfo orderID=”X”> <Company>Y</Company> </PurchaseOrderInfo>

Comunque XML é ineguagliabile come formato di scambio sul web per cui il suo utilizzo congiunto a RDF per l’interscambio dei metadati é necessario.

3.3 Approccio Ontologico

Le ontologie rappresentano una possibilità per raggiungere l’obiettivo di un Web più “machine-friendly”; attraverso un’ontologia é possibile condividere conoscen-

za attraverso l'interpretazione comune di un dominio che può essere comunicato tra persone e applicazioni.

Le Ontologie rivestono un ruolo importante nell'ambito del web semantico per diverse ragioni:

1. Lo sviluppo di un'ontologia non é un problema semplice. Condividere un'ontologia già esistente velocizza il processo di sviluppo di database che rappresentano lo stesso tipo di conoscenza (es: Una Università).

2. Se diversi database rappresentano la stessa conoscenza la condivisione di un'ontologia semplifica il processo di integrazione dei database.

3. Gli sviluppatori di database con un'ontologia sono in grado di comunicare in maniera più leggibile la semantica del loro database.

4. La condivisione di un'ontologia é importante perchè le stesse ontologie sono una forma di conoscenza.

3.3.1 OIL (*Ontology Inference Layer*)

OIL [35] é un linguaggio per la definizione di ontologie, che combina le più usate primitive di modellazione di linguaggi frame-based con le capacità offerte dalle logiche descrittive. OIL è inoltre basato sugli standard del w3c quali RDF/RDF-schema e XML/XML-schema.

Il progetto di OIL é finanziato dal programma dell'Unione Europea IST (Information Society Technologies) sotto il prgetto On-To-Knowledge[2].

Il progetto On-To-Knowledge ha lo scopo di sviluppare strumenti per poter sfruttare completamente le potenzialità dell'*approccio ontologico* al **Web Semantico**. Gli strumenti sviluppati nell'ambito del progetto On-To-Knowledge dovranno fornire un supporto per accedere alle informazioni messe a disposizione in maniera intuitiva. Poichè il progetto On-To-Knowledge si basa sull'approccio ontologico fa uso di ontologie per quanto riguarda i processi di integrazione e condivisione delle informazioni. OIL rappresenta uno dei principali strumenti sviluppati da tale progetto.

Un'ontologia in OIL é composta da due sezioni: una sezione in cui sono raccolte una serie di informazioni con scopi di documentazione: autore, linguaggio di scrittura dell'ontologia ecc..; la seconda sezione contiene la definizione dei concetti e delle proprietà che formano l'ontologia.

[2]http://www.ontoknowledge.org/

Primitive di Modellazione

Un ontologia scritta in OIL consiste in un insieme di definizioni:

import ? Una lista di riferimenti ad altre ontologie OIL da includere nell'ontologia, concetto simile a quello dei namespace di XML.

rule-base ? Una lista assiomi detti anche vincoli globali. al momento sono una semplice stringa senza una semantica precisa.

class and slot definitions Zero o più definizioni di classi (class-def) e di slot (slot-def).

Una definizione di classe (class-def) associa il nome di una classe con la sua descrizione. Una class-def consiste delle seguenti componenti:

type ? Il tipo di definizione. Questa può essere **primitive** o **defined**; il default é primitive. Quando una classe é primitive, la sua definizione é condizione necessaria ma non sufficente per l'appartenenza ad una classe (concetto simile alla distinzione fra tipi virtuali e primitivi presente in OLCD).Ad esempio la classe:

```
class-def elephant
 subclass-of animal
 slot-constraint colour
 has-filler "grey"
```

dice che tutti gli elefanti sono una specie di animale che ha la pelle di colore grigio ma, essendo **primitive**, non tutti gli animali con la pelle grigia sono elefanti.

name Il nome della classe (una stringa).

documentation Documentazione sulla classe (una stringa).

subclass-of ? Una lista di una o più **class-expressions**. La classe definita con questo **class-def** deve essere sottoclasse di ogni class-expressions nella lista.

slot-constraints Zero o più slot-constraint. La classe definita deve essere sotto classe di ogni slot-constraint.

Una **class-expression** può essere sia un nome di classe,
uno slot-constraint, o una combinazione di class expression collegate con gli operatori AND, OR o NOT.

Uno slot-constraint é una lista di uno o più vincoli applicati ad uno slot. Uno slot é una relazione binaria (le sue istanze sono coppie di oggetti), uno slot-constraint é a tutti gli effetti la definizione di una classe anonima le cui istanze sono tutte quelle che soddisfano i vincoli. Uno slot-constraint consiste delle seguenti componenti:

name Il nome dello slot (una stringa).

has-value ? Una lista di uno o più class-expression. Definisce la classe degli oggetti X per i quali esiste almeno un'istanza Y delle class-expression t.c (X,Y) é un'istanza dello slot. Questo non esclude che esistano altre istanze (X, Y') dello slot con Y' non appartenente alla class-expression. has-value esprime il quantificatore esistenziale (exists in OLCD).

value-type ? Una lista di uno o più class-expression. Definisce la classe di oggetti X per i quali se la coppia (X,Y) é un'istanza dello slot, allora Y é un'istanza della class-expression. value-type esprime il quantificatore universale (forall in OLCD).

has-filler ? Una lista di una o più istanze o valori. Definisce la classe di oggetti X per che hanno un legame con le istanze o i valori indicati tramite lo slot.

max-cardinality ? Un numero intero positivo seguito da una class-expression. Possono esistere al massimo n istanze distinte (x,y) dello slot in cui x é un'istanza della classe e y é un istanza della class-expression.

min-cardinality ? Un numero intero positivo seguito da una class-expression. Devono esistere almeno n istanze distinte (x,y) dello slot in cui x é un'istanza della classe e y é un'istanza della class-expression.

cardinality ? come min-cardinality e max-cardinality quando quest'ultimi vincolano allo stesso numero di istanze.

Mentre uno slot-constraint specifica dei vincoli locali, la definizione di uno slot (slot-def) specifica vincoli globali. Uno slot-def consiste delle seguenti componenti:

name il nome dello slot (una stringa).

documentation ? Documentazione associata allo slot (una stringa).

subslot-of ? Una lista di uno o più slot. Per esempio `slot-def daughter subslot-of child` definisce uno slot daughter che é subslot di child, ogni coppia (x,y) istanza dello slot daughter é anche istanza di child.

domain ? Una lista di uno o più class-expression. Se la coppia (x,y) é un instanza dello slot, allora x deve essere un'istanza di ogni class-expression della lista.

range ? Una lista di class-expression. Se la coppia (x,y) é un'istanza dello slot allora y é un'istanza di ogni class-expression della lista.

inverse ? Il nome di uno slot S inverso dello slot. Se (x,y) é un'istanza dello slot S, allora (y,x) deve essere un'istanza dello slot che si stà definendo. (eats inverse eaten-by).

properties ? Una lista di una o più proprietá dello slot. Proprietà valide sono: transitive e symmetric.

In OIL é possibile definire 4 tipi di assiomi:

disjoint lista di class-expression a due a due disgiunte.

covered una class-expression é data dall'unione di due o più class-expression.

disjoint-covered una class-expression é data dall'unione di due o più class-expression a due a due disgiunte.

equivalent lista di class-expression tra loro equivalenti.

SHIQ

La logica descrittiva SHIQ (**SH**F augmented with **i**nverse roles and **q**ualified number restrictions), é quella che dà il supporto a "ragionare" con la conoscenza espressa in OIL,(é allo studio un'estensione di tale logica per supportare i tipi di dato che verrà chiamata SHIQ[d]). Attualmente OIL fà uso del sistema FaCT (Fast Classification of Terminologies) in grado di comprendere i concetti espressi in SHIQ (non ancora di SHIQ[d]). Il ruolo svolto da FaCT per un'ontologia OIL é analogo a quello svolto da ODB-Tools nel progetto Momis; in quest'ottica possiamo fare le seguenti affermazioni:

$$FaCt : SHIQ = ODB - Tools : OLCD$$
$$OIL : SHIQ = ODL_{I^3} : OLCD$$

Class-def

La definizione di una classe é una coppia (**CN**,D) o una tripla (**CN**,P,D) dove

- **CN** é il nome della classe.
- `P` può assumere i valori `primitive` o `defined`.
- `D` é la definizione della classe.

(**CN**,D) é equivalente a (**CN**,**primitive**,D) e in SHIQ si esprime come **CN** $\sqsubseteq$ D (la classe **CN** é una sottoclasse della classe descritta da D).

(**CN**,**defined**,D) in SHIQ si esprime come: **CN** $\doteq$ D (la classe **CN** é equivalente alla classe descritta da D).

La descrizione di una classe D può contenere la definizione di **subclass-of** con una lista di **class-expressions** $C_1, ...C_n$, seguita da una lista di zero o più **slot-constraints** $A_1, ...A_m$.

class-expression

Una **class-expression** può essere:

- il nome di una classe **CN**.
- un elenco di istanze.
- uno **slot-constraint**
- una combinazione di **class expressions**: una congiunzione $C_1 \sqcup .... \sqcup C_n$, una disgiunzione $C_1 \sqcap .... \sqcap C_n$ o la negazione $\neg C$. Esistono claasi particolari:s **top, thing** e **bottom**; le prime due rappresentano la classe generica $\top$ mentre l'ultima é la classe inconsistente $\bot$. Un elenco di istanze é introdotto dalla parola chiave **one-of** $i_1, ...i_n$.

slot constraint

Uno **slot-constraint** consiste di un nome **SN** seguito da uno o più vincoli che si applicano allo slot: **SN**$[a_1, ..., a_n]$. I vincoli e le lor definizioni in SHIQ sono:

has-value $\exists C_1, ...C_n$.

value-type $\forall C_1, ...C_n$.

has-filler $\exists C_1, ...C_n$. .

max-cardinality n C $\leq n.C$.

min-cardinality n C $\geq n.C.$

cardinality n C $= n.C.$

concrete-type expression

Gli slot constraint possono avere come argomenti non solo class-expression ma anche espressioni che riguardano valori numerici o stringhe, queste non sono ancora esprimibili in SHIQ anche se lo saranno in SHIQ[d]:

- uno dei predicati **min** d ($\geq_d$), **max** d ($\leq_d$), **greater-than** d ($> d$), **less-than** d ($< d$), **range** d1 d2 ($\geq_{d_1} \sqcap \leq_{d_2}$) e **equal** d ($\geq_d \sqcap \leq_d$).
- Una combinazione di altre concrete-type expression: congiunzione $C_1 \sqcup .... \sqcup C_n$, disgiunzione $C_1 \sqcap .... \sqcap C_n$ o una negazione $\neg C$.

Qui si vede la differenza ditra SHIQ e OLCD il quale ha come caratteristica principale una ricca struttura per il sistema dei tipi di base: oltre ai classici tipi atomici integer, boolean, string, real, e tipi monovalore, viene considerata anche la possibilità di utilizzare dei sottoinsiemi di questi (come potrebbero essere, ad esempio, intervalli di interi).

Tale differenza riflette la diversa conoscenza che OIL e ODL_{I^3} rappresentano quest'ultimo infatti é utilizzato nel contesto dell'integrazione di sorgenti dati eterogennee caratterizzate generalmente da una forte tipizzazione delle informazioni.

slot-def

La definizione di uno slot é una coppia (**SN**,X) dove **SN** é il nome dello slot e X é la sua descrizione.

La descrizione X può avere un componente **subslot-of** ed essere seguito dalla definizione di uno o più vincoli:

domain $C_1, ...C_n$ $\downarrow |C_1, ...C_n|$

range $C_1, ...C_n$ $\uparrow |C_1, ...C_n|$

inverse RN ^{-}RN

properties $P_1, ...P_n$ elenco di proprietà: **transitive** $+$, **symmetrical** $\leftrightarrow$ e **functional** $\uparrow$

Axioms

OIL ha 4 tipi di assiomi che SHIQ esprime nel modo seguente:

disjoint $\|C_1, ...C_n\|$

covered $C \sqsubseteq (C_1, ...C_n)$

disjoint-covered $C \sqsubseteq \|C_1, ...C_n\|$

equivalent $\doteq (C_1, ...C_n)$

La funzioni di mapping σ

$\sigma(.)$ é la funzione che mappa i concetti di un'ontologia OIL in concetti della logica descrittiva SHIQ; vediamo in figura 3.3 la corrispondenza fra OIL e shiq:

3.3.2 Aumentare l'espressività di RDF

Il gruppo di lavoro su OIL ha proposto un metodo per aumentare le capacità di RDF di comunicare conoscenza articolato in tre passi [34, paragraph 6]:

1. Tradurre le primitive di modellazione di OIL (class-def, slot-constraint,...) in elementi di RDF-Schema.

2. Usare lo schema del punto precedente per definire un altro documento RDF-Schema che descriva una specifica ontologia in OIL.

3. Usare i due schemi prodotti precedentemente per descrivere istanze dell'ontologia in un documento RDF.

Questa metodologia per aumentare le capacità espressive di RDF(S) é applicabile ad ogni linguaggio di rappresentazione di conoscenza, tenendo conto che attualmente non é possibile una traduzione che usi solamente le primitive di RDF(S) il risultato che si otterrà sarà un'ontologia *parzialmente* comprensibile da applicazioni RDF-aware che non possono sfruttare l'eventuale supporto fornito dalla logica descrittiva sui cui si basa il linguaggio.

3.3.3 DAML+OIL

OIL é stato usato come base per lo sviluppo di DAML+OIL, un linguaggio di markup semantico per risorse web sviluppato da DARPA (DAML é un acronimo che stà per DARPA Agent Markup Language). DAML é basato sui più recenti

OIL	SHIQ
class-def (**primitive** | **defined**) CN	CN ($\sqsubseteq$ | $\doteq$) $\top$
subclass-of $C_1 \ldots C_n$	$\sqcap\, \sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n)$
slot-constraint$_1$	$\sqcap\, \sigma($slot-constraint$_1)$
$\vdots$	$\vdots$
slot-constraint$_m$	$\sqcap\, \sigma($slot-constraint$_m)$
top | **thing** | **bottom**	$C \sqcup \neg C \mid C \sqcup \neg C \mid C \sqcap \neg C$
(**min** d) | (**max** d)	$\geqslant_d \mid \leqslant_d$
(**greater-than** d) | (**less-than** d)	$>_d \mid <_d$
(**equal** d) | (**range** d_1 d_2)	$(\geqslant_d \sqcap \leqslant_d) \mid (\geqslant_{d_1} \sqcap \leqslant_{d_2})$
(C_1 **and** ... **and** C_n)	$(\sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n))$
(C_1 **or** ... **or** C_n)	$(\sigma(C_1) \sqcup \ldots \sqcup \sigma(C_n))$
(**not** C)	$(\neg \sigma(C))$
(**one-of** $i_1 \ldots i_n$)	$(P_{i_1} \sqcup \ldots \sqcup P_{i_n})$
slot-constraint *SN*	$\top$
has-value $C_1 \ldots C_n$	$\sqcap\, \exists SN.\sigma(C_1) \sqcap \ldots \sqcap \exists SN.\sigma(C_n)$
value-type $C_1 \ldots C_n$	$\sqcap\, \forall SN.\sigma(C_1) \sqcap \ldots \sqcap \forall SN.\sigma(C_n)$
max-cardinality n C	$\sqcap \leqslant n SN.\sigma(C)$
min-cardinality n C	$\sqcap \geqslant n SN.\sigma(C)$
cardinality n C	$\sqcap \geqslant n SN.\sigma(C) \sqcap \leqslant n SN.\sigma(C)$
has-filler d	$\sqcap\, \exists SN.\sigma(d)$
slot-def *SN*	
subslot-of $SN_1 \ldots SN_n$	$(SN \sqsubseteq SN_1) \ldots (SN \sqsubseteq SN_n)$
domain $C_1 \ldots C_n$	$\exists SN.\top \sqsubseteq \sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n)$
range $C_1 \ldots C_n$	$\top \sqsubseteq \forall SN.\sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n)$
inverse *RN*	$(SN^- \sqsubseteq RN)\ (RN^- \sqsubseteq SN)$
properties transitive	$SN \in \mathbf{S}_+$
properties symmetric	$(SN \sqsubseteq SN^-)\ (SN^- \sqsubseteq SN)$
properties functional	$\top \sqsubseteq \leqslant 1 SN$
disjoint $C_1\ C_2 \ldots C_n$	$(\sigma(C_1) \sqsubseteq \neg\sigma(C_2)) \ldots (\sigma(C_{n-1}) \sqsubseteq \neg$
covered C **by** $C_1 \ldots C_n$	$\sigma(C) \sqsubseteq \sigma(C_1) \sqcup \ldots \sqcup \sigma(C_n)$
disjoint-covered C **by** $C_1 \ldots C_n$	$(\sigma(C_1) \sqsubseteq \neg\sigma(C_2)) \ldots (\sigma(C_{n-1}) \sqsubseteq \neg$
	$(\sigma(C) \sqsubseteq \sigma(C_1) \sqcup \ldots \sqcup \sigma(C_n))$
equivalent $C\ C_1 \ldots C_n$	$(\sigma(C) \doteq \sigma(C_1)) \ldots (\sigma(C_{n-1}) \doteq \sigma(C$
instance-of $i\ C_1 \ldots C_n$	$P_i \sqsubseteq \sigma(C_1) \sqcap \ldots \sqcap \sigma(C_n)$
related *SN* $i\ j$	$P_i \sqsubseteq \exists SN.P_j$

Figura 3.3: corrispondenza OIL-SHIQ

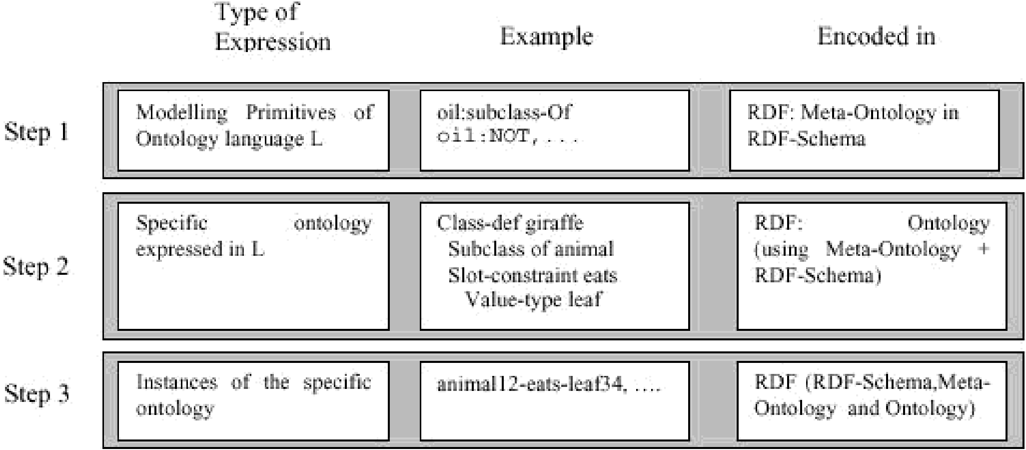


Figura 3.4: Estendere RDF(S) tratto da [34]

OIL	**RDF SCHEMA**
Class-def	rdfs:Class
Subclass-of	rdfs:subClassOf (single parent class) oil:subclass-of (parent is class expression)
Slot constraint	Slot constraints are subclass expressions in RDF-Oil
AND, ","	oil:AND
NOT	oil:NOT
Has-value	oil:has-value

Tabella 3.1: Corrispondenza OIL-RDF tratto da [34]

standard quali RDF e RDF Schema estendendoli con primitive di modellazione derivate da linguaggi frame-based. DAML+OIL (Marzo 2001) [36] estende DAML+OIL (Dicembre 2000) con i tipi di dato da XML Schema.

Una base di conoscenza in DAML+OIL é un insieme di triple RDF.

Oggetti e Tipi di Dato

L'elemento chiave di DAML+OIL é il supporto per i tipi di dato XML-Schema, supporto non presente in RFD(S), la soluzione
avanzata in DAML+OIL vuole essere un'idea per il W3C per includerlo nelle specifiche di RDF(S).

Vediamo un esempio:

```
<daml:DatatypeProperty rdf:ID="age">
  <rdfs:comment>
    age is a DatatypeProperty whose range is xsd:decimal.
    age is also a UniqueProperty (can only have one age)
  </rdfs:comment>
  <rdf:type
   rdf:resource="http://www.daml.org/2001/03/daml+oil
                 #UniqueProperty"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2000/10/XMLSchema
                 #nonNegativeInteger"/>
</daml:DatatypeProperty>
```

Un'ontologia DAML+OIL ha un'intestazione che contiene alcune informazioni sull'ontologia, vediamo un'esempio:

```
<Ontology rdf:about="">
<versionInfo>
$Id: reference.html,v 1.10 2001/04/11 16:27:53 mdean Exp $
</versionInfo>
<rdfs:comment>An example ontology</rdfs:comment>
<imports rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
</Ontology>
```

Dopo l'intestazione iniziano le definizioni di classi e proprietà:

Definizione di Classi

La definizione di una classe é introdotta dall'elemento `daml:Class` e contiene le seguenti componenti:

rdfs:subClassOf* definisce la classe C sottoclasse della class-expression definita, a differenza delle specifiche di RDF(S) DAML+OIL non vieta la ciclicità che può derivare da tale relazione.

daml:disjointWith* definisce che la classe C e la class-expression definita non hanno istanze comuni.

daml:disjointUnionOf* Tutte le classi definite dalle class-expressions elencate devono essere a due a due disgiunte e la loro unione deve essere uguale a C.

daml:sameClassAs* C e la class-expression definita hanno le stesse istanze.

Combinazioni booleane di class-expressions La classe C deve essere uguale alla classe derivante dalla combinazione di due o più class-expressions.

Elenco di elementi? La classe C contiene esattamente le istanze elencate.

Espressioni di Classe

Una class-expression può indicare:

- Il nome di una classe.
- Un elenco di istanze.
- Una restrizione di una proprietà.
- Una combinazione booleana di class-expression.

Elenco di Istanze

L'elemento `daml:oneOf` permette di definire una classe elencando le istanze che ne fanno parte:

```
<daml:oneOf parseType="daml:collection">
 <daml:Thing rdf:about="#Eurasia"/>
 <daml:Thing rdf:about="#Africa"/>
 <daml:Thing rdf:about="#North_America"/>
 <daml:Thing rdf:about="#South_America "/>
 <daml:Thing rdf:about="#Australia"/>
 <daml:Thing rdf:about="#Antarctica"/>
</oneOf>
```

Vincoli sulle Proprietà

Una restrizione di una proprietà definisce implicitamente una classe anonima (la classe delle istanze che soddisfano i vincoli). I vincoli possono riguardare proprietà che mettono in relazione due classi o proprietà che mettono in relazione una classe e un tipo di dato XML-Schema.

Un vincolo é introdotto dall'elemento `daml:Restriction` é seguito da un elemento `daml:onProperty`, che indica a quale proprietà va applicato il vincolo, e da una o più dei seguenti elementi:

daml:toClass $< class - expression >$ Definisce la classe degli oggetti x per i quali se (x,y) é un istanza della proprietá, allora y é un'istanza della class-expression indicata da daml:toClass. daml:toClass é euivalente al quantificatore universale dei predicati logici.

daml:hasValue $< istanza\ o\ valore\ di\ un\ tipo\ di\ dato >$ Se y é l'istanza, definisce la classe degli oggetti x per i quali (x,y) é un'istanza della proprietà.

daml:hasClass $< class - expression >$ Definisce la classe degli oggetti x per i quali esiste almeno un'istanza y della class-expression tale che (x,y) é un'istanza della proprietà.Questo non esclude che possoa esistere un'istanza $(x, y^{'})$ della proprietà con $y^{'}$ non appartenente
alla class-expression. daml:hasClass é equivalente al quantificatore esistenziale dei predicati logici.

daml:cardinality $< intero\ non\ negativo >$ ***N*** Definisce la classe degli oggetti che hanno esattamente N distinti valori per la proprietà a cui si applica la restrizione.

daml:maxCardinality $< intero\ non\ negativo >$ ***N*** Definisce la classe degli oggetti che hanno al massimo N distinti valori per la proprietà a cui si applica la restrizione.

daml:minCardinality $< intero\ non\ negativo >$ ***N*** Definisce la classe degli oggetti che hanno almeno N distinti valori per la proprietà a cui si applica la restrizione.

daml:cardinalityQ $< intero\ non\ negativo >$ ***N*** Definisce la classe degli oggetti che hanno esattamente N distinti valori per la proprietà a cui si applica la restrizione e tali valori sono istanze della classe o del tipo di dato indicato nell'elemento `daml:hasclassQ`.

daml:maxCardinalityQ $< intero\ non\ negativo >$ ***N*** Definisce la classe degli oggetti che hanno al massimo N distinti valori per la proprietà a cui si applica la restrizione e tali valori sono istanze della classe o del tipo di dato indicato nell'elemento `daml:hasclassQ`..

daml:minCardinalityQ $< intero\ non\ negativo >$ ***N*** Definisce la classe degli oggetti che hanno almeno N distinti valori per la proprietà a cui si

applica la restrizione e tali valori sono istanze della classe o del tipo di dato indicato nell'elemento `daml:hasclassQ`.

Combinazione di Classi

Una combinazione di classi può essere realizzata attraverso uno dei seguenti elementi:

daml:intersectionOf $< lista\ di\ class - expression >$ Definisce la classe degli oggetti in comune alle class-expression della lista (congiunzione logica).

daml:unionOf $< lista\ di\ class - expression >$ Definisce la classe i cui membri sono l'unione degli oggetti appartenenti alle class-expression della lista (disgiunzione logica).

daml:complementOf $< class - expression >$ Definisce la classe degli oggetti che NON appartengono alla class-expression (negazione logica limitata ai soli oggetti).

Definizione di Proprietà

Le proprietà definiscono delle relazioni binarie fra due entità, se quest'ultime sono due classi allora si parla di `ObjectProperty` altrimenti se la relazione é fra una classe ed un tipo di dato XML-Schema si parla di `DatatypeProperty`.

Una proprietà P contiene i seguenti elementi:

rdfs:subPropertyOf $< nome\ di\ una\ propriet\grave{a} >$ Se (x,y) é un'istanza di P allora lo é anche della proprietà indicata da subPropertyOf.

rdfs:domain $< class - expression >$ Se (x,y) é un'istanza di P allora x é un'istanza della class-expression. La presenza di più di un rdfs:domain definisce il dominio di P come l'intersezione di ogni class-expression; diversamente dalle specifiche di RDF(S).

rdfs:range $< class - expression >$ Se (x,y) é un'istanza di P allora y é un'istanza della class-expression. La presenza di più di un rdfs:range definisce il range di P come l'intersezione di ogni class-expression. Anche in questo caso si é andato contro le specifiche di RDF(S) che permettono l'indicazione di un solo range.

daml:samePropertyAs $< nome\ di\ una\ propriet\grave{a} >$ P é equivalente alla proprietà indicata.

daml:inverseOf $< nome\ di\ una\ propriet\grave{a} >$ se (x,y) é istanza di P allora (y,x) é istanza della proprietà indicata.

Le proprietá possono essere introdotte anche usando i seguenti elementi:

daml:TransitiveProperty Definisce che la proprietà P é transitiva; se (x,y) $\in$ P e (y,z) $\in$ P allora (x,z) $\in$ P.

daml:UniqueProperty Se (x,y_1) $\in$ P e (x,y_2) $\in$ P allora $y_1 = y_2$.

daml:UnambiguosProperty Se (x_1,y) $\in$ P e (x_2,y) $\in$ P allora $x_1 = x_2$.

da notare che daml:UniqueProperty e daml:UnambiguosPropert definiscono vincoli globali per una proprietà indipendenti dalla classe a cui la proprietà é applicata.

Istanze

Le istanze (oggetti) sono scritte con la sintassi usata per RDF(S), vediamo degli esempi:

```
<continent rdf:ID="Asia"/>

<rdf:Description rdf:ID="Asia">
 <rdf:type>
  <rdfs:Class rdf:about="#continent"/>
 </rdf:type>
</rdf:Description>
```

Tipi di dato

Per permettere la corretta interpretazione dei dati che hanno un tipo di dato XML-Schema si ricorre ad un particolare metodo di scrittura dei valori che indica oltre al valore del dato anche il suo tipo; ad esempio per scrivere il valore 10.5 dovremmo scrivere

```
<xsd:decimal rdf:value=''10.5''>
```

DAML+OIL in KIF

Poiché DAML+OIL ha uno stretto legame con OIL, anche DAML+OIL puó usufruire dei servizi di FaCT per fornire il supporto a *”ragionare”* con i concetti espressi in un’ontologia. Per aumentare il numero delle applicazioni in grado di ”ragionare” su tali concetti é stato definito un mapping tra concetti DAML+OIL e predicati in logica del primo ordine usando KIF. In modo che un ”reasoner” che elabora i seguenti 2 statement:

```
?Class Male and class Female are disjointWith.?

?John is type Male.?
```

possa ”capire” che l’affermazione `?John is type Female.?` é falsa.

KIF (Knowledge Interchange Format) é una proposta dell’ANS(American National Standard) che ha l’intento di definire un linguaggio col quale comunicare conoscenza; un elaboratore che legga una base di conoscenza espressa in KIF la potrà poi tradurre nella propria logica descrittiva.KIF vuole essere quello che Postscript é per le stampanti; un linguaggio leggibile da un’applicazione per facilitare lo sviluppo di programmi di manipolazione di conoscenza.

KIF definisce:

- operatori logici
- quantificatori (forall,exists)
- operazioni sui valori numerici
- definizione di funzioni
- ecc..

Poichè ogni parola chiave DAML+OIL è basata su RDF per ogni statement é possibile definire:

- una proprietà P.
- un soggetto S la risorsa a cui si applica la proprietà.
- un oggetto O il valore della proprietà.

in notazione KIF : `?(PropertyValue P S O)?`.

Qui di seguito sono riportate alcune delle assiomatizzazioni più significative dei concetti DAML+OIL; una lista completa si può trovare al sito del progetto DAML[3]

TransitiveProperty

?TransitiveProperty? é una ?Class?.

```
Ax60.    (Type TransitiveProperty Class)
```

Se Y é il valore della proprietà P di X e Z é il valore di P per Y allora Z é il valore di P per X.

[3] www.daml.org

```
Ax61.    (<=> (Type ?p TransitiveProperty)
                (forall (?x ?y ?z)
                        (=> (and (PropertyValue ?p ?x ?y)
                                 (PropertyValue ?p ?y ?z))
                            (PropertyValue ?p ?x ?z))))
```

UniqueProperty

?UniqueProperty? é una ?Class?.

```
Ax62.    (Type UniqueProperty Class)
```

P è una ?UniqueProperty? se e solo se

1. P é una ?Property?
2. se ∃ Z valore di P per X allora Y = Z.

```
Ax63.    (<=> (Type ?p UniqueProperty)
           (and (Type ?p Property)
                (forall (?x ?y ?z)
                        (=> (and (PropertyValue ?p ?x ?y)
                                 (PropertyValue ?p ?x ?z))
                            (= ?y ?z)))))
```

UnambiguousProperty

?UnambiguousProperty? é una ?Class?.

```
Ax64.    (Type UnambiguousProperty Class)
```

P é una ?UnambiguousProperty? se e solo se

1. V é il valore di P per X
2. se ∃ Y con valore V per P allora X = Y.

```
Ax65.    (<=> (Type ?p UnambiguousProperty)
           (and (Type ?p AbstractProperty)
                (forall (?x ?y ?v)
                        (=> (and (PropertyValue ?p ?x ?v)
                                 (PropertyValue ?p ?y ?v))
                            (= ?x ?y)))))
```

equivalentTo

?equivalentTo? é una ?Property?.

```
Ax68.    (Type equivalentTo Property)
```

Y è il valore di ?equivalentTo? per X se e solo se X = Y.

```
Ax69.    (<=> (PropertyValue equivalentTo ?x ?y) (= ?x ?y))
```

disjointWith

?disjointWith? è una ?Property?.

```
Ax76.    (Type disjointWith Property)
```

C2 è il valore di ?disjointWith? per C1 se e solo se C1 è una ?Class?, C2 è una ?Class?, e nessuna istanza X è di tipo C1 e C2.

```
Ax77.    (<=> (PropertyValue disjointWith ?c1 ?c2)
           (and (Type ?c1 Class)
                (Type ?c2 Class)
                (not (exists (?x)
                 (and (Type ?x ?c1) (Type ?x ?c2))))))
```

complementOf

?complementOf? è una ?Property?.

```
Ax84.    (Type complementOf Property)
```

C2 è il valore di ?complementOf? per C1 se e solo se C1 e C2 sono classe disgiunte e tutte gli oggetti sono o di tipo C1 o C2.

```
Ax85.    (<=> (PropertyValue complementOf ?c1 ?c2)
           (and (PropertyValue disjointWith ?c1 ?c2)
                (forall (?x) (or (Type ?x ?c1) (Type ?x ?c2)))))
```

toClass

La restrizione ?toClass? che ha come valore la classe C sulla proprietà P definisce la classe di tutti gli oggetti I t.c. tutti i valori di P per I sono di tipo C.

```
Ax94.    (=> (and (PropertyValue onProperty ?r ?p)
              (PropertyValue toClass ?r ?c))
          (forall (?i) (<=> (Type ?i ?r)
                            (forall (?j)
                             (=> (PropertyValue ?p ?i ?j)
                              (Type ?j ?c))))))
```

hasClass

La restrizione ?hasClass? che ha come valore la classe C sulla proprietà P é la classe di tutti gli oggetti I che hanno almeno un valore di P che é di tipo C.

```
Ax101.(=> (and (PropertyValue onProperty ?r ?p)
                    (PropertyValue hasClass ?r ?c))
                (forall (?i)
                 (<=> (Type ?i ?r)
                  (exists (?j)
                   (and (PropertyValue ?p ?i ?j)
                    (Type ?j ?c))))))
```

3.3.4 Differenze fra DAML+OIL e OIL

DAML+OIL come suggerisce il nome prende molte delle caratteristiche di OIL, per cui le capacità dei due linguaggi sono molto simili.

Le differenze più significative sono:

1. OIL ha una maggior compatibilità con RDF(S) rispetto a DAML+OIL.

2. OIL permette l'indicazione di condizioni necessarie e sufficienti in maniera più semplice che in DAML+OIL.

3.4 ODL_{I^3}

Il nostro livello logico, quello che fornisce lo strumento per definire la semantica e dare un supporto per "ragionare" con i dati é dato dall'uso congiunto di ODL_{I^3} e OLCD.

Per una ricca rappresentazione semantica degli schemi sorgenti e degli object pattern associati con le sorgenti di informazione da integrare, si è introdotto un linguaggio object-oriented, chiamato ODL_{I^3}. Seguendo le raccomandazioni ODMG e I^3 , il modello ad oggetti ODL_{I^3} é molto simile al linguaggio ODL. ODL_{I^3} è un linguaggio indipendente dalla sorgente per l'estrazione di informazioni, utilizzato per descrivere schemi eterogenei di dati più o meno strutturati in un formato comune.

Rispetto ad ODL, ODL_{I^3} introduce le seguenti estensioni:

Union: il costrutto di unione denotato con union, è stato introdotto per esprimere strutture dati alternative nella definizione
di una classe ODL_{I^3} , in modo da poter catturare le caratteristiche dei dati semistrutturati.

Attributi opzionali: sono denotati da (*) e sono stati introdotti per indicare che un determinato attributo è opzionale per un istanza (cioè potrebbe non venire specificato nell istanza). Anche questo costrutto è stato introdotto per catturare altri requisiti dei dati semistrutturati.

Regole di integrità: Questo tipo di regola è stato introdotto in ODL_{I^3} perché si potesse esprimere, in maniera dichiarativa, vincoli di integrità di tipo if-then sia all'interno della stessa sorgente, sia tra sorgenti differenti.

Relazioni intensionali: sono relazioni terminologiche che esprimono la conoscenza per lo schema sorgente. Le relazioni intensionali sono definite tra classi e attributi e sono specificate utilizzando i nomi classe/attributo,

denominati termini. In ODL_{I^3} possono essere specificate le seguenti relazioni:

- Sinonimia (SYN), definita tra due termini t_i e t_j, con $t_i \neq t_j$, che siano considerati sinonimi in tutte le sorgenti considerate (cioè, t_i e t_j possono essere utilizzati indifferentemente in ogni sorgente per indicare un certo concetto).
- Ipernimia (Boarder terms) (BT), definita tra due termini t_i e t_j tali che t_i ha un significato più generale di t_j. La relazione BT non è simmetrica. La relazione contraria a BT è NT (Narrower Terms) o iponimia.
- Termini correlati (RT) o associazione positiva, definita tra due termini t_i e t_j che siano generalmente utilizzati insieme nello stesso contesto nella sorgente considerata.

Relazioni estensionali. Le relazioni intensionali SYN, BT e NT definite tra due classi C_1 e C_2 possono essere rafforzate dichiarando che esse sono anche relazioni estensionali. Di conseguenza in ODL_{I^3} possono essere definite le seguenti relazioni estensionali:

- C_1 SYN_{ext} C_2: questo significa che le istanze di C_1 sono le stesse di C_2.
- C_1 BT_{ext} C_2: questo significa che le istanze di C_1 sono un sovrainsieme delle istanze di C_2.
- C_1 NT_{ext} C_2: questo significa che le istanze di C_1 sono un sottoinsieme delle istanze di C_2.

Inoltre, le relazioni estensionali vincolano la struttura delle due classi C_1 e C_2, ossia C_1 NT_{ext} C_2 è semanticamente equivalente a una relazione *isa* . Per riassumere:

- una relazione estensionale C_1 NT_{ext} C_2 equivale a una relazione *isa* C_1 isa C_2 più una relazione intensionale C_1 NT C_2;
- una relazione estensionale C_1 BT_{ext} C_2 equivale a una relazione *isa* C_2 isa C_1 più una relazione intensionale C_1 BT C_2;
- una relazione estensionale C_1 SYN_{ext} C_2 equivale a due relazioni *isa* C_1 isa C_2 e C_2 isa C_1 più una relazione intensionale C_1 SYN C_2.

Una relazione *isa* C_1 isa C_2 è esprimibile in ODL_{I^3} dalla seguente regola di integrità:

```
rule Rule forall X in C1 then X in C2
```

Regole di mapping (mapping rules): questo tipo di regola è stato introdotto in ODL_{I^3} per esprimere le relazioni che esistono tra la descrizione dello schema integrato e la descrizione ODL_{I^3} degli schemi delle sorgenti originali.

I wrapper di MOMIS traducono gli schemi delle varie sorgenti, in schemi ODL_{I^3} . Inoltre un wrapper è responsabile anche dell'aggiunta del nome della sorgente e del tipo (relazionale, ad oggetti, semistrutturato). La traduzione in ODL_{I^3} seguendo la corretta sintassi (si veda l Appendice D) è eseguita come segue. Data una relazione di uno schema relazionale o un pattern <l,A>, la traduzione comprende i seguenti passi:

1. Una classe ODL_{I^3} è definita con un nome corrispondente al nome della relazione o di l, rispettivamente.

2. Per ogni attributo della relazione o per ogni label $l' \in A$, viene definito un attributo nella classe ODL_{I^3} corrispondente. Quindi vengono estratti i domini degli attributi.

Come esempio, viene riportata la rappresentazione ODL_{I^3} di alcune classi locali: (un esempio completo di uno schema ODL_{I^3} può essere trovato in [31])

```
Fast-Food-pattern =

(Fast-Food,{name,address,midprice*, phone* ,
            specialty,category,nearby* ,owner*} )

interface Fast_Food
( source semistructured ED)
{
 attribute Owner owner *;
 attribute Fast_Food nearby *;
 attribute long phone *;
 attribute Address address;
 attribute string name;
 attribute string category;
 attribute set<string> specialty;
 attribute long midprice *;
};

Restaurant(r_code,name,street,zip_code,pers_id,
           special_dish,category,tourist_menu_price)

interface Restaurant
( source relational FooD_Guide
```

```
   key (r_code)
   foreign_key(pers_id) references Person )
{
 attribute string r_code;
 attribute string name;
 attribute string street;
 attribute string zip_code;
 attribute integer pers_id;
 attribute string special_dish;
 attribute integer category;
 attribute integer tourist_menu_price;
};
```

Partendo dalle descrizioni ODL_{I^3} delle varie sorgenti locali, il sistema MOMIS è in grado di produrre uno schema integrato formato da varie classi globali.

3.4.1 OLCD

Le Logiche Descrittive (DLs) [26, 27] costituiscono una restrizione delle $1^{st}OPL$[28]: esse consentono di esprimere i concetti sotto forma di formule logiche, usando solamente predicati unari e binari, contenenti solo una variabile (corrispondente alle istanze del concept).

Un grande vantaggio offerto dalle DLs, per le applicazioni di tipo DBMS, è costituito dalla capacità di rappresentare la semantica dei modelli di dati ad oggetti complessi (CODMs), recentemente proposti in ambito di basi di dati deduttive e basi di dati orientate agli oggetti. Questa capacità deriva dal fatto che, tanto le DLs quanto i CODM, si riferiscono esclusivamente agli aspetti strutturali: tipi, valori complessi, oggetti con identità, classi, ereditarietà. Unendo quindi le caratteristiche di similarità coi CODM e le tecniche di inferenza tipiche delle $1^{st}OPL$ si raggiunge l'ambizioso obiettivo di dotare i sistemi di basi di dati di componenti intelligenti per il supporto alle attività di design, di ottimizzazione edi integrazione di informazioni poste in sorgenti eterogenee.

Il formalismo OLCD deriva dal precedente ODL, proposto in [29], che estende l'espressività dei linguaggi sviluppati nell'area delle DLs. *La caratteristica principale di OLCD consiste nell'assumere una ricca struttura per il sistema dei tipi di base*: oltre ai classici tipi atomici integer, boolean, string, real, e tipi monovalore, viene ora considerata anche la possibilità di utilizzare dei sottoinsiemi di questi (come potrebbero essere, ad esempio, intervalli di interi). A partire da questi tipi di base si possono definire i tipi valore, attraverso gli usuali costruttori di tipo definiti nei CODMs, quali tuple, insiemi, liste e tipi classe, che denotano insiemi di oggetti con una identità ed un valore associato. Ai tipi può essere assegnato un nome, mantenendo la distinzione tra nomi di tipi valore e nomi di tipi

classe, che d'ora in poi denomineremo semplicemente classi: ciò equivale a dire che i nomi dei tipi vengono partizionati in nomi che indicano insiemi di oggetti (tipi classe) e nomi che rappresentano insiemi di valori (tipi valore). L'ereditarietà, sia semplice che multipla, é espressa direttamente nella descrizione di una classe tramite l'operatore di intersezione.

OLCD introduce inoltre la distinzione tra nomi di tipi virtuali, che descrivono condizioni necessarie e sufficienti per l'appartenenza di un oggetto del dominio ad un tipo (concetto che si può quindi collegare al concetto di vista), e nomi di tipi primitivi, che descrivono condizioni necessarie di appartenenza (e che quindi si ricollegano alle classi di oggetti). In [14] OLCD é stato esteso per permettere la formulazione dichiarativa di un insieme rilevante di vincoli di integrità definiti sulla base di dati. Attraverso questo logica descrittiva, é possibile descrivere, oltre alle classi, anche le regole di integrità: permettono la formulazione dichiarativa di un insieme rilevante di vincoli di integrità sotto forma di regole if-then i cui antecedenti e conseguenti sono espressioni di tipo OLCD. In tale modo, é possibile descrivere correlazioni tra proprietà strutturali della stessa classe, o condizioni sufficienti per il popolamento di sotto-classi di una classe data. In altre parole le rule costituiscono uno strumento dichiarativo per descrivere gli oggetti che popolano il sistema.

Per la rappresentazione dei vincoli di integrità é stata introdotta una sintassi intuitiva coerente con la proposta ODMG-93. In particolare si sono sfruttati il costrutto `for all`, il costrutto `exists`, gli operatori booleani e i predicati di confronto utilizzati nell OQL. Una regola di integrità é dichiarata attraverso la seguente sintassi: **rule** `<nome-regola>` **for all** `<nome-iteratore>` **in** `<nome-classe>`: `<condizione-antecedente>`
then `<condizione-conseguente>` Le `condizioni`, antecedente e conseguente, hanno la medesima forma e sono costituite da una lista di espressioni booleane in `and` tra loro; all'interno di una condizione, attributi e oggetti sono identificati mediante la *dot notation*. La nozione di ottimizzazione semantica di una query é stata introdotta, per le basi di dati relazionali, da King [15, 16] e da Hammer e Zdonik [17]. L'idea di base di queste proposte é che i vincoli di integrità, espressi per forzare la consistenza di una base di dati, possano essere utilizzati anche per ottimizzare le interrogazioni fatte dall'utente, trasformando la query in una *equivalente*, ovvero con lo stesso insieme di oggetti di risposta, ma che può essere elaborata in maniera più efficiente.

OLCD: un Formalismo per Oggetti Complessi e Vincoli di Integrità

Di seguito verrà brevemente descritta la sintassi del linguaggio OLCD, mostrando anche come vengono tradotti in tale formalismo le descrizioni ODL_{I^3} degli schemi. Sia **A** un insieme numerabile di *nomi di attributi* (denotati con $a_1, a_2, ...$),

e sia **N** un insieme numerabile di *nomi di tipi* (denotati con $N_1, N_2, ...$). L'insieme **N** include l'insieme **B** = $\{$**Integer, String, Real, Bool**$\}$ dei designatori di tipi base (atomici) ed i simboli $\top$, $\bot$, rappresentativi rispettivamente del *tipo universale* e del *tipo vuoto*. Un *path p* é una sequenza di elementi $e_1.e_2....e_n$, con $e_i \in A \cup \{\Delta, \exists\}$; con ϵ si indica il path vuoto mentre l'insieme di tutti i *path* é denominato **W**.

S(A,N) indica l'insieme di tutte
le descrizioni di tipo finite (denotate da $S_1, S_2, ...$), dette brevemente tipi, su di un dato **A,N**, ottenuto in accordo con la seguente regola sintattica, dove $a_i \neq a_j$ per $i \neq j$ (nella seguente sequenza p, $p_1, p_2, ...$, denota un *path*, d denota un valore base, θ un operatore relazionale):

$$S \rightarrow N|S_1 \sqcup S_2|S_1 \sqcap S_2|\neg S|\{S\}_\forall|\{S\}_\exists|[a_1 : S_1, ..., a_k : S_k]|\Delta S|p\theta d|p \uparrow$$

$\{\}_\forall$ corrisponde al comune costruttore di insieme, [] denota il costruttore di tupla, mentre il costruttore $\{S\}_\exists$ denota un insieme in cui *almeno* un elemento é di tipo S. Il costrutto $\sqcap$ indica la *congiunzione* (intersezione), $\sqcup$ *l'unione*, Δ é il costruttore di oggetto ed il costrutto $\neg$ indica il complemento. Infine $p\theta d$ e $p \uparrow$ rappresentano dei *predicati atomici*: $p\theta d$ definisce una restrizione ad un intervallo, $p \uparrow$ un path "indefinito" . Dato un sistema di tipi **S(A,N)**, uno *schema* σ é una funzione totale da **N** a **S(A,N)** che associa ai nomi dei tipi la loro descrizione. σ é partizionata in due funzioni: σ_p, che introduce la descrizione di tipi primitivi la cui estensione dev'essere fornita dall utente; σ_v che introduce la descrizione di tipi virtuali la cui estensione può essere ottenuta ricorsivamente dall'estensione dei tipi che compaiono nelle loro descrizioni. In OLCD sono ammessi i *nomi di tipo* ciclici: possono esistere *riferimenti circolari*, ovvero *nomi di tipo* che fanno riferimento a se stessi (direttamente o indirettamente). Vediamo ora le principali regole di traduzione da ODL_{I^3} ad OLCD.

ODL_{I^3} **classes.** La traduzione delle classi ODL_{I^3} in OLCD risulta piuttosto semplice: ogni attributo della classe ODL_{I^3} diventa un attributo della corrispondente classe OLCD. Consideriamo la classe **FD.Restaurant**

```
Restaurant(r_code,name,street,zip_code,pers_id,
special_dish,category,tourist_menu_price)
```

la sua traduzione sarà la seguente:

$$\sigma_p(FD.Restaurant) = \Delta[r_code : String, name : String,$$
$$street : String,$$
$$zip_code : String, pers_id : Integer, special_dish : String,$$
$$category : Integer, tourist_menu_price : Integer]$$

Alcuni aspetti della descrizione ODL_{I^3} delle classi non sono tradotti in OLCD, ma verranno usati nel processo di integrazione semantica delle informazioni (ad esempio primary_key,foreign_key).

Union constructor. Il costruttore **union** dell ODL_{I^3} é tradotto in OLCD usando il construtto $\sqcup$. Vediamo un esempio:

```
interface Address
( source semistructured ED )
{
 attribute string city;
 attribute string street;
 attribute string zipcode;
};
union
{
 attribute string dummy;
};
```

la traduzione della classe ED.Address sarà:

$$\sigma_p(ED.Address) = \Delta(dummy : String, \sqcup [city : String, street : String, zip_code : String])$$

Optional Constructor. Il construtto $\sqcup$ può inoltre essere usato per tradurre gli attributi *opzionali* in OLCD. Infatti, un attributo *opzionale* indica che un valore può o meno esistere per una data instanza. Tutto questo é espresso in OLCD come l'unione tra la descrizione dell'attributo e l'attributo *'non definito'* , denotato con l'operatore $\uparrow$. Se consideriamo la descrizione ODL_{I^3} della classe **Fast_Food** notiamo la presenza di attributi opzionali (quelli seguiti da *) che vengono così tradotti:

$\sigma_p(ED.Fast_Food) = \Delta([name : String, address : ED.Address, speciality : \{String\}, category : String] \sqcap ([phone : Integer] \sqcup phone \uparrow) \sqcap ([nearby : ED.Fast_Food] \sqcup nearby \uparrow) \sqcap ([midprice : Integer] \sqcup midprice \uparrow) \sqcap ([owner : ED.Owner] \sqcup owner \uparrow))$

Vincoli d'integrità. Le regole d'integrità di tipo *if-then* sono integrate in OLCD usando i costrutti $\sqcup, \sqcap, \neg$. Per esempio la rule: `rule Rule1 forall X in Restaurant: (X.category > 5) then X_tourist_menu_price > 100;` viene aggiunta alla descrizione della classe **FD.Restaurant**, viene così tradotta:

$\sigma_p(FD.Restaurant) = \Delta([r_code : String, name : String, street : String, zip_code : String, pers_id : Integer, special_dish : String, category : Integer, tourist_menu_price : Integer] \sqcap (\neg(category > 5) \sqcup (tourist_menu_price > 100)))$

Relazioni terminologiche (SYN, NT, BT, RT). Non sono tradotte.

Relazioni estensionali ($SYN_{ext}, NT_{ext}, BT_{ext}, DISJ_{ext}$). ogni relazione del tipo C_1 isa C_2 é espressa in ODL_{I^3} da una rule del tipo:

`rule Rule1 forall X in` C_1 `then X in` C_2;

ed é integrata nella descrizione della classe
C_1 usando il costrutto $\sqcap$: $\sigma_p(C_1) = C_2 \sqcap ...$

Mapping Rules. Non sono tradotte.

Le regole OLCD e l'espansione semantica di un tipo

Sia il processo di consistenza e classificazione delle classi dello schema, sia quello di ottimizzazione semantica di una interrogazione, sono basati in ODB-Tools sulla nozione di *espansione* semantica di un tipo: l'espansione semantica permette di incorporare ogni possibile restrizione che non é presente nel tipo originale, ma che é logicamente implicata dallo schema (inteso come l'insieme delle classi, dei tipi, e delle regole di integrità). L'espansione dei tipi si basa sull'iterazione di questa trasformazione: se un tipo *implica* l'antecedente di una regola di integrità, allora il conseguente di quella regola può essere aggiunto alla descrizione del tipo stesso. Le *implicazioni* logiche fra i tipi (in questo caso il tipo da espandere é l'antecedente di una regola) sono determinate a loro volta utilizzando l'algoritmo di *sussunzione*, che calcola relazioni di sussunzione, simili alle relazioni di raffinamento dei tipi definite in [18].

Il calcolo dell'espansione semantica di una classe permette di rilevare nuove relazioni *isa*, cioè relazioni di specializzazione che non sono esplicitamente definite dal progettista, ma che comunque sono logicamente implicate dalla descrizione della classe e dello schema a cui questa appartiene. In questo modo, una classe può essere automaticamente classificata all'interno di una gerarchia di ereditarietà. Oltre che a determinare nuove relazioni tra classi virtuali, il meccanismo, sfruttando la conoscenza fornita dalle regole di integrità, é in grado di riclassificare pure le classi base (generalmente gli schemi sono forniti in termini di classi base).

Analogamente, rappresentando a run-time l'interrogazione dell'utente come una classe virtuale (l'interrogazione non é altro che una classe di oggetti di cui si definiscono le condizioni necessarie e sufficienti per l'appartenenza), questa viene classificata all'interno dello schema, in modo da ottenere l'interrogazione più specializzata tra tutte quelle semanticamente equivalenti alla iniziale. In questo modo l'interrogazione viene spostata verso il basso nella gerarchia e le classi a cui si riferisce vengono eventualmente sostituite con classi più specializzate: diminuendo

l'insieme degli oggetti da controllare per dare risposta all'interrogazione, ne viene effettuata una vera ottimizzazione indipendente da qualsiasi modello di costo fisico.

Validazione e Sussunzione

Uno dei problemi principali che il progettista di una base di dati deve affrontare é quello della consistenza delle classi introdotte nello schema. Infatti, molti modelli e linguaggi di definizione dei dati sono sufficientemente espressivi da permettere la rappresentazione di classi inconsistenti, cioè classi che non potranno contenere alcun oggetto della base di dati. Tale eventualità sussiste anche in **OLCD**: ad esempio, la possibilità di esprimere intervalli di interi permette la dichiarazione di classi con attributi omonimi vincolati a intervalli disgiunti. Il prototipo rivela, durante la fase di validazione dello schema, come inconsistente una eventuale congiunzione di tali classi. Il concetto di *sussunzione* esprime invece la relazione esistente tra due classi di oggetti quando l'appartenenza di un oggetto alla seconda comporta necessariamente l'appartenenza alla prima. La relazione di sussunzione può essere calcolata automaticamente tramite il confronto sintattico tra le descrizioni delle classi; l'algoritmo di calcolo é stato presentato in [35]. Poichè accanto alle relazioni di ereditarietà definite esplicitamente dal progettista possono esistere altre relazioni di specializzazione implicite, queste possono essere esplicitate dal calcolo della relazione di sussunzione presenti nell'intero schema: il prototipo, dopo aver verificato la consistenza di ciascuna classe, determina tali relazioni di specializzazione implicite fornendo un valido strumento inferenziale per l'utente progettista della base dei dati.

3.5 MOMIS nell'architettura del Web Semantico

Finora abbiamo preso in esame le piú interessanti proposte per quanto riguarda il *livello logico* e visto il linguaggio usato in MOMIS per adempiere ai compiti di tale livello; adesso esaminiamo come é possibile inquadrare MOMIS nell'architettura del Web Semantico.

Il nostro obiettivo é quello di rendere possibile la condivisione di uno schema ODL_{I^3}, risultante dal processo d'integrazione di una serie di sorgenti dati eterogenee, cercando di mantenere il più possibile la semantica catturata.

ODL_{I^3} é nato come linguaggio per l'integrazione di sorgenti di dati eterogenee, per cui é naturalmente adatto per catturare un certo tipo di conoscenza quale: chiavi, foreign key, regole d'integrità, ecc.. . Volendo garantire il massimo della condivisione di tale conoscenza, dobbiamo muoverci all'interno degli standard che si sono affermati o che si stanno affermando.

A mio avviso la scelta migliore é quella di fare un uso congiunto di XML-Schema ed RDF(S),non menziono XML1.0 per due motivi:

1. Le specifiche XML-Schema sono arrivate a livello di Recommendation. quindi hanno un certo grado di stabilità.

2. Le difficoltà incontrate da Lenzi nella sua Tesi[37] dimostrano la non capacità di XML1.0 a esportare altro che non sia testo.

XML-Schema e RDF(S) nascono entrambi per facilitare la condivisione delle informazioni sul web, ma hanno caratteristiche *complementari*:

- XML-Schema fornisce una serie di primitive per vincolare la struttura di un documento, definire il numero di occorrenze e il tipo dei dati; mentre non fà altrettanto per quanto riguarda le primitive per specificare la semantica.

- RDF(S) fornisce alcune primitive per definire la semantica dei concetti espressi: `Class`,`Property`, `subclassOf`, `subpropertyOf`, `type` ecc..; ma non fà altrettanto per quanto riguarda la struttura del documento.

questa complementarietà é dimostrata anche dal diverso modello di dati utilizzato; XML-Schema é basato su un modello tree-based mentre RDF-Schema é basato su un modello a grafo orientato ed etichettato (Direct Labeled Graph) simile al relazionale con Risorse (Entità) e proprietà (attributi) e possibilità di indicare relazioni fra risorse (con subclassOf, type). Per questi motivi, riallacciandosi all'architettura del "web semantico" la mia proposta per esportare la conoscenza espressa dal virtual global schema e dai wrapper é la seguente:

Logic Layer: ODL_{I^3}+OLCD.

Schema Layer: XML-Schema+RDF-Schema.

Data Layer: XML+RDF.

Capitolo 4

Esportazione di schemi ODL_{I^3} in XML-Schema

4.1 Introduzione a XML-Schema

Vediamo, dopo una breve introduzione a XML-Schema, la parte Xml-Schema del traduttore:

4.1.1 Origini

L'enfasi iniziale era su internazionalizzazione, strutturazione, facilità di conversione, ecc; si pensava che XML servisse solo per esprimere documenti di testo: libri, manuali, pagine web ecc...

Raccoglieva in pieno l'eredità di SGML. Lo sviluppo di XML era difatti condotto da membri della comunità SGML.

Nasce poi l'idea che XML possa servire per qualcosa più: *XML è (anche) un linguaggio di markup per trasferire dati*: un meccanismo per convertire dati dal formato interno dell'applicazione ad un formato di trasporto, facile da convertire in altri formati interni. Non pensato per la visione umana, ma per essere prodotto ed usato da programmi.

XML è un interfaccia (Adam Bosworth):

- Un'interfaccia tra autore e lettore, attraverso XSL e XLink, per portare significato tra creatore ed utente.

- Un'interfaccia tra *applicazione ed applicazione*, attraverso XML-Schema, per esprimere contratti sui formati, e verificarne il rispetto.

Il trasferimento dei dati si semplifica: i documenti strutturati e gerarchici sono un formato ragionevole di sintassi praticamente per qualunque cosa: documenti

di testo, record di database, ecc. Nella W3C Note di
Agosto 1999 (http://www.w3.org/ TR/schema-arch):

Many data-oriented applications are being defined which build their own data structures on top of an XML document layer, effectively using XML documents as a transfer mechanism for structured data.

4.1.2 Validazione e Buona Forma

La buona forma di un documento XML è una proprietà puramente sintattica. La validazione, viceversa, è la verifica di un impegno preso sopra al formato, ad un livello già semantico:

Mi impegno a scrivere dei documenti che siano formati da capitoli, ciascuno con un titolo e vari paragrafi, e ogni immagine con la sua didascalia.

Per esprimere documenti di testo, i DTD probabilmente bastano, ma per esprimere blocchi di dati strutturati, è necessario un meccanismo di verifica più raffinato.

XML Schema è stato pensato per fornire quel supporto di validazione che i DTD permettono solo parzialmente, in particolare sul contenuto degli elementi e degli attributi del documenti XML.

4.1.3 XML-Schema e DTD

I DTD hanno una sintassi particolare diversa dall' XML, così da dover creare strumenti appositi per la validazione.

I DTD non distinguono tra nome del tag e tipo del tag, ed hanno solo due tipi: complesso (cioè strutturato) e semplice (cioè CDATA o #PCDATA).

XML Schema, invece, fornisce un set complesso di tipi, a cui i tag e il loro contenuto debbono aderire.

XML-Schema ha un approccio *object-oriented*, permettendo di ampliare i tipi disponibili e di estenderne e precisarne le proprietà.

Infine, XML Schema è scritto in XML, permettendo l'uso di applicazioni XML per la verifica della validità dei dati espressi.

4.1.4 Documento XML-Schema

Un documento di XML Schema è racchiuso in un elemento <schema>, e può contenere, i seguenti elementi:

- <import> ed <include> per inserire, altri frammenti di schema da altri documenti

- <simpleType> e <complexType> per la definizione di tipi denominati usabili in seguito.
- <element> ed <attribute> per la definizione di elementi ed attributi globali del documento.
- <attributeGroup> e <group> per definire serie di attributi e gruppi di content model complessi e denominati.
- <annotation> per esprimere commenti per esseri umani o per applicazioni diverse dal parser di XML Schema.

4.1.5 I tipi in XML Schema

XML Schema usa i tipi per esprimere vincoli sul contenuto di elementi ed attributi.

- Un tipo semplice è un tipo di dati che non può contenere markup e non può avere attributi.
- Un tipo complesso è un tipo di dati che può contenere markup e avere attributi.

Esiste un grande numero di tipi predefiniti, di tipo semplice: string, decimal, float, boolean, anyUri, date, time, ecc.

Di ogni tipo è possibile definire alcune proprietà, dette facets, che ne descrivono vincoli e formati (permessi ed obblighi).

É data possibilità di derivare nuovi tipi, sia per restrizione che per estensione di permessi ed obblighi.

4.1.6 Esempi

```
<simpleType name="bodytemp">
 <restriction base="decimal">
  <minInclusive value="36.5"/>
  <maxInclusive value="44.0"/>
 </restriction>
</simpleType>

<complexType name="name">
 <sequence>
  <element name="forename"  type="string"
           minOccurs="0"  maxOccurs="unbounded"/>
  <element name="surname" type= xsd:string />
```

```
 </sequence>
</complexType>
```

4.1.7 Derivazione di tipi

É possibile estendere i tipi in due maniere:

1. Per restrizione (<restriction>) ad esempio specificando facets aggiuntivi.
2. Per estensione (<extension>) ad esempio estendendo un content model precedentemente definito.

4.1.8 Vincoli

Per ogni tipo possiamo precisare dei facets, delle caratteristiche indipendenti tra di loro che specificano diversi aspetti:

- length, minLength, maxLength: numero richiesto, minimo e massimo di caratteri
- minExclusive, minInclusive, maxExclusive, maxInclusive: valore massimo e minimo, inclusivo ed esclusivo
- pattern: espressione regolare che il valore deve soddisfare
- enumeration: lista all'interno dei quali scegliere il valore (simile alla lista di valori leciti degli attributi nei DTD)
- ecc...

vediamo un esempio

```
<simpleType name="healthbodytemp">
 <restriction base="bodytemp">
  <maxInclusive value="37.0"/>
 </restriction>
</simpleType>
```

4.1.9 Elementi e Attributi

- Si usano gli elementi <element> e <attribute>.
- Se sono posti all'interno del tag <schema> sono elementi ed attributi globali (possono essere root di documenti).

- Altrimenti sono usabili solo all'interno di elementi che li prevedono come tipo.

Elementi ed Attributi hanno vari attributi importanti:

name : il nome del tag o dell attributo

ref : il nome di un elemento o attributo definito altrove (globale)

type : il nome del tipo, se non esplicitato come content

maxOccurs, minOccurs : il numero minimo e massimo di occorrenze

fixed, default, nillable, ecc. : specificano valori fissi, di default e determinano la possibilità di elementi nulli.

Content Model complessi

Come nei DTD si usano virgole e | per specificare obblighi e scelte tra gli elementi di un content model complesso, così in XML schema si usano <choice>, <sequence> e <all>. Questi sostituiscono anche le parentesi.

Vediamo degli esempi:

- (A, B) diventa :

```
<sequence>
 <element ref="A"/>
 <element ref="B"/>
</sequence>
```

- (A | B) diventa:

```
<choice>
 <element ref="A"/>
 <element ref="B"/>
</choice>
```

- (A, (B | C)) diventa:

```
<sequence>
 <element ref="A"/>
 <choice>
  <element ref"B"/>
  <element ref"C"/>
 </choice>
</sequence>
```

- (A & B & C) (operatore di SGML) diventa:

```
<all>
 <element ref="A"/>
  <element ref"B"/>
  <element ref"C"/>
</all>
```

Il content model misto aggiunge semplicemente l'attributo mixed con valore true. Il content model vuoto viene specificato con un tipo complesso e privo di elementi. Un content model PCDATA con l'elemento <simpleContent> . Così:

- (#PCDATA | B | C)* diventa

```
<complexType mixed="true">
 <element ref="B"/>
 <element ref="C"/>
</element>
```

- EMPTY diventa

```
<element name="vuoto">
 <complexType/>
</element>
```

Attributi

La dichiarazione di attributi può avvenire con l'elemento <attribute>, dentro alla dichiarazione dell'elemento:

```
<!ELEMENT A (B,C)>
<!ATTLIST A
          p CDATA #IMPLIED>
```

corrisponde a:

```
<element name="A">
 <complexType>
  <sequence>
   <element ref="B"/>
   <element ref="C"/>
  </sequence>
 </complexType>
 <attribute name="p" type= xsi:string/>
</element>
```

4.1.10 Gruppi di elementi ed attributi

É possibile raccogliere gli elementi e gli attributi in gruppi:

```
<element name="A">
 <group ref="elemA"/>
 <attributeGroup ref="attrsA"/>
</element>

<group name="elemA"/>
 <complexType>
  <sequence>
   <element ref="B"/>
   <element ref="C"/>
  </sequence>
 </complexType>
</group>

<attributeGroup name="attrsA">
 <attribute name="p" type="string/>
 <attribute name="q" type="int"/>
</attributeGroup>
```

4.1.11 Annotazioni

In XML Schema, esiste un posto specifico dove mettere note ed istruzioni, l'elemento <annotation>. L'elemento <annotation> può contenere elementi <documentation> (pensati per essere letti da esseri umani) oppure elementi <appInfo>, pensati per essere digeriti da applicazioni specifiche

```
<element name="pippo">
 <annotation>
  <documentation>elemento pippo</documentation>
 </annotation>
</element>
```

4.1.12 Namespace

Il concetto di namespace in XML è molto simile a quello usato in linguaggi di programmazione tipo Java e C++: permette di evitare collisioni tra i nomi di elementi e attributi tramite l'uso di nomi qualificati, consistenti in un prefisso che

indica il namespace cui l'elemento appartiene seguito da un nome locale a tale namespace.

Per esempio <ns1:tag1> e <ns2:tag1> sono due elementi distinti, pur avendo lo stesso nome locale tag1 .

I namespace sono in realtà identificati da URI (Uniform Resource Identificator) e associati a particolari prefissi nel file XML stesso, tramite l'uso dell'attributo xmlns:

```
<root xmlns:ns1="http://www.acme.org/ns1">
 <ns1:tag1 ns1:att1="value1"/>
</root>
```

Una cosa importante riguardo gli XML namespace (e che li distingue da quelli Java e C++) è che non implicano, di per sé, l'esistenza di alcun file o risorsa contenente la dichiarazione degli elementi e attributi che essi contengono. L'URI usato come valore dell'attributo xmlns in genere non si risolve in alcuna risorsa effettivamente esistente su qualche server.

Esso serve semplicemente a identificare univocamente il namespace, in modo che la applicazioni possano interpretare correttamente i tag riferentisi a quel namespace.

La dichiarazione di `targetNamespace` definisce il namespace del documento da validare. Gli attributi
`elementFormDefault` e `attributeFormDefault` permettono di controllare se l'uso del prefisso è necessario per i tipi non globali.

```
<schema xmlns="http://www.w3.org/2000/08/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="unqualified"
        attributeFormDefault="unqualified">

 <element name="A"  type="po:prova"/>

 <element name="C" type="string"/>

 <complexType name="prova">
  <sequence>
   <element name="B"  type="string"/>
   <element ref="C"/>
  </sequence>
 </complexType>
</schema>
```

Poiché gli attributi `elementFormDefault` e `attributeFormDefault` sono definiti come unqualified , solo i tipi globali debbono avere il prefisso, gli altri no[1].

```
<xx:A xmlns:xx="http://www.example.com/PO1">
 <B> ... </B>
 <xx:C> ... </xx:C>
</xx:A>
```

Alternativamente, si può usare il namespace di default e non usare mai prefissi:

```
<A xmlns="http://www.example.com/PO1">
 <B> ... </B>
 <C> ... </C>
</A>
```

Quello che i namespace permettono di fare è di specificare regole di validazione solo su alcuni e non tutti i namespace del documento:

```
<element name="htmlElement">
 <complexType>
  <sequence>
   <any namespace="http://www.w3.org/1999/xhtml"
        minOccurs="1" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

Nell'attributo namespace dell elemento <any> posso specificare o un namespace vero e proprio, o i valori:

- ##any: qualunque XML ben formato.
- ##local: qualunque XML non sia qualificato (cioè privo di dichiarazione di namespace)
- ##other: qualunque XML tranne il target namespace.

4.1.13 Inclusioni & Importazioni

In XML Schema, esistono meccanismi per dividere lo schema in più file, o per importare definizioni appartenenti ad altri namespace.

[1]Questa é la scelta fatta nel traduttore sviluppato per la tesi

- <include schemaLocation=”http://www.xxx”/>
- <import namespace=”http://www.example.com/IPO”/>

<include> serve per importare documenti con lo stesso `targetNamespace` del documento in cui vengono importati mentre <import> non ha questa limitazione permettendo di indicare nell’attributo namespace il `targetNamespace` del documento che si vuole importare.

Unicità e chiavi:

- In XML Schema, è possibile richiedere che certi valori siano unici, o che certi valori siano chiavi di riferimenti, analoghi alla coppia ID/IDREF in XML classico, tuttavia, è possibile specificare pattern molto complessi come elementi chiave.

4.2 Creazione della Schema

4.2.1 Definizione delle Interfacce

Le interfacce sono definite attraverso un `complexType` in quanto contengono dei sotto-elementi che rappresentano gli attributi dell’interfaccia.

Per come é organizzato il traduttore, con la presenza di chiamate di regole innestate, la traduzione segue il seguente schema:

- Si crea un `group` per ogni body dell’interfaccia.
- Si crea un `complexType` che fà riferimento a tali group.
- Si crea un `element` globale avente tipo il complexType definito.

ad esempio:

```
interface CS_Person
{
  attribute string first_name;
  attribute string last_name;
};
```

viene così tradotto:

```
<group name="CS_PersonBody1">
 <sequence>
  <element name="first_name" type="string"/>
  <element name="last_name" type="string"/>
```

```
 </sequence>
</group>

<complexType name="CS_PersonType">
 <sequence>
  <group ref="CS_PersonBody1/>
 </sequence>
 <attribute name="persistent" type="boolean"
            use="default" value="true">
 <attribute name="view" type="boolean"
            use="default" value="false">
</complexType>

<element name="CS_Person" type="CS_PersonType"/>
```

N.B Come si é visto gli attributi di un body sono contenuti, all'interno di un `group`, in una sezione `sequence` che vincola l'ordine di apparizione degli elementi in esso contenuti. Questo non é un vincolo di ODL_{I^3}, é stata fatta questa scelta nel traduttore perché l'alternativa di usare una sezione `all` (che non pone vincoli nell'ordine d'apparizione) non avrebbe permesso la traduzione di attributi quali variabili di tipo struct o di tipo classe con associata una dimensione maggiore di 1 (es: attribute Person relatives[2] o attribute struct xxx {....} address[2]); in quanto in una sezione `all` tutti gli elementi devono avere un occorrenza massima (il valore di `maxOccurs`) uguale ad 1 ed il traduttore per questi attributi definisce un `element` con un valore di `maxOccurs` pari alla dimensione associata all'attributo.

Se la definizione di un `group` in presenza di un solo body non é necessario, nel caso di più body in alternativa sono indispensabili.

Vediamo un esempio:

```
interface CS_Person
{
  attribute string first_name;
  attribute string last_name;
}
union
{
 attribute string name;
};
```

come detto in 4.2.1 definiamo un `group` per ogni body:

```
<group name="CS_PersonBody1">
 <sequence>
  <element name="first_name" type="string"/>
  <element name="last_name" type="string"/>
 </sequence>
</group>

<group name="CS_PersonBody2">
 <sequence>
  <element name="name" type="string"/>
 </sequence>
</group>
```

poi definiamo il `complexType` *CS_PersonType* il cui contenuto è l'alternativa fra questi due `group`:

```
<complexType name="CS_PersonType">
 <choice>
  <group ref="CS_PersonBody1"/>
  <group ref="CS_PersonBody2"/>
 </choice>
</complexType>
```

alla fine si crea l'`element` globale *CS_Person*:

```
<element name="CS_Person" type="CS_PersonType"/>
```

Quando si ha a che fare con un'interfaccia con un solo body la scelta migliore sarebbe quella di tradurlo all'interno del `complexType`:

```
<complexType ...
 <sequence>
  <element ../>
  <element ../>
 </sequence>
</complexType>
```

se nel traduttore questo non si é fatto è solo per motivi pratici in quanto la traduzione del contenuto dei body avviene prima che si sappia il loro numero.

4.2.2 Il problema delle chiavi

In XML1.0 la definizione delle chiavi e delle chiavi candidate rappresenta uno degli elementi di maggiore difficoltà nella traduzione di un'interfaccia ODL_{I^3} in quanto si possono solamente definire attributi di tipo `ID` e `IDREF` (o `IDREFS`).

XML-Schema mantiene queste definizioni per ragioni di compatibilità col passato, ma introduce nuovi meccanismi più potenti e flessibili:

- I meccanismi di XML-Schema possono essere applicati al contenuto di qualsiasi elemento od attributo indipendentemente dal suo tipo mentre ID é un tipo.
- XML-Schema permette di limitare lo *scope* dell'unicità mentre ID é relativo a tutto il documento.
- Si possono definire chiavi composte da combinazioni di elementi e attributi.

```
interface CS_Person
( source relational University
  extent CS_Persons
  key fiscal_code
  candidate_key ck00 (first_name, last_name)
  candidate_key ck01 (attributo_normale)
){
  attribute string first_name;
  attribute string last_name;
  attribute string attributo_normale
  attribute string fiscal_code
 };

<element name="CS_Person" type="CS_PersonType">
 <key name="CS_Personpk">
  <selector xpath="child::*"/>
  <field xpath="fiscal_code"/>
 </key>

 <unique name="ck00">
 <selector xpath="child::*"/>
  <field xpath="first_name"/>
  <field xpath="last_name"/>
 </unique>

 <unique name="ck01">
  <selector xpath="child::*"/>
  <field xpath="attributo_normale">
 </unique>
</element>
```

Per definire la chiave primaria si usa la parola chiave `key`; mentre per definire le chiavi candidate si usa la parola chiave `unique`, con la parola chiave `selector` si definisce il contesto di unicità e con `field` gli attributi dell'interfaccia che costituiscono la chiave.

4.2.3 Vincoli di integrità referenziale

I vincoli di integrità referenziale non sono esprimibili in una DTD, un tentativo di indicare la loro presenza in un documento XML1.0 é tramite l'uso di attributi di tipo IDREF, i quali rappresentano una debole soluzione, in quanto il tipo IDREF vincola ad avere il valore di un qualsiasi attributo definito come ID ma non specifica quale attributo. In questo caso XML-Schema permette una facile traduzione, definendo l'elemento `keyRef`; vediamo un esempio:

```
interface Chapter (
  source relational Library
  key (Chap_Num, Book_Num, Location_ID)
  foreign_key (Book_Num,Location_ID) references Book
)
{
 attribute string title;
 attribute long Chap_Num;
 attribute long Book_Num;
 attribute string Location_ID;
};
```

In XML-Schema questa interfaccia si definisce così:

```
<element name="Book" type="BookType">
 <key name="Bookpk">
  <selector xpath="child::*"/>
  <field xpath="Book_ID"/>
  <field xpath="Location_Spec"/>
 </key>
</element>

<element name="Chapter" type="ChapterType">
 ..
  <keyRef name="Chapter_fk01" refer="Bookpk">
   <selector xpath="child::*"/>
   <field xpath="Book_num"/>
   <field xpath="Location_ID"/>
```

```
  </keyref>
</element>
```

4.2.4 Ereditarietà

Consideriamo l'esempio dell'interfaccia *Research_Staff* che specializza l'interfaccia *Person*. (Preso da University.odl):

```
interface CS_Person (
  source relational University
  extent CS_Persons
  key (first_name, last_name)
){
  attribute string first_name;
  attribute string last_name; };


interface Research_Staff : CS_Person
  ( source relational University
    extent Research_Staff
    key (name)
    candidate_key ke_mail (e_mail)
    candidate_key ke_mail3 (name, relation)
    foreign_key (dept_code) references Department
    foreign_key (section_code) references Section )
  {
   attribute string relation;
   attribute string e_mail;
   attribute integer dept_code;
   attribute integer section_code;
  };
```

In XML 1.0 non è possibile definire tipi di dati personalizzati, mentre in XML-Schema è possibile derivare un tipo complesso attraverso l'estensione di un (ereditarietà singola) tipo (semplice o complesso) predefinito o definito dall'utente; nel nostro caso:

```
<complexType name="StaffType">
 <complexContent>
  <extension base="PersonType">
   <sequence>
```

```
    <group ref="StaffBody1"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>

<element name="Staff" type="StaffType"/>
```

In questo esempio il content model del tipo complesso StaffType è definito come il content model di PersonType più le dichiarazioni degli elementi presenti nel group `StaffBody1`: *relation, e_mail, dept_code e section_code*.

N.B. Nel caso di ereditarieta' multipla, non supportata da XML-Schema si procede nel seguente modo:

- Si estende la prima superclasse.
- Si introducono dei riferimenti agli elementi globali che definiscono le altre superclassi.

esempio:

```
interface Student : Person,School_Member
  {
   ....
   ......
  };
```

il `complexType` *StudentType* viene cosi definito:

```
<complexType name="StudentType">
 <complexContent>
  <extension base="PersonType">
   <sequence>
    <group ref="StudentBody1"/>
    <element ref="SchoolMember"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

4.2.5 Attributi ODL_{I^3}

In ODL_{I^3}3 gli attributi sono posti nella forma

```
attribute <attribute_type> <attribute_name>
```

Esaminiamo i vari tipi di attributo e vediamo come poterli tradurre nello Schema di un documento XML.

I tipi si dividono in tipi valore e in tipi classe.

Tipi classe

Un semplice riferimento ad un'altra interfaccia, come ad esempio:

attribute Ruota ruota_anteriore;

viene tradotto come un `element` di tipo *Ruota*:

```
<element name="ruota_anteriore" type="RuotaType"/>
```

ODL_{I^3} supporta le relazioni di tipo *is-a*; questo comporta che un attributo può assumere valori non solo conformi al suo tipo originale ma anche dei tipi che da quest'ultimo derivano. XML-Schema permette di gestire questo problema *nel documento istanza*:

```
interface Manager: Person
{
 ....
}

attribute Person boss;
```

in ODL_{I^3} l'attributo `boss` può assumere anche valori di tipo `Manager`, vediamo come gestire tale problema in XML-Schema:

Nel documento schema (file xsd):

```
<element name="boss" type="PersonType"/>

<complexType name="ManagerType">
 <complexContent>
  <extension base="PersonType">
   <sequence>
     .....
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

in un documento xml che si basa sullo schema in cui è dichiarato l'elemento `boss`:

```
<boss xsi:type="momis:ManagerType">Marchica</boss>
```

le specifiche XML-Schema permettono di usare in un documento istanza i tipi derivati, basta indicare esplicitamente il tipo attraverso l'attributo *xsi:type* il quale fà parte del namespace `XML-Schema instance`.

A differenza di XML1.0 non dobbiamo preoccuparci di ricavare tutti i tipi che hanno come superclasse il tipo di un'attributo; è compito di chi realizza il documento istanza avere ben chiaro le relazioni di ereditarietà fra i tipi. Certo rimane il problema dell'ereditarieta' multipla non supportata da XML-Schema.

Un altro problema con gli attributi classe è la possibilità di ricorsione; ad esempio consideriamo la seguente interfaccia:

```
interface Person
{
  attribute string name;
  attribute Person father;
}
```

dovendo indicare per ogni *Person* il padre che è di tipo *Person* non la finiremmo più; d'altro canto in una situazione del genere, solitamente, si è interessati solamente al primo father (non al father del father del father....) in questo caso potremmo dichiarare ogni elemento che traduce un attributo di tipo classe come *"nillable"* :

```
<element name="father" type="PersonType" nillable="true"/>
```

così facendo un documento istanza "valido" puo' contenere la seguente definizione di father:

```
<Person>
 <name>Pluto</name>
 <father>
  <name>Pippo</name>
  <father xsi:null="true"></father>
 </father>
</Person>
```

in questo modo indichiamo che l'informazione father non é disponibile o non é necessaria, anche se il complexType PersonType non permette la definizione di elemento vuoto.

Tipi valore (semplici)

I tipi valore si suddividono in tipi semplici e tipi ”complessi” (ConstrType). Per i tipi semplici è possibile usare i numerosi tipi predefiniti di XML-Schema:

tipo char

Per il tipo *char* si può usare il tipo predefinito *string* con il vincolo che la lunghezza sia pari ad 1:

```
<simpleType name="char">
 <restriction base="string">
  <length value="1"/>
 </restriction>
</simpleType>
```

tipo boolean

Si usa il tipo predefinito da XML-Schema, i valori ammessi sono: true, false, 1, 0.

tipo Floating point e tipo Integer

Tutti i vari tipi di Integer (short, long ,unsignedshort, unsignedlong, integer, int) e di Float (float e double) sono disponibili in XML-Schema per cui la traduzione è immediata.

tipo Stringa

Si usa il tipo predefinito `string`.

Octet Type

XML-Schema prevede i tipi semplici `byte` ed `unsignedByte`.

Any type

XML-Schema prevede il tipo *anyType* che rappresenta il tipo base da cui derivano tutti i tipi (semplici e complessi); anyType non pone alcun vincolo sulla struttura del suo contenuto:

```
<element name="qualsiasi" type="anyType"/>
```

poiché *anyType* é il default per il tipo degli elementi é possibile definire qualsiasi nel seguente modo:

```
<element name="qualsiasi"/>
```

Range type

Un tipo range si presenta nella forma:

```
range {17, 30} voto;
```

Una possibile traduzione del range è la seguente:

```
<element name="voto">
 <simpleType>
  <restriction base="xsd:positiveInteger">
   <minInclusive value="17">
   <maxInclusive value="30">
  </restriction>
 </simpleType>
</element>
```

Un elemento in un documento il cui contenuto è conforme a voto è:

```
<voto>21</voto>
```

4.2.6 Tipi definiti

Ad esempio:

```
typedef int Vettore[30];

 ....
attribute Vettore Voti;
```

La traduzione in questo caso è la seguente:

```
<simpleType name="listaint">
 <list itemType="integer"/>
</simpleType>

<simpleType name="Vettore">
 <restriction base="listaint">
  <length value="30"/>
 </restriction>
</simpleType>
```

è necessario definire il tipo lista *listaint* perchè il vincolo length non è applicabile agli integer.

4.2.7 Template Type

Set & List

Gli attributi set o list sono del tipo:

```
attribute set<something> a_set;
attribute list<something> a_list;
```

Se <something> è un tipo primitivo si traducono con il concetto di tipo lista:

```
<simpleType name="listsomething">
 <list itemType="something"/>
</simpleType>

<element name="a_set" type="listsomething"/>
<element name="a_list" type="listsomething"/>
```

Se <something> è di tipo classe non è possibile tradurre con un tipo lista ma bisogna definire un tipo complesso:

```
<element name="a_set" maxOccurs="unbounded">
 <complextype>
  <sequence>
   <element name="something" type="somethingType"/>
  </sequence>
 </complextype>
</element>
```

ad esempio:

```
attribute set<Person> population;

<element name="population">
 <complexType>
  <sequence>
   <element name="Person" type="PersonType" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

N.B Una cosa che si perde rispetto ai concetti di set e list di ODL_{I^3} é il concetto di ordine degli elementi presente in list ed assente in set.

Array

```
attribute int numeri[15];

<simpleType name="listint">
 <list itemType="int"/>
</simpleType>

<simpleType name="arrayint15">
 <restriction base="listint">
  <length value="15"/>
 </restriction>
</simpleType>


<element name="numeri" type="arrayint15"/>
```

N.B. Nel caso di attributi aventi una dimensione $>$ di 1 la convenzione usata per dare i nomi è: `"array"<tipo><dim>`

4.2.8 Tipi ”costruiti” (ConstrType)

Si dividono in tipi enumerativi, tipi union e strutture.

Tipi enumerativi

Vediamo un esempio:

```
attribute enum book_copy { one, two, three } book_copy;

<element name="book_copy">
 <simpleType>
  <restriction base="string">
   <enumeration value="one"/>
   <enumeration value="two"/>
   <enumeration value="three"/>
  </restriction>
 </simpleType>
</element>
```

in questo modo l’attributo book_copy può assumere solo una delle alternative indicate.

Tipi Union

```
union unionName switch(int) {
  case 10: int prova;
  case 20:
  case 30:
  string prova;
  default case:
  float prova;
}
```

Per la traduzione delle union si potrebbe far ricorso ad un ”Union Type”:

```
<simpleType name="unionName">
 <union memberTypes="int string float"/>
</simpleType>
```

anche se questa soluzione non traduce i vari case e il default.

N.B: il parser ODL_{I^3} non gestisce con molta accuratezza i tipi union, limitandosi a riportare una stringa con tutti i case, poichè non molto usati **non sono stati considerati nel traduttore**.

Strutture

Una struttura può essere:

```
typedef struct tagname {
  int a;
  string b;
  float c;
} tipoStruttura
```

viene tradotta in:

```
<complexType name="tipoStruttura">
 <sequence>
  <element name="a" type="int"/>
  <element name="b" type="string"/>
  <element name="c" type="float"/">
 </sequence>
</complexType>
```

```
attribute tipoStruttura nomeAttributo;
```

diventa:

```
<element name="nomeAttributo" type="tipoStruttura"/>
```

4.2.9 Attributi Globali e mapping rule

Ogni attributo globale può essere mappato su uno o più attributi locali. In alcuni casi l'attributo è mappato su un insieme predefinito di possibili valori; vi é una regola per ogni sorgente locale dalla quale la classe globale é stata ricavata, le regole di mapping (mapping rule) sono raccolte nella mapping table.

La mapping table è una tabella persistente che memorizza tutte le infomazioni sul mapping intensionale, le cui colonne rappresentano l'insieme delle classi locali appartenenti al cluster e le righe rappresentano gli attributi globali.

La traduzione diretta in XML-Schema non può conservare il concetto " per la sorgente X segui il mapping 1 nella sorgente Y il mapping 2...." ho preferito evitare una traduzione parziale e pasticciata e optare per l'indicazione dell'URI della mapping table in un documento RDF associato allo schema.

4.2.10 Relationship

Qui bisogna descrivere una relazione binaria tra due elementi XML-Schema. Vediamo una possibile traduzione:

```
interface professor....
  ....
 relationship set<Course> teaches
 inverse Course::is_taught_by;
```

Una traduzione proposta per l'elemento
`teaches` nell'interfaccia `professor` potrebbe essere:

```
<element name="teaches">
 <complextype>
  <sequence>
   <element ref="Course" maxOccurs="unbounded"/>
  </sequence>
 </complextype>
</element>
```

La traduzione per l'elemento `is_taught_by` nell'interfaccia `Course` è:

```
<element name="is_taught_by">
 <complextype>
  <sequence>
   <element ref="Professor" />
  </sequence>
 </complextype>
</element>
```

Per ogni relazione in ODL_{I^3} abbiamo l'indicazione del percorso diretto e di quello inverso; il diretto viene tradotto definendo un riferimento all'elemento globale che traduce l'interfaccia con cui esiste la relazione; mentre per il percorso inverso si usa una definizione di proprietà RDF-Schema che permette *parzialmente* di mettere in evidenza la relazione fra teaches e is_taught_by.

4.3 XML-Schema & RDF(S)

Nel paragrafo 3.5 ho detto che é necessario un uso congiunto di XML-Schema e RDF(S), vediamo adesso in che modo si possono combinare questi due meccanismi per garantire la leggibilità e comprensibilità dello schema risultante dal processo di traduzione.

Attualmente non esiste un meccanismo per la combinazione di RDF(S) e XML-Schema, di conseguenza bisogna cercare di ingegnarsi in qualche modo. Una possibilità, suggeritami da [38], é data dal meccanismo degli *openAttrs* previsto nelle specifiche di XML-Schema il quale permette di utilizzare nella definizione di uno schema attributi provenienti da namespace esterni.

4.3.1 Attributi openAttrs

Consideriamo la definizione di *openAttrs* (vedi appendice E):

```
<complexType name="openAttrs">
 <annotation>
  <documentation xml:lang="en">
   This type is extended by almost all schema types
   to allow attributes from other namespaces to be
   added to user schemas.
  </documentation>
 </annotation>
 <complexContent>
  <restriction base="anyType">
   <anyAttribute namespace="##other"
                processContents="lax"/>
  </restriction>
 </complexContent>
</complexType>
```

openAttrs é una restrizione di anyType (la radice della gerarchia dei tipi di dato per uno schema), può avere qualsiasi attributo (anyAttribute) proveniente da qualsiasi

namespace (namespace="##other) e se il parser non riesce ad ottenere lo schema in cui l'attributo è definito (processContents="lax") non segnala errore altrimenti procede alla validazione.

Nel nostro caso possiamo definire uno schema XML-Schema, chiamiamolo meta.xsd d'ora in avanti, in cui definire tutti quegli attributi che permettono di migliorare la comprensione dello schema che tradurrà le nostre classi ODL_{I^3}.

Questo metodo permette di separare la semantica dai vincoli sintattici e strutturali permettendo di condividere la semantica di diversi schemi e di promuovere l'interoperabilità semantica.

4.3.2 Uso di RDF con ODL_{I^3}

Esaminiamo questa problematica attraverso i concetti di ODL_{I^3} che presentano le maggiori difficoltà di traduzione in XML-Schema:

Interfacce

In ODL_{I^3} é possibile fornire alcune informazioni utilizzabili in fase di interrogazione da parte del Query Manager:

source : Tipo e nome della sorgente da cui è stata estratta la classe.

extent : Insieme di tutte le istanze della classe all'interno di un particolare database. La dichiarazione di un'estensione non è obbligatoria (una classe può non averne), tuttavia la sua presenza facilita il reperimento indicizzato (quindi rapido) delle informazioni da parte del DBMS.

Mapping Table : Ogni attributo globale ha almeno una regola di mapping, cioè un corrispondente locale sia esso un attributo, un valore di default o entrambe le cose.Tutte le regole di mapping sono raccolte nella Mapping Table, ne esiste una per classe globale, contenente il mapping fra grandezze globali e grandezze locali; consultandola é possibile tradurre qualsiasi query posta dall'utente.

Integrity Rules : Insieme di regole che permettono di definire ulteriori vincoli per l'appartenenza ad una classe in modo da aumentare le restrizioni applicabili ad una query e ridurre il costo del piano d'accesso.

Un modo di rendere disponibili tali informazioni e di facilitare la loro *reperibilità* é quello di definire un documento RDF:

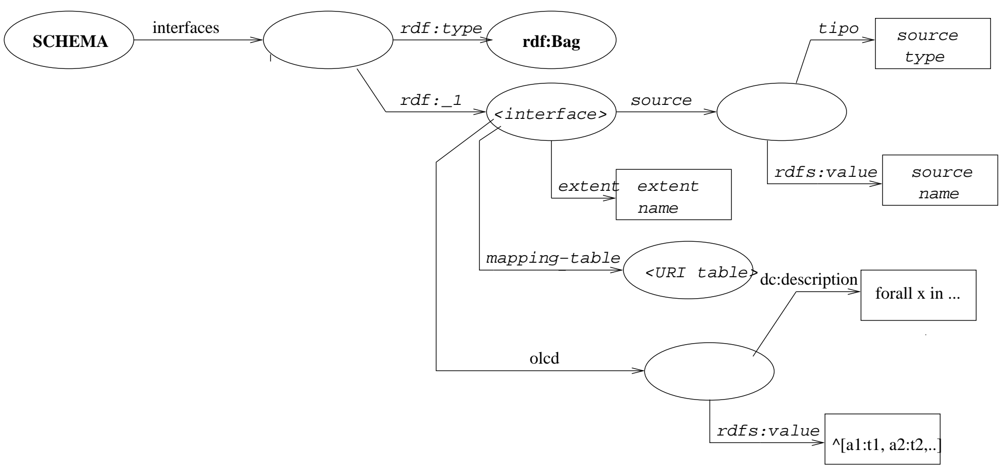


Figura 4.1: Schema RDF

il collegamento della definizione RDF di un'interfaccia nello schema XML-Schema può avvenire definendo un'attributo `interface`:

```
<xsd:attribute name="interface" type="anyURI"/>
```

`interface` é definito come attributo di tipo `anyURI` ciò indica la possibilità di contenere valori del tipo:

- http://sparc20.ing.unimo.it/interfaces.rdf.
- http://sparc20.ing.unimo.it/interfaces.rdf#Person.

ed utilizzarlo nella traduzione di una interfaccia:

```
<complexType name="PersonType"
             meta:interface="interfaces.rdf#Person">
.....
</complexType>
```

Lo **SCHEMA** é un'insieme d'interfacce ODL_{I^3}; ogni interfaccia ha le seguenti proprietà:

- source.
- extent.
- mapping-table.

- olcd.

La proprietà `source` è una proprietà *qualificata*, oltre al nome diamo anche il tipo della sorgente; una proprietà qualificata è una proprietà che oltre al valore vero e proprio porta con sè un informazione ”parte” del valore; che serve a meglio qualificare la proprietà (es: unità di misura, valuta,..).

La proprietà `extent` é una semplice stringa che rappresenta il nome dell’extent (nel caso di più extent é possibile ripetere più volte tale proprietà).

La proprietà `mapping-table` contiene l’URI della mapping table associata nel caso di interfaccia globale.

La proprietà `olcd` é una proprietà qualificata che riporta la traduzione in $OLCD$ dell’interfaccia e una descrizione per applicazioni non *OLCD-aware*; in questo modo riusciamo a trasmettere le informazioni su regole d’integrità e relazioni di tipo *is-a* che in OLCD sono integrate nella definizione di una classe:

- la rule: `rule Rule1 forall X in Restaurant: (X.category > 5) then X_tourist_menu_price > 100;` viene aggiunta alla descrizione della classe **FD.Restaurant**, viene così tradotta: $\sigma_p(FD.Restaurant) = \Delta([r_code : String, name : String,$ $street : String, zip_code : String, pers_id : Integer, special_dish : String, category : Integer, tourist_menu_price : Integer] \sqcap$ $(\neg(category > 5) \sqcup (tourist_menu_price > 100)))$

- Le relazioni del tipo $C_1\ isa\ C_2$ sono espresse in ODL_{I^3} da una rule del tipo: `rule Rule1 forall X in` C_1 `then X in` C_2; ed é integrata nella descrizione della classe C_1 usando il costrutto $\sqcap$: $\sigma_p(C_1) = C_2 \sqcap ...$

Il documento RDF ha associato uno schema-RDF che oltre ad elencare le risorse e le proprietà usate nel documento fornisce un aiuto alla leggibilità nello schema XML-Schema di alcuni concetti ODL_{I^3}.

Vorrei spendere qualche parola in più sul concetto di *leggibilità*; é importante garantire un elevato grado di non ambiguità tra schema ODL_{I^3} e schema XML-Schema per permettere ad eventuali applicazioni in grado di comprendere la logica descrittiva $OLCD$ di poter usufruire al 100% del supporto che questa DL da ad ODL_{I^3}. A mio avviso l’integrazione tra RDF(S) ed XML-Schema va in questa direzione.

Vediamo in dettaglio la definizione del documento e dello schema RDF affrontando la traduzione dei concetti ODL_{I^3} in cui RDF(S) viene utilizzato:

Relationship

Una relazione indica un legame tra due classi e comprende informazioni su tipo della classe con cui si é in relazione, la cardinalità ed informazioni sul percorso inverso presente nell'altra classe.

```
interface Professor: Person
{
 ....
 relationship set<Student> advises
              inverse Student::advisor;
 .....
};
```

Tale elemento in ODL_{I^3} introduce una corrispondenza binaria fra due risorse che in XML-Schema non è catturata:

```
<element name="advises">
 <complextype>
  <sequence>
   <element ref="Student" maxOccurs="unbounded"/>
  </sequence>
 </complextype>
</element>
```

La traduzione nell'interfaccia Student è:

```
<element name="advisor">
 <complextype>
  <sequence>
   <element ref="Professor"/>
  </sequence>
 </complextype>
</element>
```

potremmo introdurre nell'elemento che traduce il percorso diretto (anche in quello che traduce l'inverso) un'attributo con il riferimento ad una proprietà RDF che spiega la sua semantica:

```
<element name="advises"
         meta:sematica="schema.rdf/#advises"/>
```

mentre in RDF-Schema:

```
<rdf:Property ID="advises">
 <rdfs:label xml:lang="eng">advises</rdfs:label>
 <rdfs:label xml:lang="it">consiglia</rdf:label>
 <rdfs:comment>
   Relazione Binaria fra un professore
   e l'insieme dei suoi alunni
 </rdfs:comment>
 <rdfs:comment>
   L'inverso è advisor
 </rdfs:comment>
 <rdfs:seeAlso rdf:resource="#advisor"/>
</rdf:Property>

<rdf:Property ID="advisor">
 <rdfs:comment>
   Relazione fra uno studente e il suo professore
 </rdfs:comment>
 <rdfs:comment>L'inversa è advises</rdfs:comment>
 <rdfs:seeAlso rdf:resource="#advises"/>
</rdf:Property>
```

tramite `rdfs:seeAlso` informiamo che esiste una risorsa che aiuta a comprendere la definizione della proprietà mentre rdfs:comment serve per dare una spiegazione human-friendly.

Certo non é che la situazione sia risolta; ma a mio giudizio é preferibile all' uso di attributi o ulteriori elementi nel documento schema XML-Schema, che non porterebbero ad altro che ad un documento più prolisso e meno leggibile.

Ereditarietà Multipla

XML-Schema gestisce l'ereditarietà attraverso
l'elemento <extension base="..”/> tale meccanismo supporta solo l'ereditarietà singola; la gestione dell'ereditarietà multipla prevista da ODL_{I^3} comporta una traduzione in XML-Schema che non la evidenzia esplicitamente:

```
interface Engineer : Person, Graduate
{
  .....
}
```

La traduzione in XML-Schema é:

```
<complexType name="EngineerType">
 <extension base="Person">
  <element ref="Graduate"/>
</complexType>
```

in RDF-Schema:

```
<rdfs:Class rdf:ID="Person"/>

<rdfs:Class rdf:ID="Graduate"/>

<rdfs:Class rdf:ID="Engineer">
 <rdfs:label>Engineer</rdfs:label>
 <rdfs:subclassOf rdf:resource="#Person"/>
 <rdfs:subclassOf rdf:resource="#Graduate"/>
</rdfs:Class>
```

in questo modo é più chiaro, a mio avviso, la relazione di ereditarietà multipla fra Engineer, Person e Graduate.

Extent

L'extent rappresenta l'insieme di tutte le istanze della classe all'interno di un particolare database. La dichiarazione di un' estensione non è obbligatoria (una classe può non averne), tuttavia la sua presenza facilita il reperimento indicizzato (quindi rapido) delle informazioni da parte del DBMS.

```
interface Professor: Person
( extent professors
.....
```

Nel complexType che traduce la classe `Professor`:

```
<xsd:ComplexType name="ProfessorType"
meta:interface="interfaces.rdf#Professor">
.....
</xsd:ComplexType>
```

nel documento RDF:

```
<Professor>
<extent>professors</extent>
...
</Professor>
```

Source

La parola chiave source indica il tipo e il nome della sorgente dati da cui l'interfaccia ODL_{I^3} é stata estratta.

```
interface Address
( source semistructured Eating_Source )
 ...
```

Nel complexType che traduce la classe `Address`:

```
<xsd:ComplexType name="AddressType"
meta:interface="interfaces.rdf#Address">
.....
</xsd:ComplexType>
```

nel documento RDF:

```
<Address>
<source
schema:tipo="semistructured"
rdf:value="Eating_Source"/>
...
</Address>
```

Integrity Rules

Le regole d'integrità sono integrate nella definizione $OLCD$ di una classe, quest'ultima viene riportata attraverso la proprietà `olcd`, riprendiamo l'esempio della sezione sec:int: `rule Rule1 forall X in Restaurant: (X.category > 5)`
`then X_tourist_menu_price > 100;` viene riportata in *interfaces.rdf* nel seguente modo:

```
<Restaurant>
 <...> .... </...>
 <..> .. </..>
 <olcd dc:description=" "
       rdf:value="
```

$\sigma_p(FD.Restaurant) = \Delta([r_code : String,$
$name : String, street : String, zip_code : String, pers_id : Integer, special_dish :$
$String, category : Integer, tourist_menu_price : Integer]$
$\sqcap (\neg(category > 5) \sqcup$
$(tourist_menu_price > 100)))$

```
"/>
</Restaurant>
```

in questo modo un'applicazione $OLCD{-}aware$ é in grado di capire le limitazioni imposte su una classe ODL_{I^3} le altre applicazioni si dovranno accontentare in una descrizione della regola.

Il valore dell'attributo `dc:description` é stato lasciato vuoto perchè ritengo necessario un intervento del progettista per inserirlo manualmente. In questo esempio abbiamo usato una proprietà definita nel vocabolario del **Dubline Core Initiative**; (`dc:description`) un'iniziativa nata per facilitare il ritrovamento di risorse elettroniche in un modo simile a quello che si farebbe in una biblioteca.

Mapping Table

Ogni attributo globale ha almeno una regola di mapping, cioè un corrispondente locale sia esso un attributo, un valore di default o entrambe le cose.

Tutte le regole di mapping sono raccolte nella *Mapping Table*, ne esiste una per classe globale, contenente il mapping fra grandezze globali e grandezze locali; consultandola é possibile tradurre qualsiasi query posta dall'utente, e reperire ogni informazione richiesta.

```
interface Hospital_Patient
{
attribute long code
mapping_rule ID.Patient.code, ID.Dis_Patient.code;

attribute string name
mapping_rule ( ID.Patient.first_name
and ID.Patient.last_name),
CD.Patient.name;
}
```

nel documento RDF:

```
<Hospital_Patient>
 <mapping-table
  rdf:resource="http://a.b.c/Hospital_Table"/>
...
..
.
</Hospital_Patient>
```

Riepilogo

In sostanza i documenti RDF(S) da creare sono 2:

1. Il documento RDF *interfaces.rdf* che contiene per ogni interfaccia l'indicazione di sorgente extent e mapping table e le regole d'integrità associate a tutte le interfacce.

2. Il documento RDF-Schema *schema.rdf* che contiene la definizione RDF delle interfacce tradotte,usate in *interfaces.rdf*, e delle proprietà che traducono le `relationship`.

Questi due documenti fanno uso delle risorse definite nello schema *momis-schema* che contiene la definizione delle proprietà `source,extent, mapping-table,...` cioè quelle risorse comuni a tutti i processi d'integrazione e non solo ad uno in particolare.

Il documento XML-Schema risultante dal processo di traduzione invece fà uso del documento *meta.xsd*
in cui sono definiti gli attributi: `interface,semantica e int-rules`

L'uso di *meta.xsd* nello schema prodotto dal traduttore avviene attraverso l'uso di <import> perchè *meta.xsd* ha un diverso `targetNameSpace`.

N.B. Il nome dei documenti RDF e RDF-Schema prodotti e' interfaces.rdf e schema.rdf, nulla vieta di dar loro nomi diversi, in questo caso bisogna modificare i riferimenti a tali files presenti nello schema XML-Schema e nel documento RDF in quanto il sw usa questi nomi.

Nota Implementativa

L'introduzione degli attributi definiti in meta.xsd è più indicata come attributi del complexType che traduce un interfaccia piuttosto che nel suo content-model perché rappresentano caratteristiche della classe in quanto tale non di ogni singola istanza; un pò lo stesso concetto che si ha nei linguaggi OO tra variabili di classe e variabaili istanza.

4.4 Il traduttore

4.4.1 Introduzione API

Per realizzare il traduttore ODL_{I^3} $\rightarrow$XML-Schema mi sono avvalso delle nuove API della SUN per il parsing: le JAXP1.1 (**J**ava **A**PI for **X**ML **P**rocessing) che forniscono il supporto per utilizzare gli standard SAX, DOM e XSLT.

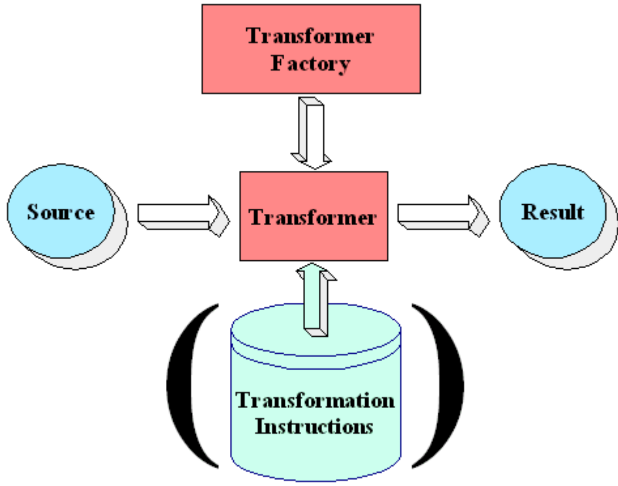


Figura 4.2: XSLT APIs

Le API Java SAX (Simple Api for XML) forniscono un meccanismo di accesso seriale ed *event-driven* per il parsing di file XML.

Le API DOM forniscono una rappresentazione ad albero in memoria del file XML elaborato, queste API richiedono un maggior consumo di risorse sia in termini di CPU che di memoria rispetto alle SAX.

Infine le API XSLT permettono di scrivere documenti XML in un file, di convertirli in un altro formato oppure usandole insieme alle API SAX di convertire una generica struttura dati in XML. *Quest'ultima opportunità é stata quella da me sfruttata per realizzare il traduttore*.

Visto il ruolo chiave rivestito da queste API nello sviluppo del traduttore vorrei spendere qualche parola sulle specifiche XSL:

La necessità di trasformare documenti XML sta diventando un'esigenza sempre più stringente con l'aumentare dell'uso di XML come standard per la distribuzione di informazioni e di conseguenza dei dispositivi in grado di leggere i nostri dati; queste considerazioni hanno spinto il W3C a rilasciare le specifiche su XSL(*eXtensible Stylesheet Language*) composte da due parti: XSLT(*eXtensible Stylesheet Language Transformation*) e XSL-FO(*eXtensible Stylesheet Language formatting Object*).

XSLT é relativa alle trasformazioni di un documento XML in un qualunque altro documento comunque di testo, mentre XSL-FO descrive trasformazioni di un documento XML in formati binari(ad esempio il PDF).

Vediamo in figura 4.2 il funzionamento di queste API:

Il Trasformer prende come input un oggetto Source che può essere creato da un SAX reader, DOM reader o da uno stream di input; l'oggetto Result è l'output del processo di trasformazione, questo oggetto può essere un gestore di eventi SAX, un DOM o uno stream di output (ad esempio un file).

Al Trasformer possono essere associate delle istruzioni di trasformazione; se tali istruzioni non sono presenti si parla di trasformazione di identità, in quanto l'oggetto Source viene copiato nell'oggetto Result.

4.4.2 Generare XML da una struttura dati arbitraria

Il traduttore si compone di due entità:

1. Un parser ODL_{I^3} che genera eventi SAX (startElement, endElement,...).
2. Un *Trasformer-XSLT* che prendendo in ingresso il *parser modificato* genera il documento schema.

Per quanto riguarda il primo passo mi sono basato sul parser realizzato da Zanoli[39], le modifiche da me fatte sono consistite nell'inserimento delle invocazioni degli opportuni eventi SAX (startElement e endElement) nelle varie *actions* della grammatica BNF di ODL_{I^3} e nell'implementazione dell'interfaccia `XMLReader` (sopratutto del metodo `parse(InputSource)`) in modo che il parser ODL_{I^3} venga visto come parser di tipo SAX.Poi con $Byacc$ si ottiene il codice java che effettua i controlli sintattici del testo specificati in BNF e lascia inalterate le actions.

Esaminiamo, piú in particolare, le estensioni fatte al parser ODL_{I^3} esaminando la gerarchia delle classi di figura 4.3:

Consideriamo il processo di traduzione di un'interfaccia; essa ha associata una lista di *IntBody*; ogni body é costituito dall'insieme delle dichiarazioni di tipi di dati, attributi, constanti, relazioni e metodi che ne fanno parte. A differenza di ODL standard, nel linguaggio ODL_{I^3} é prevista la definizione di piú implementazioni per una stessa classe. Questa caratteristica consente l'acquisizione di dati semistrutturati che, per definizione, non sono tenuti a seguire una rigida formattazione, come invece avviene per i dati strutturati.

Gestione Attributi

Quando si incontra la dichiarazione di un'attributo, per prima cosa si controlla se é la prima dichiarazione del Body oggetto della traduzione (_body == true):

```
if((_handler != null) && _body)
{
```

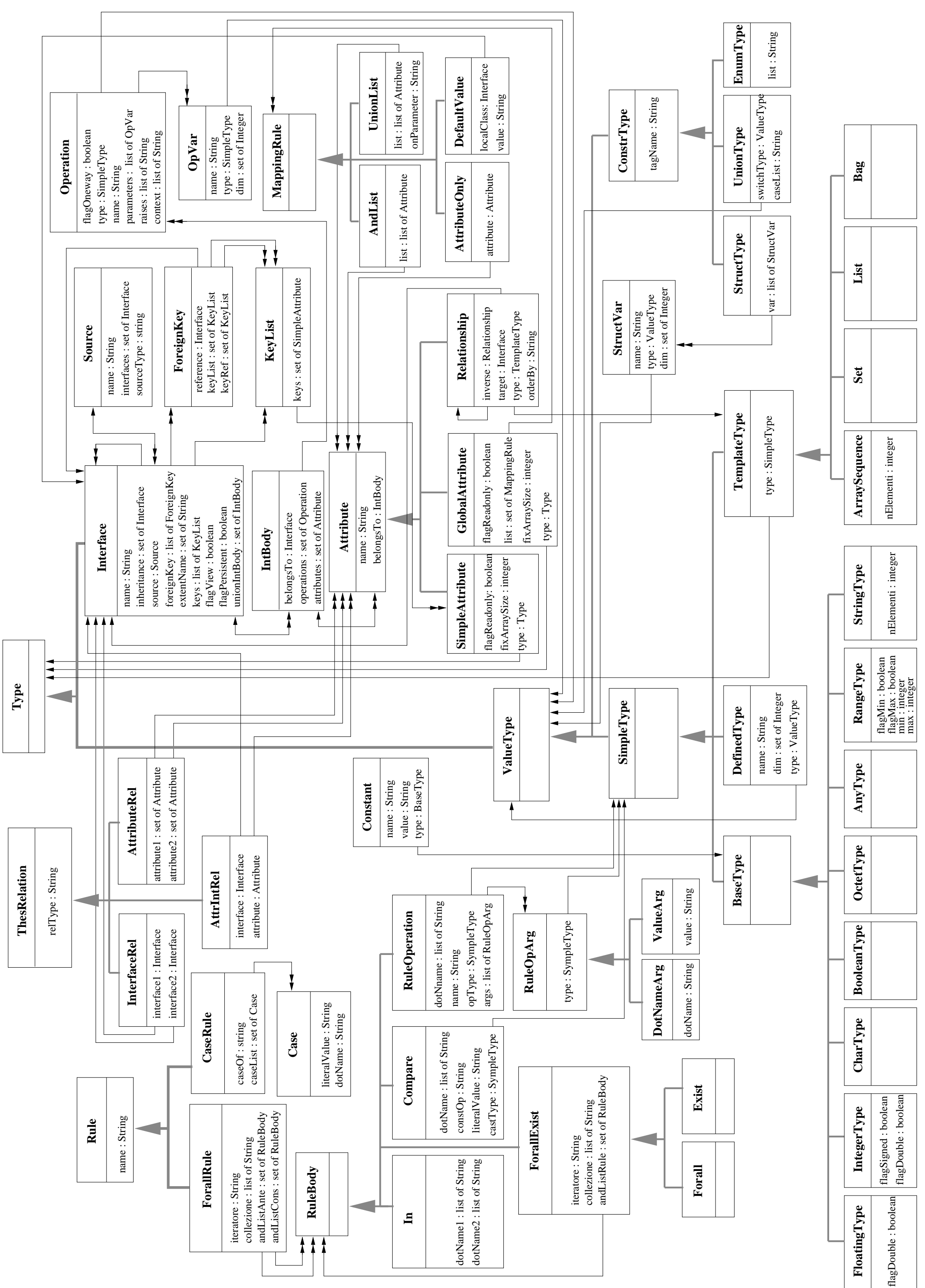


Figura 4.3: L’object model del linguaggio ODL_{I^3}

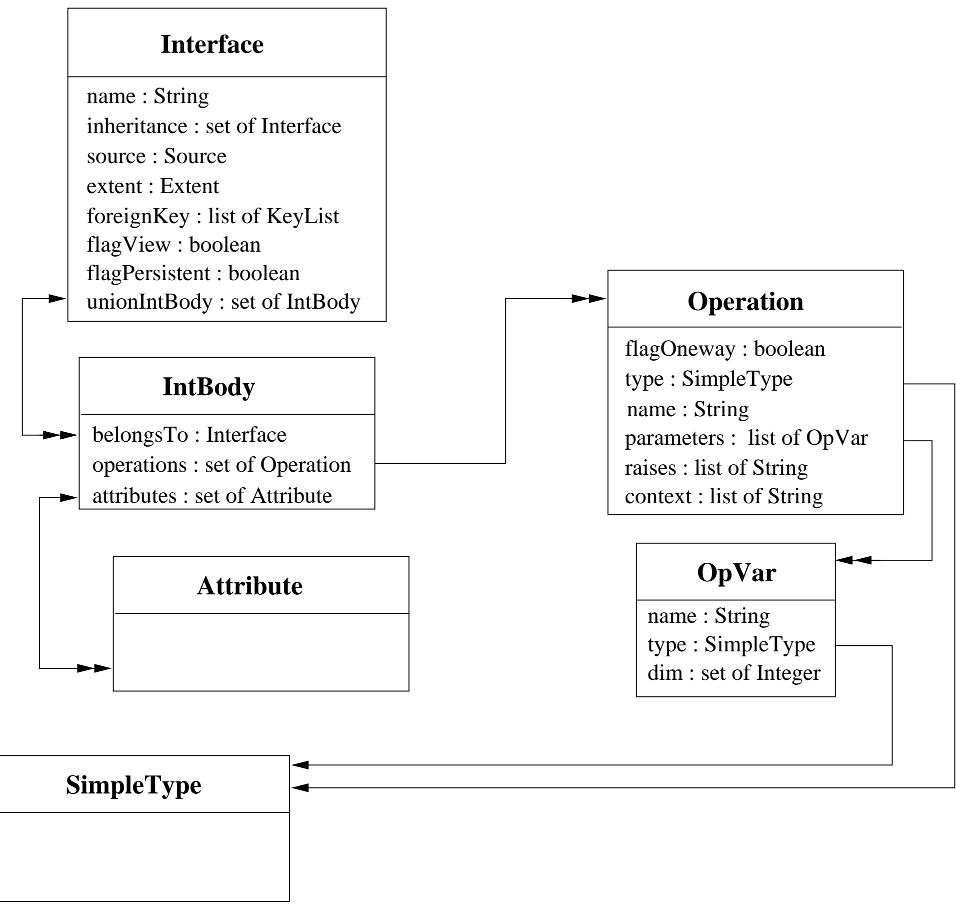


Figura 4.4: Interface Body

```
 _body = false;
 _fuoribody=false;
 _atts.clear();
 _atts.addAttribute(_nsu,"name","name",_type,"body"+_name+_nbody);
 nl("group",_atts,false);
 _atts.clear();
 nl("sequence",_atts,false);
}
```

in tal caso si procede con la definizione di un `group` che andrá a contenere tutte le dichiarazioni del body e il cui nome é

```
"body"+<nome dell'interfaccia a cui appartiene il body>
      +<numero del body dell'interfaccia>
```

contestualmente si avverte che stiamo traducendo un body e che le dichiarazioni di tipi di dato interni al body non vanno tradotti immediatamente (_fuoribody=false).

Il risultato per ora é del tipo:

```
<group name="body....">
 <sequence>
```

Dopo si procede alla traduzione dell'attributo esaminando il suo dominio; ODL (e quindi ODL_{I^3}) prevede una distinzione fondamentale nei tipi; un tipo può essere:

- **Tipo valore**
- **Tipo classe**, chiamato comunemente **Classe**

I tipi valore sono propri delle variabili e degli attributi semplici, infatti, a differenza delle classi, ogni istanza di questo tipo è priva di identificatore, cioè ha come unica proprietà il suo valore.
Al contrario gli attributi complessi e gli oggetti in generale sono istanze di classi, con OID, interfaccia e comportamento.

la figura 4.5 mostra come questa differenza si manifesta nell'**Object Model**.

I *ValueType* si suddividono in:

- **SimpleType**
- **EnumType**
- **ConstrType**

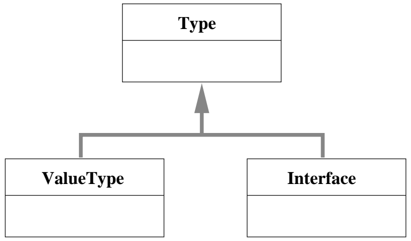


Figura 4.5: I tipi in ODL_{I^3}

I tipi semplici sono tutti i tipi atomici base, cioè i tipi predefiniti (integer, float, char, boolean, etc...), i vari tipi collezione **TemplateType** (set, list, bag, array), nonché i tipi nuovi **DefinedType**: secondo la sintassi ODL la definizione di nuovi tipi segue le stesse regole del linguaggio **ANSI C**.

Gli attributi avente come dominio un tipo predefinito sono gestiti dal metodo `defBaseElement` il quale procede nel seguente mdo:

1. Ricava il nome del tipo XML-Schema che traduce il tipo ODL_{I^3}.
2. Definisce un `element` avente come `type` il tipo ottenuto al passo precedente.

Il primo passo é ottenuto tramite la funzione `defBaseType` che nel caso di attributo con una dimensione associata provede ad inserire in una HasMap `_tmplist` le informazioni necessarie per la definizione di un tipo lista: il nome da dare al `listType` e il `simpleType` da cui derivare il tipo lista. Nel caso di attributo di tipo range con una dimensione > 1 si inserisce una entry nella struttura `_tmprange` per la definizione di un `simpletype` che traduca il range (nome del tipo ed estremi del range separati da un ":") un'altra entry é fatta in `_tmplist` per la definizione di un lista del tipo che traduce il range.

Il secondo passo é gestito dalla funzione `Elemento` che effettua un controllo sulla natura opzionale dell'attributo e se necessario memorizza in `_tmparray` le informazioni necessarie per la definizione di un array: il nome da dare al simpleType (che contiene l'informazione sulla dimensione dell'array es: arraystring3 $=>$ dimensione $= 3$)) che tradurrá l'array e il nome del listType su cui basarsi per la definizione dell'array.

Se l'attributo non ha una dimensione associata ma é di tipo range viene tradotto dalla funzione `Range` che definisce un `element` con un tipo semplice anomimo:

```
<element name="...">
```

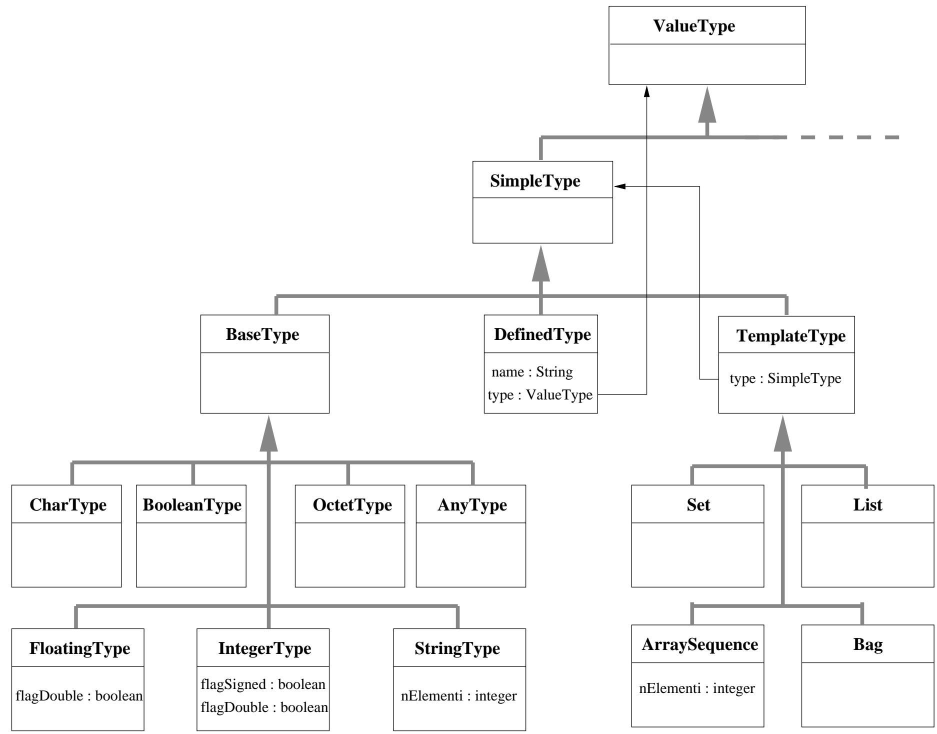


Figura 4.6: I tipi semplici

```
 <simpleType> <- TIPO SEMPLICE ANONIMO
  <restriction base="positiveInteger">
   <minInclusive value=".."/>
   <maxInclusive value="..."/>
  </restriction>
 </simpleType>
</element>
```

I tipi collezione(set, list,array,...) vengono gestiti dalla funzione `defTemplElement`, la quale riceve in ingresso il riferimento al dominio dell'attributo (il TemplateType), il nome dell'attributo e l'indicazione sull'opzionalitá dell'elemento da creare. Dal primo parametro si individua il tipo di Template: `ArraySequence` o `Collection` il primo ha associato una dimensione `nElementi` il secondo no.

- le collezioni di tipi predefiniti (es: set¡float¿) sono gestiti dalla funzione `defBaseElement` a cui, a cui nel caso di Collection, si dá l'indicazione che si dovrá definire un `listType` ponendo a `true` il valore dell'ultimo parametro di `defBaseElement`: `isCollection`.

- i Template di `DefinedType` (es: set¡Author¿) vengono gestiti dalla funzione `ArrayCompl` che crea un `element` avente come contenuto un elemento del tipo il `DefinedType` e riportate l'attributo `maxOccurs` che indica il numero delle volte che puó apparire tale elemento o `"unbounded"` se si tratta di Collection.

Per quando riguarda i `ConstrType` Fanno parte di questa classe i tipi **StructType**, **UnionType** e **EnumType**.

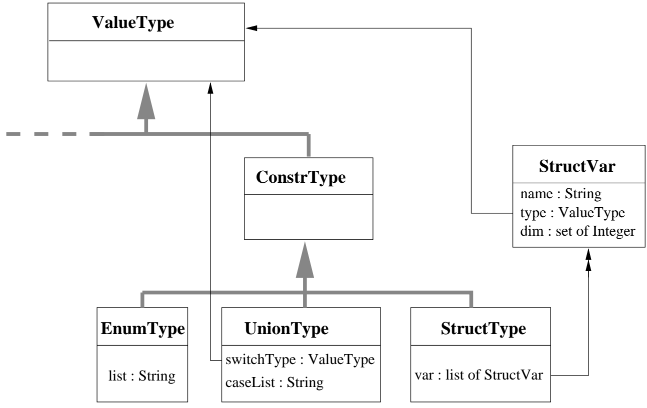


Figura 4.7: ConstrType

Gli attributi di tipo `EnumType` vengono gestiti dal metodo `defEnumElement` che definisce un `element` con un tipo semplice anonimo, definito dal metodo `defEnumType`, contenente la lista dei valori che la variabili puó assumere.

Gli attributi di tipo `StructType` vengono tradotti come elementi con un complexType anonimo avente tanti elementi locali quanti sono i campi della struct; la traduzione dei campi avviene nel metodo `defStructBody` il quale riceve in ingresso l'oggetto `StructType` contenente i campi della struct.

Gli attributi `UnionType` non sono stati trattati perché non molto usati e per la scarsa precisione con cui sono gestiti dal parser che restituisce solo una stringa con il contenuto del tipo.

N.B. Gli attributi globali (`GlobalAttribute`) sono trattati nello stesso modo degli attributi locali in quanto il riferimento alla mapping table viene inserito nella fase di creazione del documento RDF.

Gestione Costanti

Le costanti sono caratterizzate da tre attributi:

- Nome della costante
- Tipo della costante, deve essere un oggetto `BaseType`
- Valore della costante.

vengono tradotte in XML-Schema attraverso un elemento dal contenuto costante direttamente nell'action della grammatica.

```
<element type=.. fixed=<valore della costante> />
```

anche nella gestione delle costanti, come per gli attributi, se siamo alla prima dichiarazione del body si definisce l'inizio di un `group`.

Gestione Relazioni

Per quanto riguarda le Relationship la traduzione avviene direttamente nell'action della grammatica, si traduce il solo percorso diretto con un elemento il cui contenuto e il riferimento all'elemento globale che traduce l'interfaccia con cui esiste la relazione, inoltre si inserisce il legame con la definizione di una proprietá RDF-Schema che tenta di tradurre il concetto di `inverse`.

Gestione Tipi di dato

Un tipo definito ha un nome, un tipo mappato (tipo-valore, non puo essere tipo-classe) e, nel caso sia un array, un insieme di valori interi che ne indicano le dimensioni; il fatto che il tipo riferito sia un generico **ValueType** e non necessariamente un **SimpleType**, consente la dichiarazione di variabili, ad esempio strutturate, in modo semplice: prima si definisce un tipo nuovo, poi le variabili secondo la sintassi dei tipi atomici. Per quanto riguarda i **TemplateType**, essi rappresentano collezioni di istanze tutte omogenee tra loro, in particolare di tipo **SimpleType**; di conseguenza non è possibile dichiarare direttamente una variabile come set di $struct$, lo si può fare in due passi, attraverso la definizione di un tipo nuovo, come mostra l'esempio che segue:

```
typedef struct {
           int i,j;
           float f;
        } tipostruct;

set<tipostruct> var1,var2;
```

Se la definizione del tipo di dato é esterna alle interfacce, si procede direttamente alla traduzione del tipo di dato definendo un `simpleType` o un `complexType` a seconda della definizione (le regole che si seguono solo le stesse spiegate per i domini degli attributi).

Nel caso di tipo di dato interno al body di un'interfaccia non si procede immediatamente alla traduzione del tipo di dato ma si inseriscono le informazioni necessarie in apposite strutture che verrano esaminate alla fine della traduzione di tutti i body dell'interfaccia.

N.B Nel caso di tipi di dato con piú di una dimensione si é adottato un palliativo: si traduce il tipo di dato come un `complexType` che puó contenere qualsiasi cosa (tipo `anyType`) e avente un attributo che indica il vero dominio dell'elemento:

```
int i2[2][5]

 <complexType name=<nome tipo>
  <sequence>
   <element name="content" type="anyType"/>
  </sequence>
  <attribute name="tipo" type="string" fixed="int[2][3]"/>
 </complexType>
```

questo stesso procedimento é utilizzato in `defStructBody` per i campi di una struct con associate piú di una dimensione.

Nel caso di tipi di dato interni al body questi prendono come nome:

```
nome-interfaccia+nome-tipo+"Type"
```

nel caso, invece, di tipo globale il nome é dato solamente da nome-tipo+"Type".

Dopo che sono stati tradotti tutti i body di un interfaccia abbiamo altrettanti `group`, a questo punto nella action `interfacedcl` si procede alla definizione del `complexType` che traduce l'interfaccia gestendo la presenza di superclassi e di piú bodies, poi si generano tutti i tipi di dato che la traduzione delle dichiarazioni dei bodies hanno richiesto attraverso:

- defList: per definire i `listType` esaminando il contenuto dell'Hashmap `_tmplist`.
- defArray: per definire dei `listType` con il vincolo nel numero di elementi che puó contenere, esaminando il contenuto dell'Hashmap `_tmparray`.
- defRange: per definire tipi di dato range, esaminando il contenuto dell'Hashmap `_tmprange`.

- defEnu: per definire tipi di dato enumerativi (typedef enum...,... xx;), esaminando il contenuto dell'Hasmap `_enu`.

- defConstrtype: per definire tipi di dato struct, esaminando il contenuto dell'Hashmap `_Struct`.

- defArraycompltype: per definire tipi di dato che rappresentano un array o una collection di un tipo classe o un tipo definito, esaminando il contenuto della struttura `_Arraycompltype`.

- defMultidimtype: per definire tipi di dato a 2 o piú dimensioni, esaminando il contenuto dell'Hashmap `_Multidimtype`.

- defChar: definisce il tipo di dato carattere.

il `complexType` viene definito per rendere immediata la definizione di un attributo di tipo classe che altro non é che un `element` con l'attributo `type` uguale al nome del complexType del tipo classe.

Dopo si passa alla definizione di un elemento globale avente come nome quello dell'interfaccia e come tipo il complexType appena definito; in questo modo l'interfaccia é referenziabile in una relationship. In questa fase si procede alla traduzione delle chiavi primarie, candidate e delle foreignkey attraverso il metodo `defConstraint`.

Quando la fase di parsing dello schema é finita, all'interno del metodo `parse(InputSource)` si creano i documenti RDF-Schema ed RDF associati al documento XML-Schema; interrogando la variabile `odli3Schema` di tipo `Schema` che contiene i riferimenti a tutte le interfacce parserizzate.

Il risultato finale é un'applicazione che se eseguita da sola si comporta come il parser ODL_{I^3} realizzato da Zanoli vedi[39] mentre se utilizzata dal *Trasformer-XSLT* é vista come sorgente SAX.

Il trasformer usa il *parser modificato* come sorgente di tipo SAX; quando il trasformer viene lanciato si configura come gestore degli eventi SAX lanciati dal *parser modificato* (diventa il *contentHandler*) nel nostro caso il trasformer si limita a passare i dati in ingresso all'oggetto Result scelto. Io ho scelto uno StreamResult cioè un file.

Il risultato finale del trasformer è la creazione di un unico file (lo StreamResult) contenente:

1. Il documento schema in linguaggio XML-Schema (.xsd).

2. schema.rdf.

3. interfaces.rdf.

Il documento schema (il *.xsd*) é creato durante la fase di parsing delle interfacce ODL_{I^3} mentre i documenti *interfaces.rdf* e *schema.rdf* sono creati alla fine interrogando la variabile di tipo *Schema* che contiene i riferimenti a tutte le interfacce parserizzate. È opportuno, terminata la trasformazione, intervenire sui files RDF-Schema e RDF per inserire dei commenti.

Capitolo 5

XQUERY

L'uso di XML per memorizzare e scambiare informazioni é sempre più frequente, da ciò l'esigenza di poter interrogare tali sorgenti di dati. Non è difficile immaginare che sempre più sorgenti d'informazioni organizzerano le loro viste come fonti di dati XML indipendentemente da come sono internamente memorizzati i dati(relazionale, oggetti, ecc...) secondo alcuni non siamo molto lontani dal vedere *Internet come un grande e distribuito database XML*. Una delle caratteristiche principali di XML é la sua flessibilità nella rappresentazione di diversi tipi di informazioni, da quelle memorizzate in un database a quelle contenute in un file di testo; un linguaggio di interrogazione deve perciò mantenere tali caratteristiche, essere in grado di ritrovare informazioni da queste sorgenti.

Il lavoro svolto dal XML Query Working Group é stato quello di definire un modello dei dati per documenti XML, un algebra basata su tale modelllo ed un linguaggio di interrogazione che permetta di esprimere tutte le interrogazioni formulabili con l'algebra definita.

5.1 Scenari di utilizzo

Gli scenari nei quali é utile un linguaggio d'interrogazione per documenti XML sono molteplici:

- Eseguire query su singoli o collezioni di documenti, per ritrovare singoli documenti, tabelle dei contenuti, per generare nuovi documenti come risultati di una query.

- Eseguire query document-oriented e data-oriented su documenti che contengono dati quali cartelle cliniche, cataloghi di prodotti, ecc..

- Eseguire query su file di configurazione, files di log rappresentati in XML.

- Eseguire query su strutture DOM che ritornino insiemi di nodi.

- *Eseguire interrogazioni sulla rappresentazione XML di dati contenuti in una generica sorgenti di dati per estrarre informazioni, per rappresentare le informazioni estratte in formato XML, o per integrare informazioni provenienti da sorgenti eterogenee. Tutto ciò indipendentemente dal fatto che la rappresentazione in XML delle informazioni sia fisica o virtuale.*

5.2 Database Desiderata per un linguaggio d'interrogazione per XML

XML è un linguaggio usato per i più svariati scopi e da differenti comunità: ricercatori, aziende, ecc...; é naturale che molte e diverse tra loro siano le esigenze al momento in cui si vuole implementare un linguaggio d'interrogazione per dati XML.

XML Output La risposta ad una query deve essere in formato XML. In questo modo si ottengono molti vantaggi: si possono definire viste attraverso una query; trasparenza da parte dell'applicazione che esegue la query e che non deve stare a preoccuparsi se stà esaminando una vista o un database.

Server-side Processing il linguaggio di query deve essere utilizzabile per processi lato server. Quindi una query non deve essere dipendente dal contesto in cui é stata lanciata.

Operazioni Selezione, estrazione, riduzione, ristrutturazione e combinazione devono essere operazioni esprimibili in una query. Questa richiesta é anche dovuta alla necessità di una query di essere eseguita in remoto; se é possibile esprimere più operazioni in una sola query non siamo costretti a lanciare query parziali che ritornano una gran mole di dati che devono essere processati localmente. Alcuni di quete operazioni riducono di molto il volume di dati di output riducendo così il traffico di rete. Inoltre avendo a disposizione informazioni su quali dati accedere e modificare si ottiene un più efficente ottimizzazione delle query e di valutazione del piano di accesso al database.

Selezione Scegliere un documento in base al suo contenuto, alla sua struttura o ai suoi attributi.

Estrazione Filtrare da un documento alcuni elementi mantenendo i rapporti di gerarchia fra questi.

Riduzione Non considerare eventuale nodi figli di un elemento.

Ristrutturare Costruire un nuovo elemento con i dati estratti da una query.

Combinare Unire due o più elementi.

Schema Opzionale Il linguaggio di query deve poter gestire documenti XML che non hanno un DTD o schema associato. Il linguaggio di query deve poter cioè gestire documenti di cui non conosce la struttura a priori.

Sfruttare uno Schema Quando é a disposizione un DTD o uno schema, deve essere possibile valutare se la query é ben formulata in base al DTD (o schema) e calcolare un DTD per l'output. Queste capacità permettono di rilevare eventuali errori prima ancora di lanciare la query e alle applicazioni di manipolare il risultato di una query.

Preservare l'ordine e le associazioni I documenti XML sono caratterizzati da un preciso ordine di apparizione degli elementi e dalla presenze di nodi padri e figlio, una query deve mantenere tali relazioni perchè potenzialmenete rilevanti nella comprensione del contenuto di un documento.

Interazione con GUI Le query devono poter essere create e modificate da programmi. La maggior parte delle query non saranno scritte dagli utenti,ma questi useranno delle interfacce che permettono lorio di esprimersi col linguaggio naturale.

Rappresentazione XML Deve essere possibile esprimere una query in XML. Tra le molteplici sintassi con le quali si puó pensare di scrivere una query, XML non deve mancare. Questo comporta che non c'é bisogno di particolari meccanismi per il trasporto e memorizzazione delle query.

Mutua Inclusione Una query deve poter essere compresa all'interno di un documento XML e qualsiasi porzione di documento XML deve poter essere espresso in una query.

XLink e XPointer Il linguaggio di query deve essere familiare con link di tipo XLink e XPointer.

Indipendenza dai Namespace Il linguaggio di query non deve essere dipendente dai namespace dichiarati in un documento, in caso contrario sarebbe impossibile gestire query lanciate su più documenti che usano lo stesso DTD con alias diversi.

supporto per i tipi di dato Il linguaggio di query deve poter essere in grado di gestire operaioni specifiche di certi tipi di dato.

Gestione di Metadata Il linguaggio di query deve poter operare anche sui metadati, su quei dati che descrivono i dati del documento.

Semantica Formale Deve essere fornita una precisa semantica del linguaggio in modo che il significato di una query sia lo stesso indipendentemente dal contesto in cui viene lanciata. Inoltre query che producono lo stesso output devono poter apparire negli stessi contesti.

5.3 Modello dei dati

Un modello dei dati é l'insieme dei concetti usati per descrivere un insieme di dati, le loro associazioni, e le operazioni che agiscono sui dati stessi. Dato che i documenti XML hanno una struttura ad albero, il modello dei dati di XML Queryé basato sul concetto di albero con nodi etichettati e con il concetto di identità del nodo per gestire i riferimenti XML (IDREF, ecc..).

5.3.1 tipi di Nodo

Il modello dei dati si basa sul concetto di nodo. In XQuery abbiamo nove tipi di nodo:

- document.
- element.
- value.
- attribute.
- namespace (NS).
- processing instruction (PI).
- comment.
- information item.
- node reference.

```
Node =  DocNode | ElemNode | ValueNode   | AttrNode  |  NSNode  | PINode
        | CommentNode | InfoItemNode | RefNode
```

Oltre ai nodi, il modello dei dati supporta due tipi di collezioni: le liste (ordinate) e gli insiemi (non ordinati), le collezioni non possono essere innestate.

5.3.2 Tipi di dato

I nodi contengono valori appartenenti al dominio di un tipo di dato definito da XML-Schema.

```
SchemaType ::= SimpleSchemaType | ComplexSchemaType
```

Un `simple type` può essere un tipo primitivo (string, boolean, float, double, ID, IDREF) o derivato. Un complex type definisce il contenuto e la struttura di un elemento.

5.3.3 Esempio

Consideriamo il seguente documento XML:

```
<?xml version=1.0?>
 <p:part xmlns:p="http://www.mywebsite.com/PartSchema"
         xsi:schemaLocation = "http://www.mywebsite.com/PartSchema
                               http://www.mywebsite.com/PartSchema"
          name="nutbolt">

  <mfg>Acme</mfg>
  <price>10.50</price>
 </p:part>
```

lo schema associato:

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

 <xsd:element name="part" type="part_type">
  <xsd:complexType name="part_type">
   <xsd:element name = "mfg" type="xsd:string"/>
   <xsd:element name = "price" type="xsd:decimal"/>
   <xsd:attribute name = "name" type="xsd:string"/>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Una descrizione grafica dell’istanza del modello dei dati (il documento XML):

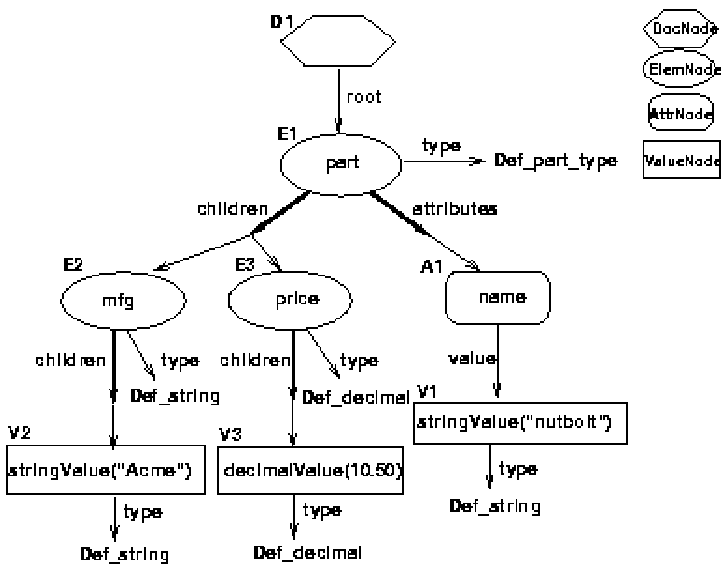


Figura 5.1: Modello dei dati XQUERY

In questo caso il documento XML ha associato uno schema, ma il modello dei dati é in grado di gestire documenti senza schema o DTD l'importante é che siano well-formed.

N.B: Il modello dei dati attualmente non rappresenta i concetti di primary key e di foreign key come definiti da XML-Schema; questi concetti rappresentano due delle direzioni in cui verrà esteso il modello dei dati.

5.4 Il Linguaggio d'interrogazione per XML

XML é un linguaggio di markup estremamente versatile, in grado di rappresentare il contenuto di diverse sorgenti dati: database relazionali, ad oggetti, dati semi-strutturati, ecc.. XQuery, é stato progettato per essere applicabile su tutti questi tipi di sorgenti di dati XML.

XQuery deriva da Quilt un linguaggio d'interrogazione per XML, il quale a sua volta prende molte delle sue caratteristiche da altri linguaggi:

1. Sintassi per navigare all'interno della struttura di un documento simile a quella di XPath e XQL.
2. Definizione di variabili globali o locali da usare all'interno di una query, preso da XML-QL.
3. Query come insieme di proposizioni come in SQL.
4. Definizione di funzioni da OQL.

In XQuery una query é rappresentata come una espressione. XQuery supporta diversi tipi di epressioni:

1. Espressioni di percorso.
2. Costruttori di elemento.
3. Espressioni FLWR (espressioni FLoWer).
4. Espressioni con operatori e funzioni.
5. Espressioni condizionali.
6. Quantificatori.
7. Filtraggio.
8. Tipi di dato.
9. Funzioni.
10. Tipi di dato definiti dall'utente.
11. Operazioni su tipi di dato.

5.4.1 Espressioni di percorso

XQuery usa una sintassi simile a quella di XPath; un'espressione di percorso é un insieme di passi, ogni passo rappresenta un movimento all'interno della struttura di un documento,il risultato di ogni passo é una lista di nodi.

.	Il nodo corrente
..	Il padre del nodo corrente
/	Il nodo radice o il separatore tra diversi passi dell'espressione di percorso
//	I figli del nodo corrente
@	Gli attributi del nodo corrente
*	Qualsiasi nodo
[ ]	Parentesi che che contengono un'espressione booleana per limitare i nodi selezionati in un certo passo
[n]	Seleziona un elemento figlio da una lista di elementi

Tabella 5.1: Espressioni di Percorso

```
document("zoo.xml")/chapter[2]//figure[caption = "Tree Frogs"]
```

In aggiunta alla sintassi di XPath, XQuery ha il predicato RANGE preso da XQL:

Trova tutti gli elementi `figure` nei capitoli 2,3,4 e 5 del documento zoo.xml:

```
document("zoo.xml")/chapter[RANGE 2 TO 5]//figure.
```

Un altro operatore aggiunto in XQuery rispetto a XPath é l'operatore di deferenza ("$->$"). Esso si applica ad attributi di tipo IDREF e ritorna l'elemento referenziato, l'operatore é seguito da un nome che indica il nome dell'elemento obiettivo:

Trovare i titoli delle immagini referenziate dagli elementi $< figref >$ nei capitoli di "zoo.xml" e aventi titolo "Frogs":

```
document("zoo.xml")/chapter[title = "Frogs"]//figref/@refid->fig/caption
```

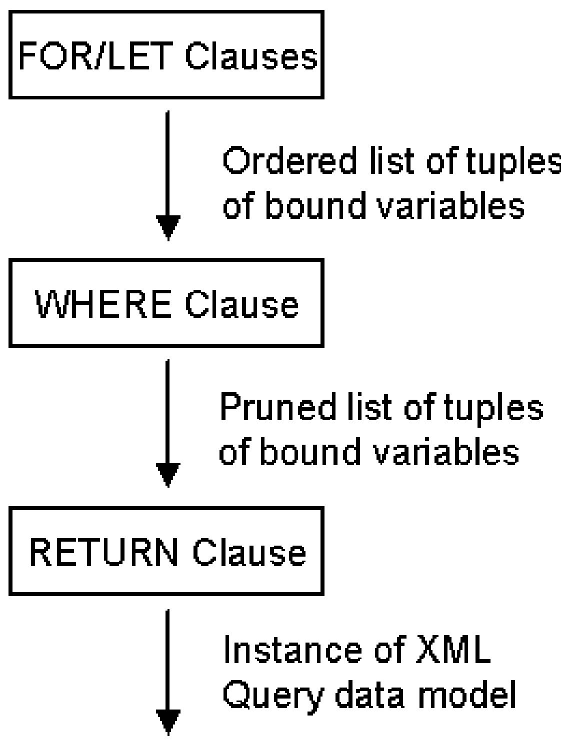
5.4.2 Costruttori di elemento

Un espressione XQuery può incapsulare il risultato di un'interrogazione, in un nuovo elemento, vediamo un tipico uso:

```
LET $tagname := name($e)
RETURN
<$tagname>
$e/@*,
2 * number($e)
</$tagname>
```

5.4.3 Espressioni FLWR (espressioni FLoWer)

Una espressione FLWR ("flower") é formata
dalle proposizioni: FOR, LET, WHERE, e RETURN. Come in una query SQL ,queste proposizioni devono apparire in un ordine ben preciso.



FOR/LET : servono per assegnare un valore ad una o più variabili referenziate nella query, mentre il FOR assegna un valore alla volta alla variabile, LET assegna una lista di nodi:

```
FOR $x IN /library/book
```

comporta tanti assegnamenti ad $x quanti sono i book in library

```
LET $x := /library/book
```

comporta il singolo assegnamento della lista dei book a $x.

Un espressione FLWR può contenere molte proposizioni FOR e LET. Il risultato della sequenza di FOR e LET é una lista di tuple di variabili assegnate.

WHERE: Solo le tuple per cui le condizioni della sezione WHERE sono vere sono usate nella sezione RETURN.I predicati presenti in questa sezione possono essere collegati con AND, OR, e NOT.

RETURN: Genera l'output della query, che può essere un nodo, un insieme di nodi o un valore primitivo.

Vediamo alcuni esempi:

(Q9) Titoli dei libri publicati da Morgan Kaufmann nel 1998.

```
FOR $b IN document("bib.xml")//book
WHERE $b/publisher = "Morgan Kaufmann"
AND $b/year = "1998"
RETURN $b/title
```

(Q13) Per ogni libro che ha un prezzo maggiore del prezzo medio, ritorniamo il titolo del libro e la differenza fra il suo prezzo e il prezzo medio.

```
<result>
 LET $a := avg(//book/price)
 FOR $b IN /book
 WHERE $b/price > $a
 RETURN
  <expensive_book>
   $b/title ,
   <price_difference>
    $b/price - $a
   </price_difference>
  </expensive_book>
</result>
```

(Q15) Per ogni editore, visualizziamo i libri che ha pubblicato ordinati in base al prezzo dal più caro al più economico. Gli editori sono elencati in ordine alfabetico.

```
<publisher_list>
 FOR $p IN distinct(document("bib.xml")//publisher)
 RETURN
  <publisher>
   <name> $p/text() </name> ,
   FOR $b IN document("bib.xml")//book[publisher = $p]
   RETURN
    <book>
     $b/title ,
     $b/price
    </book> SORTBY(price DESCENDING)
  </publisher> SORTBY(name)
</publisher_list>
```

5.4.4 Espressioni con operatori e funzioni

XQuery fornisce i soliti operatori aritmetici e logici e quelli insiemistici: UNION, INTERSECT, and EXCEPT; da XQL, XQuery eredita gli operatori BEFORE e AFTER. Un esempio può chiarire l'uso di questi due operatori:

(Q16)Tutti gli elementi presenti tra l'occorrenza del primo ed il secondo elemento incisionnella prima occorrenza di procedure.

```
<critical_sequence>
  LET $p := //procedure[1]
  FOR $e IN //* AFTER ($p//incision)[1]
      BEFORE ($p//incision)[2]
  RETURN shallow($e)
</critical_sequence>
```

La funzione shallow fa una copia del nodo,includendogli attributi ma non gli elementi figli.

5.4.5 Espressioni condizionali

(Q18) Lista delle prenotazioni ordinate per titolo. Per i giornali, visualizziamo gli editori, per le altre prenotazioni l'autore.

```
FOR $h IN //holding
RETURN
   <holding>
      $h/title,
      IF $h/@type = "Journal"
      THEN $h/editor
      ELSE $h/author
   </holding> SORTBY (title)
```

come ogni espressione XQuery, le espressioni condizionali si possono trovare dovunque é atteso un valore.

5.4.6 Quantificatori

XQuery fornisce i quantificatori esistenziale ed universale, vediamo alcuni esempi:

(Q19) Trovare i titoli dei libri nei quali ESISTE un paragrafo che contiene le parole sailing e windsurfing.

```
FOR $b IN //book
WHERE SOME $p IN $b//para SATISFIES
   contains($p, "sailing")
   AND contains($p, "windsurfing")
RETURN $b/title
```

(Q20) Trovare i titoli dei libri nei quali la parola sailing é menzionata in TUTTI i paragrafi.

```
FOR $b IN //book
WHERE EVERY $p IN $b//para SATISFIES
   contains($p, "sailing")
RETURN $b/title
```

5.4.7 Filtraggio

La funzione filter elimina parti del documento ritenute inutili, mantenendo i rapporti di gerarchia. L'esempio che segue produce una TOC (Table of Contents), mantenendo gli elementi section, title e il testo del titolo. Il primo argomento é la radice del documento, mentre il secondo é il filtro che si deve applicare:

```
LET $b := document("cookbook.xml")
RETURN
   <toc>
      filter($b, $b//section | $b//section/title | $b//section/title/text() )
   </toc>
```

5.4.8 Tipi di dato

XQuery supporta i tipi di dato di XML Schema.

Alcuni tipi di dati sono automaticamente riconosciuti da XQuery:

Type	Esempio
xsd:string	”Hello”
xsd:boolean	TRUE, FALSE
xsd:integer	47, -369
xsd:decimal	-2.57
xsd:float	$-3.805E-2$

mentre gli altri hanno bisogno di un costruttore: date(”2000-06-25”).

N.B: Non tutti i costruttori sono stati definiti.

5.4.9 Funzioni

XQuery fornisce una libreria di funzioni da usare nelle query: document, avg, sum, count, max, and min, distinct,empty, ecc.. Inoltre XQuery permette di definire proprie funzioni.

(Q22) Trovare la massima profondità del documento ”partlist.xml.”

```
NAMESPACE xsd = "http://www.w3.org/2000/10/XMLSchema-datatypes"

FUNCTION depth(ELEMENT $e) RETURNS xsd:integer
{
   -- An empty element has depth 1
   -- Otherwise, add 1 to max depth of children
   IF empty($e/*) THEN 1
   ELSE max(depth($e/*)) + 1
}


depth(document("partlist.xml"))
```

5.4.10 Tipi di dato definiti dall'utente

Qualsiasi tipo di dato definibile in XML Schema può essere usato in XQuery, una query può fare riferimento a qualsiasi elemento o tipo definito in uno schema che:

1. Sono referenziati dai documenti usati nella query.
2. Sono dichiarati attraverso la parola chiave NAMESPACE.

ad esempio

```
NAMESPACE xsd = "http://www.w3.org/2000/10/XMLSchema-datatypes"
```

Possibile estensione: una query potrebbe includere un preambolo nel quale dichiarare tipi di dato locali usando la sintassi di XML Schema.

Nell'esempio che segue, definiamo uno schema con le dichiarazioni di due tipi complessi ”emp_type” and ”dept_type” tale schema é referenziabile attraverso il namespace ”http://www.BigCompany.com/BigNames” attraverso la dichiarazione come targetNamespace.

```
<?xml version="1.0">
<schema xmlns="http://www.w3.org/2000/10/XMLSchema" <!--namespace di default-->
   targetNamespace="http://www.BigCompany.com/BigNames">

   <complexType name="emp_type">
      <sequence>
         <element name="name" type="string"/>
         <element name="deptno" type="string"/>
         <element name="salary" type="decimal"/>
         <element name="location" type="string"/>
      </sequence>
   </complexType>

   <complexType name="dept_type">
```

```
      <sequence>
         <element name="deptno" type="string"/>
         <element name="headcount" type="integer"/>
         <element name="payroll" type="decimal"/>
      </sequence>
   </complexType>

</schema>
```

(Q26) Usando lo schema definito, definiamo una funzione che ritorna il numero di impiegati per dipartimento e il loro costo totale.

```
NAMESPACE DEFAULT = "http://www.BigCompany.com/BigNames"

FUNCTION summary(LIST(emp_type) $emps) RETURNS LIST(dept_type)
   {
   FOR $d IN distinct($emps/deptno)
   LET $e := $emps[deptno = $d]
   RETURN
      <dept>
         $d,
         <headcount> count($e) </headcount>,
         <payroll> sum($e/salary) </payroll>
      </dept>
   }

summary(document("acme_corp.xml")/emp[location = "Denver"] )
```

5.4.11 Operazioni su tipi di dato

INSTANCEOF ritorna True se il primo operando é un'istanza del tipo indicato nel secondo operando.

Ad esempio

```
$x INSTANCEOF zoonames:animal
```

é True se il tipo dinamico di $x é zoonames:animal o un tipo derivato da zoonames:animal. INSTANCEOF ha la stessa sintassi di instanceof in Java.

Per i tipi primiti e derivati di XML Schema si può usare l'operatore CAST .

Per esempio:

```
CAST AS integer (x DIV y)
```

converte il risultato di x DIV y nel tipo integer. L'insieme dei tipi supportati da CAST sono ancora da definire. CAST non può essere usata per i tipi definiti dall'utente.

TREAT dice al processatore della query di trattare un espressione come se avesse come tipo uno dei tipi derivati dal suo tipo statico.

Per esempio `TREAT AS Cat($mypet)` dice di trattare $mypet come istanza del tipo Cat, anche se il suo tipo é un supertipo di Cat ad esempio Animal.

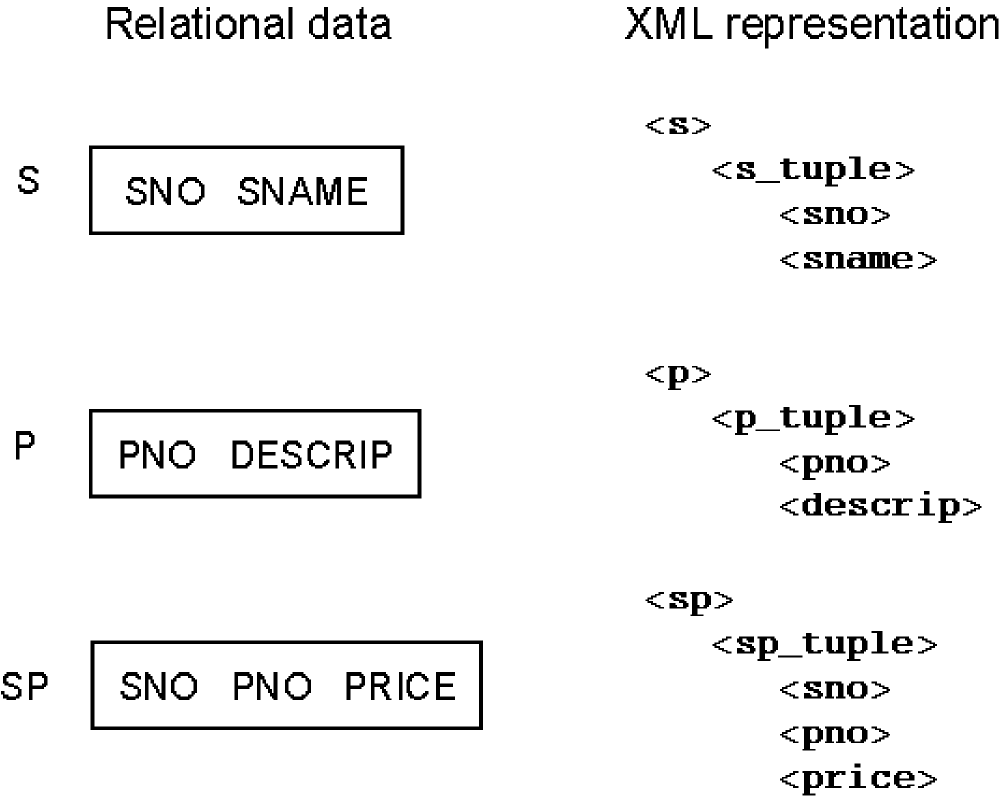
Vediamo un esempio:

```
-- First define some functions to set the stage
NAMESPACE xsd = "http://www.w3.org/2000/10/XMLSchema-datatypes"

FUNCTION quack(duck $d) RETURNS xsd:string
   { "String depends on properties of duck" }

FUNCTION woof(dog $d) RETURNS xsd:string
   { "String depends on properties of dog" }

--This function illustrates simulated subtype polymorphism

FUNCTION sound(animal $a) RETURNS xsd:string
   {
   IF $a INSTANCEOF duck THEN quack(TREAT AS duck($a))
   ELSE IF $a INSTANCEOF dog THEN woof(TREAT AS dog($a))
   ELSE "No sound"
   }

-- This query returns the sounds made by all of Billy’s pets

FOR $p IN /kid[name="Billy"]/pet
RETURN sound($p)
```

5.5 Interrogare Basi di Dati Relazionali

Il modello relazionale é molto diffuso per memorizzare informazioni, per cui é vitale che un linguaggio di interrogazione per XML sappia accedere a tali sorgenti.Le query SQL possono essere tradotte in espressioni.

Consideriamo il seguente schema relazionale:



S contiene i dati sui fornitori (suppliers); P contiene le informazioni su le parti fornite e SP rappresenta la relazzione fra le prime due tabelle.

Vediamo alcuni esempi:

5.5.1 Selezione

```
SQL:SELECT pno
FROM p
WHERE descrip LIKE 'Gear'
ORDER BY pno;
```

XQuery:

```
FOR $p IN document("p.xml")//p_tuple
WHERE contains($p/descrip, "Gear")
RETURN $p/pno SORTBY(.)
```

SORTBY(.), significa ordinare gli elementi $< pno >$ in base al loro contenuto.

5.5.2 Raggruppamento

(Q30) Trovare il codice e il prezzo medio dei componenti che hanno almeno 3 fornitori.

```
SQL:SELECT pno, avg(price) AS avgprice
FROM sp
GROUP BY pno
HAVING count(*) >= 3
ORDER BY pno;
```

XQuery:

```
FOR $pn IN distinct(document("sp.xml")//pno)
LET $sp := document("sp.xml")//sp_tuple[pno = $pn]
WHERE count($sp) >= 3
RETURN
   <well_supplied_item>
      $pn,
      <avgprice> avg($sp/price) </avgprice>
   </well_supplied_item> SORTBY(pno)
```

5.5.3 Join

(Q31) Inner Join

```
FOR $sp IN document("sp.xml")//sp_tuple,
    $p IN document("p.xml")//p_tuple[pno = $sp/pno],
    $s IN document("s.xml")//s_tuple[sno = $sp/sno]
RETURN
   <sp_pair>
      $s/sname ,
      $p/descrip
   </sp_pair> SORTBY (sname, descrip)
```

Q31 ritorna solamente le componenti che hanno almeno un fornitore e i fornitori che forniscono almeno una parte. Un outer join riporta anche le informazioni di una tabella che hanno un corrispondente nell'altra tabella. Per esempio un "left outer join" tra fornitori e componenti può ritornare le informazioni sui fornitori che non forniscono alcun componente. La mancanza di corrispondenza nei databse relazionali é rappresentata dal valore `null`; in una query XML non esiste tale valore, si possono adottare due soluzioni: usare unn empty element o non usare alcun elemento.

(Q32) Left Outer Join

```
FOR $s IN document("s.xml")//s_tuple
RETURN
   <supplier>
      $s/sname,
      FOR $sp IN document("sp.xml")//sp_tuple[sno = $s/sno],
          $p IN document("p.xml")//p_tuple[pno = $sp/pno]
      RETURN $p/descrip SORTBY(.)
   </supplier> SORTBY(sname)
```

(Q33) Full Outer Join

```
<master_list>
    (FOR $s IN document("s.xml")//s_tuple
     RETURN
        <supplier>
           $s/sname,
           FOR $sp IN document("sp.xml")//sp_tuple[sno = $s/sno],
               $p IN document("p.xml")//p_tuple[pno = $sp/pno]
           RETURN
              <part>
                 $p/descrip,
                 $sp/price
              </part> SORTBY (descrip)
        </supplier> SORTBY(sname)
    )
 UNION
    -- parts that have no supplier
    <orphan_parts>
       FOR $p IN document("p.xml")//p_tuple
       WHERE empty(document("sp.xml")//sp_tuple[pno = $p/pno])
       RETURN $p/descrip SORTBY(.)
    </orphan_parts>
</master_list>
```

5.6 Parser XQuery

Anche se le specifiche di XQuery sono solo a livello di Working Draft e potrebbero subire pesanti modifiche nel futuro, ho sviluppato un parser che verifica la correttezza sintattica di una query espressa nella sintassi astratta di XQuery. Per lo sviluppo del parser mi sono basato sul BNF fornito nel documento[40].

Il BNF utilizzato si basa sulla sintassi astratta di XQuery; si é usato il termine *astratta* in quanto come detto in 5.2 deve essere possibile esprimere una query

con piú di una sintassi. Al momento della stesura di questa tesi é allo studio il linguaggio XQueryX[41], una rappresentazione in XML di una query XQuery.

Penso che anche a fronte di modifiche nelle specifiche del linguaggio questo parser possa servire come base di partenza per lo sviluppo di un componente che lavori al fianco del Query-Manager per l'esecuzione delle query.

5.7 Conclusioni

Con l'affermarsi di XML, la distinzione fra le diverse forme di memorizzazione delle informazioni andrà scomparendo e XQuery é flinalizzato ad interrogare tutte queste sorgenti, dato che ha le caratteristiche di tanti linguaggi.

Le future versioni di XQuery potrebbero includere ulteriori funzionalità, tra le quali:

1. Overloading di funzioni.
2. Polimorfismo di funzioni.
3. Update di dati in XML.

Questioni aperte

Alcune questioni rimangono aperte, legate al legame di XQuery con altri attività di ricerca su XML:

- XQuery non permette ancora di esprimere tutte le interrogazioni formulabili dalla sua algebra.
- La corrispondenza fra gli operatori di XQuery e quelli dell'algebra é ancora in via di definizione e *potrebbe portare ad alcuni cambiamenti sia in XQuery che nell'algebra*.
- XQuery ha lo stesso insieme di tipi di dato di XML Schema. Si sta lavorando per garantire la completa coincidenza in questo senso fra XQuery, XML Query Algebra e XML Schema.
- Gli operatori in grado di agire sui tipi semplici di XML Schema saranno definiti da una task force dei gruppi di lavoro su XSLT/Schema/Query.

Conclusioni e Sviluppi Futuri

I temi indagati in questa tesi sono stati due:

1. Relazioni esistenti tra il progetto MOMIS e il progetto Web Semantico.

2. Sviluppo del traduttore $ODL_{I^3} \rightarrow XML - Schema$.

In merito al primo argomento di ricerca di questa tesi, le relazioni fra MOMIS e il Web Semantico, si é studiato l'approccio seguito in quest'ultimo ambito, la cosidetta architettura a 3 livelli (livello dati, livello schema e livello logico) e alcune delle proposte piú interessanti emerse per quanto riguarda il livello logico: OIL e DAML+OIL; due linguaggi nati per definire un'ontologia.

Quello che é emerso é che MOMIS e il Web Semantico partono da una stessa esigenza di fondo:

far sí che la conoscenza che si vuole condividere sia non solo leggibile ma anche ***comprensibile*** *da parti terze.*

Sino ad ora, il Web é stato utilizzato principalmente per scambiare dei documenti, interpretati da degli umani. Sul Web di oggi si hanno anche scambi "automatici" di informazioni, ma non sono la norma. L'idea del Web Semantico é che in futuro, strutturando l'informazione presente sul Web in modo opportuno, sará possibile *processare automaticamente le informazioni scambiate con il Web.*

I meccanismi definiti dal W3C per rendere questo possibile sono essenzialmente due: RDF(**R**esource **D**escription **F**ormat) per descrivere i meta dati e XML per garantire l'interoperabilità sintattica, cioé la capacità di leggere i dati ed ottenere una rappresentazione utilizzabile da un'applicazione.

Un terzo elemento importante per creare il Web semantico é il concetto di *"ontologia"* che, nell'ambito del Web Semantico, rappresenta un documento dove sono definite le relazioni tra termini distinti. Essenzialmente, il ruolo di una ontologia é di permettere la combinazione e la comparazione di informazioni che provengono da basi di dati distinte.

Queste tecnologie insieme disegnano un Web semantico fortemente decentralizzato, e per questo in linea con la tradizione passata del suo sviluppo. Tra le implicazioni del Web semantico vi é la possibilita' di ammettere un ampio uso di

"agenti", programmi in grado di combinare opportunamente l'informazione disponibile su Web per risolvere problemi concreti in modo automatico - per esempio, prenotare un ristorante per giovedí sera non piú lontato di tre km dal luogo dell'ultimo impegno della giornata, con un menú che dipenda dalla dieta dell'ultima settimana e dalle attivitá fisica condotta, come registrato dalle macchine delle palestre frequentate.

L'idea del progetto MOMIS é fornire un accesso *integrato* a sorgenti informative eterogenee, mettendo in condizione l'utente di porre una singola query ed ottenere una risposta unificata dei dati provenienti dalle singole sorgenti.

Da un lato, Web Semantico, abbiamo la necessitá di elaborare automaticamente le informazioni per poter fare un'ampio uso di agenti software; dall'altro, MOMIS, si ha la necessitá di rendere trasparente all'utente le diversitá delle informazioni che ha a disposizione; questo comporta che in MOMIS la definizione di un'ontologia comune, il *Thesaurus*, rappresenta il punto di partenza del processo d'integrazione non il punto d'arrivo come nel Web Semantico.

Inoltre nel progetto MOMIS é fondamentale sfruttare al massimo le informazioni "nascoste" nella struttura di una sorgente, per questo motivo si utilizza ODL_{I^3}, che riesce ad esprimere meglio di OIL e DAML+OIL la tipizzazione dei dati, i vincoli d'identitá e di integritá referenziale importanti nel processo di estrazioni di relazioni inter ed intra schema.

Come si puó facilmente capire MOMIS vá oltre alle problematiche affrontate nell'ambito del Web Semantico e mira ad una **reale** integrazione delle sorgenti.

Vorrei fare le ultime considerazioni sul Web semantico, in merito alla possibiltà di dare il pieno supporto a "ragionare" sui dati (il famoso *livello logico* del Web Semantico) esportati dal traduttore, l'uso congiunto di XML-Schema e RDF non potrà molto se non si arriverá ad uno *standard unico in merito alla logica descrittiva da utilizzare*, un set di operatori condivisi dalle varie Description Logics esiste giá ma poi ognuno ha esteso questo set comune per meglio rispondere alle proprie esigenze, ad esempio OLCD la DL usata nel progetto MOMIS assume una ricca struttura per il sistema dei tipi di base. Ma questa è una considerazione che riguarda più il progetto MOMIS nella sua interezza che questa tesi in particolare.

Per quanto riguarda il secondo punto, lo sviluppo di un traduttore di schemi ODL_{I^3} in XML-Schema, lo scopo finale é quello di rendere MOMIS una sorgente di dati XML.

L'utilizzo di XML-Schema permette di produrre un traduttore in cui i tipi e la gerarchia di classi sono riportate con un elevato grado di corrispondenza con l'originale. XML Schema è in grado di supportare efficacemente ereditarietà, chiavi primarie e candidate, foreign key, in modo da produrre uno schema XML più simile all'originale rispetto a quello attualmente prodotto dal traduttore in DTD. Il traduttore si avvale, inoltre, dell'uso di RDF-Schema e RDF per ovviare

ad alcune mancanze di XML-Schema che potrebbero portare ad una non piena comprensione dello schema esportato.

Gli sviluppi futuri del traduttore, a mio avviso, saranno legati sopratutto ad una maggiore utilizzo di RDF legato alle evoluzioni che subiranno RDF ed RDF-Schema, adesso come adesso RDF é un meccanismo *generale* di rappresentazione della conoscenza, per cui dovendo mantenere la generalità non riesce a pieno a sopperire alle mancanze di XML-Schema nella rappresentazione della semantica di ODL_{I^3}. Soprattutto si dovrà vedere come evolverà la discussione sulla necessità di un meccanismo standard per l'integrazione di XML-Schema e RDF che da più parti nella comunità degli sviluppatori viene avvertita come non più rinviabile.

Ovviamente un'altra direzione di sviluppo del traduttore può essere nella revisione dello schema RDF *momis_schema* che contiene le risorse e le proprietà usate dal traduttore per costruire i files rdf e rdf schema associati allo schema ODL_{I^3} tradotto, introducendo nuove proprietà/risorse.

Un altro progetto solamente iniziato durante questa tesi riguarda la fase di esecuzione delle query, a questo scopo é stato studiato il linguaggio XQuery l'ultima proposta del W3C come query language per documenti XML; l'aspetto più interessante di questo linguaggio é la sua apertura, attraverso il supporto per i tipi di dati, verso XML-Schema.

Poichè al momento di stesura di questa tesi, le specifiche di XQuery erano a livello di Working Draft e non erano disponibili la lista definitiva degli operatori e delle funzioni, si é sviluppato solamente un parser che effettua il controllo sintattico delle query. Quando le specifiche raggiungeranno un certo grado di stabilità si potrà completare lo sviluppo di questo parser.

Appendice A

Il linguaggio descrittivo ODL_{I^3}

Si riporta la descrizione in BNF del linguaggio descrittivo ODL_{I^3}. Essendo questo una estensione del linguaggio standard ODL, si riportano in questo appendice solo le parti che differiscono dall'ODL originale, rimandando invece a quest'ultimo per le parti in comune.

⟨interface_dcl⟩	::=	⟨interface_header⟩ {[⟨interface_body⟩]};
⟨interface_header⟩	::=	**interface** ⟨identifier⟩
		[⟨inheritance_spec⟩]
		[⟨type_property_list⟩]
⟨inheritance_spec⟩	::=	: ⟨scoped_name⟩ [,⟨inheritance_spec⟩]
⟨type_property_list⟩	::=	**(** [⟨source_spec⟩] [⟨extent_spec⟩]
		[⟨key_spec⟩] [⟨f_key_spec⟩] **)**
⟨source_spec⟩	::=	**source** ⟨source_type⟩ ⟨source_name⟩
⟨source_type⟩	::=	**relational** | **nfrelational** | **object** | **file**
⟨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⟩
...		

```
⟨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⟩ | ⟨or_expression⟩
⟨and_expression⟩      ::=  ( ⟨local_attr_name⟩ and ⟨and_list⟩ )
⟨and_list⟩            ::=  ⟨local_attr_name⟩ | ⟨local_attr_name⟩ and ⟨and_list⟩
⟨or_expression⟩       ::=  ( ⟨local_attr_name⟩ or ⟨or_list⟩ )
⟨or_list⟩             ::=  ⟨local_attr_name⟩ | ⟨local_attr_name⟩ or ⟨or_list⟩
⟨local_attr_name⟩     ::=  ⟨source_name⟩.⟨class_name⟩.⟨attribute_name⟩
...
⟨relationships_list⟩  ::=  ⟨relationship_dcl⟩; | ⟨relationship_dcl⟩; ⟨relationships_list⟩
⟨relationships_dcl⟩   ::=  ⟨local_attr_name⟩ ⟨relationship_type⟩ ⟨local_attr_name⟩
⟨relationship_type⟩   ::=  syn | bt | nt | rt
...
⟨rule_list⟩           ::=  ⟨rule_dcl⟩; | ⟨rule_dcl⟩; ⟨rule_list⟩
⟨rule_dcl⟩            ::=  rule ⟨identifier⟩ ⟨rule_pre⟩ then ⟨rule_post⟩
⟨rule_pre⟩            ::=  ⟨forall⟩ ⟨identifier⟩ in ⟨identifier⟩ : ⟨rule_body_list⟩
⟨rule_post⟩           ::=  ⟨rule_body_list⟩
⟨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⟩       ::=  = | ≥ | ≤ | > | <
⟨rule_cast⟩           ::=  (⟨simple_type_spec⟩)
⟨dotted_name⟩         ::=  ⟨identifier⟩ | ⟨identifier⟩.⟨dotted_name⟩
⟨forall⟩              ::=  for all | forall
```

Appendice B

Esempio di Ontologia OIL

Quella che segue è un parte di un'ontologia OIL che mostra l'uso delle primitive del linguaggio.

```
ontology-container
title "African animals"
creator "Ian Horrocks"
subject "animal, food, vegetarians"
description "A didactic example ontology"
description.release "1.01" publisher "I. Horrocks"
type "ontology"
format "pseudo-xml"
identifier "http://www.cs.vu.nl/~dieter/oil/TR/oil.pdf"
source "http://www.africa.com/nature/animals.html"
language "OIL"
relation.hasPart "http://www.ontosRus.com/animals/jungle.onto"

ontology-definitions
slot-def has-part
 inverse is-part-of
 properties transitive
slot-def comes-from
slot-def age
 range (min 0)
 properties functional
class-def animal
class-def plant
disjoint animal plant
class-def tree
 subclass-of plant
class-def branch
 slot-constraint is-part-of
  has-value tree
class-def defined carnivore
 subclass-of animal
 slot-constraint eats
  value-type animal
class-def giraffe
 subclass-of animal
 slot-constraint eats
  value-type leaf
class-def tasty-plant
 subclass-of plant
```

```
 slot-constraint eaten-by
  has-value herbivore carnivore
class-def elephant
 subclass-of animal
 slot-constraint colour
  has-filler "grey"
class-def defined adult-elephant
 subclass-of elephant
 slot-constraint age
  has-value (min 20)
covered adult-elephant
by (slot-constraint weight has-value (range 5000 8000))
class-def kenyan-elephant
 subclass-of elephant
disjoint kenyan-elephant indian-elephant
```

Appendice C

Esempio di Ontologia in DAML+OIL

```
<!-- $Revision: 1.9 $ of $Date: 2001/05/03 16:38:38 $ -->

<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#"
  xmlns:dex ="http://www.daml.org/2001/03/daml+oil-ex#"
  xmlns:exd ="http://www.daml.org/2001/03/daml+oil-ex-dt#"
  xmlns     ="http://www.daml.org/2001/03/daml+oil-ex#"
>

<daml:Ontology rdf:about="">
  <daml:versionInfo>$Id: daml+oil-ex.daml,v 1.9 2001/05/03 16:38:38 mdean Exp $</daml:versionInfo>
  <rdfs:comment>
    An example ontology, with data types taken from XML Schema
  </rdfs:comment>
  <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
</daml:Ontology>

<daml:Class rdf:ID="Animal">
  <rdfs:label>Animal</rdfs:label>
  <rdfs:comment>
    This class of animals is illustrative of a number of ontological idioms.
  </rdfs:comment>
</daml:Class>

<daml:Class rdf:ID="Male">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</daml:Class>

<daml:Class rdf:ID="Female">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <daml:disjointWith rdf:resource="#Male"/>
</daml:Class>

<daml:Class rdf:ID="Man">
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf rdf:resource="#Male"/>
</daml:Class>
```

```
<daml:Class rdf:ID="Woman">
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf rdf:resource="#Female"/>
</daml:Class>

<daml:ObjectProperty rdf:ID="hasParent">
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="#Animal"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="hasFather">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Male"/>
</daml:ObjectProperty>

<daml:DatatypeProperty rdf:ID="shoesize">
  <rdfs:comment>
    shoesize is a DatatypeProperty whose range is xsd:decimal.
    shoesize is also a UniqueProperty (can only have one shoesize)
  </rdfs:comment>
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#decimal"/>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="age">
  <rdfs:comment>
    age is a DatatypeProperty whose range is xsd:decimal.
    age is also a UniqueProperty (can only have one age)
  </rdfs:comment>
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</daml:DatatypeProperty>

<daml:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasParent"/>
      <daml:toClass rdf:resource="#Person"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#hasFather"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#shoesize"/>
      <daml:minCardinality>1</daml:minCardinality>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:about="#Animal">
  <rdfs:comment>
    Animals have exactly two parents, ie:
    If x is an animal, then it has exactly 2 parents
    (but it is NOT the case that anything that has 2 parents is an animal).
  </rdfs:comment>
  <rdfs:subClassOf>
```

```
    <daml:Restriction daml:cardinality="2">
      <daml:onProperty rdf:resource="#hasParent"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:about="#Person">
  <rdfs:subClassOf>
    <daml:Restriction daml:maxCardinality="1">
      <daml:onProperty rdf:resource="#hasSpouse"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:about="#Person">
  <rdfs:subClassOf>
    <daml:Restriction daml:maxCardinalityQ="1">
      <daml:onProperty rdf:resource="#hasOccupation"/>
      <daml:hasClassQ rdf:resource="#FullTimeOccupation"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:UniqueProperty rdf:ID="hasMother">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Female"/>
</daml:UniqueProperty>

<daml:ObjectProperty rdf:ID="hasChild">
  <daml:inverseOf rdf:resource="#hasParent"/>
</daml:ObjectProperty>

<daml:TransitiveProperty rdf:ID="hasAncestor">
  <rdfs:label>hasAncestor</rdfs:label>
</daml:TransitiveProperty>

<daml:TransitiveProperty rdf:ID="descendant"/>

<daml:ObjectProperty rdf:ID="hasMom">
  <daml:samePropertyAs rdf:resource="#hasMother"/>
</daml:ObjectProperty>

<daml:Class rdf:ID="Car">
  <rdfs:comment>no car is a person</rdfs:comment>
  <rdfs:subClassOf>
    <daml:Class>
      <daml:complementOf rdf:resource="#Person"/>
    </daml:Class>
  </rdfs:subClassOf>
</daml:Class>

<!-- @@CAVEAT: daml:collection is an extension of RDF 1.0 syntax;
     don't expect existing tools to support it.
     See http://www.daml.org/2001/03/reference.html#collection for details.
-->

<daml:Class rdf:about="#Person">
  <rdfs:comment>every person is a man or a woman</rdfs:comment>
  <daml:disjointUnionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Man"/>
    <daml:Class rdf:about="#Woman"/>
  </daml:disjointUnionOf>
```

```
</daml:Class>

<daml:Class rdf:ID="TallMan">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#TallThing"/>
    <daml:Class rdf:about="#Man"/>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="MarriedPerson">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Person"/>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#hasSpouse"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="HumanBeing">
  <daml:sameClassAs rdf:resource="#Person"/>
</daml:Class>

<daml:Class rdf:ID="Adult">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Person"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#age"/>
      <daml:hasClass rdf:resource="http://www.daml.org/2001/03/daml+oil-ex-dt#over17"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<daml:Class rdf:ID="Senior">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Person"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#age"/>
      <daml:hasClass rdf:resource="http://www.daml.org/2001/03/daml+oil-ex-dt#over59"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<Person rdf:ID="Adam">
  <rdfs:label>Adam</rdfs:label>
  <rdfs:comment>Adam is a person.</rdfs:comment>
  <age><xsd:integer rdf:value="13"/></age>
  <shoesize><xsd:decimal rdf:value="9.5"/></shoesize>
</Person>

<daml:ObjectProperty rdf:ID="hasHeight">
  <rdfs:range rdf:resource="#Height"/>
</daml:ObjectProperty>

<daml:Class rdf:ID="Height">
  <daml:oneOf rdf:parseType="daml:collection">
    <Height rdf:ID="short"/>
    <Height rdf:ID="medium"/>
    <Height rdf:ID="tall"/>
  </daml:oneOf>
</daml:Class>

<!-- TallThing is EXACTLY the class of things whose hasHeight is tall -->
```

```
<daml:Class rdf:ID="TallThing">
  <daml:sameClassAs>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasHeight"/>
      <daml:hasValue rdf:resource="#tall"/>
    </daml:Restriction>
  </daml:sameClassAs>
</daml:Class>

<daml:DatatypeProperty rdf:ID="shirtsize">
  <rdfs:comment>
    shirtsize is a DatatypeProperty whose range is clothingsize.
  </rdfs:comment>
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource="http://www.daml.org/2001/03/daml+oil-ex-dt#clothingsize"/>
</daml:DatatypeProperty>

<rdfs:Class rdf:ID="BigFoot">
  <rdfs:comment>
  BigFoots (BigFeet?) are exactly those persons whose shosize is over12.
  </rdfs:comment>
  <daml:intersectionOf rdf:parseType="daml:collection">
    <rdfs:Class rdf:about="#Person"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#shoesize"/>
      <daml:hasClass rdf:resource="http://www.daml.org/2001/03/daml+oil-ex-dt#over12"/>
    </daml:Restriction>
  </daml:intersectionOf>
</rdfs:Class>

<Person rdf:ID="Ian">
  <rdfs:comment>
  Ian is an instance of Person. Ian has shoesize 14 and age 37. From
  the range restrictions we know that these are of type xsd:decimal
  and xsd:nonNegativeInteger respectively. Ian also has shirtsize 12,
  the type of which is the union type clothingsize; the discriminating
  type "string" has been specified, so the value is to be taken as the
  string "12" rather than the integer 12.  We may be able to infer
  that Ian is an instance of BigFoot (because 14 is a valid value for
  xsd:over12).
  </rdfs:comment>
  <shoesize>14</shoesize>
  <age>37</age>
  <shirtsize><xsd:string rdf:value="12"/></shirtsize>
</Person>

<Person rdf:ID="Peter">
  <rdfs:comment>
  Peter is an instance of Person. Peter has shoesize 9.5 and age 46. From the
  range restrictions we know that these are of type xsd:decimal and
  xsd:nonNegativeInteger respectively. Peter also has shirtsize 15, the type
  of which is the union type clothingsize; no discriminating type
  has been specified, so the value may be either a string or an integer.
  </rdfs:comment>
  <shoesize>9.5</shoesize>
  <age>46</age>
  <shirtsize>15</shirtsize>
</Person>

<daml:DatatypeProperty rdf:ID="associatedData">
  <rdfs:comment>
```

```
  associatedData is a DatatypeProperty without a range restriction.
  </rdfs:comment>
</daml:DatatypeProperty>

<daml:Class rdf:about="#Person">
  <rdfs:comment>
  Persons have at most 1 item of associatedData
  </rdfs:comment>
   <rdfs:subClassOf>
      <daml:Restriction>
        <daml:onProperty rdf:resource="#associatedData"/>
        <daml:maxCardinality>1</daml:maxCardinality>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>

<Person rdf:ID="Santa">
  <rdfs:comment>
  Santa is an instance of Person. Santa has two pieces of
  associatedData, one of which is the real number 3.14159 and the
  other of which is the string "3.14159". We may be able to infer a
  logical inconsistency (because Persons can have at most 1 item of
  associatedData, and a value cannot be both a string and a real
  number).
  </rdfs:comment>
  <associatedData><xsd:real rdf:value="3.14159"/></associatedData>
  <associatedData><xsd:string rdf:value="3.14159"/></associatedData>
</Person>

</rdf:RDF>
```

Appendice D

Metadati RDF-Schema di supporto al Traduttore

Quello che segue é *momis_schema* lo schema RDF che contiene le definizione delle risorse e proprietà utilizzate nei documenti RDF che accompagnano lo schema XML-Schema prodotto dal traduttore.

```
<rdf:RDF xml:lang="en" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description ID="Schema">
 <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:comment>Schema contenente tutte le interfacce.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="Interface">
 <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:comment>Interfaccia generica.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="source">
 <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:domain rdf:resource="#Interface"/>
 <rdfs:comment>Sorgente</rdfs:comment>
</rdf:Description>

<rdf:Description ID="olcd">
 <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:domain rdf:resource="#Interface"/>
 <rdfs:comment>Descrizione in OLCD dell'interfaccia</rdfs:comment>
 <rdfs:comment>
       In questo modo consideriamo le regole d'integrita'
       correlate ad una interfaccia
 </rdfs:comment>
</rdf:Description>

<rdf:Description ID="interfaces">
 <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>
 <rdfs:domain rdf:resource="#Schema"/>
</rdf:Description>
```

```
<rdf:Description ID="tipo">
 <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Description>

<rdf:Description ID="extent">
 <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:domain rdf:resource="#Interface"/>
</rdf:Description>

<rdf:Description ID="mapping-table">
 <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:domain rdf:resource="#Interface"/>
</rdf:Description>

</rdf:RDF>
```

Il file seguente é il documento schema, *meta.xsd*, in cui sono dichiarati gli attributi utilizzati per realizzare l'integrazione XML-Schema RDF(S):

```
<xsd:schema xmlns="http://sparc20.ing.unimo.it/Meta"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://sparc20.ing.unimo.it/Meta">

<xsd:attribute name="interface" type="xsd:anyURI"/>

<xsd:attribute name="semantica" type="xsd:anyURI"/>

</xsd:schema>
```

Appendice E

La descrizione di XML-Schema con Schema

Poichè uno schema XML-Schema é in fin dei conti un documento XML, é possibile definire una DTD ed un documento schema, le definizioni sono state prese dal documento *XML Schema Part 1: Structures*

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- XML Schema schema for XML Schemas: Part 1: Structures -->
<!DOCTYPE xs:schema PUBLIC "-//W3C//DTD XMLSCHEMA 200102//EN" "XMLSchema.dtd" [

<!-- provide ID type information even for parsers which only read the internal subset -->
<!ATTLIST xs:schema          id  ID  #IMPLIED>
<!ATTLIST xs:complexType     id  ID  #IMPLIED>
<!ATTLIST xs:complexContent  id  ID  #IMPLIED>
<!ATTLIST xs:simpleContent   id  ID  #IMPLIED>
<!ATTLIST xs:extension       id  ID  #IMPLIED>
<!ATTLIST xs:element         id  ID  #IMPLIED>
<!ATTLIST xs:group           id  ID  #IMPLIED>
<!ATTLIST xs:all             id  ID  #IMPLIED>
<!ATTLIST xs:choice          id  ID  #IMPLIED>
<!ATTLIST xs:sequence        id  ID  #IMPLIED>
<!ATTLIST xs:any             id  ID  #IMPLIED>
<!ATTLIST xs:anyAttribute    id  ID  #IMPLIED>
<!ATTLIST xs:attribute       id  ID  #IMPLIED>
<!ATTLIST xs:attributeGroup  id  ID  #IMPLIED>
<!ATTLIST xs:unique          id  ID  #IMPLIED>
<!ATTLIST xs:key             id  ID  #IMPLIED>
<!ATTLIST xs:keyref          id  ID  #IMPLIED>
<!ATTLIST xs:selector        id  ID  #IMPLIED>
<!ATTLIST xs:field           id  ID  #IMPLIED>
<!ATTLIST xs:include         id  ID  #IMPLIED>
<!ATTLIST xs:import          id  ID  #IMPLIED>
<!ATTLIST xs:redefine        id  ID  #IMPLIED>
<!ATTLIST xs:notation        id  ID  #IMPLIED>
]>

<xs:schema targetNamespace="http://www.w3.org/2001/XMLSchema" blockDefault="#all"
           elementFormDefault="qualified"
           version="Id: XMLSchema.xsd,v 1.48 2001/04/24 18:56:39 ht Exp "
```

```
          xmlns:xs="http://www.w3.org/2001/XMLSchema" xml:lang="EN">

<xs:annotation>
 <xs:documentation source="http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/structures.html">
    The schema corresponding to this document is normative,
    with respect to the syntactic constraints it expresses in the
    XML Schema language.  The documentation (within &lt;documentation> elements)
    below, is not normative, but rather highlights important aspects of
    the W3C Recommendation of which this is a part
 </xs:documentation>
</xs:annotation>

<xs:annotation>
 <xs:documentation>
     The simpleType element and all of its members are defined
     in datatypes.xsd
 </xs:documentation>
</xs:annotation>

<xs:include schemaLocation="datatypes.xsd"/>

<xs:import namespace="http://www.w3.org/XML/1998/namespace"
           schemaLocation="http://www.w3.org/2001/xml.xsd">
 <xs:annotation>
  <xs:documentation>
   Get access to the xml: attribute groups for xml:lang
   as declared on 'schema' and 'documentation' below
  </xs:documentation>
 </xs:annotation>
</xs:import>

<xs:complexType name="openAttrs">

 <xs:annotation>
  <xs:documentation>
      This type is extended by almost all schema types
      to allow attributes from other namespaces to be
      added to user schemas.
  </xs:documentation>
 </xs:annotation>

 <xs:complexContent>
  <xs:restriction base="xs:anyType">
    <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="annotated">
 <xs:annotation>
  <xs:documentation>
   This type is extended by all types which allow annotation
   other than &lt;schema&gt; itself
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="xs:openAttrs">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
  </xs:extension>
```

```
 </xs:complexContent>
</xs:complexType>

<xs:group name="schemaTop">
 <xs:annotation>
  <xs:documentation>
   This group is for the
   elements which occur freely at the top level of schemas.
   All of their types are based on the "annotated" type by extension.
  </xs:documentation>
 </xs:annotation>
 <xs:choice>
  <xs:group ref="xs:redefinable"/>
   <xs:element ref="xs:element"/>
   <xs:element ref="xs:attribute"/>
   <xs:element ref="xs:notation"/>
  </xs:choice>
 </xs:group>

 <xs:group name="redefinable">
  <xs:annotation>
   <xs:documentation>
    This group is for the
    elements which can self-redefine (see &lt;redefine> below).
   </xs:documentation>
  </xs:annotation>
  <xs:choice>
   <xs:element ref="xs:simpleType"/>
   <xs:element ref="xs:complexType"/>
   <xs:element ref="xs:group"/>
   <xs:element ref="xs:attributeGroup"/>
  </xs:choice>
 </xs:group>

 <xs:simpleType name="formChoice">
  <xs:annotation>
   <xs:documentation>
    A utility type, not for public use
   </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:NMTOKEN">
   <xs:enumeration value="qualified"/>
   <xs:enumeration value="unqualified"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="reducedDerivationControl">
  <xs:annotation>
   <xs:documentation>
    A utility type, not for public use
   </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:derivationControl">
   <xs:enumeration value="extension"/>
   <xs:enumeration value="restriction"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="derivationSet">
  <xs:annotation>
   <xs:documentation>
    A utility type, not for public use
```

```
   </xs:documentation>
   <xs:documentation>
    #all or (possibly empty) subset of (extension, restriction)
   </xs:documentation>
  </xs:annotation>
  <xs:union>
   <xs:simpleType>
    <xs:restriction base="xs:token">
     <xs:enumeration value="#all"/>
    </xs:restriction>
   </xs:simpleType>
   <xs:simpleType>
    <xs:list itemType="xs:reducedDerivationControl"/>
   </xs:simpleType>
  </xs:union>
 </xs:simpleType>

 <xs:element name="schema" id="schema">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-schema"/>
  </xs:annotation>
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="xs:openAttrs">
     <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element ref="xs:include"/>
       <xs:element ref="xs:import"/>
       <xs:element ref="xs:redefine"/>
       <xs:element ref="xs:annotation"/>
      </xs:choice>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
       <xs:group ref="xs:schemaTop"/>
       <xs:element ref="xs:annotation" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:sequence>
     <xs:attribute name="targetNamespace" type="xs:anyURI"/>
     <xs:attribute name="version" type="xs:token"/>
     <xs:attribute name="finalDefault" type="xs:derivationSet"
                   use="optional" default=""/>
     <xs:attribute name="blockDefault" type="xs:blockSet"
                   use="optional" default=""/>
     <xs:attribute name="attributeFormDefault" type="xs:formChoice"
                   use="optional" default="unqualified"/>
     <xs:attribute name="elementFormDefault" type="xs:formChoice"
                   use="optional" default="unqualified"/>
     <xs:attribute name="id" type="xs:ID"/>
     <xs:attribute ref="xml:lang"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:key name="element">
   <xs:selector xpath="xs:element"/>
   <xs:field xpath="@name"/>
  </xs:key>

  <xs:key name="attribute">
   <xs:selector xpath="xs:attribute"/>
   <xs:field xpath="@name"/>
  </xs:key>
```

```
<xs:key name="type">
 <xs:selector xpath="xs:complexType|xs:simpleType"/>
 <xs:field xpath="@name"/>
</xs:key>

<xs:key name="group">
 <xs:selector xpath="xs:group"/>
 <xs:field xpath="@name"/>
</xs:key>

<xs:key name="attributeGroup">
 <xs:selector xpath="xs:attributeGroup"/>
 <xs:field xpath="@name"/>
</xs:key>

<xs:key name="notation">
 <xs:selector xpath="xs:notation"/>
 <xs:field xpath="@name"/>
</xs:key>

<xs:key name="identityConstraint">
 <xs:selector xpath=".//xs:key|.//xs:unique|.//xs:keyref"/>
 <xs:field xpath="@name"/>
</xs:key>

</xs:element>

<xs:simpleType name="allNNI">
 <xs:annotation>
  <xs:documentation>
   for maxOccurs
  </xs:documentation>
 </xs:annotation>
 <xs:union memberTypes="xs:nonNegativeInteger">
  <xs:simpleType>
   <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="unbounded"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:union>
</xs:simpleType>

<xs:attributeGroup name="occurs">
 <xs:annotation>
  <xs:documentation>
   for all particles
  </xs:documentation>
 </xs:annotation>
 <xs:attribute name="minOccurs" type="xs:nonNegativeInteger"
               use="optional" default="1"/>
 <xs:attribute name="maxOccurs" type="xs:allNNI"
               use="optional" default="1"/>
</xs:attributeGroup>

<xs:attributeGroup name="defRef">
 <xs:annotation>
  <xs:documentation>
   for element, group and attributeGroup,
   which both define and reference
  </xs:documentation>
 </xs:annotation>
 <xs:attribute name="name" type="xs:NCName"/>
```

```
 <xs:attribute name="ref" type="xs:QName"/>
</xs:attributeGroup>

<xs:group name="typeDefParticle">
 <xs:annotation>
  <xs:documentation>
   'complexType' uses this
  </xs:documentation>
 </xs:annotation>
 <xs:choice>
  <xs:element name="group" type="xs:groupRef"/>
  <xs:element ref="xs:all"/>
  <xs:element ref="xs:choice"/>
  <xs:element ref="xs:sequence"/>
 </xs:choice>
</xs:group>

<xs:group name="nestedParticle">
 <xs:choice>
  <xs:element name="element" type="xs:localElement"/>
  <xs:element name="group" type="xs:groupRef"/>
  <xs:element ref="xs:choice"/>
  <xs:element ref="xs:sequence"/>
  <xs:element ref="xs:any"/>
 </xs:choice>
</xs:group>

<xs:group name="particle">
 <xs:choice>
  <xs:element name="element" type="xs:localElement"/>
  <xs:element name="group" type="xs:groupRef"/>
  <xs:element ref="xs:all"/>
  <xs:element ref="xs:choice"/>
  <xs:element ref="xs:sequence"/>
  <xs:element ref="xs:any"/>
 </xs:choice>
</xs:group>

<xs:complexType name="attribute">
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:sequence>
    <xs:element name="simpleType" minOccurs="0" type="xs:localSimpleType"/>
   </xs:sequence>
   <xs:attributeGroup ref="xs:defRef"/>
    <xs:attribute name="type" type="xs:QName"/>
    <xs:attribute name="use" use="optional" default="optional">
     <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
       <xs:enumeration value="prohibited"/>
       <xs:enumeration value="optional"/>
       <xs:enumeration value="required"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="default" type="xs:string"/>
    <xs:attribute name="fixed" type="xs:string"/>
    <xs:attribute name="form" type="xs:formChoice"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
```

```
 <xs:complexType name="topLevelAttribute">
  <xs:complexContent>
   <xs:restriction base="xs:attribute">
    <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
     <xs:element name="simpleType" minOccurs="0" type="xs:localSimpleType"/>
    </xs:sequence>
    <xs:attribute name="ref" use="prohibited"/>
    <xs:attribute name="form" use="prohibited"/>
    <xs:attribute name="use" use="prohibited"/>
    <xs:attribute name="name" use="required" type="xs:NCName"/>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>

 <xs:group name="attrDecls">
  <xs:sequence>
   <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="attribute" type="xs:attribute"/>
    <xs:element name="attributeGroup" type="xs:attributeGroupRef"/>
   </xs:choice>
   <xs:element ref="xs:anyAttribute" minOccurs="0"/>
  </xs:sequence>
 </xs:group>

 <xs:element name="anyAttribute" type="xs:wildcard" id="anyAttribute">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-anyAttribute"/>
  </xs:annotation>
 </xs:element>

 <xs:group name="complexTypeModel">
  <xs:choice>
   <xs:element ref="xs:simpleContent"/>
   <xs:element ref="xs:complexContent"/>
   <xs:sequence>
    <xs:annotation>
     <xs:documentation>
      This branch is short for
      &lt;complexContent>
      &lt;restriction base="xs:anyType">
      ...
      &lt;/restriction>
      &lt;/complexContent>
    </xs:documentation>
   </xs:annotation>
   <xs:group ref="xs:typeDefParticle" minOccurs="0"/>
   <xs:group ref="xs:attrDecls"/>
  </xs:sequence>
 </xs:choice>
</xs:group>

<xs:complexType name="complexType" abstract="true">
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:group ref="xs:complexTypeModel"/>
   <xs:attribute name="name" type="xs:NCName">
    <xs:annotation>
     <xs:documentation>
      Will be restricted to required or forbidden
     </xs:documentation>
    </xs:annotation>
```

```
   </xs:attribute>
   <xs:attribute name="mixed" type="xs:boolean"
                 use="optional" default="false">
    <xs:annotation>
     <xs:documentation>
      Not allowed if simpleContent child is chosen.
      May be overriden by setting on complexContent child.
     </xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="abstract" type="xs:boolean"
                 use="optional" default="false"/>
   <xs:attribute name="final" type="xs:derivationSet"/>
   <xs:attribute name="block" type="xs:derivationSet"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="topLevelComplexType">
 <xs:complexContent>
  <xs:restriction base="xs:complexType">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:group ref="xs:complexTypeModel"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:NCName" use="required"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="localComplexType">
 <xs:complexContent>
  <xs:restriction base="xs:complexType">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:group ref="xs:complexTypeModel"/>
   </xs:sequence>
   <xs:attribute name="name" use="prohibited"/>
   <xs:attribute name="abstract" use="prohibited"/>
   <xs:attribute name="final" use="prohibited"/>
   <xs:attribute name="block" use="prohibited"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="restrictionType">
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:sequence>
    <xs:choice>
     <xs:group ref="xs:typeDefParticle" minOccurs="0"/>
     <xs:group ref="xs:simpleRestrictionModel" minOccurs="0"/>
    </xs:choice>
    <xs:group ref="xs:attrDecls"/>
   </xs:sequence>
   <xs:attribute name="base" type="xs:QName" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="complexRestrictionType">
 <xs:complexContent>
```

```
   <xs:restriction base="xs:restrictionType">
    <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
     <xs:group ref="xs:typeDefParticle" minOccurs="0"/>
     <xs:group ref="xs:attrDecls"/>
    </xs:sequence>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="extensionType">
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:sequence>
     <xs:group ref="xs:typeDefParticle" minOccurs="0"/>
     <xs:group ref="xs:attrDecls"/>
    </xs:sequence>
    <xs:attribute name="base" type="xs:QName" use="required"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:element name="complexContent" id="complexContent">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-complexContent"/>
  </xs:annotation>
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="xs:annotated">
     <xs:choice>
      <xs:element name="restriction" type="xs:complexRestrictionType"/>
      <xs:element name="extension" type="xs:extensionType"/>
     </xs:choice>
     <xs:attribute name="mixed" type="xs:boolean">
      <xs:annotation>
       <xs:documentation>
        Overrides any setting on complexType parent.
       </xs:documentation>
      </xs:annotation>
     </xs:attribute>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>

 <xs:complexType name="simpleRestrictionType">
  <xs:complexContent>
   <xs:restriction base="xs:restrictionType">
    <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
     <xs:group ref="xs:simpleRestrictionModel" minOccurs="0"/>
     <xs:group ref="xs:attrDecls"/>
    </xs:sequence>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="simpleExtensionType">
  <xs:complexContent>
   <xs:restriction base="xs:extensionType">
    <xs:sequence>
     <xs:annotation>
```

```
     <xs:documentation>
      No typeDefParticle group reference
     </xs:documentation>
    </xs:annotation>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:group ref="xs:attrDecls"/>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:element name="simpleContent" id="simpleContent">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-simpleContent"/>
 </xs:annotation>
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:choice>
     <xs:element name="restriction" type="xs:simpleRestrictionType"/>
     <xs:element name="extension" type="xs:simpleExtensionType"/>
    </xs:choice>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>

<xs:element name="complexType" type="xs:topLevelComplexType" id="complexType">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-complexType"/>
 </xs:annotation>
</xs:element>

<xs:simpleType name="blockSet">
 <xs:annotation>
  <xs:documentation>
   A utility type, not for public use
  </xs:documentation>
  <xs:documentation>
   #all or (possibly empty) subset of (substitution, extension,
   restriction)    </xs:documentation>
 </xs:annotation>
 <xs:union>
  <xs:simpleType>
   <xs:restriction base="xs:token">
    <xs:enumeration value="#all"/>
   </xs:restriction>
  </xs:simpleType>
  <xs:simpleType>
   <xs:list>
    <xs:simpleType>
     <xs:restriction base="xs:derivationControl">
      <xs:enumeration value="extension"/>
      <xs:enumeration value="restriction"/>
      <xs:enumeration value="substitution"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:list>
  </xs:simpleType>
 </xs:union>
</xs:simpleType>
```

```
 <xs:complexType name="element" abstract="true">
  <xs:annotation>
   <xs:documentation>
    The element element can be used either
    at the top level to define an element-type binding globally,
    or within a content model to either reference a globally-defined
    element or type or declare an element-type binding locally.
    The ref form is not allowed at the top level.
   </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:sequence>
     <xs:choice minOccurs="0">
      <xs:element name="simpleType" type="xs:localSimpleType"/>
      <xs:element name="complexType" type="xs:localComplexType"/>
     </xs:choice>
     <xs:group ref="xs:identityConstraint" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="xs:defRef"/>
    <xs:attribute name="type" type="xs:QName"/>
    <xs:attribute name="substitutionGroup" type="xs:QName"/>
    <xs:attributeGroup ref="xs:occurs"/>
    <xs:attribute name="default" type="xs:string"/>
    <xs:attribute name="fixed" type="xs:string"/>
    <xs:attribute name="nillable" type="xs:boolean"
                  use="optional" default="false"/>
    <xs:attribute name="abstract" type="xs:boolean"
                  use="optional" default="false"/>
    <xs:attribute name="final" type="xs:derivationSet"/>
    <xs:attribute name="block" type="xs:blockSet"/>
    <xs:attribute name="form" type="xs:formChoice"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="topLevelElement">
  <xs:complexContent>
   <xs:restriction base="xs:element">
    <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
     <xs:choice minOccurs="0">
     <xs:element name="simpleType" type="xs:localSimpleType"/>
     <xs:element name="complexType" type="xs:localComplexType"/>
    </xs:choice>
    <xs:group ref="xs:identityConstraint" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="ref" use="prohibited"/>
   <xs:attribute name="form" use="prohibited"/>
   <xs:attribute name="minOccurs" use="prohibited"/>
   <xs:attribute name="maxOccurs" use="prohibited"/>
   <xs:attribute name="name" use="required" type="xs:NCName"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="localElement">
 <xs:complexContent>
  <xs:restriction base="xs:element">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:choice minOccurs="0">
```

```
     <xs:element name="simpleType" type="xs:localSimpleType"/>
     <xs:element name="complexType" type="xs:localComplexType"/>
    </xs:choice>
    <xs:group ref="xs:identityConstraint" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="substitutionGroup" use="prohibited"/>
   <xs:attribute name="final" use="prohibited"/>
   <xs:attribute name="abstract" use="prohibited"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:element name="element" type="xs:topLevelElement" id="element">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-element"/>
 </xs:annotation>
</xs:element>

<xs:complexType name="group" abstract="true">
 <xs:annotation>
  <xs:documentation>
   group type for explicit groups, named top-level groups and
   group references
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:group ref="xs:particle" minOccurs="0" maxOccurs="unbounded"/>
   <xs:attributeGroup ref="xs:defRef"/>
   <xs:attributeGroup ref="xs:occurs"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="realGroup">
 <xs:complexContent>
  <xs:restriction base="xs:group">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:choice minOccurs="0" maxOccurs="1">
     <xs:element ref="xs:all"/>
     <xs:element ref="xs:choice"/>
     <xs:element ref="xs:sequence"/>
    </xs:choice>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="namedGroup">
 <xs:annotation>
  <xs:documentation>
   Should derive this from realGroup, but too complicated
   for now
  </xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element ref="xs:annotation" minOccurs="0"/>
  <xs:choice minOccurs="1" maxOccurs="1">
   <xs:element name="all">
    <xs:complexType>
     <xs:complexContent>
```

```
      <xs:restriction base="xs:all">
       <xs:group ref="xs:allModel"/>
       <xs:attribute name="minOccurs" use="prohibited"/>
       <xs:attribute name="maxOccurs" use="prohibited"/>
      </xs:restriction>
     </xs:complexContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="choice" type="xs:simpleExplicitGroup"/>
   <xs:element name="sequence" type="xs:simpleExplicitGroup"/>
  </xs:choice>
 </xs:sequence>
 <xs:attribute name="name" use="required" type="xs:NCName"/>
 <xs:attribute name="ref" use="prohibited"/>
 <xs:attribute name="minOccurs" use="prohibited"/>
 <xs:attribute name="maxOccurs" use="prohibited"/>
</xs:complexType>

<xs:complexType name="groupRef">
 <xs:complexContent>
  <xs:restriction base="xs:realGroup">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="ref" use="required" type="xs:QName"/>
   <xs:attribute name="name" use="prohibited"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="explicitGroup">
 <xs:annotation>
  <xs:documentation>
   group type for the three kinds of group
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:restriction base="xs:group">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:group ref="xs:nestedParticle" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:NCName" use="prohibited"/>
   <xs:attribute name="ref" type="xs:QName" use="prohibited"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="simpleExplicitGroup">
 <xs:complexContent>
  <xs:restriction base="xs:explicitGroup">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:group ref="xs:nestedParticle" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="minOccurs" use="prohibited"/>
   <xs:attribute name="maxOccurs" use="prohibited"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:group name="allModel">
```

```
  <xs:sequence>
   <xs:element ref="xs:annotation" minOccurs="0"/>
   <xs:element name="element" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       restricted max/min
      </xs:documentation>
     </xs:annotation>
     <xs:complexContent>
      <xs:restriction base="xs:localElement">
       <xs:sequence>
        <xs:element ref="xs:annotation" minOccurs="0"/>
        <xs:choice minOccurs="0">
         <xs:element name="simpleType" type="xs:localSimpleType"/>
         <xs:element name="complexType" type="xs:localComplexType"/>
        </xs:choice>
        <xs:group ref="xs:identityConstraint" minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:attribute name="minOccurs" use="optional" default="1">
        <xs:simpleType>
         <xs:restriction base="xs:nonNegativeInteger">
          <xs:enumeration value="0"/>
          <xs:enumeration value="1"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="maxOccurs" use="optional" default="1">
        <xs:simpleType>
         <xs:restriction base="xs:allNNI">
          <xs:enumeration value="0"/>
          <xs:enumeration value="1"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:attribute>
      </xs:restriction>
     </xs:complexContent>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:group>

 <xs:complexType name="all">
  <xs:annotation>
   <xs:documentation>
    Only elements allowed inside
   </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
   <xs:restriction base="xs:explicitGroup">
    <xs:group ref="xs:allModel"/>
    <xs:attribute name="minOccurs" use="optional" default="1">
     <xs:simpleType>
      <xs:restriction base="xs:nonNegativeInteger">
       <xs:enumeration value="0"/>
       <xs:enumeration value="1"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="maxOccurs" use="optional" default="1">
     <xs:simpleType>
      <xs:restriction base="xs:allNNI">
```

```
      <xs:enumeration value="1"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:element name="all" id="all" type="xs:all">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-all"/>
 </xs:annotation>
</xs:element>

<xs:element name="choice" type="xs:explicitGroup" id="choice">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-choice"/>
 </xs:annotation>
</xs:element>

<xs:element name="sequence" type="xs:explicitGroup" id="sequence">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-sequence"/>
 </xs:annotation>
</xs:element>

<xs:element name="group" type="xs:namedGroup" id="group">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-group"/>
 </xs:annotation>
</xs:element>

<xs:complexType name="wildcard">
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:attribute name="namespace" type="xs:namespaceList"
                 use="optional" default="##any"/>
   <xs:attribute name="processContents"
                 use="optional" default="strict">
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="skip"/>
      <xs:enumeration value="lax"/>
      <xs:enumeration value="strict"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="any" id="any">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-any"/>
 </xs:annotation>
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:wildcard">
    <xs:attributeGroup ref="xs:occurs"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
```

```
</xs:element>

<xs:annotation>
 <xs:documentation>
  simple type for the value of the 'namespace' attr of
  'any' and 'anyAttribute'</xs:documentation>
</xs:annotation>
<xs:annotation>
 <xs:documentation>
  Value is
    ##any      - - any non-conflicting WFXML/attribute at all
    ##other    - - any non-conflicting WFXML/attribute from
                   namespace other than targetNS
    ##local    - - any unqualified non-conflicting WFXML/attribute
                   one or     - - any non-conflicting WFXML/attribute from
                   more URI        the listed namespaces
                   references
                  (space separated)
   ##targetNamespace or ##local may appear in the above list, to
                   refer to the targetNamespace of the enclosing
                   schema or an absent targetNamespace respectively
 </xs:documentation>
</xs:annotation>

<xs:simpleType name="namespaceList">
 <xs:annotation>
  <xs:documentation>
   A utility type, not for public use
  </xs:documentation>
 </xs:annotation>
 <xs:union>
  <xs:simpleType>
   <xs:restriction base="xs:token">
    <xs:enumeration value="##any"/>
    <xs:enumeration value="##other"/>
   </xs:restriction>
  </xs:simpleType>
  <xs:simpleType>
   <xs:list>
    <xs:simpleType>
     <xs:union memberTypes="xs:anyURI">
      <xs:simpleType>
       <xs:restriction base="xs:token">
        <xs:enumeration value="##targetNamespace"/>
        <xs:enumeration value="##local"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:union>
    </xs:simpleType>
   </xs:list>
  </xs:simpleType>
 </xs:union>
</xs:simpleType>

<xs:element name="attribute" type="xs:topLevelAttribute" id="attribute">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-attribute"/>
 </xs:annotation>
</xs:element>

<xs:complexType name="attributeGroup" abstract="true">
 <xs:complexContent>
```

```
  <xs:extension base="xs:annotated">
   <xs:group ref="xs:attrDecls"/>
   <xs:attributeGroup ref="xs:defRef"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="namedAttributeGroup">
 <xs:complexContent>
  <xs:restriction base="xs:attributeGroup">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:group ref="xs:attrDecls"/>
   </xs:sequence>
   <xs:attribute name="name" use="required" type="xs:NCName"/>
   <xs:attribute name="ref" use="prohibited"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="attributeGroupRef">
 <xs:complexContent>
  <xs:restriction base="xs:attributeGroup">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="ref" use="required" type="xs:QName"/>
   <xs:attribute name="name" use="prohibited"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:element name="attributeGroup" type="xs:namedAttributeGroup" id="attributeGroup">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-attributeGroup"/>
 </xs:annotation>
</xs:element>

<xs:element name="include" id="include">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-include"/>
 </xs:annotation>
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:attribute name="schemaLocation" type="xs:anyURI" use="required"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>

<xs:element name="redefine" id="redefine">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-redefine"/>
 </xs:annotation>
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:openAttrs">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
     <xs:element ref="xs:annotation"/>
     <xs:group ref="xs:redefinable"/>
    </xs:choice>
```

```
    <xs:attribute name="schemaLocation" type="xs:anyURI" use="required"/>
    <xs:attribute name="id" type="xs:ID"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>

<xs:element name="import" id="import">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-import"/>
 </xs:annotation>
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:attribute name="namespace" type="xs:anyURI"/>
    <xs:attribute name="schemaLocation" type="xs:anyURI"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>

<xs:element name="selector" id="selector">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-selector"/>
 </xs:annotation>
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:attribute name="xpath" use="required">
     <xs:simpleType>
      <xs:annotation>
       <xs:documentation>
        A subset of XPath expressions for use
        in selectors
       </xs:documentation>
       <xs:documentation>
        A utility type, not for public use
       </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:token">
       <xs:annotation>
        <xs:documentation>
         The following pattern is intended to allow XPath
         expressions per the following EBNF:
         Selector    ::=    Path ( '|' Path )*
         Path    ::=    ('.//')? Step ( '/' Step )*
         Step    ::=    '.' | NameTest
         NameTest    ::=    QName | '*' | NCName ':' '*'
         child:: is also allowed
        </xs:documentation>
       </xs:annotation>
       <xs:pattern value="(\.//)?(((child::)?((\i\c*:)?
                          (\i\c*|\*)))|\.)(/(((child::)?
                          ((\i\c*:)?(\i\c*|\*)))|\.))*
                          (\|(\.//)?(((child::)?((\i\c*:)?
                          (\i\c*|\*)))|\.)(/(((child::)?
                          ((\i\c*:)?(\i\c*|\*)))|\.))*)*">
       </xs:pattern>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
   </xs:extension>
```

```
  </xs:complexContent>
 </xs:complexType>
</xs:element>

<xs:element name="field" id="field">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-field"/>
 </xs:annotation>
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:attribute name="xpath" use="required">
     <xs:simpleType>
      <xs:annotation>
       <xs:documentation>
         A subset of XPath expressions for use  in fields
       </xs:documentation>
       <xs:documentation>
        A utility type, not for public use
       </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:token">
       <xs:annotation>
        <xs:documentation>
         The following pattern is intended to allow XPath
         expressions per the same EBNF as for selector,
         with the following change:
         Path    ::=    ('.//')? ( Step '/' )* ( Step | '@' NameTest )
        </xs:documentation>
       </xs:annotation>
       <xs:pattern value="(\.//)?((((child::)?((\i\c*:)?(\i\c*|\*)))|\.)/)*
                         ((((child::)?((\i\c*:)?(\i\c*|\*)))|\.)|((attribute::|@)
                         ((\i\c*:)?(\i\c*|\*))))(\|(\.//)?((((child::)?((\i\c*:)?
                         (\i\c*|\*)))|\.)/)*((((child::)?((\i\c*:)?(\i\c*|\*)))|\.)
                         |((attribute::|@)((\i\c*:)?(\i\c*|\*)))))*">
       </xs:pattern>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>

<xs:complexType name="keybase">
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:sequence>
    <xs:element ref="xs:selector"/>
    <xs:element ref="xs:field" minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:NCName" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:group name="identityConstraint">
 <xs:annotation>
  <xs:documentation>
   The three kinds of identity constraints, all with
   type of or derived from 'keybase'.
  </xs:documentation>
```

```
 </xs:annotation>
 <xs:choice>
  <xs:element ref="xs:unique"/>
  <xs:element ref="xs:key"/>
  <xs:element ref="xs:keyref"/>
 </xs:choice>
</xs:group>

<xs:element name="unique" type="xs:keybase" id="unique">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-unique"/>
 </xs:annotation>
</xs:element>

<xs:element name="key" type="xs:keybase" id="key">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-key"/>
 </xs:annotation>
</xs:element>

<xs:element name="keyref" id="keyref">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-keyref"/>
 </xs:annotation>
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:keybase">
    <xs:attribute name="refer" type="xs:QName" use="required"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>

<xs:element name="notation" id="notation">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-notation"/>
 </xs:annotation>
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:attribute name="name" type="xs:NCName" use="required"/>
    <xs:attribute name="public" type="xs:public" use="required"/>
    <xs:attribute name="system" type="xs:anyURI"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>

<xs:simpleType name="public">
 <xs:annotation>
  <xs:documentation>
   A utility type, not for public use
  </xs:documentation>
  <xs:documentation>
   A public identifier, per ISO 8879
  </xs:documentation>
 </xs:annotation>
 <xs:restriction base="xs:token"/>
</xs:simpleType>

<xs:element name="appinfo" id="appinfo">
 <xs:annotation>
```

```
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-appinfo"/>
 </xs:annotation>
 <xs:complexType mixed="true">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
   <xs:any processContents="lax"/>
  </xs:sequence>
  <xs:attribute name="source" type="xs:anyURI"/>
 </xs:complexType>
</xs:element>

<xs:element name="documentation" id="documentation">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-documentation"/>
 </xs:annotation>
 <xs:complexType mixed="true">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
   <xs:any processContents="lax"/>
  </xs:sequence>
  <xs:attribute name="source" type="xs:anyURI"/>
  <xs:attribute ref="xml:lang"/>
 </xs:complexType>
</xs:element>

<xs:element name="annotation" id="annotation">
 <xs:annotation>
  <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-annotation"/>
 </xs:annotation>
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:openAttrs">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
     <xs:element ref="xs:appinfo"/>
     <xs:element ref="xs:documentation"/>
    </xs:choice>
    <xs:attribute name="id" type="xs:ID"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>

<xs:annotation>
 <xs:documentation>
  notations for use within XML Schema schemas</xs:documentation>
</xs:annotation>

<xs:notation name="XMLSchemaStructures" public="structures"
             system="http://www.w3.org/2000/08/XMLSchema.xsd"/>
<xs:notation name="XML" public="REC-xml-19980210"
             system="http://www.w3.org/TR/1998/REC-xml-19980210"/>

<xs:complexType name="anyType" mixed="true">
 <xs:annotation>
  <xs:documentation>
   Not the real urType, but as close an approximation as we can
   get in the XML representation
  </xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:any minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:anyAttribute/>
</xs:complexType>
```

```
</xs:schema>
```

Appendice F

Specifiche RDF Schema

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

 <rdfs:Class rdf:ID="Resource">
  <rdfs:label xml:lang="en">Resource</rdfs:label>
  <rdfs:label xml:lang="fr">Ressource</rdfs:label>
  <rdfs:comment>The most general class</rdfs:comment>
 </rdfs:Class>

 <rdf:Property about="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">
  <rdfs:label xml:lang="en">type</rdfs:label>
  <rdfs:label xml:lang="fr">type</rdfs:label>
  <rdfs:comment>Indicates membership of a class</rdfs:comment>
  <rdfs:range rdf:resource="#Class"/>
 </rdf:Property>

 <rdf:Property ID="comment">
  <rdfs:label xml:lang="en">comment</rdfs:label>
  <rdfs:label xml:lang="fr">commentaire</rdfs:label>
  <rdfs:domain rdf:resource="#Resource"/>
  <rdfs:comment>Use this for descriptions</rdfs:comment>
  <rdfs:range rdf:resource="#Literal"/>
 </rdf:Property>

 <rdf:Property ID="label">
  <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:label xml:lang="en">label</rdfs:label>
  <rdfs:label xml:lang="fr">label</rdfs:label>
  <rdfs:domain rdf:resource="#Resource"/>
  <rdfs:comment>Provides a human-readable version of a resource name.</rdfs:comment>
  <rdfs:range rdf:resource="#Literal"/>
 </rdf:Property>

 <rdfs:Class rdf:ID="Class">
  <rdfs:label xml:lang="en">Class</rdfs:label>
  <rdfs:label xml:lang="fr">Classe</rdfs:label>
  <rdfs:comment>The concept of Class</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Resource"/>
 </rdfs:Class>

 <rdf:Property ID="subClassOf">
```

```
 <rdfs:label xml:lang="en">subClassOf</rdfs:label>
 <rdfs:label xml:lang="fr">sousClasseDe</rdfs:label>
 <rdfs:comment>Indicates membership of a class</rdfs:comment>
 <rdfs:range rdf:resource="#Class"/>
 <rdfs:domain rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property ID="subPropertyOf">
 <rdfs:label xml:lang="en">subPropertyOf</rdfs:label>
 <rdfs:label xml:lang="fr">sousPropriétéDe</rdfs:label>
 <rdfs:comment>Indicates specialization of properties</rdfs:comment>
 <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdf:Property>

<rdf:Property ID="seeAlso">
 <rdfs:label xml:lang="en">seeAlso</rdfs:label>
 <rdfs:label xml:lang="fr">voirAussi</rdfs:label>
 <rdfs:comment>
  Indicates a resource that provides information about the subject
  resource.
 </rdfs:comment>
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Property>

<rdf:Property ID="isDefinedBy">
 <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:subPropertyOf rdf:resource="#seeAlso"/>
 <rdfs:label xml:lang="en">isDefinedBy</rdfs:label>
 <rdfs:label xml:lang="fr">esDéfiniPar</rdfs:label>
 <rdfs:comment>
  Indicates a resource containing and defining the subject resource.
 </rdfs:comment>
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Property>

<rdfs:Class rdf:ID="ConstraintResource">
 <rdfs:label xml:lang="en">ConstraintResource</rdfs:label>
 <rdfs:label xml:lang="fr">RessourceContrainte</rdfs:label>
 <rdf:type resource="#Class"/>
 <rdfs:subClassOf rdf:resource="#Resource"/>
 <rdfs:comment>Resources used to express RDF Schema constraints.</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="ConstraintProperty">
 <rdfs:label xml:lang="en">ConstraintProperty</rdfs:label>
 <rdfs:label xml:lang="fr">PropriétéContrainte</rdfs:label>
 <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:subClassOf rdf:resource="#ConstraintResource"/>
 <rdfs:comment>Properties used to express RDF Schema constraints.</rdfs:comment>
</rdfs:Class>

<rdfs:ConstraintProperty rdf:ID="domain">
 <rdfs:label xml:lang="en">domain</rdfs:label>
 <rdfs:label xml:lang="fr">domaine</rdfs:label>
 <rdfs:comment>This is how we associate a class with
  properties that its instances can have</rdfs:comment>
</rdfs:ConstraintProperty>

<rdfs:ConstraintProperty rdf:ID="range">
```

```
 <rdfs:label xml:lang="en">range</rdfs:label>
 <rdfs:label xml:lang="fr">étendue</rdfs:label>
 <rdfs:comment>Properties that can be used in a
 schema to provide constraints</rdfs:comment>
 <rdfs:range rdf:resource="#Class"/>
 <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdfs:ConstraintProperty>

<rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property">
 <rdfs:label xml:lang="en">Property</rdfs:label>
 <rdfs:label xml:lang="fr">Propriété</rdfs:label>
 <rdfs:comment>The concept of a property.</rdfs:comment>
 <rdfs:subClassOf rdf:resource="#Resource"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Literal">
 <rdfs:label xml:lang="en">Literal</rdfs:label>
 <rdfs:label xml:lang="fr">Littéral</rdfs:label>
 <rdf:type resource="#Class"/>
 <rdfs:comment>
  This represents the set of atomic values, eg. textual strings.
 </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement">
 <rdfs:label xml:lang="en">Statement</rdfs:label>
 <rdfs:label xml:lang="fr">Déclaration</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Resource"/>
 <rdfs:comment>This represents the set of reified statements.</rdfs:comment>
</rdfs:Class>

<rdf:Property about="http://www.w3.org/1999/02/22-rdf-syntax-ns#subject">
 <rdfs:label xml:lang="en">subject</rdfs:label>
 <rdfs:label xml:lang="fr">sujet</rdfs:label>
 <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
 <rdfs:range rdf:resource="#Resource"/>
</rdf:Property>

<rdf:Property about="http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate">
 <rdfs:label xml:lang="en">predicate</rdfs:label>
 <rdfs:label xml:lang="fr">prédicat</rdfs:label>
 <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
 <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdf:Property>

<rdf:Property about="http://www.w3.org/1999/02/22-rdf-syntax-ns#object">
 <rdfs:label xml:lang="en">object</rdfs:label>
 <rdfs:label xml:lang="fr">objet</rdfs:label>
 <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
</rdf:Property>

<rdfs:Class rdf:ID="Container">
 <rdfs:label xml:lang="en">Container</rdfs:label>
 <rdfs:label xml:lang="fr">Enveloppe</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Resource"/>
 <rdfs:comment>This represents the set Containers.</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag">
 <rdfs:label xml:lang="en">Bag</rdfs:label>
 <rdfs:label xml:lang="fr">Ensemble</rdfs:label>
```

```
  <rdfs:subClassOf rdf:resource="#Container"/>
 </rdfs:Class>

 <rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq">
  <rdfs:label xml:lang="en">Sequence</rdfs:label>
  <rdfs:label xml:lang="fr">Séquence</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Container"/>
 </rdfs:Class>

 <rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Alt">
  <rdfs:label xml:lang="en">Alt</rdfs:label>
  <rdfs:label xml:lang="fr">Choix</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Container"/>
 </rdfs:Class>

 <rdfs:Class rdf:ID="ContainerMembershipProperty">
  <rdfs:label xml:lang="en">ContainerMembershipProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 </rdfs:Class>

 <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#value">
  <rdfs:label xml:lang="en">object</rdfs:label>
  <rdfs:label xml:lang="fr">value</rdfs:label>
 </rdf:Property>

</rdf:RDF>
```

Appendice G

Esempio di Traduzione

Aprendo il file contenente lo schema ODL_{I^3} da tradurre questo viene visualizzato nella TextArea di sinistra:

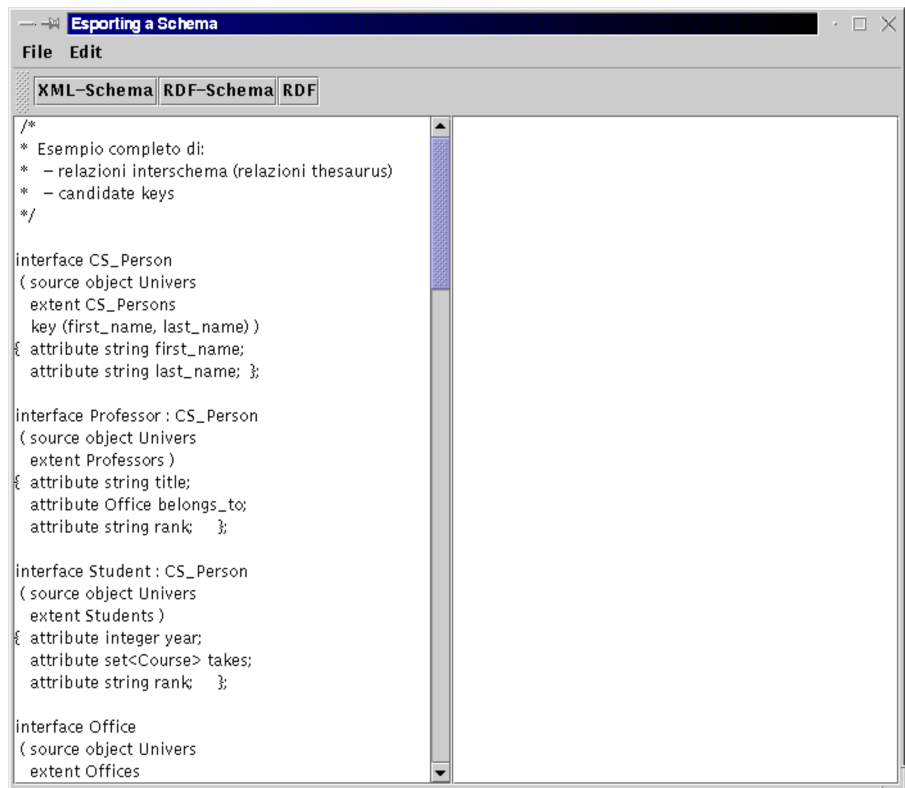


Figura G.1: Apertura file ODL_{I^3}

a questo punto i pulsanti presenti nella toolbar sono attivi ed é quindi possibile

lanciare la traduzione.

Premendo il pulsante **Xml-Schema** nella TextArea di destra verrá visualizzato il documento schema:

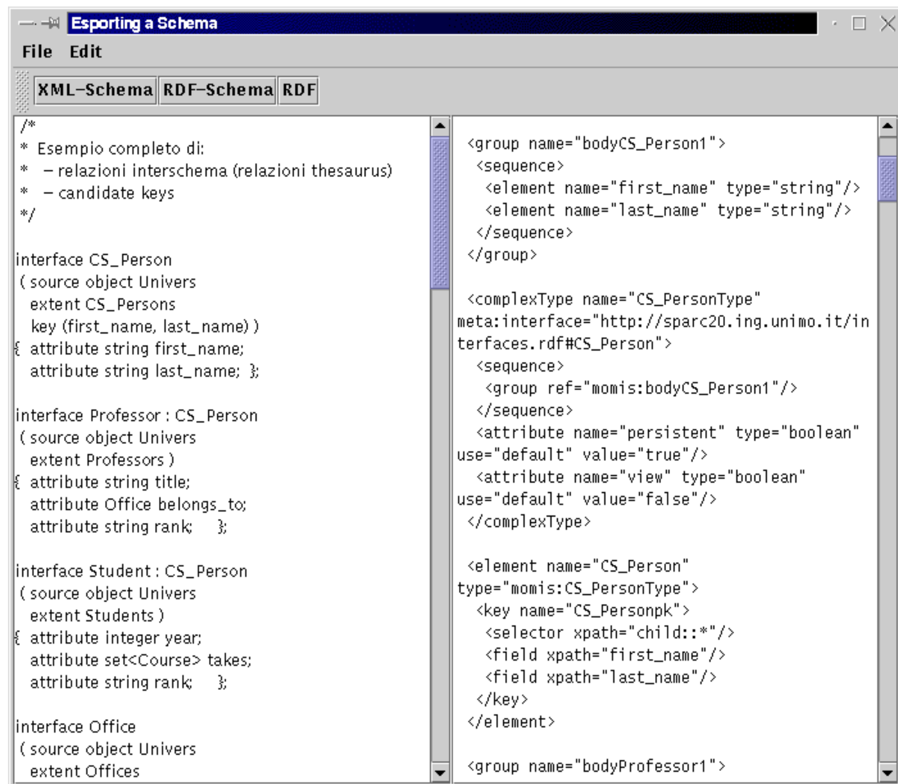


Figura G.2: documento Schema

per i motivi spiegati nel capitolo 5 é opportuna una fase di *ottimizzazione* del codice ottenuto; eliminando le definizione dei `group` nel caso di interfacce con 1 solo body.

Il pulsante **RDF-Schema** visualizza il documento RDF-Schema prodotto:

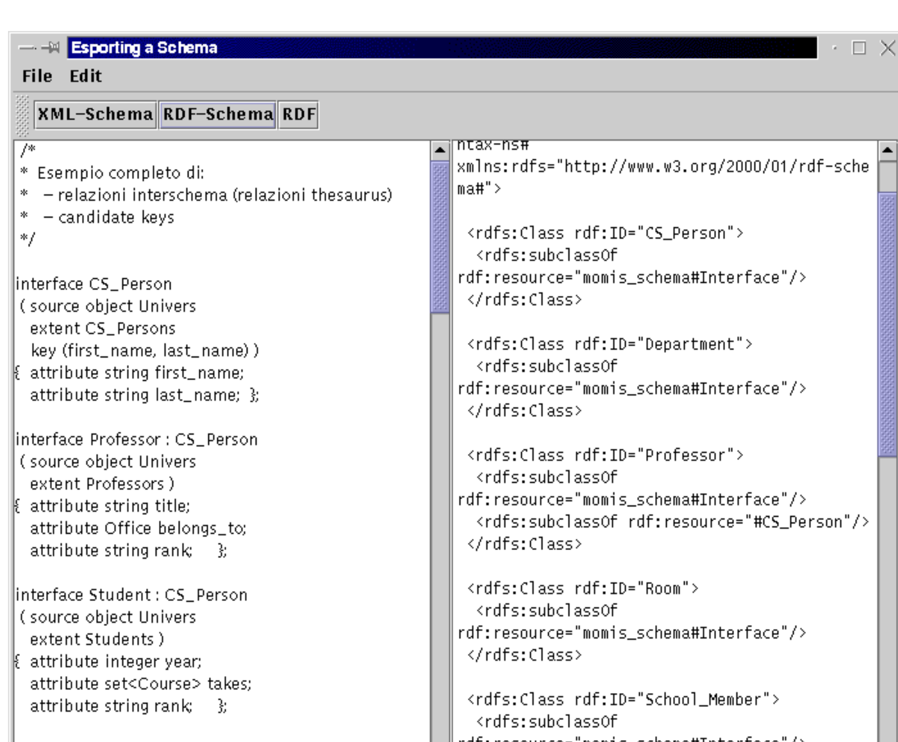


Figura G.3: documento RDF-Schema

anche per il documento RDF-Schema é opportuno l'intervento del progettista per l'inserimento di commenti che concorrano ad aumentarne la leggibilitá.

L'ultimo documento prodotto é quello RDF:

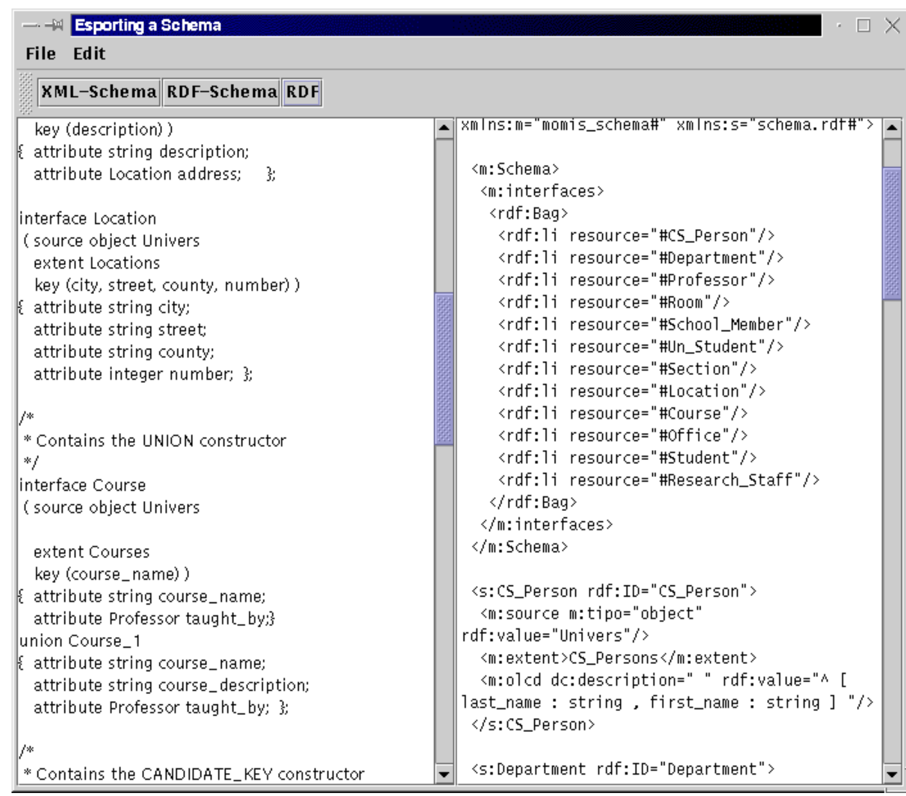


Figura G.4: documento RDF

per il documento RDF é opportuno intervenire sulle proprietá `olcd` per inserire un'opportuna descrizione del valore di tale proprietá.

Bibliografia

[1] Simone Montanari. Un approccio intelligente all' integrazione di sorgenti eterogenee di informazione. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1996-1997.

[2] Alberta Rabitti. Architettura di un mediatore per un sistema di integrazione di sorgenti distribuite ed autonome. Tesi di laurea, Università di Modena, Facoltà di ingegneria Informatica, 1997-1998. http://sparc20.dsi.unimo.it/tesi/index.html.

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

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

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

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

[9] 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.

[10] 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.

[11] 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.

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

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

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

[15] 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.

[16] 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, 1997.

[17] 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.

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

[19] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Consistency checking in complex object database schemata with integrity constraints. *IEEE Transactions on Knowledge and Data Engineering*, 10:576–598, July/August 1998.

[20] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.

[21] 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.

[22] 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.

[23] ODB-Tools staff. Odb-tools Project. Available at http://sparc20.dsi.unimo.it.

[24] S. Castano and V. De Antonellis. Semantic dictionary design for database interoperability. In *Proc. of Int. Conf. on Data Engineering, ICDE'97*, Birmingham, UK, 1997.

[25] 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.

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

[27] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[28] B. Everitt. *Computer-Aided Database Design: the DATAID Project*. Heinemann Educational Books Ltd, Social Science Research Council, 1974.

[29] D. Beneventano, A. Corni, S. Lodi, and M. Vincini. ODB: validazione di schemi e ottimizzazione semantica on-line per basi di dati object oriented. In *Quinto Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBD97, Verona*, pages 208–225, 1997.

[30] W. A. Woods and J. G. Schmolze. The KL-ONE family. In F. W. Lehman, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Pubblished as a Special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2-9.

[31] Berners-Lee. *Semantic web roadmap.* Internal note, 1998. World Wide Web Consortium.

[32] T. Berners-Lee J. Hendler and O. Lassila. *the Semantic Web*. *Scientific American*, May 2001. http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html.

[33] T.Bray. *RDF and Metadata*, January 2001. http://www.xml.com/pub/a/2001/01/24/rdf.html.

[34] S.Decker F. van Harmelen J.Broekstra M.Erdmann D.Fensel I.Horrocks M.Klein S.Melnik. *The Semantic Web - on the respective Roles of XML and RDF*. http://www.ontoknowledge.org/oil/downl/IEEE00.pdf.

[35] S.Bechhofer J.Broekstra S.Decker M.Erdmann D.Fensel C.Goble F.v.Harmelen I.Horrocks M.Klein D.McGuinness E.Motta P.Patel-Schneider S.Staab R.Studer. *An informal description of Standard OIL and Instance OIL.* whitepaper, november 2000. http://www.ontoknowledge.org/oil/downl/oil-whitepaper.pdf.

[36] F.van Harmelen P.F.Patel-Schneider and I.Horrocks. *Reference description of the DAML+OIL ontology markup language*, March 2001. http://www.daml.org/2001/03/reference.html.

[37] D.Lenzi. *Serializzazione di oggetti in formato XML nell'ambito del sistema MOMIS.* Master's thesis, Università di Modena, Italy, 2000.

[38] J. Hunter and C. Lagoze. *Combining RDF and XML Schemas to Enhance Interoperability Between Metadata Application Profiles*. http://archive.dstc.edu.au/RDU/staff/jane-hunter/www10/paper.html.

[39] Alberto Zanoli. Si-designer, un tool di ausilio all'integrazione di sorgenti di dati eterogenee distribuite: progetto e realizzazione. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998.

[40] World Wide Web Consortium. Xquery: A query language for xml - w3c working draft 15 february 2001. http://www.w3.org/TR/2001/WD-xquery-20010215, 2001.

[41] World Wide Web Consortium. Xml syntax for xquery 1.0 (xqueryx)- w3c working draft 07 june 2001. http://www.w3.org/TR/2001/WD-xqueryx-20010607, 2001.