UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Facoltà di Ingegneria - Sede di Modena

Corso di laurea in Ingegneria Informatica

Interoperabilità tra ontologie eterogenee: i traduttori OWL - ODL$_{I^3}$

Relatore

Chiar.ma Prof.ssa Sonia Bergamaschi

Tesi di Laurea di

Mirko Orsini

Correlatore

Chiar.mo Ing. Maurizio Vincini

Controrelatore

Chiar.mo Prof. Flavio Bonfatti

Anno Accademico 2003-2004

Keywords:
Semantic Web
Ontologie
ODL_{I^3}
OWL
Traduttori di ontologie

INDICE

Pag.

Pag.

Pag.

Pag.

Introduzione

Lo sviluppo del World Wide Web ha portato ad avere una quantità enorme di documenti e di informazioni fruibili, inoltre un numero sempre maggiore di utenti ha accesso al Web e desidera trovare le informazioni di cui ha bisogno in tempi brevi. Negli ultimi anni il rapido aumento delle informazioni disponibili ha reso più complicata la ricerca dei documenti e delle pagine web significative per gli utenti, infatti, se da una parte i motori di ricerca sono fondamentali per il loro recupero, dall'altra i risultati da essi presentati sono spesso insignificanti o completamente al di fuori del dominio di interesse. In questo contesto si è presentata l'esigenza di una nuova visione del Web: il Web Semantico. La maggior parte dei contenuti del Web è attualmente progettata per l'interazione umana, ma non per l'interazione con agenti software. Il Web Semantico darà una struttura ai contenuti significativi delle pagine Web, creando un ambiente in cui gli agenti software, spostandosi da pagina a pagina, possano eseguire sofisticate interrogazioni per gli utenti. Il Web Semantico non è separato dall'attuale, è invece una sua estensione nella quale ad ogni informazione viene attribuito un significato ben preciso. Le applicazioni software attuali possono analizzare facilmente i contenuti del Web per effettuarne presentazioni o semplici elaborazioni, ma in generale esse non hanno la capacità di interpretare ed elaborare la semantica dei contenuti. Nel prossimo futuro, gli sviluppi del Web Semantico, aumenteranno le capacità degli agenti software che saranno in grado di elaborare e comprendere i dati che oggi vengono semplicemente mostrati. La ricerca nell'ambito del Web Semantico è molto attiva sia in Europa (il programma IST supporta diversi progetti, On-To-Knowledge, SEWASIE, ecc.) che negli Stati Uniti.

Il World Wide Web Consortium (*W3C*) ha sviluppato, per la definizione di ontologie e di strumenti compatibili con l'architettura del Semantic Web, il linguaggio OWL (*Web Ontology Language*). OWL è stato creato per essere utilizzato da applicazioni software che intendono elaborare il contenuto delle informazioni, invece di rappresentare solamente

le informazioni agli umani. Questo linguaggio fornisce una maggiore comprensibilità dei significati e della semantica di concetti da parte di agenti software rispetto a XML, RDF e RDF-Schema. Rende inoltre più semplici le fasi di elaborazione automatica ed integrazione delle informazioni.

Questa tesi si inserisce all'interno del progetto **SEWASIE** (**SE**mantic **W**ebs and **A**gent**S** in **I**ntegrated **E**conomies), coordinato dall'Università degli studi di Modena e Reggio Emilia e sviluppato assieme a diversi partner commerciali ed accademici. Il progetto SEWASIE si propone di progettare ed implementare un motore di ricerca semantico che permetta di accedere a sorgenti di dati eterogenee nel Web. I componenti del progetto SEWASIE saranno sviluppati per creare e mantenere ontologie multilingue, con tecniche di inferenza basate sugli standards del W3C (XML, XML Schema, RDF(S), OWL), che pongono le basi per un meccanismo di ricerca avanzata e che forniscono la terminologia necessaria ad una comunicazione strutturata.

Il progetto SEWASIE si basa sullo sviluppo del sistema **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) [1–3], creato per la realizzazione di un processo semi-automatico di integrazione di sorgenti eterogenee e distribuite.
La parte centrale del sistema MOMIS è costituita dal *Mediatore* che ha lo scopo di fornire una visione integrata degli schemi locali. L'interazione con le singole sorgenti avviene attraverso dei wrappers, ognuno dei quali ottimizzato per una diversa tipologia di sorgente. Il wrapper deve fornire una rappresentazione dello schema della sorgente alla quale è connesso tramite il linguaggio comune ODL_{I^3}.

Obiettivo della presente tesi è stata l'analisi e l'implementazione di un wrapper per sorgenti di dati rappresentate tramite il linguaggio OWL. Sono stati realizzati i traduttori che consentono la trasformazione di un ontologia descritta in linguaggio OWL in uno schema ODL_{I^3}, e, viceversa, che permettono di esprimere un'ontologia rappresentata tramite il linguaggio ODL_{I^3}, in linguaggio OWL. OWL fornisce tre sottolinguaggi con espressività incrementale progettati per l'utilizzo da parte di utenti con necessità differenti per la descrizione di ontologie: rispettivamente OWL Lite, OWL DL e OWL Full. Il traduttore per ontologie ODL_{I^3} realizzato, permette la traduzione di ontologie espresse nei sottolin-

guaggi OWL DL e OWL Lite. Il traduttore per ontologie OWL permette di effettuare una traduzione dello schema ODL_{I^3} nel sottolinguaggio OWL DL o nel sottolinguaggio OWL Full.

Si mostra ora l'organizzazione della tesi ed il contenuto dei vari capitoli in essa presenti:

Il **Capitolo 1** descrive le caratteristiche del *Web Semantico* attraverso un tipico scenario del nuovo World Wide Web. Viene illustrato lo stato dell'arte per la rappresentazione della conoscenza, presentando i tre più importanti linguaggi attualmente utilizzati per lo sviluppo del Web Semantico:*XML*, *RDF* e *OWL*. Viene effettuata una descrizione dei componenti chiave per la rappresentazione e la fruibilità delle informazioni nel Web Semantico: le ontologie e gli agenti software.

Il **Capitolo 2** presenta il progetto SEWASIE (SEmantic Webs and AgentS in Integrated Economies) descrivendo gli obiettivi, le caratteristche e l'architettura del sistema. La seconda parte del capitolo è dedicata all'illustrazione del sistema MOMIS (Mediator envirOnment for Multiple Information Sources), in particolare viene presentato l'approccio semantico per l'integrazione di sorgenti eterogenee e viene descritta l'architettura del sistema.

Il **Capitolo 3** descrive il linguaggio ODL_{I^3}, utilizzato per una rappresentazione comune delle sorgenti di dati all'interno del sistema MOMIS. Nella prima parte del capitolo è presentato il linguaggio standard ODL (Object Definition Language), quindi viene descritto il linguaggio ODL_{I^3}, esteso per rispondere alle problematiche di integrazione di informazioni da fonti dati eterogenee.

Nel **Capitolo 4** viene presentato il linguaggio OWL (Web Ontology Language), un linguaggio per la definizione di ontologie per il Web Semantico sviluppato dal World Wide Web Consortium (W3C). Vengono illustrate le differenze tra i tre sottolinguaggi di OWL e viene fornita una descrizione dei costrutti del linguaggio per la rappresentazione di ontologie.

Il **Capitolo 5** illustra una proposta per la traduzione di ontologie dal linguaggio ODL_{I^3} al linguaggio OWL, tale traduzione è stata utilizzata per la realizzazione di un wrapper

impiegato all'interno del sistema MOMIS. Nel capitolo viene effettuato un confronto tra i costrutti dei due linguaggi e vengono evidenziate le differenze per la traduzione degli schemi nei diversi sottolinguaggi di OWL.

Il **Capitolo 6** presenta una proposta per la traduzione di ontologie descritte tramite il linguaggio OWL in linguaggio ODL$_{I^3}$, la traduzione è stata utilizzata per la realizzazione del wrapper utilizzato nel sistema MOMIS. Nel capitolo viene effettuato un confronto tra i costrutti dei diversi sottolinguaggi di OWL, evidenziando gli elementi non esprimibili attraverso il linguaggio ODL$_{I^3}$.

Nel **Capitolo 7** vengono presentate le principali caratteristiche del software realizzato e delle tecnologie utilizzate per la realizzazione del wrapper.

1. WEB SEMANTICO

In questo capitolo viene presentato il nuovo scenario introdotto da una visione futura del Web: il Web Semantico (*Semantic Web*). Un articolo fondamentale sul Semantic Web è quello scritto da J. Hendler, O. Lassila e T. Berners-Lee apparso su Scientific America nel 2001 [4]. Gli autori descrivono un nuovo "mondo" in cui le persone interagiscono con i loro dispositivi elettronici, in un modo completamente diverso da quello attuale. Essi immaginano che, per esempio, una persona possa essere in uno studio medico per fissare alcune sessioni di terapia, ma abbia già diversi appuntamenti. Essa potrà istruire attraverso il browser Web del palmare il proprio agente software per Web Semantico, potrà effettuare una ricerca per trovare i posti disponibili ad una distanza massima di 20 Km dalla propria casa e solo in un determinato orario del giorno. L'agente software, quindi, comincerà ad interagire con altri agenti presenti nei siti Web dei dottori dell'area circostante. In poco tempo l'agente presenterà un programma di appuntamenti per le sessioni di terapia. Se il programma è soddisfacente si potranno inserire gli appuntamenti in agenda ed inviare una email all'istruttore di tennis per disdire gli allenamenti per il mese seguente, se i risultati presentati non sono soddisfacenti si potranno aggiungere nuove preferenze ed effettuare una nuova ricerca.

Allo stato attuale il World Wide Web non è in grado di effettuare operazioni con le caratteristiche descritte precedentemente. La maggior parte dei contenuti Web è attualmente progettata per l'interazione umana, non per l'interazione con agenti software. Le applicazioni software attuali possono analizzare facilmente i contenuti del Web per effettuare presentazioni o semplici elaborazioni, ma in generale esse non hanno la capacità di interpretare ed elaborare la semantica dei contenuti del Web. Questo è il motivo per cui la ricerca nell'ambito del Web Semantico è molto attiva sia in Europa (il programma IST supporta diversi progetti, On-To-Knowledge [5], SEWASIE, ecc.) che negli Stati Uniti.

1.1 Descrizione del Web Semantico (Semantic Web)

Oggi il Web è un immenso insieme di informazioni utili per le persone ma poco comprensibili per gli agenti software. Attualmente il World Wide Web è alla sua seconda versione: la prima vedeva le pagine HTML statiche cambiare il modo in cui le persone condividevano le informazioni, mentre la seconda è quella che si sta sperimentando oggi con pagine dinamiche create in base ad interrogazioni su database. L'aumento di informazioni disponibili e dell'utilizzo del Web, sta creando nuove aspettative negli utenti. In altre parole è il momento di pensare ad una terza versione del WWW: il Web Semantico. Il Web Semantico darà una struttura ai contenuti significativi delle pagine Web, creando un ambiente in cui gli agenti software, spostandosi da pagina a pagina, possano prontamente eseguire sofisticate interrogazioni per gli utenti. Quando un agente software visita la pagina Web di un professore esso saprà non solo che la pagina ha diverse parole chiave come "ontologia, rappresentazione della conoscenza, intelligenza artificiale" ma anche che le lezioni si terranno Venerdì mattina alle 11:00. Questo contenuto semantico sarà codificato nelle pagine Web attraverso l'utilizzo di applicazioni software trasparenti agli utenti per la scrittura di pagine Web semantiche. Il Web semantico non è un Web separato dall'attuale, ne è invece un'estensione nella quale ad ogni informazione viene dato un significato ben preciso, permettendo in questo modo la cooperazione tra agenti software e persone. Il primo passo nella costruzione del Web Semantico sopra la struttura del Web attuale è già in preparazione. Nel prossimo futuro, questi sviluppi aumenteranno le capacità degli agenti software che saranno in grado di elaborare e comprendere i dati che oggi vengono semplicemente mostrati.

La caratteristica essenziale del World Wide Web è la sua universalità. La potenza di un collegamento in un ipertesto è data dal fatto che "può collegare qualsiasi cosa a qualsiasi altra cosa". La tecnologia del Web, quindi, non fa discriminazioni tra l'informazione commerciale e l'informazione accademica, così come non discrimina culture o mezzi di comunicazione. Le informazioni possono variare secondo diverse variabili: una di queste è la differenza tra le informazioni prodotte essenzialmente per la comprensione da parte delle persone e quelle prodotte principalmente per la comprensione da parte di agenti

software. Da una parte c'è tutto quello che va dalla televisione commerciale alla poesia, dall'altra parte ci sono database e programmi software. Oggi il Web viene sviluppato più velocemente per una comprensione dei documenti da parte delle persone, piuttosto che per un'elaborazione automatica dei dati e delle informazioni. Il Web Semantico punta a sviluppare queste ultime capacità. Come Internet, il Web Semantico sarà il più decentralizzato possibile. Questo nuova visione del Web sta generando molta eccitazione ad ogni livello, dalle maggiori aziende ai singoli utenti e fornirà benefici che sono difficili o impossibili da prevedere.

1.2 Rappresentazione della conoscenza

Per fornire le funzioni richieste dal Web Semantico, gli agenti software devono avere la capacità di accedere a collezioni strutturate di informazioni ed utilizzare un insieme di regole di inferenza per effettuare un ragionamento automatico. I ricercatori delle Basi di Dati e dell'Intelligenza Artificiale hanno studiato questi sistemi da prima che il Web fosse sviluppato. La rappresentazione della conoscenza è attualmente in uno stato comparabile a quello dell'ipertesto prima dell'avvento del Web. Sono stati condotti molti studi in questo ambito ma fino ad ora c'è solo qualche dimostrazione delle numerose potenziali. I sistemi di rappresentazione della conoscenza tradizionali sono tipicamente centralizzati e richiedono ad ognuno di condividere esattamente la stessa definizione per i concetti comuni. Un controllo centrale aumenta le dimensioni di tali sistemi e diventa rapidamente ingestibile. Il linguaggio per la descrizione del Web Semantico deve essere tanto espressivo quanto desiderato, in modo da permettere ai ragionatori diversi livelli di interpretazione. La sfida del Web Semantico, quindi, è quella di fornire un linguaggio che possa descrivere i dati e le regole per il ragionamento su di essi e che permetta di utilizzare regole da ogni sistema di rappresentazione della conoscenza che possa essere esportato sul Web. Al momento gli sforzi della comunità del Web Semantico sono puntati all'utilizzo, nel Web, di regole di inferenza, che possano determinare il corso delle azioni in risposta alle interrogazioni. Attualmente sono presenti tre importanti linguaggi per lo sviluppo del Web Semantico : *eXtensible Markup Language* (*XML* [6–8]), *Resource Description Fra-*

mework (*RDF* [9–12]) e *Web Ontology Language* (*OWL* [13–18]).
XML permette di creare costrutti personalizzati: i programmi possono utilizzare questi costrutti per elaborazioni sofisticate, ma ai dati non può essere data nessuna interpretazione se non ne è conosciuto a priori il significato. In altre parole XML permette di creare documenti strutturati ma non può dire niente sulle relazioni che gli oggetti hanno nella struttura.
I significati possono essere espressi tramite RDF, che li codifica in triple, ogni tripla è formata da un soggetto, un verbo ed un oggetto. Queste triple possono essere scritte utilizzando costrutti XML. In RDF si possono effettuare affermazioni del tipo: una particolare cosa (persona, pagina Web o altro) ha una proprietà ("è sorella di", "è l'autore di") con un certo valore (un'altra persona, un'altra pagina Web). Questa struttura a triple descrive in modo naturale la maggior parte dei dati elaborati dagli agenti software. Soggetti e oggetti sono entrambi identificati da un URI (Universal Resource Identifier [19]), come utilizzati nelle pagine Web. I verbi, che sono anch'essi identificati attraverso gli URIs, permettono di definire nuovi concetti dichiarando un URI per un concetto da qualche parte nel Web.
OWL è stato creato per essere utilizzato da applicazioni che intendono elaborare il contenuto delle informazioni, invece di rappresentare solamente le informazioni agli umani. Tramite questo linguaggio è possibile avere una maggiore comprensibilità dei significati e della semantica di concetti da parte di agenti software rispetto a XML, RDF e RDF-Schema. Rende inoltre più semplici le fasi di elaborazione automatica ed integrazione delle informazioni. OWL è stato costruito sulla base di RDF e RDF-Schema e rispetto ad essi aggiunge diversi costrutti per la definizione di concetti e delle loro interrelazioni: disgiunzione tra classi, restrizioni di cardinalità per le proprietà, relazioni di uguaglianza tra classi o proprietà, classi enumerate, nuovi tipi di proprietà e nuove caratteristiche per le proprietà.

1.3 Ontologie

In un sistema in cui le informazioni sono distribuite su diversi siti Web o databases, un'applicazione che voglia confrontare o combinare le informazioni contenute in due da-

tabases deve risolvere il problema dell'integrazione delle informazioni. Un certo termine può avere, per esempio, due diversi significati nelle due diverse basi di dati. Idealmente, l'applicazione dovrebbe trovare i significati comuni per ogni database essa incontri. Una soluzione a questo problema è fornita dai componenti chiave del Web Semantico, le collezioni di informazioni chiamate *ontologie*. *Ontologia* [20, 21] è un termine derivante dalla filosofia e si riferisce alla scienza che descrive i tipi di entità nel mondo e le loro relazioni. Il tipo di ontologia utilizzato per il Web è formata da una tassonomia e da un insieme di regole di inferenza. La tassonomia definisce le classi di oggetti e le relazioni che si hanno tra essi. Per esempio, un indirizzo può essere definito come un tipo di luogo, il codice di avviamento postale può essere definito come una proprietà di un luogo, e così via. Classi, sottoclassi e relazioni tra le entità sono strumenti molto efficaci per l'utilizzo sul Web. Molte relazioni tra entità, infatti, possono essere espresse assegnando proprietà alle classi e permettendo alle sottoclassi di ereditare tali proprietà.

Le regole di inferenza nelle ontologie forniscono ulteriori capacità. Un'ontologia può esprimere regole per le classi e relazioni tra esse in modo tale che un agente software possa trarre qualche conclusione. L'agente software non capisce veramente ognuna di queste informazioni, ma può manipolare i termini in modo che risultino più utili e significativi per gli utenti. Con l'utilizzo delle ontologie sul Web, cominciano a delinearsi soluzioni al problema della terminologia. Il significato dei termini o dei codici XML utilizzati in una pagina Web può essere definito attraverso dei puntatori che vanno dalla pagina ad un ontologia. Le ontologie possono ampliare le funzioni del Web in diversi modi. Esse possono essere utilizzate semplicemente per aumentare l'accuratezza delle ricerche sul Web: gli agenti software attraverso cui sono effettuate le ricerche possono, infatti, visualizzare solo le pagine che si riferiscono al concetto preciso indicato invece di visualizzare tutte le pagine che usano parole chiave ambigue. Applicazioni più avanzate utilizzeranno le ontologie per correlare l'informazione di una pagina Web con la struttura di conoscenza associata e le relative regole di inferenza.

1.4 Agenti

La reale potenza del Web Semantico potrà essere espressa quando saranno realizzate molte applicazioni che utilizzeranno contenuti Web da diverse sorgenti, elaboreranno le informazioni e condivideranno i risultati con altre applicazioni. L'efficacia di queste applicazioni, chiamate agenti software [22], aumenterà esponenzialmente con la disponibilità di contenuti Web comprensibili da essi e attraverso la fruibilità di servizi automatici (inclusi altri agenti software). Un agente è infatti un componente software che gode delle seguenti proprietà:

- autonomia: gli agenti incapsulano un qualche stato e fanno decisioni basandosi su questo stato, senza alcuna interazione umana;
- reattività: gli agenti sono situati in un ambiente, essi percepiscono i cambiamenti dell'ambiente e possono rispondere a questi cambiamenti;
- attività: gli agenti possono mostrare comportamenti diretti ad un certo scopo prendendo l'iniziativa;
- abilità sociale: gli agenti interagiscono con altri agenti e con altre applicazioni software.

Con queste caratteristiche, un agente può navigare automaticamente nel Web e spostarsi tra un sito e l'altro per compiere un determinato compito. Se un agente software naviga nel Web Semantico può avere la percezione dei contenuti memorizzati nelle sorgenti visitate. In questo modo, esso può modificare il proprio comportamento in base ai siti Web visitati e, per esempio, può seguire collegamenti specifici perchè semanticamente corretti rispetto al suo scopo. Quindi il Web Semantico permette ad un agente software di avere una conoscenza dell'ambiente in cui è istanziato e conseguentemente di reagire all'ambiente senza errori. Contributi importanti a questo fondamentale componente del Web Semantico sono stati dati da *AgentLink*[1], un'eccellente rete europea fondata dal programma UE IST, che è molto attivo nello studio e nella produzione di agenti software.

[1] http://www.agentlink.org/

2. IL PROGETTO SEWASIE ED IL SISTEMA MOMIS

2.1 SEWASIE

SEWASIE[1] è l'acronimo di "SEmantic Webs and AgentS in Integrated Economies". Il progetto è inserito nell'ambito del programma IST (IST-2001-34825) dell'Unione Europea assieme alla "Semantic Web Action Line". Il progetto è coordinato dall'Università degli studi di Modena e Reggio Emilia (Italia) e gli altri partner sono: CNA SERVIZI Modena s.c.a.r.l. (Italia), Università di Roma "LaSapienza" (Italia), Rheinisch Westfaelische Technische Hochschule Aachen (Germania), The Victoria University of Manchester (UK), Thinking Networks AG (Germania), IBM Italia SPA (Italia), Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung eingetragener Verein (Germania). Il progetto è cominciato nel Maggio del 2002 e terminerà nell'Aprile del 2005.
Il progetto SEWASIE si propone di progettare ed implementare un motore di ricerca semantico che permetta di accedere a sorgenti di dati eterogenee nel Web, fornendo le basi per una comunicazione strutturata sicura attraverso il Web.

2.1.1 Gli obiettivi del progetto SEWASIE

Gli obiettivi del progetto sono: fornire un'architettura distribuita basata su agenti software per la comunicazione e per la ricerca semantica utilizzando ontologie multilingua specifiche delle varie comunità; dotare le ontologie di una inferenza basata sugli standards del W3C; fornire prototipi che incontrino le necessità delle piccole e medie imprese in un contesto europeo; ottenere esperienza pratica (requisiti degli utenti, potenziale valore aggiunto, rischi, ecc.). In particolare il programma concentra l'attenzione su come assistere reti di piccole e medie imprese (la cosìdetta Economia Integrata) nell'aumento delle

[1] http://www.sewasie.org/

loro capacità di gestione delle informazioni intra e inter-organizzative. Il progetto include inoltre novità tecniche per l'arricchimento semantico, la gestione delle interrogazioni e le tecniche di presentazione nell'acquisizione delle informazioni multilingua dal Web.

2.1.2 Lo scenario Commerciale

In Europa, la maggior parte del tessuto industriale è composto da piccole e medie imprese agricole, manufatturiere, commerciali e società di servizi. Per ragioni storiche e sociali, esse spesso si aggregano in gruppi settoriali (che la letteratura economica chiama distretti industriali); al giorno d'oggi, questo tipo di organizzazioni economiche sono rese vulnerabili da parte della globalizzazione e nel futuro lo saranno sempre di più.

In tali condizioni, trovare (il giusto fornitore, un metodo di lavoro innovativo, un nuovo negozio, e così via) ed essere trovati (da possibili clienti, partners o sponsors) fa la differenza; gli strumenti di Internet correnti (i motori di ricerca) sono inadeguati causa la loro difficoltà di utilizzo (piccole aziende potrebbero non avere le infrastrutture tecnologiche e le conoscenze necessarie) e perchè una semplice interrogazione produce pagine e pagine di collegamenti, la maggior parte dei quali privi di utilità. Supponiamo che un'azienda debba avere informazioni su un certo argomento (un prodotto, un fornitore, una moda, una norma, ecc.): generalmente essi devono fare diverse ricerche. Per esempio, se cercano un nuovo "trattamento per la tintura del tessuto" essi devono trovare chi lo produce e se esistono norme specifiche per lo smaltimento dei materiali residui della tinteggiatura. Prima di tutto essi dovranno fare una ricerca usando il termine "tinteggiatura tessuto". Chiaramente, il motore di ricerca elencherà i collegamenti (540 nel caso di www.google.com) riguardanti non solo i produttori di attrezzature per la colorazione dei tessuti, ma anche la storia del tessuto, le tecniche di tintura, e così via. Dopo aver individuato un possibile fornitore, l'azienda deve trovare le leggi e le norme rigurdanti lo smaltimento dei rifiuti e le relative interpretazioni. Il criterio di ricerca può essere, per esempio, il numero della legge, il termine "legge", l'interpretazione, ecc. Le ricerche diventano più difficoltose quando sono effettuate all'estero, in quanto esistono terminologie specifiche che potrebbero non essere familiari all'utente.

Supponiamo invece che l'utente sia un professionista e che abbia interesse per una sola parte della normativa. In questo caso gli occorrerà recuperare rapidamente non solo le leggi rilevanti, ma anche riferimenti, problemi connessi e così via; tutto questo cercato attraverso siti Web gratuiti o a registrazione.

Per questo motivo, l'utente ha bisogno di avere a sua disposizione un motore con un'interfaccia per le interrogazioni semplice da utilizzare, capace di estrarre da Internet le informazioni richieste e di visualizzarle in un formato piacevole. Semplice da utilizzare significa "utilizzabile da utenti inesperti e con scarse infrastrutture ".

Il requisito principale è quello di fornire risultati strutturati derivanti dall'interpretazione di interrogazioni generali, seguiti da tecniche di filtraggio basate su regole dello sviluppatore oppure sull'esperienza acquisita. Per esempio, iniziando una semplice ricerca costituita dal termine "perforatrice" il motore risponde con uno o più documenti contenenti informazioni in formato facilmente accessibile su venditori, prezzi, produttori, manuali sulle tecniche di utilizzo, importatori, ecc. Particolarmente interessante per i prodotti e i produttori sarà sapere se ci sono sconti per l'acquisto in stock o se c'è qualche vendita all'asta o altri meccanismi di negoziazione per l'acquisto (da siti Web specifici).

2.1.3 L'architettura del progetto SEWASIE

I metodi e gli strumenti del progetto SEWASIE saranno sviluppati per creare e mantenere ontologie multilingue, con tecniche di inferenza basate sugli standards del W3C (XML, XML Schema, RDF(S), OWL), che pongono le basi per un meccanismo di ricerca avanzata e che forniscono la terminologia per una comunicazione strutturata. I risultati delle ricerche saranno personalizzati e visualizzati secondo le preferenze dell'utente. Il progetto SEWASIE fornirà un'architettura aperta e distribuita basata su agenti intelligenti (brokers, mediatori e traduttori) per avere scalabilità e flessibilità, ossia la capacità di addattarsi ai cambiamenti e alla crescita dell'ambiente e di interoperare con altri sistemi, offrendo un punto centrale di accesso all'utente. L'utente sarà supportato nell'interrogazione di sorgenti di dati eterogenee sul Web. L'interrogazione dell'utente sarà inviata ad un agente software per le interrogazioni che elaborerà e risolverà la richiesta spostandosi

attraverso i nodi di informazione SEWASIE (Sewasie Information Node *SINode*) per recuperare le informazioni richieste dall'utente (vedi figura 2.1). I SINode sono componenti indipendenti che arrichiscono le sorgenti di dati esistenti collegando i dati ad ontologie ed altri metadati. Il progetto SEWASIE permetterà inoltre una valutazione dei risultati in tempo utile, cercando di sviluppare un sistema e degli strumenti che non solo risolveranno il problema dell'integrazione intelligente dei dati, ma lo faranno in modo facilmente utilizzabile e commerciabile.

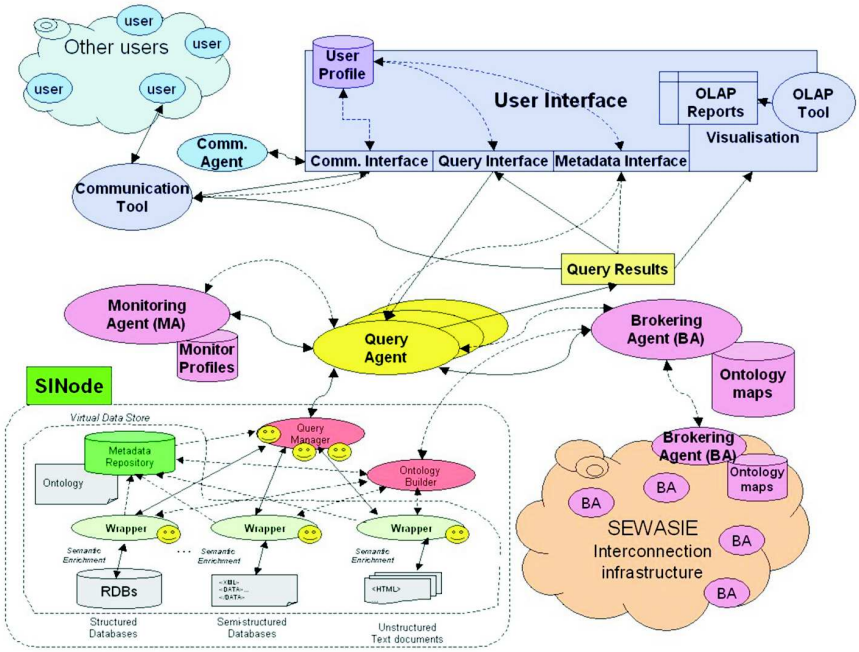

Figura 2.1. La SEWASIE Virtual Network

In particolare, il progetto SEWASIE perseguirà i seguenti obiettivi:

- Sviluppare un sistema sicuro basato su agenti software con un'architettura scalabile e distribuita per ricerche semantiche (basate su ontologie) e per una comunicazione strutturata e basata sul Web (per la negoziazione elettronica).

- Sviluppare una struttura generale responsabile dell'implementazione della fase di arricchimento semantico mettendo in primo piano l'arricchimento semantico degli

archivi dati virtuali, i quali costituiscono i nodi di informazione accessibili dagli utenti. L'ontologia creata sarà accessibile in diverse lingue, fondata su uno strato logico e codificata utilizzando gli standards W3C diffusi.

- Sviluppare una struttura generale per la gestione delle interrogazioni e l'armonizzazione delle informazioni tenendo conto degli archivi dati arricchiti semanticamente. In particolare, prima verranno trovate le similitudini tra le interrogazioni, quindi l'archivio dati virtuale responsabile per la risposta effettuerà un frazionamento delle interrogazioni, infine i sotto-risultati saranno integrati per fornire all'utente una risposta all'interrogazione originale.

- Sviluppare un componente per la mediazione delle informazioni che includerà metodi per il raggruppamento, la contestualizzazione e la visualizzazione dei dati arricchiti semanticamente. Per ottenere questo risultato, saranno sviluppati un filtraggio intelligente delle informazioni e servizi guidati dalla conoscenza basati sulle tecnologie del Web Semantico; i dati strutturati saranno collegati a dati semistrutturati o non strutturati attraverso l'utilizzo delle ontologie; per applicazioni di controllo finanziario i dati raggruppati saranno visualizzati in modo da mostrare i documenti relativi e i contesti dei risultati cercati.

- Sviluppare procedure di comunicazione strutturata che consentano l'utilizzo di ontologie. Lo strumento di comunicazione permetterà una negoziazione strutturata di supporto ai negoziatori impegnati nel commercio elettronico tra aziende impiegando agenti software per le procedure di comunicazione.

- Sviluppare interfacce utente sia per la progettazione semantica che per la gestione delle interrogazioni. Il primo è uno strumento che supporta la progettazione, la gestione e l'archiviazione delle informazioni semantiche associate agli archivi dati virtuali assieme ad una metodologia di modellazione concettuale associata al modello dei dati concepito. Il secondo è uno strumento per la gestione delle interrogazioni da parte dell'utente finale e per la navigazione intelligente effettuata

sfruttando le informazioni semantiche associate all'archivio dati virtuale e alla vista virtuale globale.

Un'altro importante obiettivo del progetto è quello di valutare l'impatto economico e organizzativo di una rete di informazioni arricchite semanticamente in un sistema industriale di piccole e medie imprese. In particolare i potenziali benefici economici attesi dal sistema SEWASIE ed i fattori interni ed esterni che occorrono per realizzare tali benefici saranno analizzati, puntando l'attenzione anche sullo studio dei cambiamenti di tipo commerciale ed organizzativo richiesti alle aziende dal nuovo strumento. Siccome aziende differenti mostrano diverse caratteristiche riguardo il tipo di informazioni richieste, le sorgenti di informazioni disponibili, le loro esperienze, le tecnologie di produzione e così via, il progetto SEWASIE ha deciso di condurre i test e le analisi commerciali sui requisiti degli utenti in due settori molto differenti tra loro, il settore della fusione industriale ed il settore tessile.

2.2 MOMIS

Negli scorsi anni le necessità di avere accesso all'informazione distribuita e il problema dell'integrazione dei dati provenienti da sorgenti eterogenee sono diventate sempre più importanti. Le aziende si sono dotate di sistemi per la memorizzazione dei dati costruiti su sistemi informativi contenenti informazioni spesso ridondanti, eterogenee e non sempre sostanziali. Dall'altra parte, l'avvento del Web, sia a livello di Internet che di Intranet, ha aumentato l'esigenza della condivisione e dell'estrazione di informazioni situate in sorgenti differenti, per ottenere una vista integrata così da eliminare ogni contraddizione e ridondanza. I problemi che devono essere considerati in questo campo sono dovuti tanto all'eterogeneità strutturale e delle applicazioni, quanto alla mancanza di un ontologia comune, che causa le differenze semantiche fra le sorgenti di informazioni. Inoltre queste differenze semantiche possono causare differenti tipi di conflitti, che possono variare da semplici contraddizioni nell'utilizzo dei nomi (quando nomi differenti sono usati da sorgenti differenti per indicare lo stesso concetto), a conflitti strutturali (quando differenti modelli/primitive sono utilizzati per rappresentare le stesse informazioni). Il problema

dell'integrazione è presente anche nell'ambiente del commercio elettronico. I cataloghi elettronici sono un componente chiave del commercio elettronico e possono essere organizzati come cataloghi di una singola azienda oppure possono fare parte di una struttura multi-catalogo. Nel secondo caso, dal punto di vista dell'utente, è molto importante avere un'interfaccia uniforme per la ricerca dei prodotti, che è data da una vista uniforme dei dati provenienti dai differenti cataloghi delle aziende e da un linguaggio per le interrogazioni unico. Dal punto di vista dell'azienda è importante garantire sia l'unicità dei loro cataloghi che la presenza in una struttura multi-catalogo. I cataloghi virtuali sintetizzano questo approccio, essi sono concepiti come strumenti per l'estrazione dinamica delle informazioni da cataloghi e dei prodotti presenti in modo uniforme, senza memorizzare direttamente i dati relativi ai diversi prodotti dai cataloghi. I clienti, invece di dover interagire con più cataloghi eterogenei, possono interagire in modo uniforme con un catalogo virtuale. In questo contesto il problema che deve essere considerato è l'identificazione delle informazioni semantiche relative ai concetti, cioè le informazioni che descrivono gli stessi concetti nelle differenti sorgenti aventi eterogeneità semantica. Infatti, le sorgenti di informazioni da integrare sono solitamente già esistenti e sono state sviluppate indipendentemente. Di conseguenza, le eterogeneità semantiche si possono presentare per gli aspetti relativi alla terminologia, alla struttura e al contesto delle informazioni e devono essere propriamente trattate durante l'integrazione per poter sfruttare correttamente ed efficacemente le informazioni disponibili. **MOMIS**[2] è l'acronimo di "**M**ediator envir**O**nment for **M**ultiple **I**nformation **S**ources". MOMIS [3, 23–25] è un sistema basato su mediatori per l'estrazione e l'integrazione delle informazioni che può trattare sorgenti di dati strutturate o semi-strutturate. Il progetto MOMIS è cominciato come collaborazione tra l'Università di Modena e Reggio Emilia, l'università di Milano e l'università di Brescia, all'interno del progetto di ricerca nazionale INTERDATA, sotto la direzione della professoressa S.Bergamaschi. Attualmente, parte l'attività di ricerca continua all'interno del progetto di ricerca nazionale D2I: Integrazione, Warehousing e Mining di Sorgenti Eterogenee di Dati.

[2] http://www.dbgroup.unimo.it/Momis/

2.2.1 L'approccio semantico del sistema MOMIS

Il sistema MOMIS segue un metodo semantico per l'integrazione delle informazioni basata sugli schemi concettuali (o metadati) delle sorgenti di dati. Nel sistema di MOMIS, ogni sorgente di dati fornisce uno schema ed in modo semiautomatico viene ottenuto uno schema virtuale globale di tutte le sorgenti. Lo schema globale è formato da un insieme di descrizioni dei collegamenti che specificano la correlazione semantica tra lo schema globale e gli schemi delle sorgenti. L'architettura del sistema è composta da elementi funzionali che comunicano utilizzando lo standard CORBA. Per descrivere le sorgenti di informazioni vengono utilizzati un modello di dati, ODM$_{I^3}$ ed un linguaggio, l'ODL$_{I^3}$. L'ODL$_{I^3}$ e l'ODM$_{I^3}$ sono stati definiti come sottoinsiemi dei corrispondenti modello e linguaggio definiti dall'ODMG, sono poi stati estesi per realizzare l'integrazione. Per interagire con una sorgente locale specifica, MOMIS utilizza i Wrapper, che devono essere disposti sopra ad ogni sorgente. Il Wrapper traduce le descrizioni dei metadati di una sorgente in una rappresentazione comune attraverso il linguaggio ODL$_{I^3}$. Il modulo costruttore dello Schema Globale (Global Schema Builder, GSB) (si veda la figura 2.2) elabora ed integra le descrizioni ricevute dai diversi wrappers per derivare lo schema comune globale interagendo con differenti moduli di servizio: *ODB-Tools*, un ambiente integrato per il ragionamento su database ad oggetti che utilizza le Logiche Descrittive [26]; *WordNet* [27] un database lessicale che supporta il mediatore nella costruzione delle relazioni lessicali; *ARTEMIS* uno strumento che effettua l'operazione di raggruppamento [28].

2.2.2 L'architettura del sistema MOMIS

Il sistema di MOMIS è progettato per l'integrazione degli schemi. La figura 2.2 mostra l'architettura del sistema in termini di moduli funzionali. Il sistema è composto da diversi oggetti che comunicano tra loro utilizzando lo standard CORBA[3] ed è formato seguendo l'architettura I^3 [29]. I moduli sono su tre livelli:

[3]Object Management Group. http://www.omg.org/

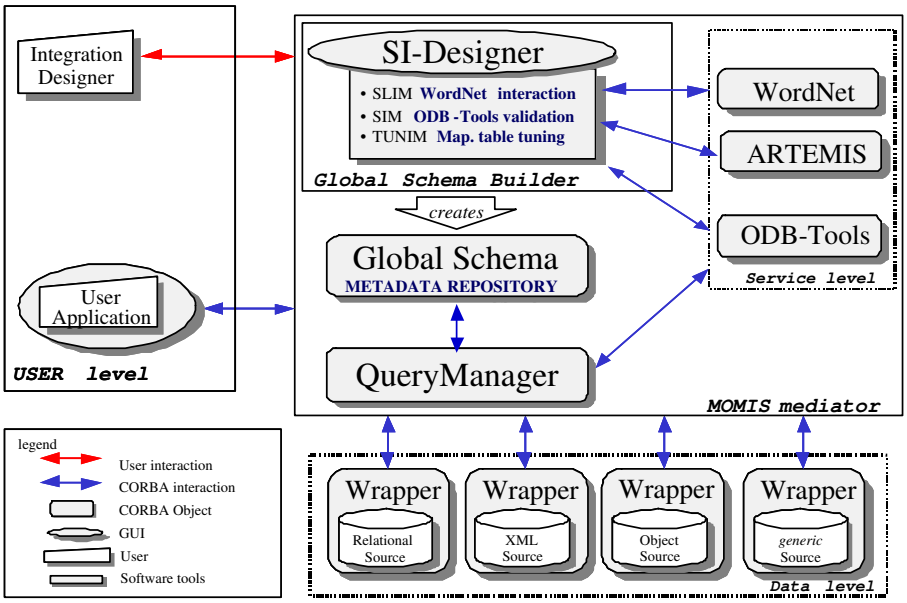

Figura 2.2. Prototipo di architettura del MOMIS

Livello Dati. Qui stanno i *Wrappers*. Essi sono disposti sopra ad ogni sorgente e rappresentano i moduli di interfaccia tra il mediatore e le sorgenti di dati locali. Hanno una doppia funzione: durante la fase di integrazione, traducono la descrizione delle informazioni mantenute nella sorgente. Questa descrizione è fornita attraverso il linguaggio ODL_{I^3}; durante la fase di elaborazione delle interrogazioni, traducono l'interrogazione ricevuta dal mediatore (espressa nel linguaggio comune per le interrogazioni OQL_{I^3}, derivato da OQL) in un'interrogazione espressa nel linguaggio relativo alla sorgente. I wrappers devono anche esportare i risultati, fornendoli al mediatore attraverso il modello comune per i dati utilizzato dal sistema.

Livello Mediatore. Il mediatore è il nucleo del sistema. Esso permette di generare una rappresentazione omogenea delle informazioni ed un accesso integrato alle sorgenti. E' composto da due moduli distinti:

- *Il Global Schema Builder* (GSB): il suo obiettivo è di supportare il progettista dall'acquisizione degli schemi delle sorgenti all'organizzazone della tabella dei riferimenti (mapping table) attraverso i vari passi dell'integrazione. Questo

modulo interagisce con ODB-Tools (una struttura per la validazione di database ad oggetti, che conserva la coerenza tassonomica ed effettua l'inferenza tassonomica e l'ottimizzazione semantica delle interrogazioni) e con il database lessicale WordNet per estrarre le relazioni lessicali intensionali tra attributi e classi. SI-Designer è la GUI (interfaccia grafica per l'utente) per il Global Schema Builder.

- *Il Query Manager* (QM): è il modulo che effettua l'elaborazione e l'ottimizzazione delle interrogazioni. Il QM genera le interrogazioni OQL_{I^3} da trasmettere ai wrappers a partire da ogni interrogazione proposta dall'utente sullo schema globale. Genera automaticamente la traduzione della richiesta in un insieme di sotto-interrogazioni per le sorgenti e sintetizza un risultato globale unificato per l'utente.

Livello Utente. Il progettista interagisce con il Global Schema Builder e realizza la vista integrata delle sorgenti. Gli utenti effettuano le interrogazioni sullo schema globale quindi il QM effettua le interrogazioni sulle sorgenti locali e presenta agli utenti una risposta unificata.

Il progetto MOMIS punta all'integrazione di dati da sorgenti strutturate e semistrutturate. SI-Designer è uno strumento di supporto per l'integrazione semiautomatica di schemi di sorgenti eterogenee (relazionali, ad oggetti, XML, e semistrutturate). Il metodo di MOMIS per effettuare la vista virtuale globale (Global Virtual View GVV) è articolato nelle seguenti fasi:

1. *Generazione di un dizionario comune (Common Thesaurus)*

   Il *Common Thesaurus* è un insieme di relazioni terminologiche di tipo estensionale ed intensionale, che descrivono la conoscenza intra e inter-schema riguardante gli attributi e le classi delle sorgenti. La conoscenza intra e inter-schema viene espressa tramite le relazioni terminologiche di tipo estensionale ed intensionale *sinonimia, ipernimia, iponimia e relazione* tra i nomi delle classi e degli attributi. In questa fase il database WordNet è usato per l'estrazione delle relazioni lessicali derivate.

2. *Analisi di affinità delle classi.*

   Le relazioni nel *Common Thesaurus* sono utilizzate per valutare il livello di affinità tra classi appartenenti alla stessa o a diverse sorgenti. Il concetto di affinità viene introdotto per formalizzare il tipo di relazione che può esserci tra classi dal punto di vista dell'integrazione. L'affinità tra due classi è stabilita attraverso coefficienti basati sui nomi delle classi, sulla loro struttura e sulle relazioni nel *Common Thesaurus*.

3. *Raggruppamento delle classi.*

   Le classi con affinità che stanno in sorgenti differenti sono raggruppate insieme utilizzando tecniche di raggruppamento gerarchiche. L'obiettivo è quello di identificare le classi che devono essere integrate perché descrivono lo stesso concetto o concetti correlati semanticamente.

4. *Generazione dello schema mediato.*

   La costituzione di gruppi di affinità conduce alla generazione dello schema previsto. Per ogni gruppo viene definita una classe che è rappresentativa per tutte le classi dell'insieme che è caratterizzata dall'unione dei loro attributi. Lo schema globale per le sorgenti analizzate è composto da tutte le classi derivate dai gruppi ed è la base per l'interrogazione delle sorgenti.

Uno strumento grafico, il *Source Integration Designer* (SI-Designer) è stato sviluppato per supportare il progettista durante la fase di integrazione.

3. IL LINGUAGGIO DESCRITTIVO ODL$_{I^3}$

Il linguaggio ODL$_{I^3}$ [30] è un'estensione del linguaggio standard ODL (Object Definition Language), definito dal gruppo di standardizzazione ODMG-93 per descrivere la conoscenza relativa ad uno schema ad oggetti in modo conforme all'ODMG Object Model. Per rispondere alle problematiche di integrazione di informazioni da fonti dati eterogenee il linguaggio ODL è stato esteso in accordo con le indicazioni del programma I3 (Integrazione Intelligente delle Informazioni) dell'ARPA, che si propone di creare una architettura di riferimento per integrare sorgenti eterogenee in maniera automatica. Il linguaggio ODL esteso viene quindi definito con il nome ODL$_{I^3}$. Utilizzando il linguaggio ODL$_{I^3}$, il Sistema MOMIS (Mediator envirOnment for Multiple Informations Sources) realizza l'integrazione di informazioni da fonti dati eterogenee, comprendenti anche sorgenti di dati semistrutturati. Nei prossimi paragrafi saranno presentati brevemente il linguaggio standard ODL e il linguaggio esteso per l'integrazione intelligente di informazioni ODL$_{I^3}$.

3.1 Il linguaggio ODL

In questo paragrafo saranno brevemente analizzati i costrutti del linguaggio ODL, tramite i quali possono essere rappresentati schemi di dati ad oggetti ma non schemi di dati semistrutturati.

3.1.1 Tipi classe e tipi valore

L'ODL distingue i suoi tipi di dati un due macrocategorie:

- *Tipi Classe*
- *Tipi Valore*

I tipi classe (o tipi complessi) sono identificati univocamente, infatti tutte le istanze possiedono un proprio OID (Object IDentifier) che deve essere unico. In pratica l'identità di un oggetto complesso è data dal proprio OID. La principale distinzione tra i tipi classe ed i tipi valore è che per i secondi l' identità è definita direttamente dal proprio valore. I tipi classe vengono impiegati per la descrizione di oggetti complessi, i tipi valore sono invece utilizzati per la dichiarazione di attributi semplici e di variabili.

Esiste un ulteriore tipo semplice oltre a quelli già elencati: il tipo *Any*. Il tipo *Any* è il supertipo da cui derivano tutti gli altri presenti nel linguaggio ODL$_{I^3}$, sia i tipi valore che quelli classe. Il contenuto di un oggetto any può essere quindi sia un attributo semplice che un oggetto complesso.

3.1.2 I tipi valore

I tipi valore (*valueType*) si differenziamo in due distinte categorie:

- *SimpleType*
- *ConstrType*

3.1.3 I tipi semplici

La categoria dei *SimpleType* (o tipi semplici) è costituita dai tipi atomici di base (*BaseType*) e dai *CollectionType*. Vi sono diversi tipi semplici di base predefiniti nel linguaggio ODL; eccone un elenco:

- *boolean*
- *char*
- *string*
- *date*
- *octet*

- *float*
- *double*
- *int*
- *short*
- *long*
- *unsigned int*
- *unsigned short*
- *unsigned long*

Oltre ai tipi semplici sopra elencati, anche il tipo *Any* può appartenere a questa categoria. I tipi Int, Short e Long che di default sono signed, possono essere anche di unsigned.

3.1.4 I tipi collezione

I tipi collezione (o *CollectionType*) vengono utilizzati per rappresentare collezioni di dati semplici. Le collezioni possono essere di quattro tipi differenti:

- *bag*
- *set*
- *list*
- *array*

Per creare una collezione di elementi non ordinati si utilizza il tipo *bag*:

```
bag <int> IntBag ;
```

Il tipo collezione *set* rappresenta un insieme non ordinato di elementi che non contiene duplicati, se si desidera una collezione di questo tipo la formulazione sarà la seguente:

```
set <string> StringSet ;
```

Se si desidera invece una collezione ordinata di elementi non duplicati si utilizzerà il tipo collezione *list*:

```
list <string> StringList ;
```

Il tipo collezione *array* viene utilizzato per creare elenchi ordinati con un numero fisso di elementi, eccone un esempio:

```
array <int,8> IntArray ;
```

3.1.5 I ConstrType

L'altra categoria di tipi valore è quella dei *ConstrType* (tipi costrutti) che è suddivisa in tre sottocategorie:

- *EnumType*
- *StructType*
- *UnionType*

I tipi enumerazione (*EnumType*) restringono il dominio di un *SimpleType* ad un elenco di valori possibili. Consideriamo il seguente esempio esplicativo:

```
enum odd {1 , 3 , 5 , 7 , 9};
```

una variabile di tipo dispari potrà assumere solamente uno dei valori (di tipo integer) contenuti all'interno dell'elenco fra le parentesi graffe. Il tipo struttura (*StructType*) viene utilizzato per definire strutture di tipi valore, segue un esempio di utilizzo:

```
typedef struct Nomestruct {
string a;
boolean b;
unsignedlong c;
} tipostruct ;
```

Un tipo struttura può contenere solo elementi di tipo valore. Una variabile strutturata come nell'esempio precedente può essere dichiarata nel seguente modo:

```
tipostruct var1 ;
```

Oppure, utilizzando il termine opzionale *Nomestruct* che deve sempre essere preceduto dalla parola chiave struct, è possibile dichiararla come segue:

```
struct Nomestruct var2;
```

L'ultima categoria dei tipi costrutti è il tipo unione che viene utilizzato per scegliere, in base al valore di una variabile in una clausola *switch*, il tipo di appartenenza di un attibuto.

```
union NumberType switch(val) {
case 1: string num;
case 2: int num;
};
```

Nel precedente esempio la variabile `NumberType` può essere una stringa di caratteri oppure un intero. Come per il tipo struttura, la variabili contenute in un tipo unione possono essere solo di tipo valore.

3.1.6 Il tipo classe

Le classi nel linguaggio ODL vengono chiamate interfacce (*interface*), la definizione di un interfaccia è costituita da due parti fondamentali:

- *Interface header*
- *Interface body*

All'interno dell'interface header vengono specificate le caratteristiche dell'interfaccia come il nome della classe, le superclassi da cui eredita (se presenti) ed una lista di proprietà chiamata *PropertyList*. Il nome di un'interfaccia è l'identificatore unico dell'oggetto (ObjectID) all'interno di uno schema ODL. Nella *PropertyList* possono essere specificate proprietà come l'*extent* che definisce l'insieme delle istanze di una classe all'interno di

un Database. Sempre all'interno di questa lista di proprietà è possibile definire una chiave (*key*) per l'interfaccia: una chiave può essere formata da un singolo attributo presente nel corpo dell'interfaccia (chiave semplice) oppure da un insieme di più attributi (chiave composita). Nella descrizione di un *Interface header* sono consentite una sola definizione di *extent* e una sola definizione di *key*. Ogni classe nel linguaggio ODL può avere una o più superclassi, è cioè ammessa l'ereditarietà multipla: da ogni superclasse vengono ereditati tutti gli attributi e tutti i metodi specificati nella definizione della sua *interface body*.

La seconda parte della definizione di un *interface* è il corpo dell'interfaccia, l'*interface body*. Il corpo dell'interfaccia descrive la struttura interna della stessa, qui è definito l'insieme di attributi e di metodi che ne fanno parte. Un *interface body* può essere composto dai seguenti elementi:

- *Attributi*
- *Relazioni*
- *Operazioni*
- *Costanti*

Gli attributi che fanno parte di una interfaccia ODL possono essere attributi di tipo semplice o complesso: gli attributi semplici sono attributi di tipo valore (*ValueType*), gli attributi complessi sono invece di tipo *interface*. La cardinalità degli attributi è di default uno, una cardinalità maggiore di uno deve essere specificata indicandola tra parentesi quadre. Le relazioni (*Relationship*) sono praticamente attributi complessi, infatti esse possono avere come oggetto solo interfacce e non tipi valore. All'interno di una relazione è possibile inoltre aggiungere informazioni sulla relazione inversa. Le operazioni definiscono i metodi della classe, sono quindi utilizzate per descrivere il comportamento della classe stessa. All'interno del corpo di un'interfaccia possono poi essere definite delle costanti indicando il nome della costante, il tipo semplice della stessa (*SimpleType*) ed il suo valore. Attributi, relazioni, operazioni e costanti devono avere nomi unici all'interno dell'*interface body*. Un esempio di definizione di interfaccia ODL è il seguente:

```
interface Professor:Person
```

```
( extent Professors
keys faculty_id, sec_no)
{
attribute string name;
attribute unsigned short faculty_id[6];
attribute long sec_no[10] ;
attribute Address address;
attribute Department department;
relationship set<Course> teach inverse Course::taught_by;
};
```

L'interfaccia `Professor` , che ha come superclasse l'interfaccia `Person` , rappresenta le istanze di tipo `Professors` all'interno del database e ha una chiave composita formata da due attributi. Il corpo dell'interfaccia è formato da tre attributi semplici, da due attributi complessi (i cui oggetti sono interfacce) e da una relazione che ha un'inversa.

3.2 L'estensione di ODL: il linguaggio ODL$_{I^3}$

Il linguaggio ODL, ben progettato per la rappresentazione della conoscenza relativa ad un singolo schema ad oggetti, risulta insufficiente per la descrizione e l'integrazione di un insieme di sorgenti di dati eterogenee. Nell'ambito del progetto MOMIS, si sono quindi rese necessarie una serie di modifiche ed estensioni per rappresentare la conoscenza relativa al processo di integrazione intelligente delle informazioni: il linguaggio così definito, seguendo le indicazioni dell'I3 workgroup, prende il nome di ODL$_{I^3}$. Di seguito saranno esaminate le estensioni effettuate per arrivare al linguaggio ODL$_{I^3}$.

3.2.1 Estensione ai tipi valore

Ai tipi semplici di base (*BaseType*) presenti nell'ODL è stato aggiunto un nuovo tipo che può essere utilizzato per rappresentare un valore intero compreso tra un estremo inferiore ed uno superiore: il tipo *range*. Eccone un esempio di utilizzo:

```
range 10,100 number;
```

l'attributo `number` può assumere tutti i valori interi compreso tra l'estremo inferiore 10 e l'estremo superiore 100. E' inoltre possibile avere come valore degli estremi il valore infinito, per rappresentare questo concetto si utilizzano i termini *+infinite* e *-infinite*. Nel seguente esempio la variabile numero può assumere tutti i valori compresi tra 100 ed infinito:

```
range 100,+infinite number;
```

3.2.2 Estensioni al tipo classe

Nella definizione del linguaggio ODL$_{I^3}$ sono state apportate diverse estensioni al tipo classe (*interface*) ODL:

- All'interno dell'*Interface header* è possibile indicare il tipo della sorgente dati a cui appartiene l'interfaccia: la sorgente può essere di tipo relazionale (*relational* o *nfrelational*), semistrutturato (*semistructured*), ad oggetti (*object*), o file (*file*). Oltre al tipo, è inoltre necessario indicare il nome della sorgente che deve essere unico nello schema ODL$_{I^3}$. Questa estensione è resa necessaria dal fatto che durante il trattamento di diverse sorgenti di dati esiste la possibilità che due classi abbiano lo stesso nome, la presenza del nome della sorgente rende unica una classe all'interno di uno schema ODL$_{I^3}$. Il nome ed il tipo della sorgente, che rappresentano una caratteristica della classe, vengono definite all'interno della *PropertyList* assieme all'extent e alla dichiarazione delle chiavi.

- Per descrivere schemi relazionali, nel linguaggio ODL$_{I^3}$ è stata aggiunta la possibilità di definire delle foreign keys. La definizione delle foreign keys è sempre contenuta nella *PropertyList* dell'interfaccia. Per dichiarare una chiave estera occorre indicare i nomi degli attributi che compongono la chiave e il nome della classe cui si riferisce la foreign key.

- Poichè il linguaggio ODL$_{I^3}$ deve essere in grado di descrivere anche schemi di dati semistrutturati, è stato aggiunto il costrutto union tramite il quale si possono definire

più *interface body* per una stessa interfaccia. In schemi di dati semistrutturati, infatti, un elemento può comparire nello stesso documento con rappresentazioni differenti, quindi in ODL$_{I^3}$ un'interfaccia potrà essere rappresentata da diverse strutture dati (differenti *interface body*). Sempre per rappresentare schemi di dati semistrutturati è stato aggiunto per gli attributi il costrutto optional. Nella definizione del corpo di un'interfaccia, postponendo un asterisco (*) alla dichiarazione di un attributo, è possibile indicare che l'attributo è opzionale, si definisce quindi una cardinalità minima uguale a zero e una cardinalità massima uguale a uno.

- Per l'operazione di integrazione delle sorgenti di dati in ODL$_{I^3}$ è possibile dichiarare attributi globali oltre ai normali attributi locali. Un attributo globale (*globalAttribute*) ha le stesse caratteristiche di un normale attributo ODL ma, in aggiunta, ha una funzione di collegamento ad un oggetto di tipo *MappingRule*.

3.2.3 Gli oggetti mappingRule

Il software del progetto MOMIS, durante la fase di integrazione delle sorgenti di dati, effettua un raggruppamento delle classi ritenute simili tra loro, generando nuove classi che contengono attributi globali. Si è resa quindi necessaria la definizione di regole di mediazione (*mappingRule*) per meglio specificare l'accoppiamento tra gli attributi globali e gli attributi locali originali. Se all'interno di un corpo di un interfaccia, un attributo presenta un riferimento ad una regola di mapping, esso è considerato automaticamente un attributo globale. Nella mediazione tra attributi globali e locali si possono verificare i seguenti casi:

- Corrispondenza fra un attributo globale ed un solo attributo locale (di una sola interfaccia).

- Corrispondenza fra un attributo globale ed un insieme di attributi locali in and tra loro: una serie di attributi appartenenti a classi differenti, ritenuti simili tra loro, vengono fusi in un attributo globale.

- Corrispondenza fra un attributo globale ed una serie di attributi locali in union tra loro: in questo caso l'attributo globale corrisponde solamente ad un attributo locale per volta.

- Corrispondenza di un attributo globale con un valore di default: può servire per esprimere un concetto nel caso in cui non ci sia corrispondenza tra un attributo globale e gli attributi locali.

3.2.4 Relazioni terminologiche

Il linguaggio ODL$_{I^3}$ offre inoltre la possibilità di definire relazioni terminologiche che possono intercorrere tra nomi di classi ed attributi. Le relazioni terminologiche vengono definite, durante il processo di integrazione, nella fase di generazione del *Thesaurus*. Le relazioni possono essere definite tra classi, attributi o miste tra classi ed attributi: le relazioni tra classi vengono memorizzate in un oggetto *InterfaceRel*, quelle tra attributi in un oggetto *AttributeRel* e quelle miste in un oggetto *AttrIntRel*. Possono essere definite relazioni terminologiche di quattro tipi:

- Ipernimia (BT)

- Iponimia (NT)

- Sinonimia (SYN)

- Associazione (RT)

Durante la fase di traduzione dal linguaggio ODL$_{I^3}$ alla logica descrittiva OLCD le equivalenze terminologiche vengono poi rese effettive. Una relazione di ipernimia o iponimia tra i nomi di due classi comporta la definizione di un vincolo di ereditarietà tra le due classi. Una relazione di sinonimia tra nomi di classi o nomi di attributi comporta un vincolo di uguaglianza tra le classi o gli attributi. Una relazione di associazione tra due classi comporta l'introduzione di un attributo con dominio la seconda classe nella prima delle due.

3.2.5 Regole d'integrità

ODL$_{I^3}$ rende possibile la dichiarazione di vincoli di integrità sotto forma di regole di tipo if-then (*rule*), in grado di definire vincoli non specificabili altrimenti. Le condizioni citate in queste regole devono essere rispettate dalle istanze delle classi ODL$_{I^3}$ specificate appartenenti ad uno schema. La sintassi di una if-then *rule* è:

```
rule nomerule forall iteratore in collezione : antecedente then
conseguente
```

oppure può essere la seguente:

```
rule nomerule [{case of indentifier : caselist }]
```

Il termine `nomerule` è il nome della regola d'integrità; l'`iteratore` rappresenta una istanza di un elemento tra quelli appartenenti a collezione; `collezione` è un insieme di istanze rappresentata da una classe o parte di essa; la parte `antecedente` definisce le condizioni della regola, formata da una serie di predicati booleani in AND fra loro; la parte `conseguente` descrive gli effetti della regola.

3.2.6 Relazioni estensionali

Una ulteriore importante funzionalità nel processo di integrazione delle sorgenti è la possibilità di esprimere, sotto forma di proposizioni, relazioni estensionali tra gli oggetti di classi distinte. Questi assiomi esprimono le relazioni insiemistiche tra le estensioni di classi definite in sorgenti autonome (indipendentemente dalla loro intensione). Ogni classe dello schema integrato può avere associati gli assiomi estensionali che predicano le relazioni di inclusione, equivalenza e disgiunzione. Gli assiomi estensionali sono espressi come *rule* nel linguaggio ODL$_{I^3}$ (e quindi tradotte in logica OLCD), senza ulteriori estensioni alla sintassi. La ragione che consente di ricorrere alla sintassi delle rule risiede nel fatto che assiomi estensionali e rule sono semanticamente equivalenti. Le relazioni di disgiunzione, inclusione ed equivalenza tra le estensioni, vengono espresse sottintendendo l'intersezione in quanto si presume che classi tra loro affini abbiano almeno estensioni la cui intersezione non è nulla.

La dichiarazione delle relazioni estensionali è la seguente:
Se A e B sono estensioni disgiunte: $A \cap B = \phi$

```
rule RE1 forall x in (A and B) then x in bottom
```

se B è inclusa in A: $B \subseteq A$

```
rule RE2 forall x in B then x in A
```

se A e B sono equivalenti: $A = B$

```
rule RE3 forall x in A then x in B
```

```
rule RE4 forall x in B then x in A
```

3.2.7 Annotazioni rispetto a WordNet

Una fase molto importante del processo di integrazione delle sorgenti del sistema MOMIS è la fase di annotazione, durante la quale un utente può selezionare manualmente uno o più significati per ogni elemento dello schema della sorgente locale utilizzando l'ontologia lessicale di WordNet. Successivamente si avrà la fase di generazione di un dizionario comune (*Common thesaurus generation*). Per poter rappresentare le annotazioni rispetto a WordNet il linguaggio ODL$_{I^3}$ aggiunge il costrutto *WNAnnotation*, eccone un esempio di utilizzo:

```
wnAnnotation ComputerScience.Professor
   lemmaValue="professor",
   lemmaSyntacticCategory=1,
   lemmaSenseNumber=1;
```

Questa annotazione assegna al nome dell'interfaccia `Professor` appartenente alla sorgente `ComputerScience` il valore `"professor"`, indicando una categoria sintattica uguale ad uno (corrispondente alla categoria sintattica "nome") ed un numero di senso uguale ad uno (corrispondente al significato: "someone who is a member of the faculty at a college or university"). Se per un termine sono previste più annotazioni diverse vengono ripetute diverse dichiarazioni di *WNAnnotation*.

4. IL LINGUAGGIO OWL PER ONTOLOGIE SU WEB

Il linguaggio OWL (*Web Ontology Language*) è stato sviluppato dal World Wide Web Consortium (*W3C*) per la definizione di ontologie e di strumenti compatibili con l'architettura del World Wide Web in generale e, più in particolare, del Semantic Web. OWL è stato creato per essere utilizzato da applicazioni che intendono elaborare il contenuto delle informazioni, invece di rappresentare solamente le informazioni agli umani. Tramite questo linguaggio è possibile avere una maggiore comprensibilità dei significati e della semantica di concetti da parte di agenti software rispetto a XML, RDF e RDF-Schema. Rende inoltre più semplici le fasi di elaborazione automatica ed integrazione delle informazioni. Questo linguaggio può essere visto come una revisione del linguaggio per ontologie su Web DAML+OIL ed incorpora quindi tutta l'esperienza fatta negli anni sui progetti e le applicazioni DAML+OIL. Il Semantic Web è una visione futura del web nella quale viene dato un significato esplicito alle informazioni, in modo da rendere più semplice per gli agenti software la loro elborazione e la loro integrazione. Il Semantic Web sarà costruito utilizzando l'abilità di definizione di schemi personalizzati del linguaggio XML e l'approccio flessibile nella rappresentazione dei dati di RDF. Perseguendo gli obiettivi sopracitati, è stato progettato il linguaggio per ontologie su Web OWL. OWL è stato costruito sulla base di RDF e RDF-Schema e rispetto ad essi aggiunge diversi costrutti per la definizione di concetti e delle loro interrelazioni: disgiunzione tra classi, restrizioni di cardinalità per le proprietà, relazioni di uguaglianza tra classi o proprietà, classi enumerate, nuovi tipi di proprietà e nuove caratteristiche per le proprietà.

4.1 I sottolinguaggi di OWL

OWL fornisce tre sottolinguaggi con espressività incrementale progettati per l'utilizzo da parte di utenti con necessità differenti per la descrizione di ontologie. Partendo dal

meno espressivo al più espressivo, i tre sottolinguaggi di OWL sono:

- *OWL Lite*
- *OWL DL*
- *OWL Full*

Nei prossimi paragrafi saranno analizzate le principali caratteristiche dei tre sottolinguaggi.

4.1.1 OWL Lite

OWL Lite permette di descrivere ontologie nelle quali siano definite principalmente classificazioni gerarchiche e semplici vincoli sulle proprietà. Per esempio, mentre permette di esprimere restrizioni sulla cardinalità delle proprietà di una classe, limita i possibili valori delle cardinalità a zero e uno. Grazie alla sua semplicità rispetto ai più espressivi OWL DL e OWL Full, un maggior numero di strumenti possono supportare OWL Lite. Considerando la minore complessità formale del linguaggio anche la definizione di ontologie risulta più semplice, infatti OWL Lite comprende solo i principali costrutti di OWL.

4.1.2 OWL DL

OWL DL è stato progettato per avere massima espressività e contemporaneamente garantire che tutte le computazioni abbiano un risultato e che tutte terminino in un tempo finito. OWL DL comprende tutti i costrutti del linguaggio OWL, ma alcuni di essi possono essere utilizzati solo sotto certe condizioni. Un esempio è dato dal fatto che in OWL DL l'insieme delle proprietà è suddiviso in *object properties* e *datatype properties*, quindi caratteristiche come la proprietà inversa, simmetrica o transitiva non possono essere definite per una *datatype properties*. OWL DL deve il suo nome alla sua corrispondenza con le logiche descrittive, DL sta infatti per Description Logics.

4.1.3 OWL Full

Il sottolinguaggio OWL Full permette di avere una massima espressività e la libertà sintattica di RDF, senza fornire nessuna garanzia dal punto di vista computazionale. In OWL Full una classe può essere trattata come un insieme di istanze o come un istanza nello stesso modo. OWL Full permette di ridefinire le caratteristiche dei costrutti di OWL o RDF. Vista la libertà sintattica, nessun ragionatore sarà mai capace di supportare tutte le caratteristiche di OWL Full.

4.2 Definizione di ontologie in OWL

Nel linguaggio OWL le informazioni sono raccolte in ontologie, le quali possono essere memorizzate sotto forma di documenti sul Web. Un documento OWL consiste di un *ontology header* nel quale sono contenute informazioni generali sull'ontologia ed un certo numero di definizioni di classi, definizioni di proprietà e dati relativi ad istanze. Un esempio di header per la definizione di un ontologia può essere il seguente:

```
<rdf:RDF
  xml:base ="http://www.example.org/University.owl"
>
...
<owl:Ontology rdf:about="">
<owl:versionInfo>v 1.17 2003/02/26 12:56:51
</owl:versionInfo>
<rdfs:comment>An example ontology</rdfs:comment>
<owl:imports rdf:resource="http://www.example.org/foo"/>
</owl:Ontology>
...
</rdf:RDF>
```

La linea `<owl:Ontology rdf:about="">` dichiara che questo blocco descrive l'ontologia corrente: essa è identificata da un URI che deve essere definito utilizzando un attributo *xml:base* nell'elemento *<rdf:RDF>* all'inizio del documento.

Il costrutto `<owl:imports>` serve per importare un'ontologia, questo ed altri costrutti appartenenti all'*ontology header* verranno meglio analizzati alla fine di questo capitolo.

4.3 Le classi

Le classi forniscono un meccanismo di astrazione per la rappresentazione di concetti, ogni classe ha quindi un contenuto semantico intensionale e viene utilizzata per raggruppare risorse che hanno caratteristiche simili tra loro. Ogni classe è associata ad un insieme di istanze (*individuals*) che ne rappresentano l'estensione, due classi possono avere la stessa estensione ma rappresentare concetti differenti. In OWL Lite e OWL DL un istanza non può essere allo stesso tempo una classe, infatti classi ed istanze hanno domini disgiunti; in OWL Full un classe può comportarsi come un istanza di un'altra classe (metaclasse). Le classi vengono definite, in OWL, attraverso una descrizione (*class descriptions*) e una serie di assiomi (*class axioms*).

4.3.1 Descrizione delle classi

La descrizione di una classe OWL deve contenere il nome per la classe ed eventualmente definire la sua estensione. Si possono inoltre avere classi anonime definite solamente dalla loro estensione (dall'insieme delle istanze che la compongono). Esistono sei tipi distinti di descrizione per una classe:

- un identificatore di classe (un riferimento URI)
- un'enumerazione di istanze che insieme formano l'estensione della classe
- una restrizione su di una proprietà
- l'intersezione di due o più descrizioni di classe
- l'unione di due o più descrizioni di classe
- il complemento di una descrizione di classe

Il primo tipo descrive la classe attraverso un nome di classe che viene sintatticamente rappresentato come un riferimento URI (*Uniform Resource Identifier*) Gli altri cinque tipi descrivono classi anonime applicando restrizioni sulle loro estensioni: il secondo è utilizzato per descrivere una classe che contiene esattamente l'insieme enumerato indicato, il terzo tipo definisce una classe le cui istanze rispettano una particolare restrizione su di una proprietà, gli ultimi tre tipi definiscono l'estensione di una classe come combinazione booleana di altre descrizioni di classe.

4.3.2 Identificatore di classe

Il costrutto necessario alla dichiarazione di un identificatore di classe è il seguente:

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

Questa proposizione definisce una classe di nome `Person` identificata dall'URI `base:#Person` , dove `base` è il namespace precedentemente dichiarato per l'ontologia corrente, per esempio `http://www.example.org/foo` .
L'URI identificativo della classe sarà quindi così composto:

```
http://www.example.org/foo#Person
```

In OWL esistono due classi predefinite per la rappresentazione dei concetti *Top* e *Bottom*, questi concetti sono rappresentati rispettivamente tramite l'identificatore *owl:Thing* e *owl:Nothing*. L'estensione della classe *owl:Thing* è formata dall'insieme di tutte le istanze, per cui essa è superclasse di tutte le classi; l'estensione della classe *owl:Nothing* è invece formata dall'insieme vuoto, quindi essa è sottoclasse di ogni altra classe.

4.3.3 Enumerazione

Una classe di tipo enumerato è descritta da un insieme di istanze che ne rappresentano l'estensione, un'esempio di descrizione è il seguente:

```
<owl:Class>
  <owl:oneOf rdf:parseType="Collection">
```

```
    <owl:Thing rdf:about="#Eurasia"/>
    <owl:Thing rdf:about="#Africa"/>
    <owl:Thing rdf:about="#NorthAmerica"/>
    <owl:Thing rdf:about="#SouthAmerica"/>
    <owl:Thing rdf:about="#Australia"/>
    <owl:Thing rdf:about="#Antarctica"/>
  </owl:oneOf>
</owl:Class>
```

Utilizzando il costrutto *owl:oneOf* si indica una collezione (*rdf:parseType="Collection"*) di istanze che descrivono esattamente l'estensione della classe, nell'esempio si descrive una classe che rappresenta tutti i continenti. La sintassi (*owl:Thing rdf:about="..."*) serve per riferirsi ad un individuo (istanza), tutte le istanze fanno infatti parte della classe *owl:Thing*. Una delle limitazioni del sottolinguaggio OWL Lite è che esso non prevede la definizione di classi enumerate.

4.3.4 Restrizioni sulle proprietà

Una restrizione su di una proprietà descrive la classe formata da tutte le istanze che soddisfano la restrizione, OWL distingue due tipi di restrizioni sulle proprietà: restrizioni sui valori e restrizioni sulla cardinalità. Una restrizione sui valori di una proprietà limita il range dei possibili valori che la proprietà può assumere quando è utilizzata dalla classe in cui è definita, una restrizione sulla cardinalità di una proprietà riduce il numero di valori che la proprietà può assumere quando è utilizzata dalla classe in cui è definita. Per effettuare restrizioni sui valori di una proprietà per una data classe si possono utilizzare i costrutti *owl:allValuesFrom*, *owl:someValuesFrom* e *owl:hasValue*; per definire una restrizione sulla cardinalità di una proprietà per una data classe si possono utilizzare i costrutti *owl:maxCardinality*, *owl:minCardinality* e *owl:cardinality*

Restrizioni sui valori delle proprietà

Il costrutto *owl:allValuesFrom* indica che tutti i valori della proprietà specificata devono essere istanze della classe indicata come oggetto, oppure valori indicati nel range. Ecco un esempio di utlizzo:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#produce" />
  <owl:allValuesFrom rdf:resource="#RedWine" />
</owl:Restriction>
```

Questa restrizione può essere utilizzata per descrivere la classe formata dai produttori di soli vini rossi.
Il costrutto *owl:someValuesFrom* è utilizzato in modo analogo al precedente ed indica che almeno un valore della proprietà specificata deve essere istanza della classe indicata come oggetto, oppure un valore appartenente al range indicato. Il seguente esempio descrive la classe formata dai produttori di vino che producono almeno una qualità di vino rosso:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#produce" />
  <owl:someValuesFrom rdf:resource="#RedWine" />
</owl:Restriction>
```

Una restrizione del tipo *owl:hasValue* viene utilizzata per indicare che la proprietà deve avere come valore l'istanza o il dato indicati. Un'esempio di utilizzo può essere il seguente:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#produce" />
  <owl:hasValue rdf:resource="#Chianti" />
</owl:Restriction>
```

Nell'esempio si descrive la classe formata dai produttori di solo Chianti. Il costrutto *owl:hasValue* non è incluso nel sottolinguaggio OWL Lite.

Restrizioni sulla cardinalità delle proprietà

In OWL le proprietà hanno di default una cardinalità che può andare da zero ad infinito. Per rendere una proprietà necessaria, permettere un determinato numero di valori per quella proprietà o per indicare che una proprietà è opzionale, vengono utlizzate le restrizioni sulla cardinalità. Il costrutto *owl:maxCardinality* viene utilizzato per specificare il massimo numero di valori che una proprietà può avere. Il seguente esempio definisce la classe degli individui che hanno al massimo due genitori:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:maxCardinality rdf:datatype="&xsd;nonNegative-
  Integer">2</owl:maxCardinality>
</owl:Restriction>
```

In modo analogo viene utilizzato il costrutto *owl:minCardinality* il quale indica il numero minimo di valori che una proprietà può avere. Eccone un esempio:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:minCardinality rdf:datatype="&xsd;nonNegative-
  Integer">2</owl:minCardinality>
</owl:Restriction>
```

L'esempio definisce la classe degli individui che hanno almeno due genitori. Per definire una cardinalità minima e massima uguali, è possibile fare uso del costrutto *owl:cardinality*, per descrivere la classe degli individui che hanno esattamente due genitori, si utilizzerà il codice seguente:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:cardinality rdf:datatype="&xsd;nonNegative-
  Integer">2</owl:cardinality>
</owl:Restriction>
```

4.3.5 Intersezione, unione e complemento

In OWL è possibile definire l'estensione di una classe come combinazione booleana di altre descrizioni di classe utlizzando i costrutti *owl:intersectionOf*, *owl:unionOf* e *owl:complementOf* che rappresentano rispettivamente gli operatori booleani AND, OR e NOT utilizzati sulle descrizioni di classi. Il costrutto *owl:intersectionOf* descrive una classe la cui estensione contiene solamente le istanze che appartengono a tutte le estensioni delle classi indicate, ha quindi il significato logico di congiunzione:

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Tosca" />
        <owl:Thing rdf:about="#Salome" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Turandot" />
        <owl:Thing rdf:about="#Tosca" />
      </owl:oneOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```

Nel precedente esempio la classe intersezione tra le due classi enumerate indicate, contiene l'unica istanza (Tosca) appartenente alle estensioni di entrambe. Il costrutto *owl:unionOf* descrive una classe la cui estensione contiene le istanze che appartengono almeno ad una tra le estensioni delle classi indicate, ha quindi il significato logico di disgiunzione, un esempio:

```
<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
```

```
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Tosca" />
        <owl:Thing rdf:about="#Salome" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Turandot" />
        <owl:Thing rdf:about="#Tosca" />
      </owl:oneOf>
    </owl:Class>
  </owl:unionOf>
</owl:Class>
```

Questa descrizione di classe definisce una classe la cui estensione contiene tre istanze (Tosca, Salome, e Turandot). Un costrutto del tipo *owl:complementOf* descrive una classe la cui estensione è formata da tutte le istanze che non appartengono all'estensione della classe indicata come oggetto. L'esempio seguente può essere utilizzato per rappresentare l'insieme di tutte le istanze che non appartengono alla classe Opera:

```
<owl:Class>
  <owl:complementOf>
    <owl:Class rdf:about="#Opera"/>
  </owl:complementOf>
</owl:Class>
```

Il sottolinguaggio OWL Lite esclude l'utilizzo dei costrutti *owl:unionOf* e *owl: complementOf.*

4.3.6 Assiomi per le classi

Gli assiomi per le classi vengono inseriti all'interno delle descrizioni di classi per aggiungere caratteristiche sufficienti e/o necessarie per una OWL fornisce tre diversi tipi

di assiomi per meglio descrivere le classi:

- *rdfs:subClassOf*
- *owl:equivalentClass*
- *owl:disjointWith*

L'assioma *rdfs:subClassOf* viene utilizzato per specificare una o più superclassi, indica che l'estensione della classe è un sottoinsieme dell'estensione della classe descritta. Un esempio:

```
<owl:Class rdf:ID="RedWine">
  <rdfs:subClassOf rdf:resource="#Wine" />
</owl:Class>
```

Questo tipo di assioma viene utilizzato per specificare le restrizioni precedentemente illustrate all'interno delle descrizioni di classi:

```
<owl:Class rdf:about="#SpeakTwoLanguages">
   <rdfs:subClassOf>
<owl:Restriction>
  <owl:onProperty rdf:resource="#speaklanguage" />
  <owl:cardinality rdf:datatype="&xsd;nonNegative-
  Integer">2</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

L'assioma di tipo *owl:equivalentClass* permette di dichiarare che due classi hanno esattamente la stessa estensione, questo non implica però l'uguaglianza intensionale delle classi, ossia due classi che hanno la stessa estensione possono rappresentare concetti differenti. Un esempio di dichiarazione di equivalenza tra le estensioni di due classi è il seguente:

```
<owl:Class rdf:ID="DaPonteOperaOfMozart">
  <owl:equivalentClass>
```

```
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <Opera rdf:about="#Nozze_di_Figaro"/>
        <Opera rdf:about="#Don_Giovanni"/>
        <Opera rdf:about="#Cosi_fan_tutte"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Un assioma di tipo *owl:disjointWith* viene utilizzato per indicare che l'estensione di una classe non ha membri in comune con l'estensione di un'altra classe, questo è un esempio di disgiunzione tra l'estensione di due classi:

```
<owl:Class rdf:about="#Man">
  <owl:disjointWith rdf:resource="#Woman"/>
</owl:Class>
```

OWL Lite non consente la definizione di disgiunzione tra estensioni tramite l'assioma *owl:disjointWith*.

4.4 Le proprietà

In OWL le proprietà permettono di definire relazioni binarie generali tra classi o specifiche tra istanze delle classi. OWL distingue le proprietà in due categorie principali:

- *datatype properties*: relazioni tra istanze di una classe e tipi di dato XML Schema;
- *object properties*: relazioni tra istanze di due classi.

Le *object properties* correlano le istanze di due classi, mentre le *datatype properties* coorelano le istanze di una classe a valori. Esistono altre due particolari categorie di proprietà che verrano analizzate nella fine del capitolo, le *annotation properties* e le *ontology properties*. In OWL Full, le *object properties* e le *datatype properties* non sono disgiunte e

siccome, in questo sottolinguaggio, i valori possono essere trattati come istanze le *datatype properties* sono effettivamente una sottoclasse delle *object properties.* Le proprietà vengono descritte attraverso degli assiomi, nella sua forma più semplice un assioma può descrivere l'esistenza di una proprietà, per esempio:

```
<owl:ObjectProperty rdf:ID="hasParent"/>
```

Questo assioma definisce una proprietà con l'unica restrizione che i suoi valori devono essere istanze, è infatti una *object properties.* Per meglio specificare le caratteristiche di una proprietà possono essere utilizzati diversi tipi di assiomi:

- Costrutti delle proprietà: *rdfs:subPropertyOf*, *rdfs:domain* e *rdfs:range*
- Relazioni tra proprietà: *owl:equivalentProperty* e *owl:inverseOf*
- Restrizioni di cardinalità: *owl:FunctionalProperty* e *owl:InverseFunctionalProperty*
- Caratteristiche logiche: *owl:SymmetricProperty* e *owl:TransitiveProperty*

Nei prossimi paragrafi i vari tipi di assiomi per la descrizione di proprietà verranno esaminati più dettagliatamente.

4.4.1 Costrutti delle proprietà

In questo paragrafo vengono analizzati i principali costrutti per la dichiarazione di proprietà, essi appartengono al linguaggio RDF Schema, qui ne verrà descritto l'utilizzo in OWL.

rdfs:subPropertyOf

In OWL è possibile definire relazioni gerarchiche anche tra proprietà, il costrutto *rdfs:subPropertyOf* è infatti utilizzato per dichiarare che una proprietà e sottoproprietà di un'altra. Formalmente questo significa che se P1 è una sottoproprietà di P2, l'estensione della proprietà P1 (l'insieme di coppie di istanze oggetto e soggetto della proprietà) deve essere un sottoinsieme dell'estensione della proprietà P2 (insieme di coppie). Un esempio:

```
<owl:ObjectProperty rdf:ID="hasMother">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
</owl:ObjectProperty>
```

Con questo esempio si dice che tutte le coppie di istanze facenti parte dell'estensione della proprietà `hasMother` fanno parte anche dell'estensione della proprietà `hasParent` . Questo tipo di assioma può essere utilizzato sia per le *object properties* che per le *datatype properties*. In OWL DL e OWL Lite il soggetto e l'oggetto di questo assioma devono essere entrambe *object properties* o *datatype properties*, le due categorie di proprietà sono infatti disgiunte.

rdfs:domain

L'assioma *rdfs:domain* vine utilizzato per specificare il soggetto di una proprietà che deve essere una descrizione di classe. Se per una proprietà vengono utilizzati più assiomi del tipo *rdfs:domain* il soggetto della proprietà viene interpretato come l'intersezione delle descrizioni di classe specificate. Se si vuole definire come soggetto di una proprietà un insieme di più classi, deve essere utilizzata una descrizione di classe nella forma *owl:unionOf*:

```
<owl:ObjectProperty rdf:ID="hasBankAccount">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person"/>
        <owl:Class rdf:about="#Corporation"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
```

In OWL Lite il soggetto di una proprietà può essere soltanto definito come un identificatore di classe.

rdfs:range

L'assioma *rdfs:range* viene utilizzato per specificare l'oggetto di una proprietà che può essere una descrizione di classe o un valore appartenente ad un certo range. Analogamente alla specificazione del soggetto di una proprietà, se vengono utilizzati più assiomi *rdfs:range* l'oggetto della proprietà viene interpretato come l'intersezione delle descrizioni di classe specificate. Per definire come oggetto di una proprietà un insieme di più classi o range di valori, deve essere utilizzata una descrizione di classe nella forma *owl:unionOf*:

```
<owl:ObjectProperty rdf:ID="produceWine">
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#RebWine"/>
        <owl:Class rdf:about="#WhiteWine"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>
```

A differenza della restrizione sui valori *owl:allValuesFrom* utilizzata all'interno delle descrizioni di classi che specifica una restrizione per la proprietà quando viene utiizzata da quella classe, l'assioma *rdfs:range* pone una restrizione globale sui valori che la proprietà può assumere. In OWL Lite l'oggetto di una proprietà può essere soltanto definito come un identificatore di classe.

4.4.2 Relazioni tra proprietà

Esistono due assiomi che possono essere utilizzati per definire una relazione di equivalenza tra due proprietà (*owl:equivalentProperty*) o una relazione inversa tra due proprietà (*owl:inverseOf*)

owl:equivalentProperty

Il costrutto *owl:equivalentProperty* dichiara che due proprietà mettono in relazione tra loro le stesse coppie di istanze. Tramite questo costrutto non viene dichiarata l'uguaglianza tra le proprietà, infatti proprietà che hanno la stessa estensione, possono avere un significato intensionale differente, quindi rappresentare concetti diversi.

owl:inverseOf

Il costrutto *owl:inverseOf* può essere utilizzato per dichiarare che una proprietà è l'inversa di un'altra, ossia il soggetto (*rdfs:domain*) e l'oggetto (*rdfs:range*) di una proprietà sono scambiati nella proprietà inversa. La dichiarazione di proprietà inverse può essere utilizzata solo con le *object property*. Eccone un esempio di utilizzo:

```
<owl:ObjectProperty rdf:ID="hasSon">
  <owl:inverseOf rdf:resource="#hasParent"/>
</owl:ObjectProperty>
```

L'esempio dichiara che per ogni coppia `(Parent,Son)` nell'estensione della proprietà `hasSon` , esiste la corrispondente coppia `(Son,Parent)` nell'estensione della proprietà `hasParent` e viceversa.

4.4.3 Restrizioni globali di cardinalità

In OWL esistono due costrutti per la dichiarazione di restrizioni di cardinalità globali, sono il costrutto *owl:FunctionalProperty* e il costrutto *owl:InverseFunctionalProperty*. Le restrizioni di cardinalità globale definiscono la cardinalità che una proprietà può avere indipendentemente dalle classi a cui sono applicate, mentre le restrizioni di cardinalità specificate all'interno delle descrizioni di classi analizzate precedentemente definiscono la cardinalità di una proprietà solo per l'utilizzo con quella classe.

owl:FunctionalProperty

Una *functional property* è una proprietà che può avere un solo valore `y` per ogni istanza `x` , ossia non possono esistere due valori distinti `y1` e `y2` tali che le coppie `(x,y1)` e `(x,y2)` siano entrambe istanze della proprietà. Sia le *object properties* che le *datatype properties* possono essere dichiarate come proprietà funzionali. Il seguente esempio definisce la proprietà `husband` come funzionale, dichiara cioè che una donna può avere al più un marito:

```
<owl:ObjectProperty rdf:ID="husband">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Woman"/>
  <rdfs:range rdf:resource="#Man"/>
</owl:ObjectProperty>
```

owl:InverseFunctionalProperty

In una proprietà dichiarata come inversamente funzionale, l'oggetto della proprietà determina in modo univoco il soggetto. Quindi se una proprietà è una *owl:InverseFunctionalProperty* dato un valore `y` della proprietà, non possono esistere due istanze distinte `x1` e `x2` , tali che le coppie `(x1,y)` e `(x2,y)` appartengano entrambe all'estensione della proprietà. Un esempio di proprietà inversamente funzionale può essere il seguente:

```
<owl:InverseFunctionalProperty rdf:ID="biologicalMotherOf">
  <rdfs:domain rdf:resource="#Woman"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:InverseFunctionalProperty>
```

Questo esempio dichiara che per ogni `Human` si può identificare una sola `biologicalMotherOf` .

4.4.4 Caratteristiche logiche

In OWL esistono due costrutti per la definizione di caratteristiche logiche delle proprietà, il costrutto *owl:TransitiveProperty* e il costrutto *owl:SymmetricProperty*.

owl:TransitiveProperty

Quando una proprietà è definita come transitiva, questo sta ad indicare che se una coppia `(x,y)` è un'istanza della proprietà e la coppia `(y,z)` è anch'essa un'istanza della proprietà, allora anche la coppia `(x,z)` apparterrà all'estensione della proprietà: Un esempio di proprietà transitiva può essere il seguente:

```
<owl:TransitiveProperty rdf:ID="subClassOf">
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range  rdf:resource="#Class"/>
</owl:TransitiveProperty>
```

Nell'esempio si vuole evidenziare che se una classe C è sottoclasse di B e la classe B è sottoclasse di A, allora C è sottoclasse anche di A. Soltanto le *object properties* possono essere dichiarate come proprietà transitive. Il sottolinguaggio OWL DL richiede inoltre che non ci siano restrizioni locali o globali sulla cardinalità di una proprietà dichiarata come transitiva, sulle sue superclassi, sull'inversa o sulle superclassi dell'inversa.

owl:SymmetricProperty

Una proprietà definita come simmetrica è una proprietà nella quale, per ogni coppia `(x,y)` istanza della proprietà, anche la coppia `(y,x)` è istanza della proprietà. Ecco un esempio di proprietà simmetrica:

```
<owl:SymmetricProperty rdf:ID="friendOf">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range  rdf:resource="#Person"/>
</owl:SymmetricProperty>
```

Naturalmente il soggetto e l'oggetto di una proprietà simmetrica sono istanze della stessa classe.

4.5 Le istanze (Gli individui)

Nel linguaggio OWL le istanze di una classe vengono dichiarate attraverso due categorie di assiomi, la prima categoria ne definisce l'appartenenza all'estensione di una classe ed i valori delle proprietà, la seconda categoria è invece utilizzata per definire le relazioni di uguaglianza o disuguaglianza che possono intercorrere tra esse.

4.5.1 Appartenenza alle classi e valori delle proprietà

L'appartenenza di un istanza ad una classe ed i valori delle proprietà vengono specificati in OWL come segue:

```
<Person>
<name rdf:datatype="&xsd;string">Paolo</name>
<surname rdf:datatype="&xsd;string">Rossi</surname>
<bornIn rdf:resource="#Roma"/>
<born rdf:datatype="&xsd;date">1956-04-14</born>
<cf rdf:datatype="&xsd;string">RSSPLL56D14F231P</cf>
</Person>
```

Questo esempio dichiara un'istanza per la classe Persona, specificando i valori corrispondenti ad ogni proprietà.

4.5.2 Identità delle istanze

Essendo OWL un linguaggio per ontologie su Web, non può fare l'assunzione che nomi diversi si riferiscano sempre ad oggetti diversi nel Web. Per esempio la stessa persona potrebbe essere identificata in modi diversi (da diversi URI). Quindi in OWL, a meno che non venga specificato attraverso specifici costruttori che due URI identificano la stessa o

diverse istanze, si può assumere che entrambe le situazioni siano possibili. Il linguaggio fornisce tre costruttori per dichiarare fatti relativi all'identità delle istanze:

- *owl:sameAs* viene utilizzato per dichiarare che due URI identificano la stessa istanza;
- *owl:differentFrom* è usato per dichiarare che due URI identificano due diverse istanze;
- *owl:AllDifferent* fornisce un metodo per dichiarare che una lista di istanze sono tutte differenti tra loro.

Per dichiarare che due URI identificano la stessa istanza, un esempio può essere il seguente:

```
<rdf:Description rdf:about="#Paolo_Rossi">
  <owl:sameAs rdf:resource="#Rossi_Paolo"/>
</rdf:Description>
```

L'esempio seguente mostra come dichiarare che due URI identificano diverse istanze:

```
<Persona rdf:ID="Paolo_Rossi">
  <owl:differentFrom rdf:resource="#Mario_Bianchi"/>
</Opera>
```

Il costrutto *owl:AllDifferent* viene aggiunto per una maggiore semplicità nella dichiarazione di un insieme di istanze tutte differenti tra loro:

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <Persona rdf:about="#Paolo_Rossi"/>
    <Persona rdf:about="#Mario_Bianchi"/>
    <Persona rdf:about="#Fabio_Verdi"/>
    <Persona rdf:about="#Simona_Neri"/>
  </owl:distinctMembers>
</owl:AllDifferent>
```

La dichiarazione dell'esempio può essere infatti effettuata anche utilizzando più costrutti del tipo *owl:differentFrom*.

4.6 I tipi di dato

OWL permette, attraverso i cosìdetti *Data Range*, di definire un range di valori da utilizzare per esempio come oggetto (*owl:range*) per una *datatype property*. I tipi di dato utilizzabili con OWL si possono suddividere in tre categorie:

- tipi di dato RDF;
- tipi di dato enumerati;
- altri tipi di dato.

4.6.1 Tipi di dato RDF

OWL utilizza lo schema per i tipi di dato di RDF, il quale fornisce un meccanismo per il riferimento ai tipi di dato XML Schema (*XML Schema datatypes*). Per riferirsi ad un tipo di dato XML Schema viene utilizzato un URI nella forma

`http://www.w3.org/2001/XMLSchema#NAME`

dove `NAME` è il nome del tipo di dato XML Schema. Per esempio, il riferimento URI del tipo XML Schema stringa sarà il seguente:

`http://www.w3.org/2001/XMLSchema#string`

Il namespace di XML Schema è uno dei namespace definiti per convenzione e viene abbreviato con `xsd:` , il riferimento al tipo di dato stringa abbreviato sarà quindi `xsd:string` . I tipi di dato XML Schema utilizzabili in OWL sono quindi i seguenti:

- il tipo di dato primitivo *xsd:string*, più i tipi di dato derivati da esso *xsd:normalizedString*, *xsd:token*, *xsd:language*, *xsd:NMTOKEN*, *xsd:Name* e *xsd:NCName*;
- il tipo di dato primitivo *xsd:boolean*;

- i tipi di dato numerici primitivi *xsd:decimal*, *xsd:float*, *xsd:double* e *xsd:decimal*, più i tipi derivati da *xsd:decimal*, *xsd:integer*, *xsd:positiveInteger*. *xsd:nonPositiveInteger*, *xsd:negativeInteger*, *xsd:nonNegativeInteger*, *xsd:long*, *xsd:int*, *xsd:short*, *xsd:byte*, *xsd:unsignedLong*, *xsd:unsignedInt*, *xsd:unsignedShort*, *xsd:unsignedByte*;

- i tipi di dato primitivi relativi al tempo *xsd:dateTime*, *xsd:time*, *xsd:date*, *xsd:gYearMonth*, *xsd:gYear*, *xsd:gMonthDay*, *xsd:gDay* e *xsd:gMonth*;

- i tipi di dato primitivi *xsd:hexBinary*, *xsd:base64Binary*, and *xsd:anyURI*.

Il tipo di dato deve essere specificato ogni volta che una proprietà viene utilizzata, anche se il range della proprietà è già stato specificato nella sua definizione. Ecco un esempio:

```
<owl:DatatypeProperty rdf:about="#Height">
  <rdfs:domain rdf:resource="#Misure"/>
  <rdf:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>

<Misure>
  <Height rdf:datatype="&xsd;double">374</Height>
</Misure>
```

4.6.2 Tipi di dato enumerati

Oltre ai tipi di dato RDF, OWL fornisce un costrutto per la definizione di tipi di dato enumerati, costituiti cioè da un insieme di valori specificati. Sfortunatamente, per la definizione di tipi di dato enumerato non è possibile utilizzare il costrutto *rdf:parseType=-"Collection "* (utilizzato per la dichiarazione di una collezzione di elementi), occorre invece definire una lista RDF facendo uso dei costrutti *rdf:List*, *rdf:first*, *rdf:rest* e *rdf:nil*. Come si può notare dal seguente esempio, questa modalità di descrizione richiede la scrittura di molto codice:

```
<owl:DatatypeProperty rdf:ID="tennisGameScore">
```

```
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf>
        <rdf:List>
           <rdf:first rdf:datatype="&xsd;integer">0</rdf:first>
           <rdf:rest>
             <rdf:List>
               <rdf:first rdf:datatype="&xsd;integer">15
               </rdf:first>
               <rdf:rest>
                 <rdf:List>
                   <rdf:first rdf:datatype="&xsd;integer">30
                   </rdf:first>
                   <rdf:rest>
                     <rdf:List>
                       <rdf:first rdf:datatype="&xsd;integer">40
                       </rdf:first>
                       <rdf:rest rdf:resource="&rdf;nil" />
                     </rdf:List>
                   </rdf:rest>
                 </rdf:List>
              </rdf:rest>
            </rdf:List>
          </rdf:rest>
        </rdf:List>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
</owl:DatatypeProperty>
```

L'esempio definisce la proprietà `tennisGameScore` che può assumere i valori interi 0, 15, 30, 40. I tipi di dato enumerato non possono essere definiti nel sottolinguaggio OWL

Lite.

4.6.3 Altri tipi di dato

Oltre ai tipi di dato XML Schema di base, è possibile definire altri tipi derivati da essi modficando alcune delle proprietà attraverso cui sono descritti, queste proprietà sono chiamate in XML Schema *facets*. Ecco un esempio di definizione per un tipo di dato *range-1-100* che può avere valori interi compresi tra uno e cento:

```
<xsd:simpleType name="range-1-100">
  <xsd:restriction base="integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="100"/>
  </xsd:restriction>
</xsd:simpleType>
```

La definizione di nuovi tipi di dato non è però permessa per i sottolinguaggi OWL DL e OWL Lite, è quindi permessa solo per il sottolinguaggio OWL Full.

4.7 Annotazioni

In OWL esiste un tipo di proprietà appositamente definito per le annotazioni, sono le *Annotation Properties*. OWL Full non vincola in nessun modo l'utilizzo delle annotazioni in un ontologia. OWL DL, invece, permette di inserire le annotazioni soltanto all'interno delle descrizioni di classi, proprietà, istanze e ontology headers; in OWL DL, inoltre gli insiemi delle *object properties*, *datatype properties*, *annotation properties* e *ontology properties* devono essere completamente disgiunti tra loro. Nel sottolinguaggio OWL DL gli assiomi per la descrizione di proprietà non possono essere utilizzati nelle *annotation properties*, quindi non possono essere definite sottoproprietà o restrizioni sui valori o sulle cardinalità per le annotazioni. Infine, OWL DL vincola l'oggetto di una *annotation pro-*

perties ad essere un tipo di dato XML Schema, un URI o un istanza. In OWL esistono 5 tipi predefiniti di annotazioni:

- owl:versionInfo
- rdfs:label
- rdfs:comment
- rdfs:seeAlso
- rdfs:isDefinedBy

Queste annotazioni possono essere utilizzate per indicare informazioni su un oggetto, rispettivamente la versione, un'etichetta o un commento, un collegamento ad una risorsa simile, un riferimento a chi ha definito l'oggetto. Ecco un'esempio di utilizzo delle annotazioni:

```
<Canzone rdf:about="#StarwayToHeaven">
<rdfs:label rdf:datatype="&xsd;string">Starway to Heaven
</rdfs:label>
<rdfs:comment rdf:datatype="&xsd;string">Led Zeppelin
</rdfs:comment>
</Canzone>
```

Oltre ai cinque tipi predefiniti, è possibile dichiarare altri tipi di annotazioni da utilizzare nelle descrizioni di ontologie, un insieme di metadati utilizzabile per la descrizione di documenti è definito dalla Dublin Core Metadata Initiative. Questo insieme comprende i seguenti tipi di annotazioni definite sotto il namespace `dc` che punta all'URI `http://dublincore.org/documents/2003/06/02/dces/`:

- *dc:title*: A name given to the resource.
- *dc:creator*: An entity primarily responsible for making the content of the resource.
- *dc:subject*: The topic of the content of the resource.

- *dc:description*: An account of the content of the resource.
- *dc:publisher*: An entity responsible for making the resource available.
- *dc:contributor*: An entity responsible for making contributions to the content of the resource.
- *dc:date*: A date associated with an event in the life cycle of the resource.
- *dc:type*: The nature or genre of the content of the resource.
- *dc:format*: The physical or digital manifestation of the resource.
- *dc:identifier*: An unambiguous reference to the resource within a given context.
- *dc:source*: A reference to a resource from which the present resource is derived.
- *dc:language*: A language of the intellectual content of the resource.
- *dc:relation*: A reference to a related resource.
- *dc:coverage*: The extent or scope of the content of the resource.
- *dc:rights*: Information about rights held in and over the resource.

Un esempio per la definizione di una *annotation property* è il seguente:

```
<owl:AnnotationProperty rdf:ID="&dc;creator"/>
```

4.8 Importazione e versioning di ontologie

Il linguaggio OWL mette a dispozione alcuni costrutti per l'importazione di ontologie all'interno di altre e per fornire informazioni sulla versione di un'ontologia e sulla sua compatibilità con le versioni precedenti.

4.8.1 Importazione di ontologie

Per importare un'ontologia all'interno di un'altra, OWL permette di utilizzare il costrutto *owl:imports*, che deve essere inseririto all'interno dell' *ontology header*. Importare un'ontologia significa includere nell'ontologia corrente la definizione di tutte le classi, proprietà e istanze specificate nell'ontologia importata. Per importare un'ontologia occorre indicare, utilizzando il costrutto *owl:imports*, il suo URI che specifica da dove l'ontologia deve essere importata. L'importazione di ontologie è transitiva, quindi se un'ontologia A importa un'ontologia B e l'ontologia B importa un'ontologia C, A importa tutte le difnizioni di B e di C. Se un'ontologia A importa un'ontologia B e B importa A, le due ontologia sono considerate equivalenti. L'utilizzo del costrutto *owl:imports* deve essere come nel seguente esempio:

```
<owl:Ontology rdf:about="">
  <owl:versionInfo>v 1.17 2003/02/26 12:56:51
  </owl:versionInfo>
  <rdfs:comment>An example ontology</rdfs:comment>
  <owl:imports rdf:resource="http://www.example.org/foo"/>
</owl:Ontology>
```

La differenza tra la dichiarazione di un namespace all'inizio del documento e l'utilizzo del costrutto *owl:imports* è che la dichiarazione di un namespace fornisce un collegamento per il riferimento agli oggetti di un ontologia, mentre il costrutto *owl:imports* importa le difinizioni degli oggetti.

4.8.2 Informazioni sulle versioni delle ontologie

OWL permette di fornire informazioni sulla versione di un'ontologia e sulla sua compatibilità con le precedenti versioni attraverso alcuni costrutti:

- *owl:versionInfo*
- *owl:priorVersion*
- *owl:backwardCompatibleWith*

- *owl:incompatibleWith*
- *owl:DeprecatedClass*
- *owl:DeprecatedProperty*

Il costrutto *owl:versionInfo* fa parte delle *annotation property* predefinte dall'OWL e viene utilizzato per indicare la versione di un oggetto, se viene utilizzato all'interno dell'*ontology header* indica la versione dell'ontologia corrente. I costrutti *owl:priorVersion*, *owl:backwardCompatibleWith*, *owl:incompatibleWith*, *owl:DeprecatedClass* e *owl:DeprecatedProperty* fanno parte delle *ontology property*, utilizzate per fornire informazioni relative all'ontologia. La proprietà *owl:priorVersion* contiene il riferimento ad un'altra ontologia ed indica che l'ontologia è la versione precedente a quella corrente. Le proprietà *owl:backwardCompatibleWith* e *owl:incompatibleWith* vengono utlizzate per definire la compatibilità dell'ontologia corrente con altre ontologie. Le proprietà *owl:DeprecatedClass* e *owl:DeprecatedProperty* indicano che le classi o le proprietà specificate saranno modificate o eliminate nelle prossime versioni dell'ontologia.

4.9 Sommario dei costrutti per il linguaggio

In questo paragrafo viene fornito un sommario dei costrutti per il linguaggio OWL, in particolare viene fornito un elenco dei costrutti disponibili per il sottolinguaggio OWL Lite, quindi vengono presentati i costrutti aggiuntivi utilizzabili nei sottolinguaggi OWL DL e OWL Full. I prefissi *rdf:* o *rdfs:* sono impiegati per indicare i costrutti appartenenti a RDF o RDF Schema.

4.9.1 Sommario dei costrutti per OWL Lite

Nella tabella 4.1 sono mostrati i costrutti per il sottolinguaggio OWL Lite.

Costrutti RDF Schema	**Equivalenza**	**Proprietà**
Class (Thing, Nothing)	*equivalentClass*	*ObjectProperty*
rdfs:subClassOf	*equivalentProperty*	*DatatypeProperty*
rdf:Property	*sameAs*	*inverseOf*
rdfs:subPropertyOf	*differentFrom*	*TransitiveProperty*
rdfs:domain	*AllDifferent*	*SymmetricProperty*
rdfs:range	*distinctMembers*	*FunctionalProperty*
Individual		*InverseFunctionalProperty*
Restrizioni sulle proprietà	**Restrizioni di cardinalità**	**Header dell'ontologia**
Restriction	*minCardinality (0 or 1)*	*Ontology*
rdfs:subClassOf	*maxCardinality (0 or 1)*	*imports*
onProperty	*cardinality (0 or 1)*	
allValuesFrom		
someValuesFrom		
Intersezione tra classi	**Versioning**	**Annotation Properties**
intersectionOf	*versionInfo*	*rdfs:label*
	priorVersion	*rdfs:comment*
	backwardCompatibleWith	*rdfs:seeAlso*
Tipi di dato	*incompatibleWith*	*rdfs:isDefinedBy*
xsd datatypes	*DeprecatedClass*	*AnnotationProperty*
	DeprecatedProperty	*OntologyProperty*

Tabella 4.1
Costrutti di OWL Lite

4.9.2 Sommario dei costrutti aggiuntivi per OWL DL e OWL Full

Nella tabella 4.2 sono mostrati i costrutti per i sottolinguaggi OWL DL e OWL Full che estendono il sottolinguaggio OWL Lite.

Assiomi per le classi	**Combinazioni booleane**
oneOf ... dataRange *disjointWith* *equivalentClass* *rdfs:subClassOf*	*unionOf* *complementOf* *intersectionOf*
Restrizioni di cardinalità	**Restrizioni sui valori**
minCardinality *maxCardinality* *cardinality*	*hasValue*

Tabella 4.2
Costrutti aggiuntivi di OWL DL e OWL Full

5. SPECIFICHE DEL TRADUTTORE PER ONTOLOGIE OWL

All'interno di questo capitolo viene proposta una possibile traduzione dal linguaggio ODL_{I^3} al linguaggio OWL, tale traduzione è stata utilizzata per la realizzazione di un wrapper impiegato all'interno del sistema MOMIS. Il linguaggio ODL_{I^3} è un linguaggio definito per la descrizione e l'integrazione di sorgenti di dati eterogenee, siano esse database ad oggetti, relazionali, sorgenti di dati semistrutturati o file. OWL è invece un linguaggio nato per la definizione di ontologie compatibili con l'architettura del Semantic Web, una visione futura del Web in cui viene dato un significato esplicito a tutte le informazioni, in modo da rendere più semplice e rapido agli agenti software la loro integrazione. La principale differenza tra i due linguaggi sta proprio nella modalità di descrizione dei concetti, ODL_{I^3} è facilmente leggibile e comprensibile anche dalle persone, mentre OWL, essendo orientato ad una massima comprensibilità da parte degli agenti software, è più verboso e risulta faticosamente leggibile dalle persone. Sebbene vengano entrambi utilizzati per la descrizione di ontologie, quindi, i due linguaggi sono stati definiti per scopi diversi tra loro, risulta quindi comprensibile come alcuni concetti non possano essere tradotti fedelmente nel passaggio da un linguaggio all'altro. La traduzione è stata effettuata con l'intento di avere una trasposizione il più possibile bidirezionale, l'ontologia ODL_{I^3} e l'ontologia OWL risultante dalla traduzione devono infatti esprimere gli stessi concetti. Informazioni come l'architettura dell'ontologia, il nome degli elementi e la loro struttura, le proprietà e le relazioni che legano gli elementi tra loro, sono di fondamentale importanza per una traduzione bidirezionale, occorre quindi che tutte queste informazioni non vengano perdute.

5.1 Traduzione dei tipi semplici

La prima questione considerata nella trasposizione dal linguaggio ODL$_{I^3}$ linguaggio OWL è la traduzione dei tipi semplici, composti dai tipi semplici di base e dai tipi collezione che vengono utilizzati per rappresentare collezioni di dati semplici.

5.1.1 Traduzione dei BaseType

Tra i tipi atomici di base o *BaseType* di ODL$_{I^3}$ e i tipi di dato XML Schema utilizzati con OWL è possibile ottenere una traduzione uno ad uno, questo consente di avere una rappresentazione fedele dei dati in OWL. La corrispondenza tra i tipi semplici di base e i tipi di dato XML Schema è mostrata nella tabella [5.1].

5.1.2 Traduzione del tipo Range

Il tipo Range, aggiunto nel linguaggio esteso ODL$_{I^3}$ ai tipi semplici di base facenti parte dell'ODL, è utilizzato per rappresentare un valore intero compreso tra un estremo inferiore ed uno superiore. Eccone un esempio di utilizzo:

```
range 1,100 number;
```

l'attributo numero può assumere tutti i valori interi compreso tra l'estremo inferiore 1 e l'estremo superiore 100. XML Schema non prevede un tipo di dato per la specificazione di un range di interi ed il linguaggio OWL non fornisce, almeno per ora, nessun costrutto per la rappresentazione di questo concetto. Come specificato dalla documentazione OWL **–>OWL Web Ontology Language Use Cases and Requirements**, infatti, il linguaggio dovrebbe supportare in futuro la possibilità di specificare range di valori per le proprietà, il motivo che fino ad ora non ha permesso di aggiungere un costrutto per questo scopo è l'impossibilità da parte di RDF e RDF Schema di esprimere l'inconsistenza di valori. Una possibile soluzione per la traduzione di questo costrutto è quella di creare, per ogni range presente nell'ontologia, un tipo di dato XML Schema derivato dal tipo integer in cui le proprietà *minInclusive* e *maxInclusive* vengono settate ai limiti inferiore e superiore del

Tipo ODL$_{I^3}$	Esempio	Tipo OWL
boolean	true, false	xsd:boolean
char	0, 126	xsd:unsignedByte
string	stringa di prova	xsd:string
date	2004-10-27	xsd:date
timestamp	2004-10-27T11:20:00.000-05:00	xsd:dateTime
octet	-1, 0, 126, +100	xsd:byte
float	-1E4, 12.78e-2, 12	xsd:float
double	1267.43233E12, -1234.46e-14	xsd:double
int	-1, 0, 1267896754, +100000	xsd:int
short	-1, 0, 12678, +10000	xsd:short
long	+92233720368547758, -72036854775807720	xsd:long
unsigned int	1267896754, 100000	xsd:unsignedInt
unsigned short	12678, 64535	xsd:unsignedShort
unsigned long	0, 12678967543233, 100000	xsd:unsignedLong
any	true, 1267896754, 2004-10-27	xsd:anyType

Tabella 5.1
Corrispondenza tra tipi di dato ODL$_{I^3}$ e XML Schema

range. Ecco un esempio di traduzione per un tipo range *range-1-100* che può avere valori interi compresi tra uno e cento:

```
<xsd:simpleType name="range-1-100">
  <xsd:restriction base="integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="100"/>
  </xsd:restriction>
</xsd:simpleType>
```

La definizione di nuovi tipi di dato è consentita solo per il sottolinguaggio OWL Full e non per i sottolinguaggi OWL DL e OWL Lite, la soluzione presentata potrà quindi essere utilizzata solo per OWL Full. In OWL non si hanno operatori per il confronto tra valori, quindi la soluzione adottata per OWL DL è stata quella di creare per ogni attributo range ODL_{I^3} una *datatype property* OWL che può assumere valori interi, quindi vincolare la proprietà corrispondente all'attributo range ad una lista enumerata di valori compresi tra gli estremi inferiore e superiore del range.

5.2 Traduzione di CollectionType e ArrayType

In questa sezione verranno descritte le traduzioni proposte per i tipi *CollectionType* e *ArrayType* del linguaggio ODL_{I^3}, in linguaggio OWL.

5.2.1 Traduzione di Set, List e Bag

I tipi collezione (o *CollectionType*) vengono utilizzati per rappresentare collezioni di dati semplici. Le collezioni ODL_{I^3} possono essere di tre tipi differenti:

- *bag*: collezione di elementi non ordinati;
- *set*: collezione di elementi non ordinati che non contiene duplicati;
- *list*: collezione ordinata di elementi non duplicati.

Per la definizione di liste di elementi in OWL si possono utilizzare i tipi RDF *Container Type*, i costruttori corrispondenti sono i seguenti:

- *rdf:Bag*: collezione di elementi non ordinati;
- *rdf:Alt*: collezione di elementi non ordinati che non contiene duplicati;
- *rdf:Seq*: collezione ordinata di elementi non duplicati.

Il tipo *bag* ODL$_{I^3}$ verrà quindi tradotto in OWL con il costrutto *rdf:Bag*, il tipo ODL$_{I^3}$ *set* verrà tradotto con il costrutto OWL *rdf:Alt* e il tipo ODL$_{I^3}$ *list* con il costrutto OWL *rdf:Seq*. Nei sottolingaggi OWL DL e OWL Lite i *Container Type* RDF non possono essere utilizzati, occorre quindi creare i tipi corrispondenti a quelli RDF in un ontologia OWL contenente le definizioni dei tipi, tutte le definizioni saranno poi importate dalle ontologie create per la traduzione. Nell'esempio ODL$_{I^3}$ seguente la collezione di tipo Bag `type` è definita come attributo della classe `Report` :

```
interface Report( ){
      attribute bag <string> type *;
    }
```

Ecco la traduzione OWL corrispondente:

```
<owl:Class rdf:ID="Report.type_Bag">
  <rdfs:subClassOf rdf:resource="&rdf;Bag"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="Report.type">
  <rdfs:domain rdf:resource="#Report"/>
  <rdfs:range rdf:resource="#Report.type_Bag"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="Report.type_Bag_member_of">
  <rdfs:domain rdf:resource="Report.type_Bag"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

5.2.2 Traduzione del tipo ArrayType

Il tipo *ArrayType* viene utilizzato in ODL$_{I^3}$ per definire una collezione ordinata con un numero fisso di elementi. In OWL una collezione ordinata di elementi viene descritta attraverso il costrutto *rdf:Seq*, per indicare un numero fisso di elementi viene utilizzata una restrizione di cardinalità sulla proprietà corrispondente. Per i sottolinguaggi OWL DL e OWL Lite il *Container Type rdf:Seq* non può essere utilizzato, occorre quindi creare il tipo corrispondente in un ontologia OWL contenente le definizioni dei tipi.

5.3 Traduzione dei tipi costruttori (ConstrType)

I tipi costrutti (*ConstrType*) di ODL$_{I^3}$ sono suddivisi in tre sottocategorie:

- *UnionType*
- *StructType*
- *EnumType*

Il tipo unione (*UnionType*) viene utilizzato per scegliere, in base al valore di una variabile in una clausola *switch*, il tipo di appartenenza di un attibuto. In OWL questa caratteristica viene espressa tramite il costrutto *owl:unionOf* attraverso il quale è possibile definire come oggetto di una proprietà un insieme di più classi o range di valori. Il seguente esempio ODL$_{I^3}$ definisce l'attributo `TypeNumber` come stringa o intero:

```
union TypeNumber switch(val) {
case 1: string;
case 2: int;
};
```

Lo stesso concetto viene espresso in OWL tramite il codice seguente:

```
<owl:DatatypeProperty rdf:ID="TypeNumber">
  <rdfs:range>
    <owl:DataRange>
```

```
      <owl:unionOf rdf:parseType="Collection">
        <rdfs:Datatype rdf:about="&xsd;string"/>
        <rdfs:Datatype rdf:about="&xsd;int"/>
      </owl:unionOf>
    </owl:DataRange>
  </rdfs:range>
</owl:DatatypeProperty>
```

I costrutti OWL *owl:DataRange* e *owl:unionOf* non possono essere utilizzati per il sottolinguaggio OWL Lite, quindi per questo sottolinguaggio non si può avere una traduzione fedele del tipo ODL_{I^3} *UnionType*.
Il tipo struttura (*StructType*) di ODL_{I^3} viene utilizzato per definire strutture di tipi valore, la traduzione OWL di questo tipo è data da una classe e da una serie di *datatype properties* corrispondenti ai tipi valore della struttura. Un esempio ODL_{I^3} per la definizione di una struttura è il seguente:

```
typedef struct {
string a;
boolean b;
unsignedlong c;
} structType ;
```

La trasposizione in linguaggio OWL della struttura sarà espressa dal seguente codice:

```
<owl:Class rdf:ID="structType"/>
<owl:DatatypeProperty rdf:ID="a">
  <rdfs:domain rdf:resource="structType"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="b">
  <rdfs:domain rdf:resource="structType"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="c">
```

```
  <rdfs:domain rdf:resource="structType"/>
  <rdfs:range rdf:resource="&xsd;unsignedLong"/>
</owl:DatatypeProperty>
```

5.3.1 Traduzione dei tipi enumerati

I tipi enumerazione (*EnumType*) del linguaggio ODL$_{I^3}$ restringono il dominio di un *SimpleType* ad un elenco di valori possibili. Nel linguaggio OWL i tpi di dato enumerati devono essere definiti attraverso l'utilizzo di una lista RDF, la descrizione di tipi di dato enumerati è quindi piuttosto verbosa, come si può notare dall'esempio successivo. Il tipo di dato enumerato tennisGameScore viene rappresentato in ODL$_{I^3}$ nel seguente modo:

```
enum tennisGameScore {0 , 15 , 30 , 40};
```

la traduzione in OWL sarà invece la seguente:

```
<owl:DatatypeProperty rdf:ID="tennisGameScore">
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf>
        <rdf:List>
           <rdf:first rdf:datatype="&xsd;integer">0</rdf:first>
           <rdf:rest>
             <rdf:List>
               <rdf:first rdf:datatype="&xsd;integer">15</rdf:first>
               <rdf:rest>
                 <rdf:List>
                   <rdf:first rdf:datatype="&xsd;integer">30
                   </rdf:first>
                   <rdf:rest>
                     <rdf:List>
                       <rdf:first rdf:datatype="&xsd;integer">40
                       </rdf:first>
                       <rdf:rest rdf:resource="&rdf;nil" />
                     </rdf:List>
                   </rdf:rest>
```

```
                </rdf:List>
              </rdf:rest>
            </rdf:List>
          </rdf:rest>
        </rdf:List>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
</owl:DatatypeProperty>
```

Anche i tipi di dato enumerato non possono essere definiti in OWL Lite, essendo rappresentati attraverso i costrutti *owl:DataRange* e *owl:oneOf* proibiti in questo sottolinguaggio.

5.4 Traduzione delle interfacce

Le classi nel linguaggio ODL$_{I^3}$ vengono chiamate interfacce (*interface*). La definizione di un interfaccia è costituita dall'interface header, in cui vengono specificate diverse caratteristiche dell'interfaccia tra cui il nome della classe, le superclassi da cui eredita ed una lista di proprietà chiamata *PropertyList* e dall'*interface body*, nel quale è definito l'insieme di attributi e di metodi dell'interfaccia. Il nome di un'interfaccia in ODL$_{I^3}$ è l'identificatore unico dell'oggetto (ObjectID) all'interno di uno schema, anche nel linguaggio OWL una classe è identificata univocamente dal suo *rdf:ID* rappresentato da un URI. Sia in ODL$_{I^3}$ che in OWL è ammessa l'ereditarietà multipla. Il seguente esempio rappresenta una interfaccia descritta in linguaggio ODL$_{I^3}$:

```
interface Report : Reference ( ){
  attribute string title;
  attribute Institution institution;
  attribute Date date;
  attribute int number;
};
```

la traduzione della classe in linguaggio OWL è la seguente:

```
<owl:Class rdf:ID="Report">
  <rdfs:subClassOf rdf:resource="#Reference"/>
</owl:Class>
<owl:DatatypeProperty rdf:ID="title">
  <rdfs:domain rdf:resource="#Report"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="institution">
  <rdfs:domain rdf:resource="#Report"/>
  <rdfs:range rdf:resource="#Institution"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="date">
  <rdfs:domain rdf:resource="#Report"/>
  <rdfs:range rdf:resource="#Date"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="number">
  <rdfs:domain rdf:resource="#Report"/>
  <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
```

5.4.1 Specificazione del tipo di Source e delle proprietà Extent e Persistent

All'interno della *PropertyList* di una classe ODL$_{I^3}$ è possibile indicare il nome ed il tipo della sorgente dati a cui appartiene l'interfaccia: la sorgente può essere di tipo relazionale (*relational* o *nfrelational*), semistrutturato (*semistructured*), ad oggetti (*object*), o file (*file*). In OWL ogni sorgente viene tradotta come superclasse di tutte le classi appartenenti ad essa, il tipo di sorgente viene quindi specificato una sola volta come proprietà della superclasse: la traduzione delle sorgenti e della struttura dello schema verrà trattata in modo dettagliato nel paragrafo [5.10]. Sempre nella *PropertyList* di una classe ODL$_{I^3}$ vengono indicate le proprietà *extent* e *persistent*. La traduzione OWL di queste caratteristiche viene

effettuata con il codice seguente:

```
<owl:Class rdf:ID="#CS_Person">
  <rdfs:subClassOf rdf:resource="#Computer_Science"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#CS_Person.extent"/>
      <owl:hasValue rdf:datatype="&xsd;string">Persons
      </owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#CS_Person.persistent"/>
      <owl:hasValue rdf:datatype="&xsd;boolean">false
      </owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

L'esempio descrive la classe `CS_Person` appartenente alla sorgente `Computer_Science` e specifica i valori per le proprietà *extent* e *persistent*.

5.4.2 Traduzione di Key e ForeignKey

La definizione delle keys e foreign keys è sempre contenuta nella *PropertyList* dell'interfaccia ODL_{I^3}. La possibilità di specificazione delle chiavi in ODL_{I^3} deriva dall'integrazione nello schema di database ad oggetti e relazionali. Il linguaggio OWL non prevede nessun costrutto per la definizione dei concetti di chiave, per cui le proprietà *Key* e *ForeignKey* sono definite all'interno di un ontologia OWL contenente le definizioni dei tipi. La proprietà Key di una classe deve essere una *inverse functional property* (l'oggetto della

proprietà determina in modo univoco il soggetto) e la sua inversa deve avere cardinalità uguale ad uno (di modo che una classe abbia soltanto una chiave), inoltre il valore della proprietà *Key* deve avere un valore non nullo. L'insieme dei valori di una proprietà *Key* deve essere completamente disgiunto (non posso avere due valori di chiave uguali). In questo modo ogni valore di chiave identifica univocamente una ed una sola istanza della classe ed ogni istanza ha un solo valore di chiave. Per chiavi composte da più attributi ogni attributo dovrà rispettare le condizioni indicate precedentemente, ma sarà l'insieme delle proprietà *Key* (e non la singola proprietà) ad identificare in modo univoco la classe.

5.5 Traduzione degli attributi

Gli attributi di una interfaccia ODL_{I^3} possono essere attributi di tipo semplice o complesso: gli attributi semplici sono attributi di tipo valore (*ValueType*), gli attributi complessi sono invece di tipo *interface*. Gli attributi sono identificati univocamente dal loro nome all'interno dell'*interface body*. Gli attributi complessi vengono tradotti in OWL con le *object properties* ossia con relazioni tra due classi. Gli attributi di tipo valore vengono tradotti sia con *datatype properties* che con *object properties* a seconda del tipo. Il tipo di attributi atomici di base *BaseType* viene tradotto con *datatype properties* che hanno come oggetto il tipo di dato XML Schema corrispondente al tipo di base ODL_{I^3}, come indicato nella tabella [5.1]. Il tipo collezione *CollectionType* ed il tipo costrutti *ConstrType* vengono invece rappresentati attraverso *object properties*. Il seguente esempio rappresenta la descrizione degli attributi di una interfaccia in linguaggio ODL_{I^3}:

```
interface Book ( ){
  attribute string title;
  attribute Author author;
  attribute Date date;
  attribute int pages;
};
```

la traduzione delle proprietà in linguaggio OWL è la seguente:

```
  <owl:Class rdf:ID="Book"/>
```

```
<owl:DatatypeProperty rdf:ID="title">
  <rdfs:domain rdf:resource="#Book"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="author">
  <rdfs:domain rdf:resource="#Book"/>
  <rdfs:range rdf:resource="#Author"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="date">
  <rdfs:domain rdf:resource="#Book"/>
  <rdfs:range rdf:resource="#Date"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="pages">
  <rdfs:domain rdf:resource="#Book"/>
  <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
```

Gli attributi `title` e `pages` sono tipi di base ODL_{I^3} e vengono tradotti in OWL con *datatype properties*, gli attributi `author` e `date` sono attributi ODL_{I^3} complessi e vengono rappresentati da *object properties* nel linguaggio OWL.

5.5.1 Cardinalità degli attributi

In ODL_{I^3} la cardinalità degli attributi è di default uno, postponendo un asterisco (*) alla dichiarazione di un attributo si indica che l'attributo è opzionale, si definisce quindi una cardinalità minima uguale a zero e una cardinalità massima uguale a uno. Una cardinalità maggiore di uno viene specificata indicandola tra parentesi quadre. Nel linguaggio OWL le proprietà hanno una cardinalità che va da zero ad infinito, per effettuare restrizioni sulle cardinalità delle proprietà si utilizzano i costrutti *owl:maxCardinality*, *owl:minCardinality* e *owl:cardinality*. Se per un attributo ODL_{I^3} non è specificata nessuna cardinalità esso ha cardinalità minima e massima uguali ad uno, in OWL questo si traduce con una restrizione

di cardinalità sulla proprietà corrispondente:

```
interface Book ( ){
  attribute Author author;
};
```

in OWL la cardinalità della proprietà `author` viene vincolata ad uno utilizzando il costrutto *owl:cardinality* che definisce cardinalità minima e massima uguali:

```
<owl:Class rdf:ID="Book">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#author"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Se un attributo ODL_{I^3} è dichiarato come opzionale esso ha cardinalità minima uguale a zero e massima uguale ad uno:

```
interface Book ( ){
  attribute Author author*;
};
```

in OWL la cardinalità massima della proprietà `author` viene vincolata ad uno utilizzando il costrutto *owl:maxCardinality*, la cardinalità minima è di default uguale a zero:

```
<owl:Class rdf:ID="Book">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#author"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
      1</owl:maxCardinality>
```

```
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Se per un attributo ODL$_{I^3}$ viene specificata una cardinalità (minima e massima) maggiore di uno:

```
interface Book ( ){
  attribute Author author[3];
};
```

in OWL le cardinalità minima e massima della proprietà `author` vengono vincolate a 3 utilizzando il costrutto *owl:cardinality*:

```
<owl:Class rdf:ID="Book">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#author"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">3
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

5.6 Traduzione di Relazioni inverse

Le relazioni (*Relationship*) definite all'interno di un interfaccia ODL$_{I^3}$ sono attributi complessi: esse possono avere come oggetto solo interfacce. All'interno di una relazione è possibile inoltre specificare la relazione inversa. In OWL le relazioni vengono tradotte come *object property*, la relazione inversa viene dichiarata utilizzando il costrutto *owl:inverseOf*. Ecco un esempio di relazione inversa tradotta in OWL:

```
<owl:ObjectProperty rdf:ID="author">
<rdfs:domain rdf:resource="#Book"/>
```

```
    <rdfs:range rdf:resource="#Author"/>
  <owl:inverseOf rdf:resource="#isauthor"/>
</owl:ObjectProperty>
```

La proprietà `author` che ha come soggetto `Book` e come oggetto `Author` viene dichiarata come inversa della proprietà `isauthor` che ha come soggetto `Author` e come oggetto `Book`.

5.7 Traduzione delle Rule

Attraverso le *Rule* ODL$_{I^3}$ è possibile definire vincoli di integrità sotto forma di regole di tipo if-then: le condizioni citate all'interno di una regola devono essere rispettate da tutte le istanze delle classi ODL$_{I^3}$ specificate. Il linguaggio OWL non fornisce alcun costrutto per la definizione di regole di questo tipo, la traduzione viene effettuata aggiungendo restrizioni sulle proprietà alle classi specificate. Un esempio di regola d'integrità può essere il seguente:

```
interface Book ( ) {
attribute string title;
attribute range {1, 10} category;
attribute int price;
};
rule rule1 forall X in Book: X.category = 8
then X.price = 25;
```

la traduzione per questo tipo di regola in OWL è il seguente:

```
<owl:Class rdf:about="Book_category_8">
  <rdfs:subClassOf rdf:resource="#Book"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#category"/>
      <owl:hasValue rdf:datatype="&xsd;Int">8</owl:hasValue>
    </owl:Restriction>
    <owl:Restriction>
```

```
      <owl:onProperty rdf:resource="#price"/>
      <owl:hasValue rdf:datatype="&xsd;Int">25</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Una regola del tipo seguente:

```
rule rule1 forall X in Book: X.category >=5 and
X.category <=10
then X.price = 25;
```

sarà traducibile nel sottolinguaggio OWL Full definendo un nuovo tipo di dato XML Schema:

```
<owl:Class rdf:about="Book_category_8">
  <rdfs:subClassOf rdf:resource="#Book"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#category"/>
      <owl:allValuesFrom rdf:datatype="&xsd;rangeCategory"/>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#price"/>
      <owl:hasValue rdf:datatype="&xsd;int">25</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
...
<xsd:simpleType name="rangeCategory">
  <xsd:restriction base="integer">
    <xsd:minInclusive value="5"/>
    <xsd:maxInclusive value="10"/>
  </xsd:restriction>
</xsd:simpleType>
```

Per il sottolinguaggio OWL DL che non prevede la definizione di nuovi tipi di dato XML Schema dovrà invece essere utilizzato un insieme enumerato di valori:

```
<owl:Class rdf:ID="Book_category_5-10">
  <rdfs:subClassOf rdf:resource="#Book"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#category"/>
      <owl:allValuesFrom rdf:resource="#enumCategory"/>
      </owl:allValuesFrom>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#price"/>
      <owl:hasValue rdf:datatype="&xsd;Int">25</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:DataRange rdf:ID="enumCategory">
 <owl:oneOf>
   <rdf:List>
    <rdf:first rdf:datatype="&xsd;int">5</rdf:first>
    <rdf:rest>
      <rdf:List>
       <rdf:first rdf:datatype="&xsd;int">6</rdf:first>
       <rdf:rest>
         <rdf:List>
          <rdf:first rdf:datatype="&xsd;int">7</rdf:first>
          <rdf:rest>
            <rdf:List>
             <rdf:first rdf:datatype="&xsd;int">8</rdf:first>
             <rdf:rest>
               <rdf:List>
                <rdf:first rdf:datatype="&xsd;int">9</rdf:first>
```

```
                <rdf:rest>
                  <rdf:List>
                   <rdf:first rdf:datatype="&xsd;int">10
                   </rdf:first>
                   <rdf:rest rdf:resource="&rdf;nil"/>
                  </rdf:List>
                 </rdf:rest>
                </rdf:List>
               </rdf:rest>
              </rdf:List>
             </rdf:rest>
            </rdf:List>
           </rdf:rest>
          </rdf:List>
         </rdf:rest>
        </rdf:List>
       </rdf:rest>
      </rdf:List>
     </owl:oneOf>
    </owl:DataRange>
```

5.8 Traduzione delle Relazioni terminologiche (Thesaurus relations)

Nel sistema MOMIS, tramite il linguaggio ODL$_{I^3}$, è possibile esprimere relazioni terminologiche di diversi tipi che possono intercorrere tra nomi di classi (*InterfaceRel*), tra attributi (*AttributeRel*) e miste tra nomi di classi ed attributi (*AttrIntRel*). Le relazioni terminologiche possono essere di quattro tipi:

- Ipernimia (BT):
- Iponimia (NT):
- Sinonimia (SYN):
- Associazione (RT):

5.8.1 Relazioni di tipo estensionale

Le relazioni terminologiche di tipo estensionale esprimono un legame tra istanze. Una relazione estensionale di ipernimia (*BT*) o iponimia (*NT*) tra due classi viene tradotta in linguaggio OWL tramite l'assioma *rdfs:subClassOf* il quale dichiara che l'estensione della classe soggetto è un sottoinsieme dell'estensione della classe oggetto. Date le due classi `Wine` e `RedWine` , le relazioni `Wine BT RedWine` o `RedWine NT Wine` implicano la seguente definizione OWL:

```
<owl:Class rdf:about="RedWine">
  <rdfs:subClassOf rdf:resource="#Wine"/>
</owl:Class>
```

Una relazione estensionale di ipernimia o iponimia tra due attributi ODL_{I^3} viene tradotta con il linguaggio OWL in modo analogo all'esempio precedente utilizzando l'assioma *rdfs:subPropertyOf*. La relazione estensionale di sinonimia (*SYN*) tra due classi implica una equivalenza tra le estensioni delle due classi e viene espressa in OWL utilizzando l'assioma *owl:equivalentClass*. Date le due classi `Book` e `Novel` , la relazione `Book SYN Novel` viene tradotta in OWL nel seguente modo:

```
<owl:Class rdf:about="Book">
  <owl:equivalentClass rdf:resource="#Novel"/>
</owl:Class>
```

In modo analogo al precedente esempio, una relazione estensionale di sinonimia tra due attributi ODL_{I^3}, viene tradotta utilizzando l'assioma *owl:equivalentProperty*.

5.8.2 Relazioni di tipo intensionale

Le relazioni terminologiche di tipo intensionale implicano un legame tra due concetti rappresentati dalle classi o dagli attributi ODL_{I^3}. Questo tipo di relazioni, quindi, non può essere espresso tramite i costrutti OWL *rdfs:subClassOf*, *rdfs:subPropertyOf*, *owl:equivalentClass* o *owl:equivalentProperty* che definiscono legami solamente tra le estensioni dei concetti. Per tradurre efficacemente le relazioni terminologiche di tipo intensionale vengono quindi create delle *annotation property* chiamate *Thesaurus Relations*.

Le *Thesaurus Relations* fanno parte dell'ontologia OWL e vengono utilizzate per definire le relazioni terminologiche intensionali che intercorrono tra due classi, due proprietà o tra una classe ed una proprietà. Segue un esempio di definizione di relazione terminologica intensionale espressa tramite una *thesaurus relation*:

```
<sew:ThesRelation rdf:ID="ThesRelation_1">
  <sew:RelClass rdf:datatype="&xsd;string">InterfaceRel
  </sew:RelClass>
  <sew:Attribute1 rdf:datatype="&xsd;anyURI">&university;Room
  </sew:Attribute1>
  <sew:RelType rdf:datatype="&xsd;string">rt</sew:RelType>
  <sew:Attribute2 rdf:datatype="&xsd;anyURI">&university;Section
  </sew:Attribute2>
  <sew:ProducerID rdf:datatype="&xsd;int">900</sew:ProducerID>
</sew:ThesRelation>
```

La relazione descritta nell'esempio è una relazione intensionale di associazione (*RT*) tra le due classi (*InterfaceRel*) `Room` e `Section`.

5.9 Traduzione delle Annotazioni (WordNet annotations)

Il linguaggio ODL_{I^3} esprime le annotazioni lessicali rispetto a WordNet utilizzando il costrutto *WNAnnotation*, eccone un esempio di utilizzo:

```
wnAnnotation ComputerScience.Professor
   lemmaValue="professor",
   lemmaSyntacticCategory=1,
   lemmaSenseNumber=1;
```

Questa annotazione assegna al nome dell'interfaccia `Professor` appartenente alla sorgente `ComputerScience` il valore `"professor"`, indicando una categoria sintattica uguale ad uno (corrispondente alla categoria sintattica "nome") ed un numero di senso uguale ad uno (corrispondente al significato: "someone who is a member of the faculty at a

college or university"). In OWL le annotazioni lessicali rispetto a Wordnet vengono tradotte attraverso la creazione di apposite *annotation property* chiamate *WordNetAnnotations*. Le *WordNetAnnotations* create sono di 6 tipi diversi:

- *sew:lemmaValue*
- *sew:lemmaSyntacticCategory*
- *sew:lemmaSenseNumber*
- *sew:nodeUri*
- *sew:ontoVersionInfo*
- *sew:ontoCreator*

Le annotazioni *lemmaValue*, *lemmaSyntacticCategory*, e *lemmaSenseNumber* sono corrispondenti alle *WNAnnotation* di ODL$_{I^3}$. Le proprietà *nodeUri*, *ontoVersionInfo* e *ontoCreator* permettono di annotare rispettivamente l'URI su cui viene effettuata l'annotazione, le informazioni sulla versione dell'ontologia e sul creatore dell'annotazione.

5.10 Traduzione dello Schema

Lo *Schema* nel linguaggio ODL$_{I^3}$ rappresenta l'oggetto nel quale sorgenti eterogenee vengono integrate. In OWL questo concetto viene tradotto attraverso il costrutto *owl:imports* che consente di includere in un'ontologia la definizione di tutte le classi, proprietà e istanze specificate all'interno dell'ontologia importata. Nella fase di traduzione di uno *Schema* ODL$_{I^3}$, verrà quindi creata in OWL un'ontologia globale che importa le ontologie corrispondenti alle diverse sorgenti. Ogni sorgente viene tradotta in OWL come superclasse di tutte le classi appartenenti ad essa, il tipo di sorgente (*relational*, *nfrelational*, *semistructured*, *object*, o *file*) viene quindi specificato come proprietà della superclasse. Segue un esempio di *Schema* ODL$_{I^3}$ tradotto in OWL:

```
<!DOCTYPE rdf:RDF [
   <!ENTITY owl "http://www.w3.org/2002/07/owl#">
```

```
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
    <!ENTITY xsdsew "http://dbgroup.unimo.it/samples/sewasie.xsd#"
    >
    <!ENTITY sew "http://dbgroup.unimo.it/samples/sewasie.owl#">
    <!ENTITY Computer_Science
    "http://dbgroup.unimo.it/samples/Computer_Science.owl#">
    <!ENTITY Tax_Position
    "http://dbgroup.unimo.it/samples/Tax_Position.owl#">
    <!ENTITY University
    "http://dbgroup.unimo.it/samples/University.owl#">
]>
<rdf:RDF
  xmlns="http://dbgroup.unimo.it/samples/
  Schema_statoCompleto.owl#"
  xml:base="http://dbgroup.unimo.it/samples/
  Schema_statoCompleto.owl"
  xmlns:sew="&sew;"
  xmlns:owl="&owl;"
  xmlns:rdf="&rdf;"
  xmlns:rdfs="&rdfs;"
  xmlns:xsd="&xsd;"
  xmlns:xsdsew="&xsdsew;"
  xmlns:Computer_Science="&Computer_Science;"
  xmlns:Tax_Position="&Tax_Position;"
  xmlns:University="&University;"
>
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource="&sew;"/>
  <owl:imports rdf:resource="&Computer_Science;"/>
```

```
  <owl:imports rdf:resource="&Tax_Position;"/>
  <owl:imports rdf:resource="&University;"/>
  <rdfs:comment>Schema name="Schema_statoCompleto"
  momisCodeVersion="momis.0.4.3 compiled.06-September-2004"
  </rdfs:comment>
  <rdfs:label>Schema_statoCompleto Ontology</rdfs:label>
  <owl:versionInfo>04-09-06</owl:versionInfo>
</owl:Ontology>
...
</rdf:RDF>
```

L'ontologia globale `Schema_statoCompleto` importa, oltre all'ontologia di definizione dei tipi `sew` , le ontologie corrispondenti alle sorgenti `Computer_Science` , `Tax_Position` e `University` .

5.11 Esempio di traduzione ODL$_{I^3}$-OWL

Un esempio completo di traduzione di un'ontologia ODL$_{I^3}$ in un'ontologia OWL è reperibile all'indirizzo *http://www.dbgroup.unimo.it/ orsini/esempioOdli3-Owl.pdf*.

5.12 Confronto tra i costrutti ODL$_{I^3}$ e OWL

Nelle seguenti tabelle viene presentato un confronto tra i costrutti del linguaggio ODL$_{I^3}$ e i costrutti dei sottolinguaggi OWL DL e OWL Lite, in particolare vengono evidenziate le differenze tra i due sottolinguaggi. Nel sottolinguaggio OWL Lite, infatti, molti dei costrutti necessari alla traduzione di un ontologia ODL$_{I^3}$ non sono presenti.
Il prefisso *sew:* è impiegato per indicare le proprietà definite all'interno del progetto SEWASIE.

Costrutti ODL$_{I^3}$	**Costrutti OWL DL**	**Costrutti OWL Lite**	**Interpretazione Logica**
–	owl:Thing	owl:Thing	insieme di tutte le istanze
–	owl:Nothing	owl:Nothing	insieme vuoto
interface	owl:Class	owl:Class	concetto di classe (intensionale)
view	owl:Class	owl:Class	concetto di vista
isa	rdfs:subClassOf rdfs:subPropertyOf	rdfs:subClassOf (classi o restrizioni)	gerarchia estensionale
nt$_{ext}$ bt$_{ext}$	rdfs:subClassOf rdfs:subPropertyOf	rdfs:subClassOf (classi o restrizioni)	gerarchia estensionale
syn$_{ext}$	owl:equivalentClass owl:equivalentProperty	owl:equivalentClass (classi o restrizioni)	equivalenza estensionale
and	owl:intersectionOf (classi o restrizioni)	owl:intersectionOf	intersezione
union	owl:unionOf	–	unione
enum	owl:DataRange...owl:oneOf ... rdf:List...rdf:Rest	–	enumerazione
range	owl:DataRange...owl:oneOf ... rdf:List...rdf:Rest (solo per range limitati)	–	range di valori
–	owl:complementOf	–	negazione
–	owl:disjointWith	–	disgiunzione estensionale
bt	sew:ThesRelation... sew:RelType...bt	sew:ThesRelation... sew:RelType...bt	ipernimia intensionale (ODL$_{I^3}$)
nt	sew:ThesRelation... sew:RelType...nt	sew:ThesRelation... sew:RelType...nt	iponimia intensionale (ODL$_{I^3}$)
rt	sew:ThesRelation... sew:RelType...rt	sew:ThesRelation... sew:RelType...rt	associazione (ODL$_{I^3}$)
syn	sew:ThesRelation... sew:RelType...syn	sew:ThesRelation... sew:RelType...syn	sinonimia intensionale (ODL$_{I^3}$)

Tabella 5.2
Confronto tra costrutti ODL$_{I^3}$ e OWL

Costrutti ODL$_{I^3}$	**Costrutti OWL DL**	**Costrutti OWL Lite**	**Interpretazione Logica**
–	owl:sameAs (solo per istanze)	owl:sameAs (solo per istanze)	uguaglianza tra istanze
–	owl:differentFrom	owl:differentFrom	disuguaglianza tra istanze
–	owl:AllDifferent... owl:distinctMembers	owl:AllDifferent... owl:distinctMembers	disuguaglianza tra istanze di una lista
attribute DomainType Interface or View	owl:ObjectProperty	owl:ObjectProperty	relazione tra istanze
attribute DomainType DataType	owl:DatatypeProperty	owl:DatatypeProperty	relazione tra un'istanza e un valore
attribute	rdfs:domain	rdfs:domain (solo classi)	soggetto della proprietà
attribute Domain	rdfs:range	rdfs:range (solo classi)	oggetto della proprietà
–	rdfs:subPropertyOf	rdfs:subPropertyOf	gerarchia tra proprietà
–	owl:equivalentProperty	owl:equivalentProperty	equivalenza estensionale tra proprietà
relationship...inverse	owl:inverseOf (Object properties)	owl:inverseOf (Object properties)	proprietà inversa
relationship...inverse	owl:SymmetricProperty (Object properties)	owl:SymmetricProperty (Object properties)	proprietà simmetrica (uguale all'inversa)
key (singola)	owl:FunctionalProperty + owl:Cardinality (1)	owl:FunctionalProperty + owl:Cardinality (1)	proprietà funzionale restrizione di cardinalità
key (multipla)	sew:Key	sew:Key	concetto di chiave (ODL$_{I^3}$)
foreign key... references	rdfs:subPropertyOf KEY Properties	rdfs:subPropertyOf KEY Properties	concetto di chiave estera (ODL$_{I^3}$)
–	owl:TransitiveProperty (Object properties)	owl:TransitiveProperty (Object properties)	proprietà transitiva
*	owl:minCardinality = 0 owl:maxCardinality = 1	owl:minCardinality = 0 owl:maxCardinality = 1	attributo opzionale

Tabella 5.3
Confronto tra costrutti ODL$_{I^3}$ e OWL

Costrutti ODL$_{I^3}$	**Costrutti OWL DL**	**Costrutti OWL Lite**	**Interpretazione Logica**
attribute set forall	owl:allValuesFrom (classi o data range)	owl:allValuesFrom (classi)	quantificazione universale
exists	owl:someValuesFrom (classi o data range)	owl:someValuesFrom (classi)	quantificazione esistenziale
–	owl:hasValue	–	quantificazione universale di istanze
–	owl:minCardinality	owl:minCardinality(0,1)	cardinalità minima
–	owl:maxCardinality	owl:maxCardinality(0,1)	cardinalità massima
FixedArraySize	owl:Cardinality	owl:Cardinality(0,1)	cardinalità minima e massima
rule	owl:Restriction... owl:onProperty	owl:Restriction... owl:onProperty	restrizioni sulle proprietà
mapping rule	URI reference	URI reference	relazioni tra schema integrato e sorgenti locali
– – – – – –	owl:AnnotationProperty owl:versionInfo rdfs:label rdfs:comment rdfs:seeAlso rdfs:isDefinedBy	owl:AnnotationProperty owl:versionInfo rdfs:label rdfs:comment rdfs:seeAlso rdfs:isDefinedBy	Proprietà per l'annotazione di documenti definite dal Dublin Core metadataset
–	owl:Ontology... owl:imports	owl:Ontology... owl:imports	importa un'ontologia (transitiva)
–	owl:Ontology... owl:priorVersion	owl:Ontology... owl:priorVersion	riferimento alla versione precedente di un'ontologia
–	owl:Ontology... owl:backwardCompatibleWith	owl:Ontology... owl:backwardCompatibleWith	riferimento ad una versione compatibile di una ontologia
–	owl:Ontology... incompatibleWith	owl:Ontology... incompatibleWith	riferimento ad una versione incompatibile di un'ontologia

Tabella 5.4
Confronto tra costrutti ODL$_{I^3}$ e OWL

Costrutti ODL$_{I^3}$	**Costrutti OWL DL**	**Costrutti OWL Lite**	**Interpretazione Logica**
–	owl:DeprecatedClass	owl:DeprecatedClass	classe che sarà modificata in futuro
–	owl:DeprecatedProperty	owl:DeprecatedProperty	proprietà che sarà modificata in futuro
wnAnnotation	sew:lemmaValue sew:lemmaSyntacticCategory sew:lemmaSenseNumber sew:nodeUri sew:ontoVersionInfo sew:ontoCreator	– – – – – –	Annotazioni terminologiche di Wordnet (ODL$_{I^3}$)

Tabella 5.5
Confronto tra costrutti ODL$_{I^3}$ e OWL

6. SPECIFICHE DEL TRADUTTORE PER ONTOLOGIE ODL$_{I^3}$

In questo capitolo viene descritta una possibile traduzione dal linguaggio OWL al linguaggio ODL$_{I^3}$ utilizzata per la realizzazione di un wrapper impiegato all'interno del sistema MOMIS. La traduzione proposta viene effettuata con l'intento di avere una trasposizione il più possibile bidirezionale, l'ontologia OWL e l'ontologia ODL$_{I^3}$ risultante devono infatti esprimere gli stessi concetti. Per una traduzione bidirezionale, la presenza delle seguenti informazioni deve essere garantita: architettura dell'ontologia, nome degli elementi e loro struttura, proprietà e relazioni che legano gli elementi tra loro. OWL fornisce tre sottolinguaggi con espressività incrementale progettati per la descrizione di ontologie. Partendo dal meno espressivo al più espressivo, i tre sottolinguaggi di OWL sono:

- *OWL Lite* : che permette di descrivere ontologie nelle quali siano definite principalmente classificazioni gerarchiche e semplici vincoli sulle proprietà;
- *OWL DL* : progettato per avere massima espressività e contemporaneamente garantire che tutte le computazioni abbiano un risultato e che tutte terminino in un tempo finito;
- *OWL Full* : che permette di avere massima espressività e la libertà sintattica di RDF, senza fornire nessuna garanzia dal punto di vista computazionale.

Date le caratteristiche dei tre sottolinguaggi di OWL, per ontologie espresse in OWL Lite e in OWL DL è garantita la traduzione in linguaggio ODL$_{I^3}$. Per ontologie descritte nel sottolinguaggio OWL Full, non è invece possibile assicurare che tutti i concetti definiti siano tradotti fedelmente. OWL Full permette di trattare classi come istanze e viceversa, inoltre permette di ridefinire le caratteristiche dei costrutti di OWL e RDF. Queste caratteristiche non possono supportate da nessun ragionatore ed essere espresse efficacemente nel linguaggio ODL$_{I^3}$. La trasposizione è comunque garantita per ontologie OWL Full

contenenti le proprietà di ODL$_{I^3}$ non esprimibili nel sottolinguaggio OWL DL, come la definizione di tipi di dati *range*, la dichiarazione dei *Container Type* RDF e la dichiarazione di regole d'integrità. In questo modo un'ontologia espressa tramite il linguaggio ODL$_{I^3}$ può essere descritta utilizzando anche alcuni costrutti OWL Full e quindi essere tradotta nuovamente in linguaggio ODL$_{I^3}$.

6.1 Traduzione dei tipi di dato

Nela traduzione dal linguaggio OWL linguaggio ODL$_{I^3}$ la prima cosa da considerare è la traduzione dei tipi di dato. I tipi di dato utilizzabili con OWL si possono suddividere in tre categorie:

- tipi di dato RDF;
- tipi di dato range;
- tipi di dato enumerati.

6.1.1 Traduzione dei tipi di dato RDF

OWL utilizza lo schema per i tipi di dato di RDF, il quale fornisce un meccanismo per il riferimento ai tipi di dato XML Schema (*XML Schema datatypes*) Tra i tipi di dato XML Schema utilizzati da OWL e i tipi atomici di base o *BaseType* di ODL$_{I^3}$, non è possibile ottenere una traduzione uno ad uno, la tabella [6.1] mostra la corrispondenza tra i tipi di dato XML Schema e i tipi semplici di base ODL$_{I^3}$.

Tipo OWL	Esempio	Tipo ODL$_{I^3}$
xsd:boolean	true, false	boolean
xsd:unsignedByte	0, 126	char
xsd:byte	-1, 0, 126, +100	char
xsd:string	example string	string
xsd:normalizedString	example string	string
xsd:token	example string	string
xsd:language	en-GB, en-US, fr	string
xsd:NMTOKEN	US UK, Brsil Canada, Mexique	string
xsd:Name	shipTo	string
xsd:NCName	USAddress	string
xsd:decimal	-1.23, 0, 123.4, 1000.00	float
xsd:float	-1E4, 12.78e-2, 12	float
xsd:double	1267.43233E12, -1234.46e-14	double
xsd:integer	-1, 0, 1267896754, +100000	int
xsd:positiveInteger	1, 1267896754, +100000	int
xsd:nonPositiveInteger	-1, 0, -1267896754, -100000	int
xsd:negativeInteger	-1, -1267896754, -100000	int
xsd:nonNegativeInteger	1, 0, 1267896754, +100000	int
xsd:short	-1, 0, 12678, +10000	short
xsd:long	+922337203685477, -720368547758077	long
xsd:unsignedInt	1267896754, 100000	unsigned int
xsd:unsignedShort	12678, 64535	unsigned short
xsd:unsignedLong	0, 12678967543233, 100000	unsigned long

Tabella 6.1
Corrispondenza tra tipi di dato XML Schema e ODL$_{I^3}$

Tipo OWL	Esempio	Tipo ODL$_{I^3}$
xsd:date	2004-10-27	date
xsd:dateTime	2004-10-27T11:20:00.000-05:00	timestamp
xsd:time	13:20:00-05:00	timestamp
xsd:gYearMonth	2004-10	date
xsd:gMonthDay	10-27	date
xsd:gYear	2004	date
xsd:gDay	27	date
xsd:gMonth	10	date
xsd:hexBinary	0FB7	string
xsd:base64Binary	GpM7	string
xsd:anyURI	http://www.example.org/exontology	string
xsd:anyType	true, 1267896754, 2004-10-27	any

Tabella 6.2
Corrispondenza tra tipi di dato XML Schema e ODL$_{I^3}$

Come si può notare dalla tabella i tipi di base *char*, *string*, *float*, *int*, *date* e *timestamp* del linguaggio ODL$_{I^3}$, mappano più tipi di dato XML Schema.

6.1.2 Traduzione dei tipi di dato Range

XML Schema non prevede un tipo di dato per la specificazione di un range di interi ed il linguaggio OWL non fornisce nessun costrutto per la rappresentazione di questo concetto. I tipi di dato range sono quindi tipi XML Schema derivati dal tipo integer in cui le proprietà *minInclusive* e *maxInclusive* sono settate ai limiti inferiore e superiore del range. Ecco un esempio di tipo range *range-1-100* che può avere valori interi compresi tra uno e cento:

```
<xsd:simpleType name="range-1-100">
  <xsd:restriction base="integer">
    <xsd:minInclusive value="1"/>
```

```
    <xsd:maxInclusive value="100"/>
  </xsd:restriction>
</xsd:simpleType>
```

La definizione di nuovi tipi di dato XML Schema è consentita solo per il sottolinguaggio OWL Full e non per i sottolinguaggi OWL DL e OWL Lite. In OWL DL il tipo corrispondente all'attributo range ODL_{I^3} è una *datatype property* che può assumere valori interi, vincolata ad una lista enumerata di valori compresi tra gli estremi inferiore e superiore del range. Sia il tipo di dato XML Schema derivato di OWL Full che la lista enumerata di OWL DL, vengono tradotti nel linguaggio ODL_{I^3} attraverso il costrutto *range*:

```
range 1,100 number;
```

l'attributo numero può assumere tutti i valori interi compreso tra l'estremo inferiore 1 e l'estremo superiore 100.

6.1.3 Traduzione dei tipi enumerati

Nel linguaggio OWL i tipi di dato enumerati sono definiti attraverso l'utilizzo di una lista RDF. Il tipo di dato enumerato tennisGameScore viene rappresentato in OWL nel seguente modo:

```
<owl:DatatypeProperty rdf:ID="tennisGameScore">
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf>
        <rdf:List>
          <rdf:first rdf:datatype="&xsd;integer">0</rdf:first>
          <rdf:rest>
            <rdf:List>
              <rdf:first rdf:datatype="&xsd;integer">15</rdf:first>
              <rdf:rest>
                <rdf:List>
                  <rdf:first rdf:datatype="&xsd;integer">30
                  </rdf:first>
```

```
                    <rdf:rest>
                      <rdf:List>
                        <rdf:first rdf:datatype="&xsd;integer">40
                        </rdf:first>
                        <rdf:rest rdf:resource="&rdf;nil" />
                      </rdf:List>
                    </rdf:rest>
                  </rdf:List>
                </rdf:rest>
              </rdf:List>
            </rdf:rest>
          </rdf:List>
        </owl:oneOf>
      </owl:DataRange>
    </rdfs:range>
</owl:DatatypeProperty>
```

I tipi enumerazione (*EnumType*) del linguaggio ODL$_{I^3}$ restringono il dominio di un *SimpleType* ad un elenco di valori possibili. Come si può notare la descrizione in ODL$_{I^3}$ dei tipi enumerati è molto più compatta rispetto al linguaggio OWL:

```
enum tennisGameScore {0 , 15 , 30 , 40};
```

6.2 Traduzione dei Container Type

In questa sezione verranno descritte le traduzioni proposte per i tipi RDF *Container Type* del sottolinguaggio OWL Full e per i tipi corrispondenti definiti nei sottolinguaggi OWL DL e OWL Lite, in linguaggio ODL$_{I^3}$.

6.2.1 Traduzione di Alt, Seq e Bag

Per la definizione di liste di elementi in OWL si possono utilizzare i tipi RDF *Container Type*, i costruttori corrispondenti sono i seguenti:

- *rdf:Bag*: collezione di elementi non ordinati;

- *rdf:Alt*: collezione di elementi non ordinati che non contiene duplicati;
- *rdf:Seq*: collezione ordinata di elementi non duplicati.

Nel linguaggio ODL$_{I^3}$, per rappresentare collezioni di dati semplici, vengono utilizzati i tipi collezione (o *CollectionType*). Le collezioni ODL$_{I^3}$ possono essere di tre tipi differenti:

- *bag*: collezione di elementi non ordinati;
- *set*: collezione di elementi non ordinati che non contiene duplicati;
- *list*: collezione ordinata di elementi non duplicati.

Il tipo *rdf:Bag* di OWL verrà quindi rappresentato in ODL$_{I^3}$ con il tipo *bag*, il tipo *rdf:Alt* di OWL sarà tradotto in ODL$_{I^3}$ con il tipo *set* e il tipo OWL *rdf:Seq* con il tipo ODL$_{I^3}$ *list*. Nei sottolingaggi OWL DL e OWL Lite i *Container Type* RDF non possono essere utilizzati, occorre quindi creare i tipi corrispondenti a quelli RDF in un ontologia OWL contenente le definizioni dei tipi, tutte le definizioni saranno poi importate dalle ontologie create per la traduzione. I costruttori corrispondenti ai *Container Type* RDF nelle ontologie OWL DL e OWL Lite saranno:

- *sew:Bag*;
- *sew:Alt*;
- *sew:Seq*.

Nell'esempio seguente viene presentata la traduzione di una proprietà del tipo *sew:Seq* definita per un ontologia OWL DL:

```
<owl:Class rdf:ID="Report.type_Seq">
  <rdfs:subClassOf rdf:resource="&sew;Seq"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="Report.type">
  <rdfs:domain rdf:resource="#Report"/>
```

```
  <rdfs:range rdf:resource="#Report.type_Seq"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="Report.type_Seq_member_of">
  <rdfs:domain rdf:resource="Report.type_Seq"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

In linguaggio ODL$_{I^3}$ l'esempio viene tradotto con un attributo (*Collection Type*) di tipo list:

```
interface Report( ){
      attribute list <string> type *;
    }
```

6.2.2 Traduzione di Seq con restrizioni sulla cardinalità

In OWL, attraverso il costrutto *rdf:Seq* (*sew:Seq* per i sottolinguaggi OWL DL e OWL Lite), viene descritta una collezione ordinata di elementi. Se la proprietà che ha come oggetto il *Container Type Seq* ha una restrizione sulla cardinalità del tipo *owl:cardinality*, viene descritta una collezione ordinata di elementi non duplicati con cardinalità fissa. In ODL$_{I^3}$, una collezione ordinata con un numero fisso di elementi, viene rappresentata attraverso il tipo collezione *ArrayType*. La traduzione di una proprietà OWL *Seq* con restrizione di cardinalità fissa sarà quindi la seguente:

```
interface Book( ){
      attribute array <string,5> keywords;
    }
```

6.3 Traduzione delle classi

Nel linguaggio OWL una classe è identificata univocamente dal suo nome (*rdf:ID*) rappresentato come un riferimento URI, le classi nel linguaggio ODL$_{I^3}$ vengono chiamate interfacce (*interface*) ed anche il nome di un'interfaccia in ODL$_{I^3}$ è l'identificatore unico dell'oggetto (ObjectID) all'interno di uno schema. Sia in OWL che in ODL$_{I^3}$ è ammessa

l'ereditarietà multipla. Il seguente esempio rappresenta una classe descritta in linguaggio OWL:

```
  <owl:Class rdf:ID="Report">
    <rdfs:subClassOf rdf:resource="#Reference"/>
  </owl:Class>
  <owl:DatatypeProperty rdf:ID="title">
    <rdfs:domain rdf:resource="#Report"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:ID="institution">
    <rdfs:domain rdf:resource="#Report"/>
    <rdfs:range rdf:resource="#Institution"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="date">
    <rdfs:domain rdf:resource="#Report"/>
    <rdfs:range rdf:resource="#Date"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="number">
    <rdfs:domain rdf:resource="#Report"/>
    <rdfs:range rdf:resource="&xsd;int"/>
  </owl:DatatypeProperty>
```

la traduzione della classe in interfaccia ODL_{I^3} è la seguente:

```
interface Report : Reference ( ){
  attribute string title;
  attribute Institution institution;
  attribute Date date;
  attribute int number;
};
```

6.3.1 Traduzione delle proprietà Extent e Persistent

Se un ontologia OWL deriva dalla traduzione di una ontologia ODL$_{I^3}$, ogni classe presente avrà una restrizione sui valori delle proprietà *extent* e *persistent*, che definiscono rispettivamente l'insieme delle istanze di una classe all'interno di un Database e il tempo di vita di un oggetto. La classe OWL dell'esempio seguente descrive la classe `CS_Person` appartenente alla sorgente `Computer_Science` e specifica i valori per le proprietà *extent* e *persistent*:

```
<owl:Class rdf:ID="#CS_Person">
  <rdfs:subClassOf rdf:resource="#Computer_Science"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#CS_Person.extent"/>
      <owl:hasValue rdf:datatype="&xsd;string">Persons
      </owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#CS_Person.persistent"/>
      <owl:hasValue rdf:datatype="&xsd;boolean">true
      </owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

La traduzione della classe in linguaggio ODL$_{I^3}$ è la seguente:

```
interface CS_Person persistent : Computer_Science
(extent Persons)
{ ...
};
```

6.3.2 Traduzione di Key e ForeignKey

Un ontologia OWL derivante dalla traduzione di una ontologia ODL_{I^3}, può avere proprietà che esprimono i concetti di *Key* e *ForeignKey*. Il linguaggio OWL non prevede nessun costrutto per la definizione dei concetti di chiave, per cui le proprietà *Key* e *ForeignKey* sono definite all'interno di un ontologia OWL contenente le definizioni dei tipi. La possibilità di specificazione delle chiavi in ODL_{I^3} deriva dall'integrazione nello schema di database ad oggetti e relazionali. In ODL_{I^3} la definizione delle keys e foreign keys è sempre contenuta nella *PropertyList* dell'interfaccia:

```
interface Professor persistent : Computer_Science
(extent Professors
 keys faculty_id, sec_no)
{
attribute string name;
attribute unsigned short faculty_id[6];
attribute long sec_no[10] ;
attribute Address address;
attribute Department department;
};
```

Nell'esempio l'interfaccia `Professor` ha una chiave composita formata dai due attributi `faculty_id` e `sec_no`.

6.4 Traduzione delle proprietà

Le proprietà OWL possono essere di due tipi principali: *object properties* o *datatype properties*. Le *object properties* hanno come oggetto della proprietà una classe, rappresentano quindi una relazioni tra due classi, mentre le *datatype properties* hanno come oggetto della proprietà un tipo di dato e rappresentano una relazione tra una classe e un insieme di valori. Gli attributi di una interfaccia ODL_{I^3} possono essere attributi di tipo semplice o complesso: gli attributi semplici sono attributi di tipo valore (*ValueType*), gli attributi complessi sono invece di tipo *interface*. Le *object properties* di OWL vengono tradotte in

ODL_{I^3} con attributi complessi, le *datatype properties* con attributi di tipo valore atomici di base (*BaseType*), secondo le corrispondenze indicate nella tabella [6.1]. Alcune *object properties* vengono tradotte con i tipi collezione *CollectionType*(vedi paragrafo [6.2]) o attraverso tipi costrutti *ConstrType* (vedi paragrafo [6.5]). Il seguente esempio rappresenta in linguaggio OWL le proprietà di una classe:

```
<owl:Class rdf:ID="Book"/>
<owl:DatatypeProperty rdf:ID="title">
  <rdfs:domain rdf:resource="#Book"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="author">
  <rdfs:domain rdf:resource="#Book"/>
  <rdfs:range rdf:resource="#Author"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="date">
  <rdfs:domain rdf:resource="#Book"/>
  <rdfs:range rdf:resource="#Date"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="pages">
  <rdfs:domain rdf:resource="#Book"/>
  <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
```

la traduzione dell'esempio in linguaggio ODL_{I^3} è la seguente:

```
interface Book ( ){
  attribute string title;
  attribute Author author;
  attribute Date date;
  attribute int pages;
};
```

Le *datatype properties* OWL `title` e `pages` vengono tradotte in ODL$_{I^3}$ con tipi di base, le *object properties* `author` e `date` vengono rappresentate in ODL$_{I^3}$ attraverso attributi complessi.

6.4.1 Cardinalità delle proprietà

le proprietà, nel linguaggio OWL, hanno una cardinalità che va da zero ad infinito, per effettuare restrizioni sulle cardinalità delle proprietà si utilizzano i costrutti *owl:maxCardinality*, *owl:minCardinality* e *owl:cardinality*. In ODL$_{I^3}$ la cardinalità degli attributi è di default uno, postponendo un asterisco (*) alla dichiarazione di un attributo si indica che l'attributo è opzionale, si definisce quindi una cardinalità minima uguale a zero e una cardinalità massima uguale a uno. Una cardinalità maggiore di uno viene specificata indicandola tra parentesi quadre. Se in OWL una proprietà ha cardinalità minima e massima uguali ad uno, per l'attributo ODL$_{I^3}$ corrispondente non sarà specificata nessuna cardinalità. Nel seguente esempio la cardinalità della proprietà `author` viene vincolata ad uno utilizzando il costrutto *owl:cardinality* che definisce cardinalità minima e massima uguali:

```
<owl:Class rdf:ID="Book">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#author"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

la corrispondente traduzione in linguaggio ODL$_{I^3}$ è la seguente:

```
interface Book ( ){
  attribute Author author;
};
```

Se una proprietà OWL ha una restrizione sulla cardinalità massima uguale ad uno e nessuna restrizione sulla cardinalità minima (che di default è zero):

```
<owl:Class rdf:ID="Book">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#author"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
      1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

La proprietà viene tradotta in linguaggio ODL$_{I^3}$ attraverso un attributo dichiarato come opzionale (*):

```
interface Book ( ){
  attribute Author author*;
};
```

Se le cardinalità minima e massima di una proprietà OWL vengono ristrette ad un valore maggiore di uno, l'attributo ODL$_{I^3}$ corrispondente avrà la cardinalità espressa tra parentesi quadre. Nell'esempio seguente le cardinalità minima e massima della proprietà `author` vengono vincolate a 3 utilizzando il costrutto *owl:cardinality*:

```
<owl:Class rdf:ID="Book">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#author"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">3
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Ecco la traduzione in linguaggio ODL$_{I^3}$:

```
interface Book ( ){
```

```
  attribute Author author[3];
};
```

Se una proprietà OWL non ha nessuna restrizione di cardinalità essa viene tradotta in linguaggio ODL$_{I^3}$ attraverso il tipo collezione *Set* che può rappresentare una cardinalità minima uguale a zero ed una cardinalità massima infinita:

```
interface Book ( ){
  attribute set <Author> author *;
};
```

6.5 Traduzione di unioni e strutture di valori

In OWL, tramite il costrutto *owl:unionOf*, è possibile definire una proprietà che ha come oggetto un insieme di più classi o range di valori. Lo stesso concetto viene espresso nel linguaggio ODL$_{I^3}$ tramite il tipo unione (*UnionType*), il quale viene utilizzato per scegliere, in base al valore di una variabile in una clausola *switch*, il tipo di appartenenza di un attibuto. Nel seguente esempio OWL il valore della proprietà `TypeNumber` può essere una stringa o un intero:

```
<owl:DatatypeProperty rdf:ID="TypeNumber">
  <rdfs:range>
    <owl:DataRange>
      <owl:unionOf rdf:parseType="Collection">
        <rdfs:Datatype rdf:about="&xsd;string"/>
        <rdfs:Datatype rdf:about="&xsd;int"/>
      </owl:unionOf>
    </owl:DataRange>
  </rdfs:range>
</owl:DatatypeProperty>
```

Lo stesso concetto viene espresso in linguaggio ODL$_{I^3}$ utilizzano il costrutto *union*:

```
union TypeNumber switch(val) {
```

```
case 1: string;
case 2: int;
};
```

Un ontologia OWL derivante dalla traduzione di una ontologia ODL$_{I^3}$, può avere classi e serie di *datatype properties* che esprimono i concetti di strutture di tipi valore. Una struttura di tipi valore viene descritta in ODL$_{I^3}$ utilizzando il tipo struttura. Il seguente esempio rappresenta una struttura di tipi valore in OWL formata da una classe e da una serie di *datatype properties*:

```
<owl:Class rdf:ID="structType"/>
<owl:DatatypeProperty rdf:ID="a">
  <rdfs:domain rdf:resource="structType"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="b">
  <rdfs:domain rdf:resource="structType"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="c">
  <rdfs:domain rdf:resource="structType"/>
  <rdfs:range rdf:resource="&xsd;unsignedLong"/>
</owl:DatatypeProperty>
```

La trasposizione in linguaggio ODL$_{I^3}$ della struttura sarà espressa dal seguente codice:

```
typedef struct {
string a;
boolean b;
unsignedlong c;
} structType ;
```

6.6 Traduzione di Relazioni inverse

In OWL, per ogni *object property*, è possibile definire una relazione inversa. La relazione inversa viene dichiarata utilizzando il costrutto *owl:inverseOf*. Nel linguaggio ODL$_{I^3}$ le relazioni inverse vengono definite come *Relationship*. Le relazioni definite all'interno di un interfaccia ODL$_{I^3}$ sono attributi complessi: esse possono avere come oggetto solo interfacce ed è inoltre possibile specificare la relazione inversa. Ecco un esempio di definizione di relazione inversa in OWL:

```
<owl:ObjectProperty rdf:ID="author">
<rdfs:domain rdf:resource="#Book"/>
    <rdfs:range rdf:resource="#Author"/>
  <owl:inverseOf rdf:resource="#isauthor"/>
</owl:ObjectProperty>
```

La proprietà `author` che ha come soggetto `Book` e come oggetto `Author` viene dichiarata come inversa della proprietà `isauthor` che ha come soggetto `Author` e come oggetto `Book` . La traduzione in linguaggio ODL$_{I^3}$ sarà la seguente:

```
interface Book ( ) {
relationship set<Author> author inverse Author::isauthor;
};
```

6.7 Traduzione delle Restrizioni sui valori

Se una classe di un ontologia OWL ha restrizioni multiple sui valori delle proprietà, questi vincoli possono essere espressi in linguaggio ODL$_{I^3}$ attraverso le *Rule*, tramite cui è possibile definire vincoli di integrità per le istanze dell'interfaccia. Nell'esempio viene presentata una classe OWL con restrizioni sui valori delle proprietà:

```
<owl:Class rdf:about="Book_category_8">
  <rdfs:subClassOf rdf:resource="#Book"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#category"/>
```

```
      <owl:hasValue rdf:datatype="&xsd;Int">8</owl:hasValue>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#price"/>
      <owl:hasValue rdf:datatype="&xsd;Int">25</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Queste restrizioni possono essere descritte in ODL$_{I^3}$ tramite una regola d'integrità:

```
interface Book ( ) {
attribute string title;
attribute range {1, 10} category;
attribute int price;
};
rule rule1 forall X in Book: X.category = 8
then X.price = 25;
```

6.8 Traduzione delle Relazioni terminologiche (Thesaurus relations)

In un ontologia OWL derivante facente parte del sistema MOMIS, è possibile che siano presenti relazioni terminologiche di diversi tipi che possono intercorrere tra nomi di classi, tra proprietà e miste tra nomi di classi e proprietà. Le relazioni terminologiche possono essere di quattro tipi:

- Ipernimia (BT):
- Iponimia (NT):
- Sinonimia (SYN):
- Associazione (RT):

6.8.1 Relazioni di tipo estensionale

Le relazioni terminologiche di tipo estensionale esprimono un legame tra istanze.
Una relazione estensionale di (*BT*) o iponimia (*NT*) tra due proprietà viene espressa in linguaggio OWL utilizzando l'assioma *rdfs:subPropertyOf* il quale dichiara che l'estensione della proprietà soggetto è un sottoinsieme dell'estensione della proprietà oggetto nel seguente modo:

```
<owl:ObjectProperty rdf:about="produceRedWine">
  <rdfs:subropertyOf rdf:resource="#produceWine"/>
</owl:ObjectProperty>
```

Date le due proprietà `produceWine` e `produceRedWine` , il codice dell'esempio OWL implica la seguente relazione:

```
produceWine BT produceRedWine
```

o

```
produceRedWine NT produceWine
```

La relazione estensionale di sinonimia (*SYN*) tra due classi implica una equivalenza tra le estensioni delle due classi e viene espressa in OWL utilizzando l'assioma *owl:equivalentClass*:

```
<owl:Class rdf:about="Book">
  <owl:equivalentClass rdf:resource="#Novel"/>
</owl:Class>
```

Date le due classi `Book` e `Novel` , il codice dell'esempio esprime la relazione:

```
Book SYN Novel
```

In modo analogo al precedente esempio, utilizzando l'assioma *owl:equivalentProperty* è possibile esprimere una relazione estensionale di sinonimia tra due proprietà.

6.8.2 Relazioni di tipo intensionale

Le relazioni terminologiche di tipo intensionale implicano un legame tra due concetti rappresentati dalle classi o dalle proprietà. Questo tipo di relazioni, non può essere espresso tramite i costrutti OWL *rdfs:subClassOf*, *rdfs:subPropertyOf*, *owl:equivalentClass* o *owl:equivalentProperty* che definiscono legami solamente tra le estensioni dei concetti. Per tradurre efficacemente le relazioni terminologiche di tipo intensionale vengono quindi create delle *annotation property* chiamate *Thesaurus Relations*. Le *Thesaurus Relations* fanno parte dell'ontologia OWL e vengono utilizzate per definire le relazioni terminologiche intensionali che intercorrono tra due classi, due proprietà o tra una classe ed una proprietà. Segue un esempio di definizione di relazione terminologica intensionale espressa tramite una *thesaurus relation*:

```
<sew:ThesRelation rdf:ID="ThesRelation_1">
  <sew:RelClass rdf:datatype="&xsd;string">InterfaceRel
  </sew:RelClass>
  <sew:Attribute1 rdf:datatype="&xsd;anyURI">&university;Room
  </sew:Attribute1>
  <sew:RelType rdf:datatype="&xsd;string">rt</sew:RelType>
  <sew:Attribute2 rdf:datatype="&xsd;anyURI">&university;Section
  </sew:Attribute2>
  <sew:ProducerID rdf:datatype="&xsd;int">900</sew:ProducerID>
</sew:ThesRelation>
```

La relazione descritta nell'esempio è una relazione intensionale di associazione (*RT*) tra le due classi (*InterfaceRel*) `Room` e `Section` e sarà tradotta in ODL_{I^3} tramite la relazione seguente: `Room RT Section`

6.9 Traduzione delle Annotazioni (WordNet annotations)

In OWL le annotazioni lessicali rispetto a Wordnet vengono descritte attraverso l'utilizzo di apposite *annotation property* chiamate *WordNetAnnotations*. Un esempio di utilizzo delle annotazioni lessicali è il seguente:

```
<owl:Class rdf:about="Computer_Science.Professor">
  <sew:lemmaValue rdf:datatype="&xsd;string">professor
  </sew:lemmaValue>
  <sew:lemmaSyntacticCategory rdf:datatype="&xsd;int">1
  </sew:lemmaSyntacticCategory>
  <sew:lemmaSenseNumber rdf:datatype="&xsd;int">1
  </sew:lemmaSenseNumber>
</owl:Class>
```

Questa annotazione assegna al nome della classe `Professor` appartenente alla sorgente `ComputerScience` il valore `"professor"`, indicando una categoria sintattica uguale ad uno (corrispondente alla categoria sintattica "nome") ed un numero di senso uguale ad uno (corrispondente al significato: "someone who is a member of the faculty at a college or university"). Il linguaggio ODL$_{I^3}$ esprime le annotazioni lessicali rispetto a WordNet utilizzando il costrutto *WNAnnotation*, l'esempio precedente sarà tradotto in ODL$_{I^3}$ come segue:

```
wnAnnotation ComputerScience.Professor
   lemmaValue="professor",
   lemmaSyntacticCategory=1,
   lemmaSenseNumber=1;
```

6.10 Traduzione delle ontologie importate

Il costrutto *owl:imports* di OWL consente di includere in un'ontologia la definizione di tutte le classi, proprietà e istanze specificate all'interno dell'ontologia importata. L'ontologia che importa altre ontologie viene quindi considerata l'ontologia globale corrispondente ad uno *Schema* ODL$_{I^3}$ che importa le descrizioni relative alle diverse sorgenti. Lo *Schema*, nel linguaggio ODL$_{I^3}$, rappresenta infatti l'oggetto nel quale sorgenti eterogenee vengono integrate. Le sorgenti vengono specificate in OWL come superclassi di tutte le classi appartenenti all'ontologia, il tipo di sorgente (*relational*, *nfrelational*, *semistructured*, *object*,

o *file*) viene specificato come proprietà della superclasse. Nel linguaggio ODL$_{I^3}$ il tipo di sorgente ed il suo nome vengono invece specificati per ogni classe dello *Schema*.

6.11 Esempio di traduzione OWL-ODL$_{I^3}$

Un esempio completo di traduzione di un'ontologia OWL in un'ontologia ODL$_{I^3}$ è reperibile all'indirizzo *http://www.dbgroup.unimo.it/ orsini/esempioOwl-Odli3.pdf*.

7. IMPLEMENTAZIONE DEI TRADUTTORI

Il wrapper che sovrintende ai documenti di tipo OWL, come ogni altro wrapper utilizzato nell'ambito del progetto MOMIS, deve deve implementare due operazioni fondamentali: fornire una descrizione in ODL_{I^3} della sorgente alla quale è connesso e consentire l'esecuzione di query, generate dal query manager, sulla specifica sorgente. La prima funzionalità indicata viene svolta completamente dai due traduttori OWL-ODL_{I^3} e ODL_{I^3}-OWL, utilizzando le tabelle e gli algoritmi di conversione presentati nei capitoli precedenti. In particolare, i traduttori implementati, consentono la conversione di uno schema ODL_{I^3} in un ontologia descritta tramite i sottolinguaggi OWL DL o OWL Full e la conversione di ontologie descritte in OWL Lite o OWL DL in uno schema ODL_{I^3}.

Per l'esecuzione di query su sorgenti OWL, un linguaggio di interrogazione specifico non è ancora stato sviluppato, è però possibile effettuare interrogazioni su ontologie OWL tramite il linguaggio RDQL (RDF Query Language) progettato per ontologie RDF. Si è preferito non implementare l'esecuzione di query locali, per focalizzare maggiormente l'attenzione sull'implementazione dei due traduttori.

Il wrapper è stato implementato utilizzando il linguaggio java, pertanto non necessita di alcuna ricompilazione nel passaggio tra diverse piattaforme.

Di seguito vengono presentati alcuni diagrammi delle classi del sistema MOMIS, utilizzati per la progettazione del software. Nella figura 7.1 viene mostrato il package relativo alla definizione del linguaggio ODL_{I^3}, dalla classe MomisObject che rappresenta un generico oggetto del linguaggio, derivano le sei sottoclassi rappresentanti le diverse categorie di costrutti presenti nel linguaggio. Le sottoclassi sono completamente disgiunte una dall'altra. Per le classi TypeContainer, AttributeInterface, Rule e Type seguono i relativi diagrammi per una descrizione più dettagliata.

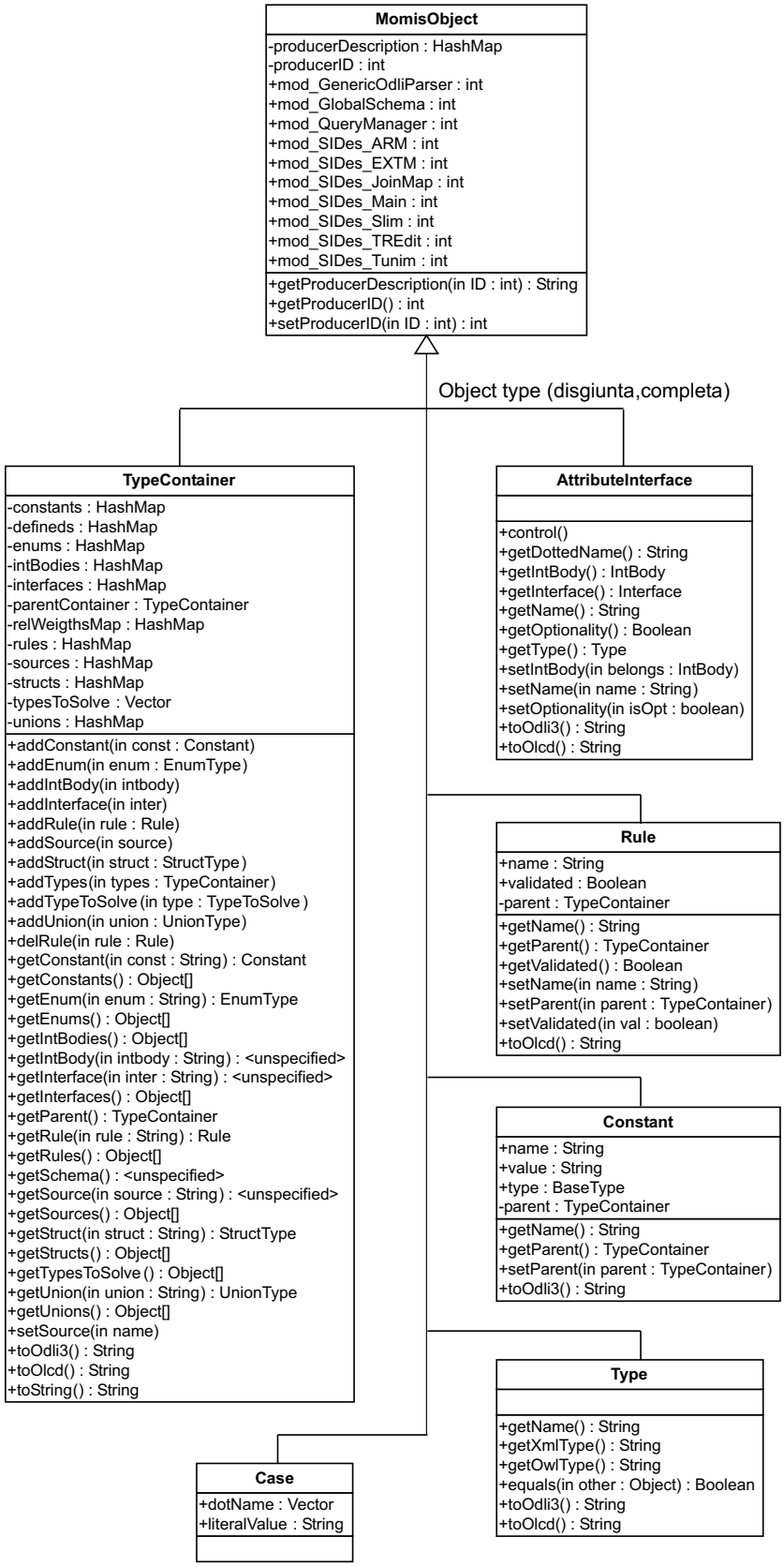

Figura 7.1. Diagramma delle classi: MomisObject

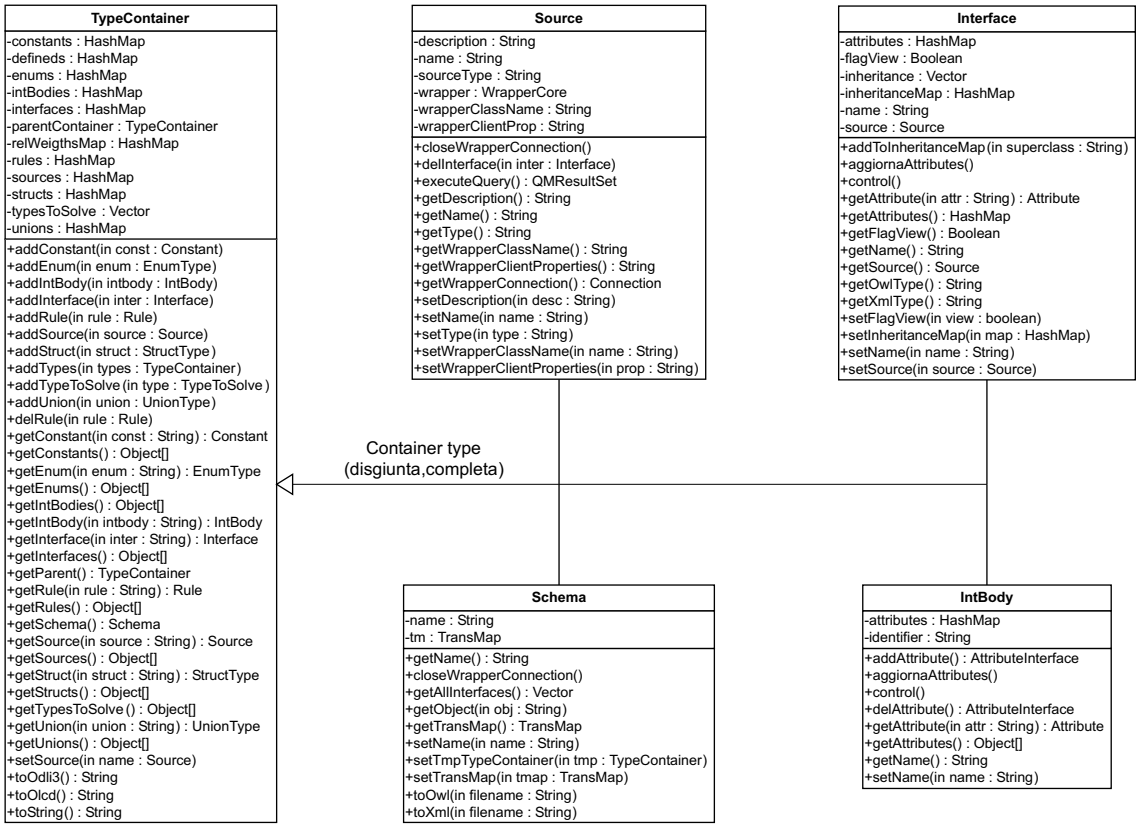

Figura 7.2. Diagramma delle classi: TypeContainer

La figura 7.2 mostra la suddivisione della classe TypeContainer nei quattro diversi tipi di Contenitori del linguaggio ODL$_{I^3}$. La classe Schema rappresenta lo schema di una ontologia in ODL$_{I^3}$, la classe Source rappresenta una sorgente di dati, la classe Interface rappresenta una classe in ODL$_{I^3}$, la classe IntBody rappresenta il corpo di una classe.

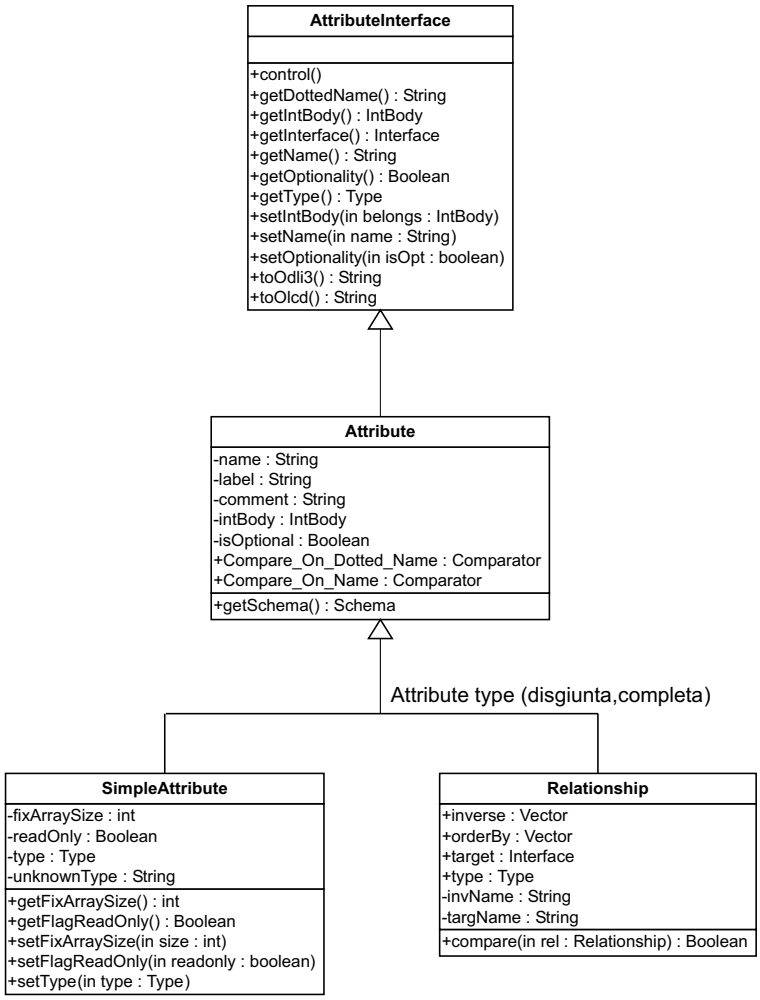

Figura 7.3. Diagramma delle classi: AttributeInterface

Nella figura 7.3 viene visualizzata la classe Attribute, sottoclasse di AttributeInterface e le sue 2 sottoclassi completamente disgiunte: SimpleAttribute che rappresenta gli attributi di tipo semplice e relationship che rappresenta gli attributi di tipo relazione.

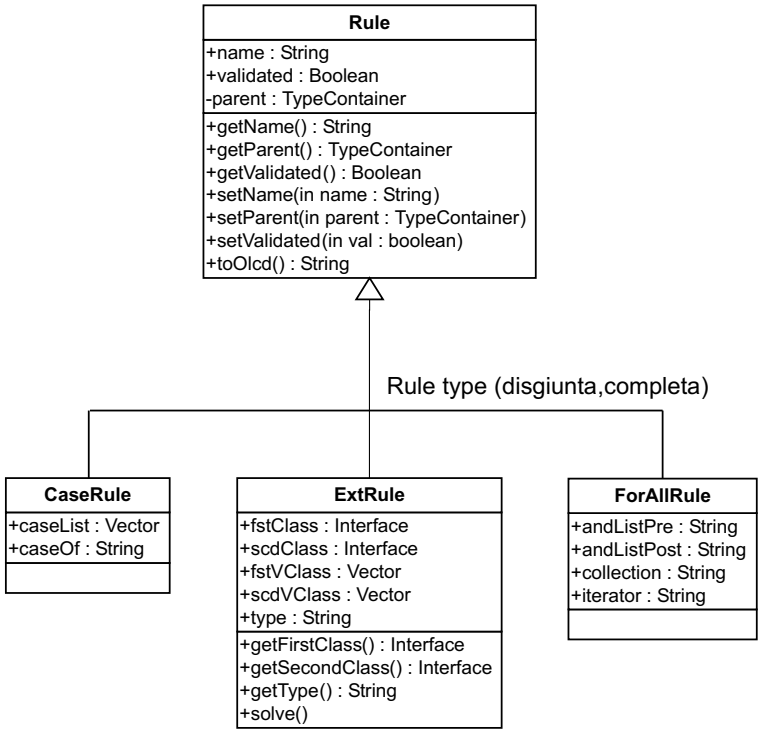

Figura 7.4. Diagramma delle classi: Rule

Il diagramma della figura 7.4 mostra le informazioni relative agli oggetti di tipo Rule del linguaggio ODLI3. La suddivisione in sottoclassi è completamente disgiunta in quanto le sottoclassi rappesentano regole di tipi diversi, rispettivamente regole di tipo Case, regole di tipo Esterno e regole di tipo ForAll.

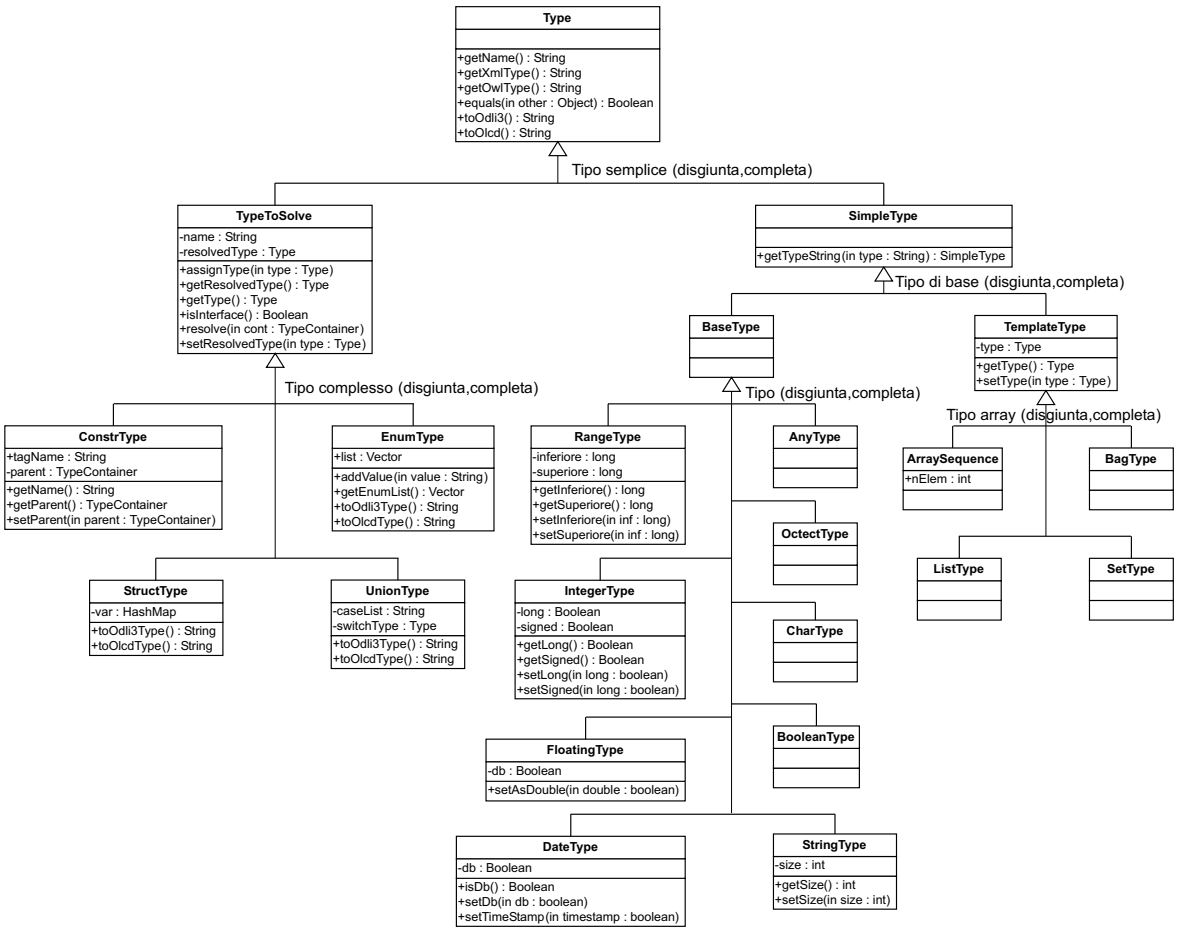

Figura 7.5. Diagramma delle classi: Type

Il diagramma della figura 7.5 mostra le informazioni relative ai tipi che si possono avere nel linguaggio ODL$_{I^3}$. La classe Type è suddivisa nelle due sottoclassi SimpleType che rappresenta i tipi semplici e nella classe TypeToSolve che rappresenta i tipi complessi. La classe TypeToSolve ha poi quattro sottoclassi che definiscono i tipi complessi Costruttore, Struttura, Enumerato e Unione. La classe SimpleType è di nuovo suddivisa in due sottoclassi: BaseType rappresenta i tipi di base di ODL$_{I^3}$, la classe TemplateType definisce i tipi Array.

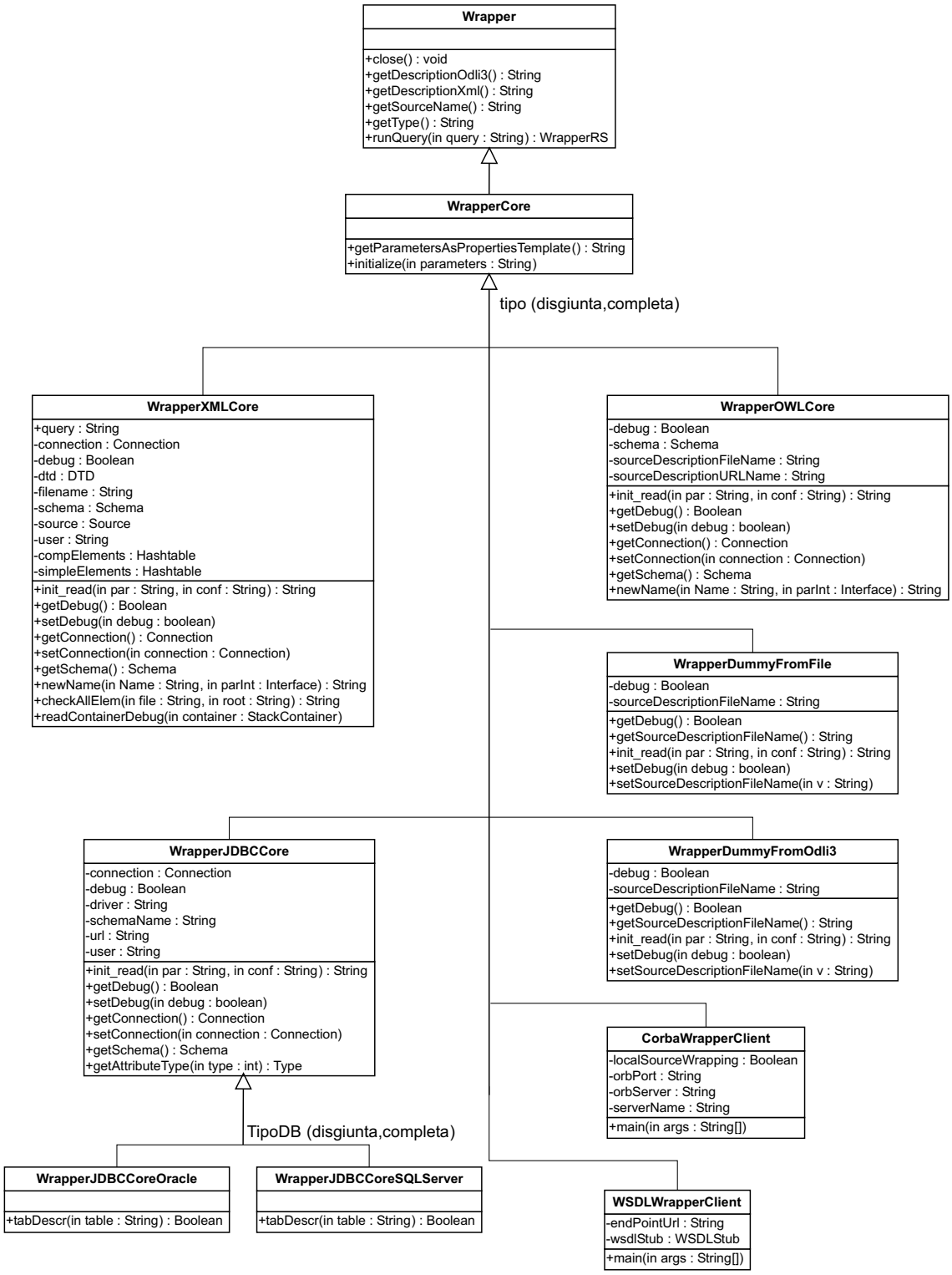

Figura 7.6. Diagramma delle classi: Wrapper

Nel diagramma di figura 7.6 viene mostrato il package relativo all'implementazione dei Wrapper, vengono mostrate le informazioni relative ai traduttori per i diversi linguaggi, la classe WrapperCore è suddivisa nelle sette sottoclassi relative ai linguaggi Xml, Owl, Odli3, Corba e alle sorgenti di dati di tipo File, JDBC, WSDL. La classe rappresentante le sorgenti JDBC è poi suddivisa nelle due sottoclassi relativa a DB di tipo Oracle o SQLServer.

Conclusioni e Lavoro Futuro

In questa tesi è stato studiato un metodo innovativo per la traduzione di ontologie descritte attraverso il linguaggio ODL_{I^3} in linguaggio OWL (sviluppato dal W3C) e la traduzione di ontologie espresse tramite il linguaggio OWL in linguaggio ODL_{I^3}. La tesi si inserisce all'interno del progetto SEWASIE (SEmantic Webs and AgentS in Integrated Economies), che si propone di progettare ed implementare un motore di ricerca semantico che permetta di accedere a sorgenti di dati eterogenee nel Web. Il metodo proposto per la traduzione si basa sull'utilizzo del sistema mediatore MOMIS (Mediator envirOnment for Multiple Information Sources) che è in grado di operare su schemi, descriventi le informazioni presenti in una sorgente locale, solamente dopo che essi sono stati tradotti, da appositi wrapper, in schemi ODL_{I^3}. Dopo un'analisi approfondita delle sintassi del linguaggio OWL (e dei tre sottolinguaggi OWL Lite, OWL DL e OWL Full) e del linguaggio per la mediazione ODL_{I^3} utilizzato da MOMIS, sono state proposte le traduzioni per ontologie da ODL_{I^3} a OWL e da OWL a ODL_{I^3} in modo da permettere l'implementazione del wrapper. Si è quindi realizzato il modulo wrapper che realizza la traduzione di ontologie descritte nel linguaggio ODL_{I^3} specifico del sistema MOMIS in ontologie OWL e la traduzione di ontologie OWL in ontologie ODL_{I^3}. In particolare, visti i differenti livelli di espressività dei sottolinguaggi di OWL, è stata implementata la traduzione di schemi ODL_{I^3} nei sottolinguaggi OWL DL e Full, il sottolinguaggio OWL Lite è infatti decisamente limitato per l'espressione di schemi complessi, esso permette di descrivere ontologie nelle quali siano definite principalmente classificazioni gerarchiche e semplici vincoli sulle proprietà. La traduzione di ontologie OWL in schemi ODL_{I^3} è stata implementata per i sottolinguaggi OWL Lite e OWL DL, in quanto il sottolinguaggio OWL Full è caratterizzato da una grande libertà sintattica e non fornisce nessuna garanzia dal punto di vista computazionale.

Durante l'analisi delle sintassi dei due linguaggi, si sono evidenziate le differenze che li caratterizzano. Il linguaggio utilizzato dal sistema MOMIS è stato creato per la descrizione e l'integrazione di sorgenti di dati eterogenee mentre il linguaggio OWL è stato progettato in modo specifico per la rappresentazione di ontologie nel Web Semantico. Nella traduzione OWL-ODL$_{I^3}$, si sono notate alcune carenze nel linguaggio ODL$_{I^3}$. In particolare, è stata proposta un'estensione al linguaggio per l'espressione dei concetti di disgiunzione e complemento tra le estensioni delle classi, per la dichiarazione di proprietà transitive e per il versioning di ontologie.

Un possible sviluppo futuro del wrapper per ontologie OWL è la realizzazione del modulo per la gestione delle interrogazioni delle sorgenti. Il modulo dovrà effettuare la traduzione delle interrogazioni provenienti dal Query Manager del sistema MOMIS, nel query language specifico per ontologie OWL. Un linguaggio di interrogazione specifico per OWL non è ancora stato sviluppato, è però possibile effettuare interrogazioni su ontologie OWL tramite il linguaggio RDQL (RDF Query Language) progettato per ontologie RDF.

APPENDICE A

SINTASSI ODL$_{I^3}$

120

121

122

123

124

125

126

127

128

129

APPENDICE B

SINTASSI OWL

Ontologies

Un'ontologia OWL può contenere una serie di annotazioni, assiomi(axioms) e fatti(facts)

```
ontology ::= 'Ontology(' [ ontologyID ] { directive } ')'
directive ::= 'Annotation(' ontologyPropertyID ontologyID ')'
   | 'Annotation(' annotationPropertyID URIreference ')'
         | 'Annotation(' annotationPropertyID dataLiteral ')'
         | 'Annotation(' annotationPropertyID individual ')'
         | axiom
         | fact
```

Un ontologia contiene informazioni su classi, proprietà e istanze ognuna delle quali deve essere identificata tramite un riferimento URI

```
datatypeID ::= URIreference
classID ::= URIreference
individualID ::= URIreference
ontologyID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference
annotationPropertyID ::= URIreference
ontologyPropertyID ::= URIreference
```

Molti costrutti OWL utilizzano le annotazioni

```
annotation ::= 'annotation(' annotationPropertyID URIreference ')'
            | 'annotation(' annotationPropertyID dataLiteral ')'
            | 'annotation(' annotationPropertyID individual ')'
```

Facts

Esistono due tipi di fatti nella sintassi OWL. Il primo tipo serve per dare informazioni circa un istanza (individual)

```
fact ::= individual
individual ::= 'Individual(' [ individualID ] { annotation }
{ 'type(' type ')' } { value } ')'
value ::= 'value(' individualvaluedPropertyID individualID ')'
        | 'value(' individualvaluedPropertyID  individual ')'
        | 'value(' datavaluedPropertyID  dataLiteral ')'
```

In OWL Lite, i tipi possono essere classi o restrizioni

```
type ::= classID
       | restriction
```

In OWL DL i tipi possono essere più in generale descrizioni

```
type ::= classID
       | description

dataLiteral ::= typedLiteral | plainLiteral
typedLiteral ::= lexicalForm^^URIreference
plainLiteral ::= lexicalForm | lexicalForm@languageTag
lexicalForm ::= as in RDF, a unicode string in normal form C
languageTag ::= as in RDF, an XML language tag
```

Il secondo tipo di fatti per rendere due istanze uguali o distinte

```
fact ::= 'SameIndividual(' individualID individualID
{individualID} ')'
      | 'DifferentIndividuals(' individualID individualID
      {individualID} ')'
```

OWL Lite: assiomi per le classi

```
axiom ::= 'Class(' classID ['Deprecated'] modality { annotation }
{ super } ')'
modality ::= 'complete' | 'partial'
super ::= classID | restriction
axiom ::= 'EquivalentClasses(' classID classID { classID } ')'
axiom ::= 'Datatype(' datatypeID ['Deprecated']  { annotation } )'
```

OWL Lite: restrizioni

```
restriction ::= 'restriction(' datavaluedPropertyID
dataRestrictionComponent ')'
            | 'restriction(' individualvaluedPropertyID
            individualRestrictionComponent ')'
dataRestrictionComponent ::= 'allValuesFrom(' dataRange ')'
            | 'someValuesFrom(' dataRange ')'
            | cardinality
individualRestrictionComponent ::= 'allValuesFrom(' classID ')'
            | 'someValuesFrom(' classID ')'
            | cardinality
cardinality ::= 'minCardinality(0)' | 'minCardinality(1)'
            | 'maxCardinality(0)' | 'maxCardinality(1)'
            | 'cardinality(0)'    | 'cardinality(1)'
```

OWL Lite: assiomi per le proprietà

```
axiom ::= 'DatatypeProperty(' datavaluedPropertyID ['Deprecated']
{ annotation }
              { 'super(' datavaluedPropertyID ')' } ['Functional']
              { 'domain(' classID' ')' } { 'range(' dataRange ')' } ')'
       | 'ObjectProperty(' individualvaluedPropertyID ['Deprecated']
       { annotation }
              { 'super(' individualvaluedPropertyID ')' }
              [ 'inverseOf(' individualvaluedPropertyID ')' ]
              [ 'Symmetric' ]
              [ 'Functional' | 'InverseFunctional' | 'Functional'
              'InverseFunctional' | 'Transitive' ]
              { 'domain(' classID ')' } { 'range(' classID ')' } ')'
       | 'AnnotationProperty(' annotationPropertyID { annotation } ')'
       | 'OntologyProperty(' ontologyPropertyID { annotation } ')'
dataRange ::= datatypeID | 'rdfs:Literal'
axiom ::= 'EquivalentProperties(' datavaluedPropertyID
datavaluedPropertyID { datavaluedPropertyID } ')'
```

```
        | 'SubPropertyOf(' datavaluedPropertyID  datavaluedPropertyID ')'
        | 'EquivalentProperties(' individualvaluedPropertyID
        individualvaluedPropertyID  { individualvaluedPropertyID } ')'
        | 'SubPropertyOf(' individualvaluedPropertyID
        individualvaluedPropertyID ')'
```

OWL DL: assiomi per le classi

```
axiom ::= 'Class(' classID  ['Deprecated'] modality { annotation }
{ description } ')'
modality ::= 'complete' | 'partial'
axiom ::= 'EnumeratedClass(' classID ['Deprecated'] { annotation }
{ individualID } ')'
axiom ::= 'DisjointClasses(' description description { description } ')'
        | 'EquivalentClasses(' description { description } ')'
        | 'SubClassOf(' description description ')'
axiom ::= 'Datatype(' datatypeID ['Deprecated']  { annotation } )'
```

OWL DL: descrizioni

```
description ::= classID
            | restriction
            | 'unionOf(' { description } ')'
            | 'intersectionOf(' { description } ')'
            | 'complementOf(' description ')'
            | 'oneOf(' { individualID } ')'
```

OWL DL: restrizioni

```
restriction ::= 'restriction(' datavaluedPropertyID
dataRestrictionComponent { dataRestrictionComponent } ')'
            | 'restriction(' individualvaluedPropertyID
individualRestrictionComponent { individualRestrictionComponent } ')'
dataRestrictionComponent ::= 'allValuesFrom(' dataRange ')'
            | 'someValuesFrom(' dataRange ')'
            | 'value(' dataLiteral ')'
```

```
            | cardinality
individualRestrictionComponent ::= 'allValuesFrom(' description ')'
            | 'someValuesFrom(' description ')'
            | 'value(' individualID ')'
            | cardinality
cardinality ::= 'minCardinality(' non-negative-integer ')'
            | 'maxCardinality(' non-negative-integer ')'
            | 'cardinality(' non-negative-integer ')'
dataRange ::= datatypeID | 'rdfs:Literal'
            | 'oneOf(' { dataLiteral } ')'
```

OWL DL: assiomi per le proprietà

```
axiom ::= 'DatatypeProperty(' datavaluedPropertyID ['Deprecated']
{ annotation }
                { 'super(' datavaluedPropertyID ')'} ['Functional']
                { 'domain(' description ')' } { 'range(' dataRange ')'
 } ')'
        | 'ObjectProperty(' individualvaluedPropertyID ['Deprecated']
{ annotation }
                { 'super(' individualvaluedPropertyID ')' }
                [ 'inverseOf(' individualvaluedPropertyID ')' ]
                [ 'Symmetric' ]
                [ 'Functional' | 'InverseFunctional' | 'Functional'
                'InverseFunctional' | 'Transitive' ]
                { 'domain(' description ')' } { 'range(' description ')'
} ')'
        | 'AnnotationProperty(' annotationPropertyID { annotation } ')'
        | 'OntologyProperty(' ontologyPropertyID { annotation } ')'
axiom ::= 'EquivalentProperties(' datavaluedPropertyID
datavaluedPropertyID { datavaluedPropertyID } ')'
        | 'SubPropertyOf(' datavaluedPropertyID  datavaluedPropertyID
')'
        | 'EquivalentProperties(' individualvaluedPropertyID
        individualvaluedPropertyID { individualvaluedPropertyID } ')'
        | 'SubPropertyOf(' individualvaluedPropertyID
```

```
individualvaluedPropertyID ')'
```

Bibliografia

[1] 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.

[2] 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.

[3] S. Bergamaschi, S. Castano, D. Beneventano, and M. Vincini. Semantic integration of heterogeneous information sources. *Special Issue on Intelligent Information Integration, Data and Knowledge Engineering*, 36(1):215–249, 2001.

[4] J. Hendler, O. Lassila, and T. Berners-Lee. The semantic web, a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, May 2001.

[5] Y. Sure. On-to-knowledge – ontology based knowledge management tools and their application. *German Journal Kuenstliche Intelligenz, Special Issue on Knowledge Management*, (1/02), 2002.

[6] Tim Bray, C. M. Sperberg-McQueen Jean Paoli, and Eve Maler. Extensible markup language (xml) 1.0 (second edition), 2000. Available at http://www.w3.org/XML.

[7] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. Xml schema part 1: Structures second edition, 2004. W3C Recommendation, available at http://www.w3.org/TR/xmlschema-1/.

[8] Paul V. Biron and Ashok Malhotra. Xml schema part 2: Datatypes, 2001. W3C Recommendation, available at http://www.w3.org/TR/xmlschema-2/.

[9] Dave Beckett. Rdf/xml syntax specification (revised), 2004. W3C Recommendation, available at http://www.w3.org/TR/rdf-syntax-grammar/.

[10] Graham Klyne and Jeremy J. Carroll. Resource description framework (rdf): Concepts and abstract syntax, 2004. W3C Recommendation, available at http://www.w3.org/TR/rdf-concepts/.

[11] Dan Brickley and R. V. Guha. Rdf vocabulary description language 1.0: Rdf schema, 2004. W3C Recommendation, available at http://www.w3.org/TR/rdf-schema/.

[12] Patrick Hayes. Rdf semantics, 2004. W3C Recommendation, available at http://www.w3.org/TR/rdf-mt/.

[13] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. Owl web ontology language guide, 2004. W3C Recommendation, available at http://www.w3.org/TR/owl-guide/.

[14] Mike Dean and Guus Schreiber. Owl web ontology language reference, 2004. W3C Recommendation, available at http://www.w3.org/TR/owl-ref/.

[15] Peter F. Patel-Schneider, Pat Hayes, and Ian Horrocks. Owl web ontology language semantics and abstract syntax, 2004. W3C Recommendation, available at http://www.w3.org/TR/owl-semantics/.

[16] Jeremy J. Carroll and Jos De Roo. Owl web ontology language test cases, 2004. W3C Recommendation, available at http://www.w3.org/TR/owl-test/.

[17] Jeff Heflin. Owl web ontology language use cases and requirements, 2004. W3C Recommendation, available at http://www.w3.org/TR/webont-req/.

[18] Michael K. Smith. Web ontology issue status, 2003. Available at http://www.w3.org/2001/sw/WebOnt/webont-issues.html.

[19] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (uri): Generic syntax, 1998. IETF (Internet Engineering Task Force) RFC 2396.

[20] M. Uschold and M. Gruninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[21] N. Guarino. Formal ontologies and information systems. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, june 1998.

[22] M. Wooldridge and N.R. Jennings. Intelligent agents: Theories and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[23] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Records*, 28(1), March 1999.

[24] Domenico Beneventano, Sonia Bergamaschi, Silvana Castano, Alberto Corni, R. Guidetti, G. Malvezzi, Michele Melchiori, and Maurizio Vincini. Information integration: The momis project demonstration. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 611–614. Morgan Kaufmann, 2000.

[25] I. Benetti, D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. An information integration framework for e-commerce. *IEEE Intelligent Systems Magazine*, Jannuary/February 2002.

[26] 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.

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

[28] Castano S., De Antonellis V., and De Capitani Di Vimercati S. Global viewing of heterogeneous data sources. *IEEE Transactions TKDE*, (13(2)):277–297, 2001.

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

[30] S. Bergamaschi, D. Beneventano, S. Castano, and M. Vincini. Integrazione di informazione: il linguaggio odl_i^3 e la logica descrittiva olcd. Technical Report INTERDATA T3-R03, MURST 40%, 1998. http://bsing.ing.unibs.it/ deantone/interdata_tema3/.

[31] Veronica Guidetti. Intelligent information integration systems: extending lexicon ontology. Master's thesis, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, Modena, Italy, 2001-2002. Available from http://www.dbgroup.ing.unimo.it/.

[32] N. F. Noy and D. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KLS-01-05, Standford KSL, 2000.

[33] M. Klein and D. Fensel. Ontology versioning on the semantic web. In *International Semantic Web Working Symposium (SWWS)*, pages 75–91, Stanford University, USA, July 30 - August 1 2001.

[34] Ljiljana Stojanovic and Boris Motik. Ontology evolution within ontology editors. In *2002 in 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW2002)*, Siguenza, Spain, October 1-4 2002.

[35] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Daml+oil reference description, 2001. W3C Note, available at http://www.w3.org/TR/daml+oil-reference.

[36] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.