InterData

Programma di ricerca (cofinanziato dal MURST, esercizio 1997)

Metodologie e tecnologie per la gestione di dati e processi su reti Internet e Intranet

Integrazione di informazione: il linguaggio $\mathrm{ODL}_I{}^3$ e la logica descrittiva OLCD

Sonia Bergamaschi, Domenico Beneventano, Maurizio Vincini, Silvana Castano

T3-R03 **16 luglio 1998**

Sommario

In questo Rapporto viene presentata la definizione del linguaggio $\mathrm{ODL}_I{}^3$, utilizzato per rispondere alle problematiche di estrazione/integrazione di sorgenti testuali eterogenee e le proposte di estensione alla logica descrittiva OLCD per l'integrazione di informazioni da fonti eterogenee. Per quanto riguarda queste ultime, si mostrerà come semplici regole di equivalenza permettano la gestione dell'integrazione di dati da parte della logica descrittiva OLCD senza dover ricorrere ad estensioni vere e proprie.

Tema	Tema 3: Integrazione di informazioni estratte da siti WWW
Codice	T3-R03
Data	16 luglio 1998
Tipo di prodotto	Rapporto tecnico
Unità responsabile	MO
Unità coinvolte	MO, BS
Autori	Sonia Bergamaschi, Domenico Beneventano, Maurizio Vincini, Silvana Castano
Autore da contattare	Sonia Bergamaschi Dipartimento di Scienze dell'Ingegneria Università degli Studi di Modena Via Campi 213/B, 41100 Modena, Italia sonia@dsi.unimo.it

Integrazione di informazione: il linguaggio ODL$_{I^3}$e la logica descrittiva OLCD

Sonia Bergamaschi Domenico Beneventano Maurizio Vincini
Silvana Castano

Sommario

In questo Rapporto viene presentata la definizione del linguaggio ODL$_I{}^3$, utilizzato per rispondere alle problematiche di estrazione/integrazione di sorgenti testuali eterogenee e le proposte di estensione alla logica descrittiva OLCD per l'integrazione di informazioni da fonti eterogenee. Per quanto riguarda queste ultime, si mostrerà come semplici regole di equivalenza permettano la gestione dell'integrazione di dati da parte della logica descrittiva OLCD senza dover ricorrere ad estensioni vere e proprie.

Nella sezione 1 viene compiuta una breve panoramica sulle proposte in ambito di ricerca riguardo l'integrazione di informazione da fonti eterogenee. Per una più esaustiva e puntuale descrizione sullo stato dell'arte si faccia riferimento al rapporto tecnico del Progetto INTERDATA - TEMA 3 "Stato dell'arte della ricerca nell'ambito dell'estrazione/integrazione intelligente di informazioni".
La sezione 2 presenta gli strumenti esistenti che sono da noi utilizzati nell'ambito dell'integrazione: la logica descrittiva OLCD, sviluppata in [BN94a, BB97] e il sistema ODB-Tools, presentato in [BBSV97a], in grado di utilizzare la logica descrittiva per compiere *ragionamento tassonomico* sullo schema ed *espansione semantica* sulle query.
In sezione 3 viene presentata la nostra proposta per l'integrazione di dati: dapprima il sistema MOMIS (Mediator envirOnment for Multiple Information Sources) che realizza l'integrazione delle informazioni da fonti eterogenee linguaggio; quindi viene compiuta l'analisi delle estensioni al linguaggio standard ODL-ODMG93 per l'integrazione di informazioni e l'analisi delle possibili estensioni alla logica descrittiva OLCD.

1 Integrazione di dati da fonti eterogenee: breve panoramica

Il problema comune cui oggigiorno molte organizzazioni e istituzioni debbono far fronte è la dispersione del loro patrimonio informativo, frammentato in numero sempre maggiore e disparato di sorgenti informative quali possono essere DBMS, Object Store, Knowledge Base, File System, Digital Library, Information Retrieval System, ... I potenziali utenti delle informazioni, che possono essere individuati nei *decision maker*, sempre più spesso richiedono informazioni da sorgenti diverse, senza essere in grado di compiere autonomamente la ricerca e la fusione delle informazioni. Questo a causa delle difficoltà tecnologiche per l'accesso a sistemi diversi e a problemi semantici derivanti dall'incosistenza e contraddizione delle informazioni ottenute.
I modelli dei dati e gli schemi si differenziano infatti gli uni dagli altri fornendo ciascuno una propria struttura logica di rappresentazione dei numerosi tipi di dati da immagazzinare e questo crea una eterogeneità *semantica* (o *logica*) non risolvibile dagli attuali standard. Inoltre si devono trovare le soluzioni a problemi quali l'*overload* delle informazioni, la riduzione dei tempi di accesso, la salvaguardia della sicurezza e della consistenza, gli elevati costi di mantenimento da affrontare per modificare, eliminare od introdurre una nuova sorgente.

Tutti i problemi elencati sottolineano la complessità degli aspetti che le architetture dedicate all'integrazione devono comprendere. Per facilitare il processo di progettazione e realizzazione di tali moduli dedicati, che siano affidabili, flessibili e tali da far fronte efficacemente ad un panorama in continua evoluzione, si cerca di sviluppare un'architettura di moduli che assicuri il riuso e la capacità di interagire con altri sistemi esistenti.

Gli approcci all'integrazione, descritti in letteratura o effettivamente realizzati, presentano diverse metodologie che vanno dalla reingegnerizzazione delle sorgenti mediante standardizzazione degli schemi e creazione di un database distribuito, all'approccio complementare denominato *repository independence*, che prevede di isolare al di sotto di una vista integrata le applicazioni ed i dati integrati dalle sorgenti, consentendo la massima autonomia e nascondendo al contempo le differenze esistenti. Nel seguito si descrive la proposta dell'ARPA (Advanced Research Projects Agency)[ARP] per un'architettura che favorisca l'autonomia delle sorgenti, assicurando flessibilità e riusabilità e si presenta l'approccio seguito per il progetto MOMIS.

1.1 L'Integrazione Intelligente delle Informazioni

L'integrazione delle Informazioni (I^2) si distingue da quella dei dati e dei database in quanto non cerca di collegare semplicemente alcune sorgenti quanto risultati opportunamente selezionati da esse [Wea96]. Per ottenere risultati selezionati è richiesta *conoscenza* ed *intelligenza* finalizzate alla scelta delle sorgenti e dei dati, alla loro fusione e alla conseguente sintesi. L'integrazione comporta perciò grandi difficoltà, sia a livello teorico che pratico, oltre a concrete differenze realizzative. La dimensione e la quantità delle problematiche che insorgono qualora si desideri fondere informazioni provenienti da sorgenti autonome fanno sì che si cerchi di ideare una metodologia riusabile, per diminuire i costi di sviluppo di tali applicazioni, e flessibile per far fronte intelligentemente alle evoluzioni fisiche e logiche delle sorgenti interessate.

Un'ambiziosa ricerca, finalizzata ad indicare un'architettura di riferimento che realizzi l'integrazione di sorgenti eterogenee in maniera automatica, è quella sviluppata dalla ARPA nel programma I^3. L'integrazione aumenta il valore dell'informazione ma richiede una forte adattabilità realizzativa: si devono poter gestire i casi di aggiornamento e sostituzione delle sorgenti, dei loro ambienti e/o piattaforme, della loro ontologia e della semantica. Le tecniche sviluppate dall'*Intelligenza Artificiale*, potendo efficacemente dedurre informazioni utili dagli schemi delle sorgenti, diventano uno strumento prezioso per la costruzione automatica di soluzioni integrate flessibili e riusabili.

Costruire supersistemi ad hoc che interessino una gran quantità di sorgenti non correlate semanticamente in modo premeditato, è faticoso ed il risultato è un sistema scarsamente manutenibile, adattabile e strettamente finalizzato alla risoluzione dei problemi per cui è stato implementato. Secondo il programma I^3, una soluzione a questi problemi deriva dall'introduzione di architetture modulari sviluppabili secondo i principi posti da uno standard che ponga le basi dei servizi che devono essere soddisfatti dall'integrazione ed abbassi i costi di sviluppo e di mantenimento. Costruire nuovi sistemi risulta realizzabile con minor difficoltà e minor tempo (e costi) se si riesce a supportare lo sviluppo delle applicazioni riutilizzando la tecnologia già sviluppata. Per potere riusare è fondamentale l'esistenza di interfacce ed architetture standard. Il paradigma suggerito[1] per la suddivisione dei servizi e delle risorse nei diversi moduli si basa su due partizionamenti:

- quello orizzontale distingue tre livelli in database sorgenti, basi di conoscenza intermedi, applicazioni utente.
- quello verticale distingue i domini in cui raggruppare le sorgenti (il cui numero si deve aggirare fra cinque e nove).

[1] Le specifiche architetturali del programma I^3 non sono ancora state messe a punto in maniera definitiva, ma è stato sviluppato un Glossario comune.

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

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

1.1.1 Motivazioni

Per capire come il programma I^3 sia arrivato a suggerire una certa architettura occorre mettere in evidenza i problemi che essa si deve prefiggere di risolvere:

- *L'eterogeneità delle sorgenti:* si manifesta a tutti i livelli di rappresentazione. Distingue i dati per tipi, schemi logici, lessico utilizzato per descriverli, meccanismi di accesso (interfacce di programmazione, linguaggi di interrogazione, protocolli per la gestione delle transazioni), rendendo necessaria la creazione di un modello globale per i dati da integrare.

- *La semantica nascosta:* molti sistemi di grandi dimensioni consentono di accedere ai propri dati attraverso interfacce predefinite: in questi casi diventa spesso molto complicato dedurre le regole che consentono di interpretare ed elaborare i dati da integrare.

- *La costruzione di applicazioni indipendenti dai diversi gradi di dettaglio delle sorgenti*: spesso integrare grandi sistemi comporta una scelta sui segmenti di dati accessibili e su come farne uso: se renderli semplicemente accedibili o viceversa mantenerli trasparenti ma tradurli in informazioni implicite utili ai servizi di integrazione.

- *Le dimensioni:* non solo le sorgenti da integrare sono sempre più spesso di grandi dimensioni, ma anche il numero di sorgenti appartenenti ad uno stesso dominio è spesso molto elevato.

- *La necessità di sviluppare sistemi modulari riusabili:* è importante disporre di strumenti che facilitno e standardizzino le operazioni necessarie all'integrazione in modo da ridurre tempi e costi di sviluppo.

- *La continua evoluzione delle sorgenti:* non solo le sorgenti migliorano le piattaforme, i programmi e le rappresentazioni dei dati, ma si aggiungono sempre nuove e diverse sorgenti, perciò i sistemi progettati devono essere in grado di adattarsi a questi cambiamenti con flessibilitá e velocitá.

1.1.2 L'Architettura di Riferimento

In questa prima fase, il progetto proposto dall'ARPA intende dare una visione esauriente dei servizi necessari all'integrazione. In particolare individua cinque famiglie di attività omogenee, la cui reciproca interazione consente di eseguire le operazioni di comunicazione, traduzione ed integrazione dei dati nelle sorgenti. In Figura 1 si illustrano i principali legami fra le famiglie di servizi. I sevizi principali sono quelli di Coordinamento: essi usufruiscono delle funzionalità messe a loro disposizione dalle altre famiglie e coordinano le operazioni da eseguire sia in fase di progettazione dei "link" fra i livelli ed i domini informativi che di esecuzione in tempo reale di una particolare richiesta dell'utente. Per svolgere quest'importanti attività si devono individuare le sorgenti informative ed i diversi servizi messi a disposizione dagli altri moduli, e generare ed invocare i sistemi run-time in grado di eseguire le applicazioni stesse.

Alcune parole chiave per capire il fitto intreccio di interazioni fra queste famiglie sono qui definite:

- *Ambiente(Environment):* l'insieme dei servizi di Coordinamento e delle sorgenti e degli altri servizi I^3 che possono essere trovati per invocare un certo insieme di attività (task). Esso fornisce un accesso intelligente ad un sottoinsieme dei dati e degli applicativi messi a disposizione dall'architettura complessiva in modo finalizzato.

- *Configurazione(Configuration):* sono i sistemi run-time incaricati di eseguire uno specifico task, in particolare sono cosituiti da un insieme di tool e sorgenti interconnesse. Sono costruite sia a run che a compile time. Un ambiente può invocare più configurazioni che collaborino per raggiungere un determinato risultato.

- *Strumento(Tool):* sono incaricati di eseguire le attività, indipendentemente dalla famiglia di servizi che li invoca.

- *Schema di comportamento(Template):* costituisce la codifica di una configurazione (o di parte di essa): generalmente si tratta di strutture dati che specificano le sorgenti, i tool ed il reciproco ordine di interazione.

L'analisi del diagramma in Figura 1 bene evidenzia la dimensione orizzontale citata precedentemente. L'asse centrale rappresenta il flusso informativo di un dominio (o interdomini) fra le sorgenti e l'applicazioni finali. I tre livelli orizzontali rappresentano i livelli di gestione e manipolazione dei dati: quello delle sorgenti, rappresentato dai Wrapper, traduce l'informazione in un'interfaccia comune, quello intermedio analizza e sintetizza le informazioni provenienti dai diversi domini e quello di vertice mantiene riferimento dei domini e dei servizi disponibili e li invoca. Illustriamo brevemente le cinque famiglie di servizi.

Servizi di Coordinamento Questi servizi forniscono supporto sia in fase di progettazione e programmazione di nuove Configurazioni che di esecuzione a run-time delle stesse. Per poter fare questo devono disporre di conoscenza sulle sorgenti (metadati ed ontologie) e sulla natura e la struttura dei servizi esistenti: queste informazioni sono gestite dai servizi di Amministrazione. Il Coordinamento costituisce l'interfaccia con l'utente dandogli l'impressione di trattare con un sistema omogeneo: ogni richiesta dell'utente non viene gestita direttamente da questi servizi, ma si inserisce in un certo Ambiente ed origina o invoca una Configurazione: a questo punto il controllo passa ai servizi di Amministrazione che trovano le sorgenti ed i tool preposti e ne controllano lo scheduling delle attività. Illustriamo sinteticamente i principali moduli di Coordinamento.

- **Broker:** si occupa di selezionare ed invocare i Tool capaci di soddisfare una richiesta: in questo senso interagisce coi sevizi di Amministrazione che trovano le risorse adeguate e coordinano le richieste e le risposte fra i tool che servono la richiesta specifica.

- **Facilitator:** gestisce le richieste più complesse: le traduce, le scompone, le instrada verso gli opportuni Tool che le soddisfano in maniera autonoma ed infine compone le risposte ottenute: in altre parole costruisce dinamicamente le Configurazioni. Trattandosi di un

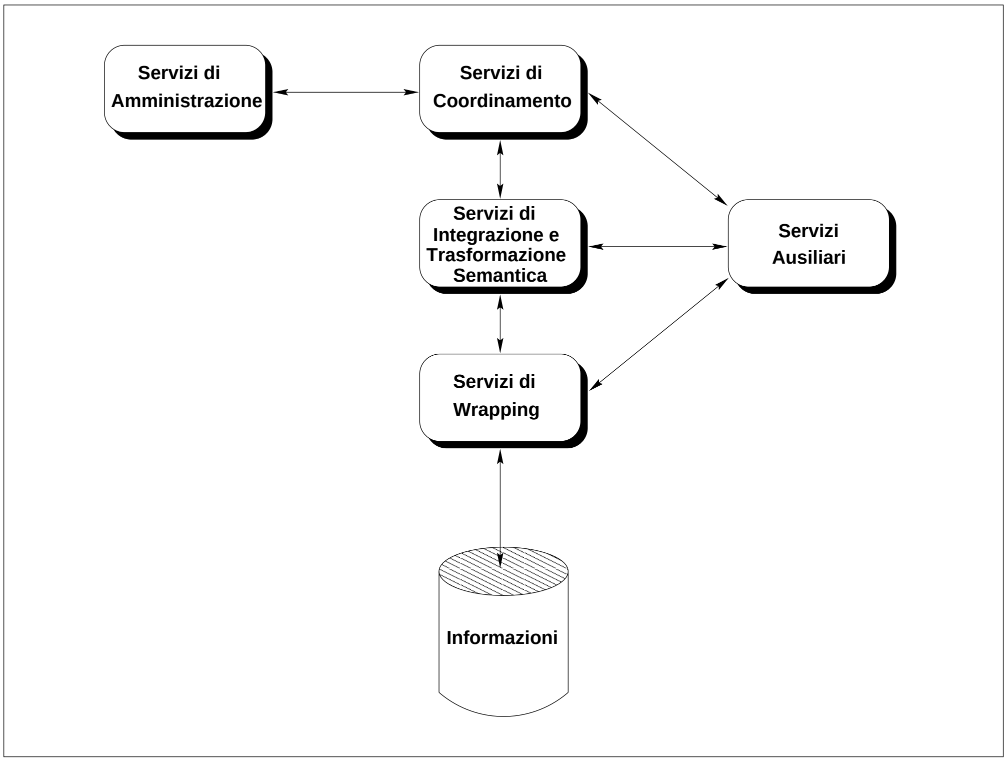


Figura 1: Diagramma dei servizi I^3

Broker con funzionalità più sofisticate oltre ai servizi di Amministrazione già citati, interagisce con sevizi Ausiliari di inferenza e con quelli di Tasformazione ed Integrazione Semantica per riuscire nel processo di scomposizione delle query.

- **Matchmaker:** appronta configurazioni dedicate a certi servizi: i componenti e le loro comunicazioni sono settate una volta per tutte, e non calcolate dinamicamente.

Servizi di Amministrazione Forniscono le informazioni relative alle sorgenti, agli strumenti disponibili per soddisfare una certa attività ed ai passi in da eseguire per soddisfare una richiesta, sono quindi indispensabili ai servizi di Coordinamento nella costruzione (statica o dinamica) delle configurazioni e dei relativi template. Inoltre una volta creati i template sono questi servizi a controllarne e schedularne l'attività. I moduli più interessanti sono:

- **Resource Discovery:** sono in grado di riconoscere e ritrovare gli strumenti che gestiscono una determinata richiesta a run time. Questi servizi sono in grado di acquisire ed aggiornare dinamicamente informazioni sui tool (nomi e servizi forniti) e sul loro stato. Inoltre acquisiscono e mantengono le informazioni sui domini informativi.
- **Browser:** servono per mostrare all'utente gli schemi integrati e quelli delle sorgenti singole, in questo senso è indispensabile l'interazione coi sevizi di Wrapping, che recuperano e traducono in uno standard dichiarativo queste informazioni, e di Trasformazione Semantica per le informazioni relative all'ontologia ed al vocabolario per il mappaggio fra lo schema integrato e quelli sorgenti.
- **Iterative Query Formulation:** sono di supporto all'utente in fase di formulazione di query particolarmente complesse sullo schema integrato, specialmente qualora la query

formulata non abbia prodotto informazioni interessanti, suggeriscono quasli condizioni rilasciare, come rendere più specifica o più generale la query.

- **Primitive di costruzione delle configurazioni:** servono a scegliere i servizi, i tool e le sorgenti appropriati per svolgere un opportuno task e come collegarli fra loro per creare una configurazione.

Servizi di Integrazione e Trasformazione Semantica Sono preposti alla manipolazione semantica delle informazioni e delle applicazioni delle sorgenti: hanno come input il risultato dei servizi di Traduzione e da questi generano una vista integrata diversa dalla semplice unione delle informazioni delle sorgenti, oppure una applicazione ottenuta riusando opportunamente applicazioni che operano in ambienti autonomi. I pricipali servizi sono:

- **Integrazione degli Schemi:** si occupano della creazione dei vocabolari e delle ontologie condivise delle sorgenti, integrano gli schemi in una vista globale, mantengono il mappaggio tra gli schemi globali e le sorgenti,

- **Integrazione delle Informazioni:** aggregano, riassumono ed astraggono i dati per fornire presentazioni analitiche significative.

- **Supporto:** forniscono le funzionalità per la scomposizione delle query su un certo schema integrato e per l'aggregazione dei risultati, generano la propagazione degli update (quando prevista) e risolvono i problemi di incongruenze fra i dati da integrare. Sono forniti anche servizi di supporto all'integrazione fisica: caching delle risposte, indexing, replicazione, etc... .

Servizi Ausiliari Sono servizi di varia natura che aumentano le funzionalità e le possibilità degli altri servizi. I servizi atti ad arricchire l'espressività semantica delle sorgenti sono utilizzati maggiormente da quelli di Coordinamento, di Integrazione e di Wrapping. Altri servizi forniti sono quelli di monitoraggio e di mantenimento della consistenza fra le sorgenti.

Servizi di Traduzione Realizzano il primo passo verso l'integrazione rendendo tutte le informazioni (dati ed applicazioni) omogenee grazie alla traduzione delle stesse in un linguaggio standard. Costituiscono un'interfaccia fra le sorgenti e gli altri servizi, dunque un grande vantaggio si ha qualora questi moduli siano riusabili e coerenti con gli standard giá definiti (SQL, CORBA, OLE) e se se ne rendende il processo di costruzione automatico attraverso la costruzione di una libreria di servizi addetti.

1.1.3 OMG versus I^3

L'architettura proposta dall'ARPA non deve essere considerata indipendente dalle tecnologie che altri gruppi stanno sviluppando, anzi deve essere collocata all'interno del panorama informatico: il rispetto reciproco del lavoro realizzato dalle diverse comunità aumenta il valore delle tecnologie e della loro interazione. In particolare evidenziamo lo stretto rapporto con l'architettura realizzata dall'OMG (Object Management Group) [OMG98], i cui obiettivi di fondo sono lo sviluppo di uno standard e di interfacce per le comunicazioni fra gli oggetti.

Il primo obiettivo dell'ARPA è stato identificare i servizi e le loro relazioni, il successivo deve affrontare la scelta dei paradigmi di supporto all'interazione fra i vari I^3 Tool, come ad esempio il linguaggio per le comunicazioni, lo standard per le interfacce e per comporre servizi complessi. Per realizzare questa seconda fase di definizione delle interazioni fra i vari servizi dell'architettura, il confronto con la tecnologia OMG, sviluppata per supportare la produzione di Sw riusabile, scalabile e portabile per sistemi aperti, distribuiti ed interagenti, è basilare.

L'architettura proposta dall'OMG è più generale di quella I^3: mentre la seconda si occupa di descrivere e coordinare solo i servizi necessari all'integrazione, la prima si rivolge a tutti quei servizi che possono essere trattati come oggetti, indipendentemente dal loro scopo. Questo non significa però che l'approccio I^3 sia inutile, anzi esso può supportare l'architettura OMG in relazione ai compiti di integrazione, mentre OMG riesce a fornire lo standard per implementare i servizi I^3: il vantaggio di ricorrere ad essi deriva dal fatto che sono già implementati e provati ed assicurano un continuo sviluppo delle loro funzionalità.

Un limite offerto dallo standard OMG alla gestione dell'architettura I^3 è rappresentato dal linguaggio IDL. Esso è descrittivo, indipendente dal codice, orientato agli oggetti e adatto a descrivere importanti concetti del mondo dei DB, ma non è semanticamente ricco e quindi non consente di trattare adeguatamente informazioni relative alla *conoscenza*, come rule e metadati, indispensabili ai servizi di Integrazione Semantica, Ausiliari e di Amministazione. Altri linguaggi, sviluppati nell'area dei paradigmi ad Agenti, sembrano invece maggiormente adatti a favorire la comunicazione fra quei sevizi che devono trattare principalmente con la conoscenza. Al momento si sta ancora cercando di capire quale dei due protocolli sia maggiormente vantaggioso per implementare gli I^3 Tools o se sia possibile realizzare una combinazione nell'uso di entrambi.

2 Gli strumenti esistenti

2.1 OLCD: una logica descrittiva per Oggetti Complessi e Vincoli di Integrità

In questa sezione viene presentata la logica descrittica OLCD (Object Languages with Complements allowing Descriptive cycles), nonché una breve descrizione dei concetti di *relazione di sussunzione* e *espansione semantica.*
OLCD deriva dalla $1^{st}OPL$, opportunamente ridotta per consentire la decidibilità di ogni suo processo di inferenza. Nel corso della descrizione, si farà inoltre riferimento ad un esempio esplicativo, di un dominio `Magazzino`, di cui riportiamo la descrizione in sintassi ODMG-93 nella tabella 1, e le regole di integrità su di esso definite in tabella 2.

L'esempio considerato è relativo alla struttura organizzativa di una società: i materiali (`Material`) sono descritti da un nome, un rischio e da un insieme di caratteristiche; gli `SMaterial` sono un sottoinsieme dei `Material`. I `DMaterial` (materiali "pericolosi") sono un sottoinsieme dei `Material`, caratterizzati da un rischio compreso tra 15 e 100. I manager hanno un nome, un salario compreso tra 40K e 100K dollari e un livello compreso tra 1 e 15. I `TManager` sono manager che hanno un livello compreso tra 8 e 12. I magazzini (`Storage`) sono descritti da una categoria, sono diretti (`managed_by`) da un manager e contengono (`stock`) un insieme di articoli (`item`) che sono dei materiali, per ciascuno dei quali è indicata la quantità presente; è inoltre riportato il rischio massimo (`maxrisk`) ammissibile per i materiali conservati. Gli `SStorage` sono un sottoinsieme dei magazzini. Infine vi sono i magazzini "pericolosi" (`DStorage`) che sono un sottoinsieme degli `Storage` caratterizzati dallo stoccaggio di soli materiali "pericolosi" (`DMaterial`).

2.1.1 Schema e istanza del database

Sia $\mathbf{D}$ l'insieme infinito numerabile dei valori atomici (che saranno indicati con $d_1, d_2, \ldots$), e.g., l'unione dell'insieme degli interi, delle stringhe e dei booleani. Sia $\mathbf{B}$ l'insieme di designatori di tipi atomici, con $\mathbf{B} = \{\texttt{integer}, \texttt{string}, \texttt{boolean}, \texttt{real}, i_1\text{–}j_1, i_2\text{–}j_2, \ldots, d_1, d_2, \ldots\}$, dove i d_k indicano tutti gli elementi di $\texttt{integer} \cup \texttt{string} \cup \texttt{boolean}$ e dove gli $i_k\text{–}j_k$ indicano tutti i

Classi dello Schema

```
interface Material
{
   attribute string name;
   attribute integer risk;
   attribute string code;
   attribute set <string> feature;
};
interface DMaterial: Material
{
   attribute range {15, 100} risk;
};
interface SMaterial: Material{ };
interface Storage
{
   attribute string category;
   attribute Manager managed_by;
   attribute set < struct {
         attribute Material item;
         attribute range {10, 300} qty; } > stock;
   attribute integer maxrisk;
};
interface Manager
{
   attribute string name;
   attribute range {40K, 100K} salary;
   attribute range {1, 15} level;
};
interface TManager: Manager
{
   attribute range {8, 12} level;
};
interface SStorage: Storage{ };
interface DStorage: Storage
{
   attribute set < struct {
         attribute DMaterial item; } > stock;
};
```

Tabella 1: Schema del dominio `Magazzino` in sintassi ODMG–93

```
rule R1   for all X in Material (X.risk ≥ 10)
          then   X in SMaterial
rule R2   for all X in Storage (
                 for all X1 in X.stock (X1.item in SMaterial))
          then   X in SStorage
rule R3   for all X in Storage (X.managed_by.level ≥ 6 and
                                X.managed_by.level ≤ 12)
          then   X.category = "A2"
rule R4   for all X in Storage (X.category = "A2" and
                 exists X1 in X.stock (X1.item.risk ≥ 10))
          then   X.managed_by in SMaterial)
```

Tabella 2: Regole di integrità sul dominio `Magazzino` possibili intervalli di interi (i_k può essere $-\infty$ per denotare il minimo elemento di `integer` e j_k può essere $+\infty$ per denotare il massimo elemento di `integer`).

Sia $\mathbf{A}$ un insieme numerabile di *attributi* (denotati da $a_1, a_2, \ldots$) e $\mathcal{O}$ un insieme numerabile di *identificatori di oggetti* (denotati da $o, o', \ldots$) disgiunti da $\mathbf{D}$. Si definisce l'insieme $\mathcal{V}(\mathcal{O})$ dei *valori su* $\mathcal{O}$ (denotati da v, v') come segue (assumendo $p \geq 0$ e $a_i \neq a_j$ per $i \neq j$):

$$v \ \rightarrow d \mid o \mid \{v_1, \ldots, v_p\} \mid [a_1 : v_1, \ldots, a_p : v_p]$$

Gli identificatori di oggetti sono associati a valori tramite una *funzione totale* δ da $\mathcal{O}$ a $\mathcal{V}(\mathcal{O})$; in genere si dice che il valore $\delta(o)$ è lo *stato* dell'oggetto identificato dall'oid o.
Sia $\mathbf{N}$ l'insieme numerabile di *nomi di tipi* (denotati da $N, N', \ldots$) tali che $\mathbf{A}$, $\mathbf{B}$, e $\mathbf{N}$ siano a due a due disgiunti. $\mathbf{N}$ è partizionato in tre insiemi $\mathbf{C}$, $\mathbf{V}$ e $\mathbf{T}$, dove $\mathbf{C}$ consiste di nomi per *tipi-classe base* $(C, C' \ldots)$, $\mathbf{V}$ consiste di nomi per *tipi-classe virtuali* $(V, V' \ldots)$, e $\mathbf{T}$ consiste di nomi per *tipi-valori* $(t, t', \ldots)$. Un *path* p è una sequenza di elementi $p = e_1 . e_2 . \ldots . e_n$, con $e_i \in \mathbf{A} \cup \{\triangle, \forall, \exists\}$. Con ϵ si indica il path vuoto.
$\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$[2] indica l'insieme di tutte le *descrizioni di tipo finite* (S, S', ...), dette brevemente *tipi*, su di un dato $\mathbf{A}, \mathbf{B}, \mathbf{N}$, ottenuto in accordo con la seguente regola sintattica:

$$S \quad \rightarrow \quad \top \mid B \mid N \mid [a_1 : S_1, \ldots, a_k : S_k] \mid \forall\{S\} \mid \exists\{S\} \mid \ \triangle S \mid S \sqcap S' \mid (p : S)$$

$\top$ denota il *tipo universale* e rappresenta tutti i valori; [ ] denota il costruttore di tupla. $\forall\{S\}$ corrisponde al comune costruttore di insieme e rappresenta un insieme i cui elementi sono *tutti* dello stesso tipo S. Invece, il costruttore $\exists\{S\}$ denota un insieme in cui *almeno* un elemento è di tipo S. Il costrutto $\sqcap$ indica la *congiunzione*, mentre $\triangle$ è il costruttore di oggetto. Il tipo $(p \colon S)$ è detto *tipo path* e rappresenta una notazione abbreviata per i tipi ottenuti con gli altri costruttori.

Dato un dato sistema di tipi $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, uno *schema* σ su $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ è una funzione totale da $\mathbf{N}$ a $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, che associa ai nomi di tipi la loro descrizione. Diremo che un nome di tipo N *eredita* da un altro nome di tipo N', denotato con $N \prec\!\!|_\sigma N'$, se $\sigma(N) = N' \sqcap S$. Si richiede che la relazione di ereditarietà sia priva di cicli, i.e., la chiusura transitiva di $\prec\!\!|_\sigma$, denotata $\prec$, sia un ordine parziale stretto.

Dato un dato sistema di tipi $\mathbf{S}$, una *regole di integrità* R è espressa nella forma $R = S^a \rightarrow S^c$, dove S^a e S^c rappresentano rispettivamente l'antecedente e il conseguente della regola R, con $S^a, S^c \in \mathbf{S}$. Una regola R esprime il seguente vincolo: per tutti gli oggetti v, se v è di tipo S^a allora v deve essere di tipo S^c. Con $\mathbf{R}$ si denota un insieme finito di regole.

Uno *schema con regole* è una coppia $(\sigma, \mathbf{R})$, dove σ è uno schema e $\mathbf{R}$ un insieme di regole. Ad esempio, il dominio Magazzino rappresentato in Tabella 1 ed esteso in 2 con l'aggiunta di alcune regole di integrità, è descritto in OLCD tramite lo schema con regole mostrato in Tabella 3.

La *funzione interpretazione* $\mathcal{I}$ è una funzione da $\mathbf{S}$ a $2^{\mathcal{V}(\mathcal{O})}$ tale che: $\mathcal{I}[\top] = \mathcal{V}(\mathcal{O})$, $\mathcal{I}[B] = \mathcal{I}_{\mathbf{B}}[B]$[3] , $\mathcal{I}[C] \subseteq \mathcal{O}$, $\mathcal{I}[V] \subseteq \mathcal{O}$, $\mathcal{I}[t] \subseteq \mathcal{V}(\mathcal{O}) - \mathcal{O}$. L'interpretazione è estesa agli altri tipi come segue:

$$\begin{aligned}
\mathcal{I}[[a_1 \colon S_1, \ldots, a_p \colon S_p]] &= \Big\{ [a_1 \colon v_1, \ldots, a_q \colon v_q] \Big| \, p \leq q, v_i \in \mathcal{I}[S_i], 1 \leq i \leq p, \\
&\qquad\qquad v_j \in \mathcal{V}(\mathcal{O}), p{+}1 \leq j \leq q \Big\} \\
\mathcal{I}[\forall\{S\}] &= \Big\{ \{v_1, \ldots, v_p\} \Big| v_i \in \mathcal{I}[S], 1 \leq i \leq p \Big\} \\
\mathcal{I}[\exists\{S\}] &= \Big\{ \{v_1, \ldots, v_p\} \Big| \exists i, 1 \leq i \leq p, v_i \in \mathcal{I}[S] \Big\} \\
\mathcal{I}[\triangle S] &= \Big\{ o \in \mathcal{O} \Big| \delta(o) \in \mathcal{I}[S] \Big\} \\
\mathcal{I}[S \sqcap S'] &= \mathcal{I}[S] \cap \mathcal{I}[S']
\end{aligned}$$

[2]In seguito, scriveremo $\mathbf{S}$ in luogo di $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ quando i componenti sono ovvi dal contesto.
[3]Assumendo $\mathcal{I}_{\mathbf{B}}$ funzione di *interpretazione standard* da $\mathbf{B}$ a $2^{\mathbf{B}}$ tale che per ogni $d \in \mathbf{D} : \mathcal{I}_{\mathbf{B}}[d] = \{d\}$.

$$
\begin{array}{l}
\mathbf{C} = \{\texttt{Material}, \texttt{SMaterial}, \texttt{DMaterial}, \texttt{Storage}, \texttt{SStorage}, \texttt{DStorage}, \texttt{Manager}, \\
\quad \texttt{TManager}\}, \\
\mathbf{V} = \emptyset, \\
\mathbf{T} = \emptyset
\end{array}
$$

$$
\sigma \left\{
\begin{array}{lcl}
\sigma(\texttt{Material}) & = & \triangle[\texttt{name}: \texttt{string}, \texttt{risk}: \texttt{integer}, \texttt{code}: \texttt{string}, \texttt{feature}: \{\texttt{string}\}] \\
\sigma(\texttt{SMaterial}) & = & \texttt{Material} \\
\sigma(\texttt{DMaterial}) & = & \texttt{Material} \sqcap \triangle[\texttt{risk}: 15 \div 100] \\
\sigma(\texttt{Storage}) & = & \triangle[\texttt{managed\_by}: \texttt{Manager}, \texttt{category}: \texttt{string}, \texttt{maxrisk}: \texttt{integer}, \\
 & & \quad \texttt{stock}: \{[\texttt{item}: \texttt{Material}, \texttt{qty}: 10 \div 300]\}] \\
\sigma(\texttt{SStorage}) & = & \texttt{Storage} \\
\sigma(\texttt{DStorage}) & = & \texttt{Storage} \sqcap \triangle[\texttt{stock}: \{[\texttt{item}: \texttt{DMaterial}]\}] \\
\sigma(\texttt{Manager}) & = & \triangle[\texttt{name}: \texttt{string}, \texttt{salary}: 40K \div 100K, \texttt{level}: 1 \div 15] \\
\sigma(\texttt{TManager}) & = & \texttt{Manager} \sqcap \triangle[\texttt{level}: 8 \div 12]
\end{array}
\right.
$$

$$
\mathbf{R} \left\{
\begin{array}{ll}
R_1 : & \texttt{Material} \sqcap (\triangle.\texttt{risk}: 10 \div \infty) \rightarrow \texttt{SMaterial} \\
R_2 : & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\forall.\texttt{item}: \texttt{SMaterial}) \rightarrow \texttt{SStorage} \\
R_3 : & \texttt{Storage} \sqcap (\triangle.\texttt{managed\_by}.\triangle\texttt{level}: 6 \div 12) \rightarrow (\triangle.\texttt{category}: \text{``}\mathit{A2}\text{''}) \\
R_4 : & \texttt{Storage} \sqcap (\triangle.\texttt{category}: \text{``}\mathit{A2}\text{''}) \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{risk}: 10 \div \infty) \\
 & \qquad \rightarrow \triangle[\texttt{managed\_by}: \texttt{TManager}]
\end{array}
\right.
$$

Tabella 3: Schema con Regole di Integrità in linguaggio OLCD

Per i tipi cammino abbiamo $\mathcal{I}[(p: S)] = \mathcal{I}[(e: (p': S))]$ se $p = e.p'$ dove

$$
\mathcal{I}[(\epsilon: S)] = \mathcal{I}[S],\ \mathcal{I}[(a: S)] = \mathcal{I}[[a: S]],\ \mathcal{I}[(\triangle: S)] = \mathcal{I}[\triangle S],
$$
$$
\mathcal{I}[(\forall: S)] = \mathcal{I}[\forall\{S\}],\ \mathcal{I}[(\exists: S)] = \mathcal{I}[\exists\{S\}]
$$

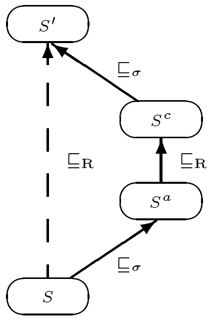
Quindi il tipo path è un'abbreviazione per un altro tipo: ad esempio, il tipo path ($\triangle$.`stock`.$\forall$.`item`: `SMaterial`) è equivalente a $\triangle$[`stock`: $\forall$\{[`item`: `SMaterial`]\}].

Si introduce ora la nozione di istanza legale di uno schema con regole come una interpretazione nella quale le interpretazioni di classi e tipi vengono vincolate alla loro descrizione e sono valide le relazioni di inclusioni stabilite tramite le regole.

Definizione 1 (Istanza Legale) *Una funzione di interpretazione $\mathcal{I}$ è una* istanza legale *di uno schema con regole* $(\sigma, \mathbf{R})$ *sse l'insieme $\mathcal{O}$ è* finito *e per ogni $C \in \mathbf{C}, V \in \mathbf{V}, T \in \mathbf{T}, R \in \mathbf{R}$: $\mathcal{I}[C] \subseteq \mathcal{I}[\sigma(C)]$, $\mathcal{I}[T] = \mathcal{I}[\sigma(T)]$, $\mathcal{I}[V] = \mathcal{I}[\sigma(V)]$, $\mathcal{I}[S^a] \subseteq \mathcal{I}[S^c]$.*

Si noti come, in una istanza legale $\mathcal{I}$, l'interpretazione di un nome di classe base (C) *è contenuta* nell'interpretazione della sua descrizione, mentre per un nome di classe virtuale (V), come per un nome di tipo-valore (T), l'interpretazione *coincide* con l'interpretazione della sua descrizione. In altri termini, mentre l'interpretazione di una classe base è fornita dall'utente, l'interpretazione di una classe virtuale è calcolata sulla base della sua descrizione.

Pertanto, in particolare, le classi virtuali possono essere utilizzate per rappresentare sia viste che query effettuate sulla base di dati. Se lo schema è aciclico, l'istanza di classi virtuali e tipi può essere univocamente calcolata sulla base della sua descrizione, in presenza di classi virtuali cicliche questa computazione non è univoca: fissata un'interpretazione per le classi base, una classe virtuale ciclica può avere più di una interpretazione possibile. Tra tutte queste interpretazioni, in OLCD viene selezionata come istanza legale quella *più grande*, cioè viene adottata una semantica di *massimo punto fisso*. Le definizioni formali di tale semantica e le motivazioni della scelta sono discusse in [BN94b]; in questa sede si vuole soltanto evidenziare il fatto che il modello ammette delle classi virtuali cicliche e quindi il metodo di ottimizzazione proposto è applicabile anche alle query cicliche o ricorsive [BB96].



$$
\begin{aligned}
S &= \ \texttt{Manager} \sqcap (\triangle.\texttt{level}\colon 5 \div 8)\\
S' &= \ (\triangle.\texttt{salary}\colon 30 \div 70)\\
R_1 &= \ \texttt{Manager} \sqcap (\triangle.\texttt{level}\colon 5 \div 10)\\
&\qquad \rightarrow (\triangle.\texttt{salary}\colon 40 \div 60)
\end{aligned}
$$

$$S \sqsubseteq_{\mathrm{R}} S', \quad S \not\sqsubseteq_\sigma S'$$

Figura 2: Relazione di sussunzione dovuta alle regole

2.1.2 Sussunzione ed Espansione Semantica di un tipo

Introduciamo la definizione di relazione di sussunzione in uno schema con regole.

Definizione 2 (Sussunzione) *Dato uno schema con regole* $(\sigma, \mathbf{R})$*, la* relazione di sussunzione *rispetto a* $(\sigma, \mathbf{R})$*, scritta* $S \sqsubseteq_{\mathrm{R}} S'$ *per ogni coppia di tipi* $S, S' \in \mathbf{S}$*, è data da:* $S \sqsubseteq_{\mathrm{R}} S'$ *sse* $\mathcal{I}[S] \subseteq \mathcal{I}[S']$ *per tutte le istanze legali* $\mathcal{I}$ *di* $(\sigma, \mathbf{R})$.

Segue immediatamente che $\sqsubseteq_{\mathrm{R}}$ è un preordine (i.e., transitivo e riflessivo ma antisimmetrico) che induce una *relazione di equivalenza* $\simeq_{\mathrm{R}}$ sui tipi: $S \simeq_{\mathrm{R}} S'$ sse $S \sqsubseteq_{\mathrm{R}} S'$ e $S' \sqsubseteq_{\mathrm{R}} S$. Diciamo, inoltre, che un tipo S è *inconsistente* sse $S \simeq_{\mathrm{R}} \bot$, cioè per ciascun dominio l'interpretazione del tipo è sempre vuota.

È importante notare che la relazione di sussunzione rispetto al solo schema σ, cioè considerando $\mathbf{R} = \emptyset$, denotata con $\sqsubseteq_\sigma$, è simile alle relazioni di *subtyping* o *refinement* tra tipi definite nei *CODMs* [BBS93, LR89]. Questa relazione può essere calcolata attraverso una comparazione sintattica sui tipi; per il nostro modello tale l'algoritmo è stato presentato in [BN94b, BBS96, BB97].

È immediato verificare che, per ogni S, S', se $S \sqsubseteq_\sigma S'$ allora $S \sqsubseteq_{\mathrm{R}} S'$. Comunque, il viceversa, in generale, non vale, in quanto le regole stabiliscono delle relazioni di inclusione tra le interpretazioni dei tipi che fanno sorgere *nuove* relazioni di sussunzione. Intuitivamente, come viene mostrato nell'esempio di Figura 2, se $S \sqsubseteq_\sigma S^a$ e $S^c \sqsubseteq_\sigma S'$ allora $S \sqsubseteq_{\mathrm{R}} S'$.

La relazione esistente tra $\sqsubseteq_{\mathrm{R}}$ e $\sqsubseteq_\sigma$ può essere espressa formalmente attraverso la nozione di *espansione semantica*.

Definizione 3 (Espansione Semantica) *Dato uno schema con regole* $(\sigma, \mathbf{R})$ *e un tipo* $S \in \mathbf{S}$*, l'*espansione semantica *di* S *rispetto a* $\mathbf{R}$*, EXP(S), è un tipo di* $\mathbf{S}$ *tale che:*

1. $EXP(S) \simeq_{\mathrm{R}} S$*;*
2. *per ogni* $S' \in \mathbf{S}$ *tale che* $S' \simeq_{\mathrm{R}} S$ *si ha* $EXP(S) \sqsubseteq_\sigma S'$.

In altri termini, EXP(S) è il tipo più specializzato (rispetto alla relazione $\sqsubseteq_\sigma$) tra tutti i tipi $\simeq_{\mathrm{R}}$-equivalenti al tipo S.

L'espressione $EXP(S)$ permette di esprimere la relazione esistente tra $\sqsubseteq_{\mathrm{R}}$ e $\sqsubseteq_\sigma$: per ogni $S, S' \in \mathbf{S}$ si ha $S \sqsubseteq_{\mathrm{R}} S'$ se e solo se $EXP(S) \sqsubseteq_\sigma S'$. Questo significa che, dopo aver determinato l'espansione semantica, anche la relazione di sussunzione nello schema con regole può essere calcolata tramite l'algoritmo presentato in [BN94b].

È facile verificare che, per ogni $S \in \mathbf{S}$ e per ogni $R \in \mathbf{R}$, se $S \sqsubseteq_\sigma (p : S^a)$ allora $S \sqcap (p\colon S^c) \simeq_{\mathrm{R}} S$. Questa trasformazione di S in $S \sqcap (p\colon S^c)$ è la base del calcolo della $EXP(S)$: essa viene effettuata iterativamente, tenendo conto che l'applicazione di una regola può portare all'applicazione di altre regole. Per individuare tutte le possibili trasformazioni di un tipo implicate da uno schema con regole $(\sigma, \mathbf{R})$, si definisce la funzione totale $\Gamma : \mathbf{S} \longrightarrow \mathbf{S}$, come segue:

$$\Gamma(S) \quad = \quad \begin{cases} S \sqcap (p : S^c) & \text{se esistono } R \text{ e } p \text{ tali che } S \sqsubseteq_\sigma (p\colon S^a) \text{ e } S \not\sqsubseteq_\sigma (p\colon S^c) \\ S & \text{altrimenti} \end{cases}$$

e poniamo $\tilde{\Gamma} = \Gamma^i$, dove i è il più piccolo intero tale che $\Gamma^i = \Gamma^{i+1}$. L'esistenza di i è garantita dal fatto che il numero di regole è finito e una regola non può essere applicata più di una volta con lo stesso cammino ($S \not\sqsubseteq_\sigma (p\colon S^c)$). Si può dimostrare che, per ogni $S \in \mathbf{S}$, EXP(S) è effettivamente calcolabile tramite $\tilde{\Gamma}(S)$.

2.1.3 Inferenza ed ottimizzazione: un esempio

Riprendiamo l'esempio descritto precedentemente e vediamo a quali vantaggi pratici può portare l'uso della logica OLCD.
L'attività di inferenza iniziale da parte del sistema consiste nel determinare l'espansione semantica di ogni classe dello schema. Questo permette di rilevare relazioni *isa* non esplicitate nella definizione dell classi stesse. Ad esempio, nel calcolo dell'espansione semantica della classe `DMaterial` si rileva che tale classe è sussunta (implica) l'antecedente della regola R1: congiungendone quindi il conseguente si determina che `DMaterial` è una specializzazione di `SMaterial`. Nel caso in cui vengano applicate più regole durante l'espansione semantica di una classe, le relazioni *isa* ricavate sono meno ovvie di quella appena descritta. Ad esempio, nel calcolo dell'espansione semantica della classe `DStorage` viene applicata prima la regola R1 e successivamente la regola R2, concludendo che `DStorage` è una specializzazione di `SStorage`.

L'altra componente del sistema, ODB-QOptimizer, si occupa invece dell'ottimizzazione semantica delle interrogazioni a run-time, anch'essa basata sull'espansione semantica, in modo da specializzare le classi interessate dall'interrogazione. Prendiamo ad esempio la seguente query, espressa in *OQL*, che vuole selezionare dallo schema tutti i magazzini che contengono materiali che hanno tutti rischio maggiore o uguale a 15.

```
select   * from Storage S
where    for all X in S.stock : (  X.item in Material and
                                   X.item.risk ≥ 15 )
```

L'espansione semantica di quest'interrogazione, ottenuta trasformando la query stessa in un classe virtuale OLCD ed applicando prima la regola R1 e successivamente la regola R2, porta alla seguente interrogazione equivalente:

```
select   * from SStorage S
where    for all X in S.stock : (  X.item in SMaterial and
                                   X.item.risk ≥ 15 )
```

L'esempio mostra un'effettiva ottimizzazione della query, indipendente da un qualsiasi modello di costo: sostituendo `Storage` con `SStorage` e `Material` con `SMaterial` si riduce l'insieme di oggetti da controllare per individuare il risultato dell'interrogazione.

2.2 Architettura e funzionalità di ODB-Tools

ODB-Tools, sviluppato al Dipartimento di Scienze dell'Ingegneria dell'Università di Modena, è un prototipo software per la validazione di schemi e l'ottimizzazione di interrogazioni in ambiente OODB.

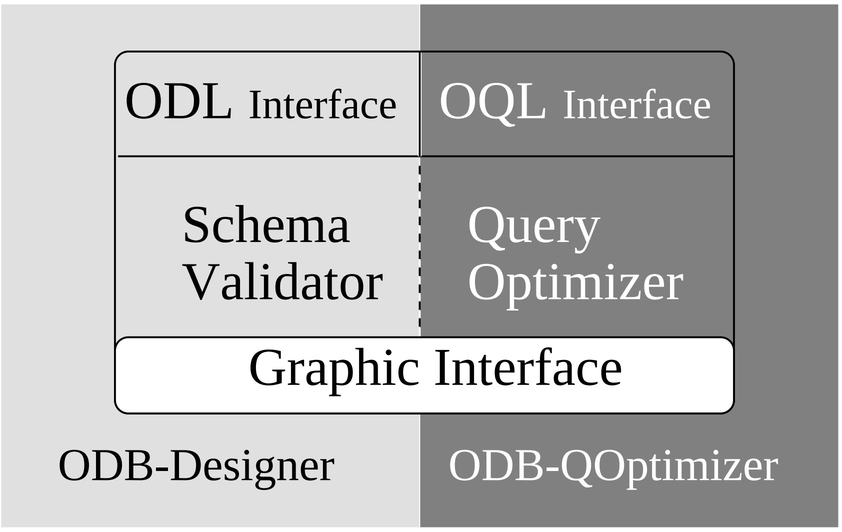


Figura 3: ODB-Tools

ODB-Tools è dotato di un nucleo che interpreta e supporta la logica descrittiva OLCD ed esegue le funzioni: controllo di coerenza di descrizioni OLCD, calcolo della sussunzione ed espansione semantica di un tipo [BBB+95, SBV95, CA97a, BSBV97].
L'architettura, mostrata in figura 3, presenta i vari moduli integrati che definiscono un ambiente *user-friendly* basato sul linguaggio standard ODMG-93. L'utente inserisce gli schemi in linguaggio ODL e le query in OQL ottenendo come risultato la validazione dello schema, l'ottimizzazione dell'interrogazione (in OQL) e la rappresentazione grafica della gerarchia di ereditarietà e di aggregazione dello schema. ODB-Tools fornisce i seguenti servizi:

- *validazione di schemi* (ODB-Designer): l'utente può inserire la descrizione di uno schema di una base di dati, usando il linguaggio ODL, e il sistema realizza automaticamente la validazione dello schema (verificando che non esistano classi incoerenti, ovvero che non possono essere popolate da alcun oggetto) e la sua riclassificazione (determinando eventuali relazioni di specializzazione non esplicitate dallo schema stesso). Al suo interno è quindi presente un'interfaccia tra ODL_{I^3} e OLCD;

- *ottimizzazione semantica delle interrogazioni* (ODB-QOptimizer): l'utente può inserire una query, in OQL, posta su uno schema predefinito, e questa viene automaticamente riformulata attraverso il procedimento precedentemente descritto. Fa uso di un'interfaccia di traduzione da OQL a OLCD e viceversa.

Considerazioni Un'importante osservazione riguarda la **versatilità** degli ODB-Tools: potrebbe infatti sembrare che il sistema, alla cui base sono delle complesse teorie e ricerche nel campo della logica, sia unicamente finalizzato alla validazione degli schemi ad oggetti ed all' ottimizzazione semantica delle query. In realtà, grazie proprio al completo impianto teorico alla base di questi strumenti, ODB-Tools può essere impiegato per realizzare altre funzioni purchè la conoscenza da elaborare sia esprimibile (e tradotta in modo automatico) in OLCD.

Gli strumenti software che si basano sull'uso della logica descrittiva OLCD, e che sono stati sviluppati presso il Dipartimento di Scienze dell'Ingegneria dell'Università di Modena, sono visibili ed utilizzabili attraverso il sito web *http://www.sparc20.dsi.unimo.it*.

3 Estensione di OLCD e ODB-Tools per l'integrazione di informazioni

Obiettivo principale del progetto INTERDATA, TEMA 3 è quello di fornire funzioni per l'integrazione di dati provenienti da sorgenti eterogenee ed assicurare la consistenza delle informazioni recuperate. L'approccio seguito propone un sistema, chiamato MOMIS (Mediator envirOnment for Multiple Information Sources), che si ispira ad altre architetture già studiate e sviluppate (ad esempio TSIMMIS di Stanford [GMea95, YAGM96]): in particolare si è ripresa l'idea dei *Wrapper*, sistemi incaricati di tradurre i dati delle sorgenti rendendoli omogenei ad un'interfaccia comune, e dei *Mediator*, che si occupano di combinare, integrare ed arricchire gli schemi ed i modelli delle sorgenti e di fornire facilitazioni e supporto per l'interrogazione e la generazione delle risposte.

I principali problemi nella realizzazione di queste architetture si riscontrano in relazione ai seguenti aspetti:

- nello sviluppo di generatori di Wrapper, che permettano di sviluppare velocemente Wrapper ad hoc per ogni tipo di sorgente (relazionale, ad oggetti, file system, semistrutturata, ...) in grado di interpretare il modello logico dei dati della sorgente e di convertirlo in un modello standard, di tradurre la query nel linguaggio di interrogazione della sorgente e di convertire i dati in risposta nel formato comune (ad esempio da tupla ad oggetto);
- nello sviluppo del Mediatore che a partire dal modello standard dei dati riesca a realizzare lo schema integrato in modo automatico o almeno semiautomatico secondo una procedura standardizzata che sia riutilizzabile;
- nella generazione di un modulo che fornisca le funzionalità di *Decomposizione delle Interrogazioni* e di *Fusione delle Risposte* che fornisca risposte corrette e complete. La correttezza è legata alla fase di fusione delle istanze, la completezza all'ottenimento dell'insieme di oggetti strettamente necessari: si devono selezionare efficacemente fra le grandi moli di dati e di sorgenti quelle strettamente necessarie a dare una risposta esauriente evitando quelle inutili.

La base teorica dell'approccio proposto nell'architettura del Mediatore si fonda sullo standard ODMG93, su ODB-Tools, un componente basato sulla logica descrittiva [BBSV97a, BBSV97b] e sull'architettura presentata in TSIMMIS [YAGM96, GMea95]. In particolare ODM ed ODL di ODMG93 sono presi come modello dei dati e linguaggio di descrizione degli schemi interni e quindi rappresentano lo standard di riferimento dei Wrapper. Inoltre ODL viene qui esteso per renderlo in grado di esprimere conoscenza non solo intensionale ed ontologica ma anche estensionale e vincoli, attraverso relazioni terminologiche e rule. OQL è stato limitato in modo che le query siano traducibili anche su sorgenti non a oggetti assumendo il linguaggio proposto come standard dal progetto DARPA I^3 [ARP].
Nella sottosezione successiva viene presentata una prima proposta di architettura di un sistema di integrazione.

3.1 Architettura e funzionalità del sistema MOMIS

Il sistema di integrazione proposto mantiene la struttura classica di un sistema I^3. In Figura 4 è mostrata l'architettura del sistema MOMIS: i livelli raffigurativi sono tre:

1. **Wrapper:** si posizionano a livello delle sorgenti costituendone l'interfaccia col Mediatore vero e proprio, perciò devono essere appositamente sviluppati per il tipo di sorgente che

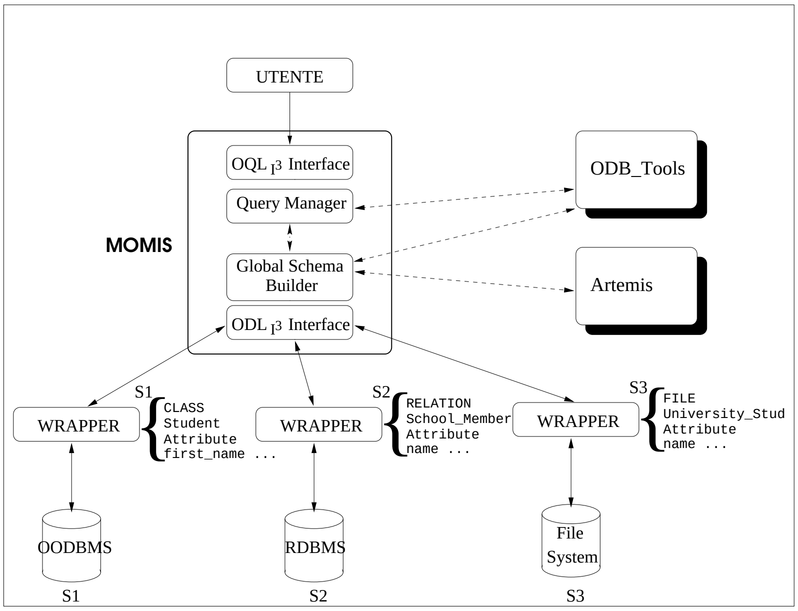


Figura 4: Architettura di MOMIS

andranno a servire. Le sorgenti sono le fonti di informazioni da integrare: possono essere dei database tradizionali (sia ad oggetti, sia basati sul modello relazionale), oppure dei semplici file system. A differenza di altri progetti analoghi, in questa prima fase del progetto, non si è ipotizzato di avere a che fare con dati di tipo multimediale o semi-strutturati. La funzione è dei Wrapper è duplice:

- in fase di integrazione forniscono, consultando il catalogo della sorgente, la descrizione degli schemi nel linguaggio ODL$_{I^3}$ (descritto in Sezione 3.2);
- in fase di query processing, devono tradurre la query ricevuta, espressa nel linguaggio comune di interrogazione OQL$_{I^3}$, in una interrogazione comprensibile (e realizzabile) dalla sorgente stessa. Devono inoltre esportare i dati ricevuti in risposta all'interrogazione, presentandoli al mediatore attraverso il modello comune di dati utilizzato dal sistema;

2. **Mediatore:** è il cuore del sistema, ed è composto da diversi sottomoduli che si occupano dei diversi stadi del processo di integrazione:

   - Global Schema Builder: è il modulo di integrazione degli schemi locali e di manipolazione delle conoscenze estensionali. Partendo dalle descrizioni delle sorgenti, espresse in ODL$_{I^3}$, genera un unico schema globale da presentare all'utente, traducendo la conoscenza estensionale in una tassonomia. Questa fase di integrazione, realizzata in modo semi-automatico dall'interazione del progettista del sistema, fa uso degli ODB-Tools, delle tecniche di clustering e di quelle di fusione delle gerarchie;
   - Query Manager: è il modulo di gestione delle interrogazioni. Provvede a gestire la query dell'utente, deducendo, in base alle conoscenze intensionali ed estensionali generate dal Global Schema Builder, un insieme di sottoquery da spedire alle fonti locali. In fase di ritorno delle informazioni provvede a *ricomporre* i dati ricevuti per mostrarli in forma univocae coerente . Tra i suoi compiti si evidenzia l'ottimizzazio-

ne semantica delle interrogazioni, realizzata, come la fase di *Query Decomposition*, utilizzando gli ODB-Tools.

- ODL_{I^3} interface: è l'interfaccia che serve al Mediatore per comunicare con i Wrapper in maniera uniforme, utilizzando il modello comune.
- OQL_{I^3} interface: è l'interfaccia che serve all'utente per interrogare il Mediatore essendo consistente con ODL_{I^3}. Essendo una riduzione di OQL non genera alcuna complicazione nè per l'utente nè per il sistema.

3. **Utente:** è l'utilizzatore del sistema, colui che interagisce con il Mediatore. Ha la visione della conoscenza attraverso lo schema globale, all'interno del quale sono rappresentati tutti i concetti interrogabili: è come se si trovasse di fronte ad un unico database ad oggetti (o relazionale), da interrogare in modo tradizionale, mentre le sorgenti locali rimangono completamente trasparenti.

Lo scopo che ci si è preposti con il Progetto MOMIS è la realizzazione di un sistema di mediazione che, a differenza di molti altri progetti analizzati, contribuisca a fornire un'integrazione delle sorgenti che faciliti l'interrogazione per gli utenti, senza comprometterne la libertà od inficiare la completezza e la sinteticità della risposta.

Per realizzare l'integrazione degli schemi, vengono accoppiate le tecniche di integrazione basate su clustering con le tecniche di *intelligenza artificiale* di ODB-Tools, in modo da diminuire il più possibile l'apporto manuale del progettista del sistema.

L'approccio *semantico* utilizzato si articola nei seguenti punti:

1. *Generazione del Thesaurus di relazioni terminologiche:* attraverso l'interazione col progettista, che pone relazioni terminologiche e l' utilizzo di ODB-Tools, si cerca di derivare in maniera semi_automatica nuove relazioni di sinonimia e corrispondenza tra i termini (nomi di classi e di attributi) delle diverse sorgenti. Questo passo viene realizzato da un opportuno modulo realizzato presso l'Università di Modena;

2. *Analisi delle affinità* ***intensionali*** *fra le classi:* questa fase è realizzata considerando le relazioni terminologiche memorizzate nel Thesaurus ed i *coefficienti di affinità* in [CA97b]. Questo passo, ed il successivo *3.*, viene realizzato dal componente ARTEMIS, sviluppato presso l'Università di Brescia;

3. *Creazione dei Cluster*, cioè di raggruppamenti di classi affini: si tratta di classi intensionalmente affini per le quali si presume esista anche una qualche sovrapposizione fra le estensioni;

4. *Generazione dello Schema Globale del Mediatore:* da ogni Cluster si definisce una Classe Globale la cui estensione è costituita dall'unione delle estensioni delle classi sorgenti che costituiscono il cluster, mentre l'intensione è ricavata dall'unione "ragionata" degli attributi delle stesse. Questa fase porta alla definizione ODL_{I^3} delle Classi Globali con le regole di mapping fra gli attributi dalla Casse Globale alle classi sorgenti.

Risultati preliminari di questo approccio sono stati presentati in [BCdV$^+$98a, BCdV$^+$98b].

3.2 Analisi delle estensioni alla logica descrittiva OLCD

Come visto precedentemente, il sistema ODB-Tools, basato sulla logica descrittiva OLCD, è aperto verso gli utenti esterni attraverso il linguaggio ODL dello standard ODMG-93 (esteso co regole *if then*).

Uno degli obiettivi di questo rapporto è quello di proporre le nuove primitive del linguaggio di input di ODB-Tools al fine ottenere un nuovo linguaggio, che chiameremo ODL_{I^3}, in grado di rappresentare la conoscenza necessaria al processo di integrazione delle informazioni e di studiare le corrispondenti estensioni alla logica descrittiva OLCD.
Il risultato di questa analisi preliminare porta a ritenere sufficiente, in presenza di sorgenti eterogenee strutturate, l'attuale logica descrittiva OLCD utilizzata con gli ooportuni accorgimenti mostrati in seguito. L'eventuale impatto dovuto a sorgenti semi-strutturate verrà analizzato in un prossimo rapporto.

Punto di partenza per la definizione del linguaggio ODL_{I^3} è stata la proposta di standardizzazione per i linguaggi di mediazione [BRU96]. Partendo da questa proposta (secondo la quale i diversi sistemi di mediazione avrebbero potuto supportare sorgenti con modelli complessi, come quelli ad oggetti, e sorgenti molto più semplici, come file di strutture), si è cercato di discostarsi il meno possibile dal linguaggio ODL, a sua volta proposto dal gruppo di standardizzazione ODMG-93.

È stato quindi definito il linguaggio ODL_{I^3} come una estensione allo standard ODL, perseguento le seguenti finalità (per ciascuna viene analizzato l'impatto sulla logica descrittiva OLCD):

- *dichiarare relazioni terminologiche fra nomi di classi ed attributi*:

⟨relationships_list⟩	::=	⟨relationship_dcl⟩**;** | ⟨relationship_dcl⟩**;** ⟨relationships_list⟩
⟨relationships_dcl⟩	::=	⟨local_attr_name⟩ ⟨relationship_type⟩ ⟨local_attr_name⟩
⟨relationship_type⟩	::=	**syn** | **bt** | **nt** | **rt**
...		

  Le relazioni terminologiche sono definite, durante il processo di integrazione, nella fase di generazione del *Thesaurus*. La logica descrittiva OLCD viene utilizzata per inferire nuove relazioni terminologiche sui termini, secondo l'assunzione di equivalenza, a livello intensionale, tra le seguenti relazioni terminologiche e tassonomiche[4]:

A **bt** B	equivalente alla relazione	$B \prec A$
A **nt** B	equivalente alla relazione	$A \prec B$
A **syn** B	equivalente alla relazione	$A \prec B$
		$B \prec A$
A **rt** B	equivalente alla relazione	$A \longrightarrow B$

  In fase di traduzione dal linguaggio ODL_{I^3} alla logica descrittiva OLCD si compie l'armonizzazione delle descrizioni di classe per rendere effettive le equivalenze terminologiche e tassonomiche. Ad esempio, la relazione A **bt** B comporta la definizione di una relazione di *ereditarietà* tra la classe B e la classe A.
  Invece, la relazione A **rt** B comporta l'introduzione di un attributo nella classe A con dominio la classe B.

- *descrivere tutte le fonti di informazioni in maniera uniforme*, come depositi di oggetti, e quindi utilizzare il concetto di *classe*, indipendentemente dal modello originale utilizzato[5]: integrare database relazionali, ad oggetti, e file system. Si illustra di seguito l'estensione ODL per esprimere il tipo della sorgente ed altri concetti propri del modello originario:

[4] La notazione $\longrightarrow$ indica una relazione di aggregazione tra le classi A e B

[5] Sarà compito del Wrapper provvedere alla traduzione dal modello originale di descrizione all'ODL_{I^3}, aggiungendo eventualmente in questa descrizione tutte le informazioni necessarie al Mediatore (il nome e il tipo della sorgente,...)

⟨interface_dcl⟩	::=	⟨interface_header⟩ {[⟨interface_body⟩]};
⟨interface_header⟩	::=	**interface** ⟨identifier⟩
		[⟨inheritance_spec⟩]
		[⟨type_property_list⟩]
⟨inheritance_spec⟩	::=	: ⟨scoped_name⟩ [,⟨inheritance_spec⟩]
⟨type_property_list⟩	::=	**(** [⟨source_spec⟩] [⟨extent_spec⟩]
		[⟨key_spec⟩] [⟨f_key_spec⟩] **)**
⟨source_spec⟩	::=	**source** ⟨source_type⟩ ⟨source_name⟩
⟨source_type⟩	::=	**relational** | **nfrelational** | **object** | **file**
⟨source_name⟩	::=	⟨identifier⟩
⟨extent_spec⟩	::=	**extent** ⟨extent_list⟩
⟨extent_list⟩	::=	⟨string⟩ | ⟨string⟩ **,** ⟨extent_list⟩
⟨key_spec⟩	::=	**key**[**s**] ⟨key_list⟩
⟨f_key_spec⟩	::=	**foreign_key** ⟨f_key_list⟩

...

Ciascuna **interface** coincide, in logica OLCD, ad una classe dello schema. L'unica attenzione particolare la si pone alla definizione di **foreign_key** che equivale ad una asserzione terminologica **rt** tra le due classi coinvolte.

- *supportare la dichiarazione di vincoli di integrità sottoforma di if then rule*, siano esse definite sugli schemi locali (e magari da questi ricevute), siano esse riferite allo Schema Globale, e quindi inserite dal progettista. Vediamo quindi l'estensione di ODL per l' espressione *dichiarativa* dei vincoli:

⟨rule_list⟩	::=	⟨rule_dcl⟩**;** | ⟨rule_dcl⟩**;** ⟨rule_list⟩
⟨rule_dcl⟩	::=	**rule** ⟨identifier⟩ ⟨rule_pre⟩ **then** ⟨rule_post⟩
⟨rule_pre⟩	::=	⟨forall⟩ ⟨identifier⟩ **in** ⟨rule_identifier⟩ **:** ⟨rule_body_list⟩
⟨rule_identifier⟩	::=	⟨identifier⟩ | **(** ⟨identifier⟩ **and** ⟨identifier⟩ **)**
⟨rule_post⟩	::=	⟨rule_body_list⟩
⟨rule_body_list⟩	::=	**(** ⟨rule_body_list⟩ **)** | ⟨rule_body⟩ |
		⟨rule_body_list⟩ **and** ⟨rule_body⟩ |
		⟨rule_body_list⟩ **and** **(** ⟨rule_body_list⟩ **)**
⟨rule_body⟩	::=	⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨literal_value⟩ |
		⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨rule_cast⟩ ⟨literal_value⟩ |
		⟨dotted_name⟩ **in** ⟨dotted_name⟩ |
		⟨forall⟩ ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩ |
		exists ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩
⟨rule_const_op⟩	::=	= | ≥ | ≤ | > | <
⟨rule_cast⟩	::=	**(**⟨simple_type_spec⟩**)**
⟨dotted_name⟩	::=	⟨identifier⟩ | ⟨identifier⟩**.**⟨dotted_name⟩
⟨forall⟩	::=	**for all** | **forall**

I vincoli di integrità sono già parte della definizione di schema con regole della logica OLCD.

- *supportare la dichiarazione di regole di mediazione, o mapping rule, utilizzate per meglio specificare l'accoppiamento tra i concetti globali e i concetti locali originali*;

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

...

Le regole di *mapping* sono di supporto all'azione del *Query Manager* per la riscrittura finale delle query; essendo il risultato della fase di ragionamento che porta alla definizione dello schema integrato (in cui si utilizza OLCD) non sono tradotte in logica OLCD.

Essendo ODL$_{I^3}$ tradotto automaticamente nella logica descrittiva OLCD, si consente l'utilizzo di una *semantica di mondo aperto*, che permette di effettuare i controlli di consistenza e l'ottimizzazione semantica delle interrogazioni, forniti da ODB-Tools.

Una ulteriore importante funzionalità nel processo di integrazione è la possibilità di esprimere, sotto forma di proposizioni, relazioni estensionali tra gli oggetti di classi distinte.

Tali relazioni, che chiameremo *assiomi estensionali*, esprimono le relazioni insiemistiche tra le estensioni di classi definite in sorgenti autonome (indipendentemente dalla loro intensione).

Ogni classe dello schema integrato avrà (o potrà avere) associato gli assiomi estensionali che predicano le relazioni di *inclusione*, *equivalenza* e *disgiunzione*.

Gli assiomi possono essere espressi come rule nel linguaggio ODL$_{I^3}$ (e quindi tradotte in logica OLCD), senza ulteriori estensioni alla sintassi e senza creare complicazioni all'utilizzatore. La ragione che consente di ricorrere alla sintassi delle rule risiede nel fatto che assiomi estensionali e rule sono semanticamente equivalenti. Attraverso le rule, infatti, possiamo asserire che un insieme di istanze di un dato concetto C1, che godono di certe proprietà, appartengono ad un altro concetto C2, vincolando così delle relazioni estensionali a proprietà intensionali. Gli assiomi, anche se indipendenti dalle caratteristiche intensionali, aspirano a *ridefinire* i tipi delle istanze esistenti: dire che A e B hanno estensioni disgiunte significa dire che non esistono tipi di istanze con le proprietà e di A e di B, mentre affermare che A e B hanno estensioni equivalenti significa che tutte le istanze di A sono anche istanze di B e tali istanze hanno le proprietà sia di A sia di B.

È stato pensato di esprimere le relazioni di disgiunzione, inclusione ed equivalenza fra le estensioni, sottintendendo l'intersezione in quanto si presume che classi tra loro affini abbiano almeno estensioni la cui intersezione non è nulla. In particolare avremo:

1. A e B sono estensioni disgiunte: $A \cap B = \emptyset$

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

2. B è inclusa in A: $B \subseteq A$

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

3. 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
   ```

Sorgente University (S_1)

```
University_Worker(first_name,last_name, dept_code,pay)
Research_Staff(first_name,last_name,relation,email,
               dept_code,section_code,pay)
School_Member(first_name,last_name,faculty,year)
Department(dept_name,dept_code,budget,dept_area)
Section(section_name,section_code,length,room_code)
Room(room_code,seats_number,notes)
```

Sorgente Computer_Science (S_2)

```
CS_Person(name)
Professor:CS_Person(title,belongs_to:Division,rank)
Student:CS_Person(year,takes:set⟨Course⟩,rank)
Division(description,address:Location,fund,sector,employee_nr)
Location(city,street,number,county)
Course(course_name,taught_by:Professor)
```

Sorgente Tax_Position (S_3)

```
University_Student(name,student_code,faculty_name,tax_fee)
```

Figura 5: Esempio di Riferimento: Sorgenti dell'Università

Le descrizioni ODL$_{I^3}$ di tutte le classi da integrare sono fornite al Mediatore dai Wrapper: da sottolineare che le descrizioni ricevute rappresentano tutte e sole le classi che una determinata sorgente vuole mettere a disposizione del sistema. Non è quindi detto che lo schema locale ricevuto dal mediatore rappresenti l'intera sorgente, bensì ne descrive il sottoinsieme di informazioni che possono essere rese accessibili ad un utente esterno.

3.3 Esempio applicativo

Introduciamo ora brevemente un esempio di descrizione di schemi in linguaggio ODL$_{I^3}$. L'esempio si riferisce ad una realtà universitaria: le sorgenti da integrare sono tre (viene rappresentato anche in modo schematico in Figura 5).
La prima sorgente, `University` (S_1), è un database di tipo relazionale e contiene informazioni sui dipendenti e sugli studenti di una determinata università. È composta da sei tabelle: `University_Worker`, `Research_Staff`, `School_Member`, `Department`, `Section` e `Room`. Per ogni professore (presente nella tabella `Research_Staff`), sono memorizzate informazioni sul suo dipartimento (attraverso la foreign key `dept_code`), sul suo indirizzo di posta elettronica (`email`), e sul corso da lui tenuto (`section_code`). Per il corso, viene memorizzata pure l'aula (`Room`) dove questo si svolge, mentre del dipartimento sono descritti, oltre al nome (`dept_name`) ed al codice (`dept_code`), il budget (`budget`) che ha a disposizione e l'area (`dept_area`) a cui appartiene, sia essa Scientifica, Economica,... Per gli studenti presenti nella tabella `School_Member` sono invece mantenuti il nome (nella coppia `first_name` e `last_name`), la facoltà di appartenenza (`faculty`) e l'anno di corso (`year`).
La sorgente `Computer_Science` (S_2) contiene invece informazioni sulle persone afferenti a questa facoltà, ed è un database ad oggetti. Sono presenti sei classi: `CS_Person`, `Professor`, `Student`, `Division`, `Location` e `Course`. I dati mantenuti sono comunque abbastanza simili a quelli della sorgente S_1: per quanto riguarda i professori, sono memorizzati il titolo (`title`), e la divisione di appartenenza (`belongs_to`), che a sua volta fa parte di un dipartimento (e ne può quindi essere considerata una specializzazione); per gli studenti sono memorizzati i corsi seguiti (`takes`) e

l'anno di corso (year). Il corso ha poi un attributo complesso che lo lega al professore che ne è titolare (taught_by), mentre per la divisione si tiene l'indirizzo (address), i fondi (fund) e il numero di impiegati (employee_nr).
È presente inoltre una terza sorgente, Tax_Position (S_3), facente capo alla segreteria studenti, che mantiene i dati relativi alle tasse da pagare (tax_fee). In questo caso (S_3), non si tratta di un database ma di un file system, che contiene quindi semplici tracciati record.

Di seguito riportiamo, come esempio, la descrizione fornita dal wrapper per le classi School_Member e Computer_Science:

```
interface School_Member
( source relational University
  extent School_Member
  key name )
{ attribute string first_name;
  attribute string last_name;
  attribute string faculty;
  attribute integer year; };
```

```
interface Student : CS_Person
( source object Computer_Science
  extent Students )
{ attribute integer year
  attribute set<Course> takes;
  attribute string rank; };
```

La nuova definizione aggiunge, dopo il nome della classe, le informazioni sul tipo e sul nome della sorgente di origine, sulla sua estensione, sugli attributi chiave, ed eventualmente sulle foreign key. Nell'esempio sono riportate una classe Student derivata da una sorgente ad oggetti (Computer_Science) ed una classe School_Member proveniente da una sorgente relazionale, rappresentata quindi originariamente da una tabella di tuple, con attributo chiave name.

Come già visto in sezione 3.1, è possibile esplicitare relazioni terminologiche tra classi (o attributi) di sorgenti diverse. Ad esempio è possibile esprimere la sinonimia tra le classi Section e Course, o una relazione di ipernimia tra gli attributi first_name della classe Research_Staff e name della classe CS_Person nel seguente modo:

```
University.Section syn Computer_Science.Course
Computer_Science.CS_Person.name bt University.Research_Staff.first_name
```

Per quanto riguarda la definizione di *mapping rules*, vediamo l'esempio seguente:

```
interface University_Person
(extent University_Worker, Research_Staffer, School_Member,
        CS_Person, Professor, Student, University_Student
 key    name)
{ attribute string name
     mapping_rule  (University.Research_Staff.first_name and
                   University.Research_Staff.last_name)
                   (University.School_Member.first_name and
                   University.School_Member.last_name),
                   Computer_Science.CS_Person.name
                   Computer_Science.Professor.name
                   Computer_Science.Student.name
                   Tax_Position.University_Student.name;
  attribute string rank
     mapping_rule  University.Research_Staff = 'Professor',
                   University.School_Member = 'Student',
     ... }
```

Tale dichiarazione permette la definizione di una *classe globale* chiamata `University_Person` che rappresenta l'integrazione delle classi sorgenti descritte nella parte `extent` (`University_Worker, Research_Staffer, School_Member, CS_Person, Professor, Student, University_Student`). Inoltre, per ciascun attributo della *classe globale* è possibile assegnare i collegamenti agli attributi delle classi locali componenti, in modo da definire il mapping tra lo schema integrato e gli schemi originali.

Infine è possibile introdurre relazioni estensionali che forniscono relazioni insiemistiche tra gli oggetti delle classi locali. Con riferimento alla classe globale `University_Person` introdotta alcuni assiomi estensionali possono essere:

1. `School_Member` e `University_Student` hanno estensioni equivalenti
2. l'estensione di `Professor` è un sottoinsieme di quella di `Research_Staff`
3. le estensioni delle classi `Student` e `Professor` sono disgiunte

Tali assiomi posso essere inseriti nel sistema attraverso i seguenti vincoli di integrità:

```
rule RE1a forall x in School_Member then x in University_Student;
rule RE1b forall x in University_Student then x in School_Member;
rule RE2 forall x in Professor then x in Research_Staff;
rule RE3 forall x in (Professor and Student) then x in bottom;
```

3.4 Il linguaggio descrittivo ODL$_{I^3}$

Riportiamo quindi la sintassi BNF del linguaggio ODL$_{I^3}$. Essendo, ODL$_{I^3}$, una estensione del linguaggio standard ODL, si riportano solo le parti che differiscono dall'ODL originale, rimandando invece a quest' ultimo per le parti in comune.

⟨interface_dcl⟩	::=	⟨interface_header⟩ {[⟨interface_body⟩]};
⟨interface_header⟩	::=	**interface** ⟨identifier⟩
		[⟨inheritance_spec⟩]
		[⟨type_property_list⟩]
⟨inheritance_spec⟩	::=	: ⟨scoped_name⟩ [,⟨inheritance_spec⟩]
⟨type_property_list⟩	::=	**(** [⟨source_spec⟩] [⟨extent_spec⟩]
		[⟨key_spec⟩] [⟨f_key_spec⟩] **)**
⟨source_spec⟩	::=	**source** ⟨source_type⟩ ⟨source_name⟩
⟨source_type⟩	::=	**relational** | **nfrelational** | **object** | **file**
⟨source_name⟩	::=	⟨identifier⟩
⟨extent_spec⟩	::=	**extent** ⟨extent_list⟩
⟨extent_list⟩	::=	⟨string⟩ | ⟨string⟩ **,** ⟨extent_list⟩
⟨key_spec⟩	::=	**key**[**s**] ⟨key_list⟩
⟨f_key_spec⟩	::=	**foreign_key** ⟨f_key_list⟩
...		

Si veda ora la sintassi proposta per la dichiarazione dei mapping fra i nomi delle classi globali e quelli nelle sorgenti:

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

La seguente sintassi è stata introdotta per definire le relazioni terminologiche:

⟨relationships_list⟩ ::= ⟨relationship_dcl⟩**;** | ⟨relationship_dcl⟩**;** ⟨relationships_list⟩
⟨relationships_dcl⟩ ::= ⟨local_attr_name⟩ ⟨relationship_type⟩ ⟨local_attr_name⟩
⟨relationship_type⟩ ::= **syn** | **bt** | **nt** | **rt**
...

Di seguito si mostra la sintassi per la dichiarazione di vincoli e rule estensionali:

⟨rule_list⟩ ::= ⟨rule_dcl⟩**;** | ⟨rule_dcl⟩**;** ⟨rule_list⟩
⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_pre⟩ **then** ⟨rule_post⟩
⟨rule_pre⟩ ::= ⟨forall⟩ ⟨identifier⟩ **in** ⟨rule_identifier⟩ **:** ⟨rule_body_list⟩
⟨rule_identifier⟩ ::= ⟨identifier⟩ | **(** ⟨identifier⟩ **and** ⟨identifier⟩ **)**
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨rule_body_list⟩ ::= **(** ⟨rule_body_list⟩ **)** | ⟨rule_body⟩ | ⟨rule_body_list⟩ **and** ⟨rule_body⟩ | ⟨rule_body_list⟩ **and (** ⟨rule_body_list⟩ **)**
⟨rule_body⟩ ::= ⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨literal_value⟩ | ⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨rule_cast⟩ ⟨literal_value⟩ | ⟨dotted_name⟩ **in** ⟨dotted_name⟩ | ⟨forall⟩ ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩ | **exists** ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩
⟨rule_const_op⟩ ::= $= \mid \geq \mid \leq \mid > \mid <$
⟨rule_cast⟩ ::= **(**⟨simple_type_spec⟩**)**
⟨dotted_name⟩ ::= ⟨identifier⟩ | ⟨identifier⟩**.**⟨dotted_name⟩
⟨forall⟩ ::= **for all** | **forall**

Riferimenti bibliografici

[ARP] Arpa i^3 reference architecture. Available at http://www.isse.gmu.edu/I3_Arch/index.html.

[BB96] D. Beneventano and S. Bergamaschi. Incoherence and subsumption for recursive views and queries in object-oriented data models. *Data & Knowledge Engineering*, 1996. To appear.

[BB97] D. Beneventano and S. Bergamaschi. Incoherence and subsumption for recursive views and queries in object-oriented data models. *Data & Knowledge Engineering*, 21, 1997.

[BBB+95] J.P. Ballerini, D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. A semantics-driven query optimizer for oodbs. In *Int. Workshop on Description Logics*, Rome, Italy, June 1995.

[BBS93] D. Beneventano, S. Bergamaschi, and C. Sartori. Taxonomic reasoning with cycles in LOGIDATA+. In P. Atzeni, editor, *LOGIDATA+: Deductive Databases with Complex Objects*. Springer-Verlag, Heidelberg - Germany, 1993. Lecture Notes in CS - N. 701.

[BBS96] D. Beneventano, S. Bergamaschi, and C. Sartori. Semantic query optimization by subsumption in oodb. In *Proc. of the Int. Workshop on Flexible Query-Answering Systems (FQAS96)*, pages 167–185, Roskilde, Denmark, May 1996.

[BBSV97a] Domenico Beneventano, Sonia Bergamaschi, Claudio Sartori, and Maurizio Vincini. Odb-qoptimizer: a tool for semantic query optimization in oodb. In *Proc. of Int. Conf. on Data Engineering, ICDE'97*, Birmingham, UK, April 1997.

[BBSV97b] Domenico Beneventano, Sonia Bergamaschi, Claudio Sartori, and Maurizio Vincini. Odb-tools: a description logics based tool for schema validation and semantic query optimization in object oriented databases. In *Proc. of Int. Conference of the Italian Association for Artificial Intelligence (AI*IA97)*, Rome, 1997.

[BCdV+98a] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. An intelligent approach to information integration. In *Proc. of Int. Conf. on Formal Ontology in Information Systems, FOIS '98*, Trento, IT, June 1998.

[BCdV+98b] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. Exploiting schema knowledge for the integration of heterogeneous sources. In *Proc. of VI Convegno Nazionale Sistemi di Basi di Dati Evolute, SEDB98*, Ancona, IT, June 1998.

[BN94a] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.

[BN94b] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks and Complex Problem Solving Technologies*, 4:185–203, 1994.

[BRU96] P. Buneman, L. Raschid, and J. Ullman. Mediator languages - a proposal for a standard, April 1996. Available at ftp://ftp.umiacs.umd.edu/pub/ONRrept/medmodel96.ps.

[BSBV97] D. Beneventano, C.Sartori S. Bergamaschi, and M. Vincini. Odb-tools: a description logics based tool for schema validation and semantic query optimization in object oriented databases. In *Fifth Conference of the Italian Association for Artificial Intelligence (AI*IA97)*, Rome, Italy, 97.

[CA97a] S. Castano and V. De Antonellis. Semantic dictionary design for database interoperability. In *Proc. of Int. Conf. on Data Engineering, ICDE'97*, Birmingham, UK, April 1997.

[CA97b] S. Castano and V. De Antonellis. Semantic dictionary design for database interoperability. In *Int. Conf. on Data Engineering, ICDE'97*, Birmingham, UK, apr 1997.

[GMea95] H. Garcia-Molina and et al. The tsimmis approach to mediation: Data models and languages. In *NGITS workshop*, 1995. Available ftp://db.stanford.edu/pub/garcia/1995/tsimmis-models-languages.ps.

[LR89] C. Lecluse and P. Richard. Modelling complex structures in object-oriented databases. In *Symp. on Principles of Database Systems*, pages 362–369, Philadelphia, PA, 1989.

[OMG98] *"The Object Database Standard. Release 2.0"*. "Morgan Kauffman Inc.", "1998".

[SBV95] C.Sartori S. Bergamaschi and M. Vincini. Dl tecniques for intensional query answering in oodbs. In *2th KRDB Workshop Proceedings.*, Bielfield, Germany, 1995.

[Wea96] Gio Wiederhold and et al. *Integrating Artificial Intelligence and Database Technology*, volume 2/3. Journal of Intelligent Information Systems, June 1996.

[YAGM96] Y.Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *VLDB Int. Conf.*, Bombay, India, September 1996.