UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Facoltà di Ingegneria – Sede di Modena
Corso di Laurea Specialistica in Ingegneria Informatica

DATA INTEGRATION E DATA QUALITY NEGLI ECOSISTEMI DIGITALI

Candidato
ANDREA GALAVOTTI

Relatore
Chiar.ma Prof.ssa SONIA BERGAMASCHI

ANNO ACCADEMICO 2005 – 2006

PAROLE CHIAVE

NeP4B

DBE

Data Integration

Data Quality

Ontologie

Ringraziamenti

Desidero ringraziare la Professoressa Sonia Bergamaschi per la disponibilità e l'ottimismo dimostratomi durante la realizzazione di questa tesi. Desidero inoltre ringraziare Mirko e Antonio per il supporto fornitomi in questi mesi.

Un ringraziamento speciale va alla mia famiglia per il costante appoggio datomi nei cinque anni di studi universitari e a tutti i miei amici, in particolare al Drinkin' Team (Manuel, Giacomo, Lory, Pampuriella, Guido, Tilly, Elis, Loris, Bigna, Andrea B), Sandro, Lele e Paolo per tutti quei momenti in cui non mi hanno fatto pensare allo studio.

Voglio ringraziare infine tutti i miei compagni di corso, Elena, Mauri, Enri, Rivva, Pelle, Cave, Daniela, Mattia, Bompa, Tania, Mux, Matteo, Matte, Matteooooo!!, Mareo, Marisa, lo Squatter, Mirandla, From e Fabiana e tutti i ragazzi che mi hanno fatto compagnia nelle lunghe giornate passate in laboratorio: Mauro, Fede, Daniele e Biagio.

A tutti un sincero Grazie!!!

Andrea Galavotti

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1. Il Progetto Networked Peers For Business (NeP4B)

Il progetto *Networked Peers for Business*[1] (*NeP4B*), coordinato dal Dipartimento di Ingegneria dell'Informazione dell'Università di Modena e Reggio Emilia, ha come obbiettivo lo sviluppo di un'infrastruttura tecnologica avanzata che consenta alle aziende di qualsiasi natura, dimensione e posizione geografica di cercare partner, scambiarsi informazioni e conoscenza, negoziare e collaborare senza limitazioni o vincoli.

Si tratta di un progetto FIRB, iniziato nel Luglio 2006, che coinvolge, oltre all'unità di Modena, altri importanti istituzioni nello scenario scientifico italiano, tra cui il CEFRIEL di Milano, il Dipartimento di Informatica, Sistemistica e Comunicazione dell'Università di Milano-Bicocca e ISTI-CNR di Pisa.

La visione del progetto è orientata ad un mercato strutturato basato su internet, dove le aziende, in particolare Piccole e Media Imprese (PMI), possono accedere a grandi quantità d'informazioni già presenti in portali verticali e basi di dati aziendali, usandoli per una collaborazione dinamica a valore aggiunto.

Da un punto di vista tecnologico, NeP4B si propone di sviluppare una rete di peer semantici nella quale le imprese possano classificare i propri profili, offerte, servizi, ricercare partner e avviare delle collaborazioni industriali. Diversi problemi devono essere affrontati al fine di sviluppare lo scenario descritto:

- *Estrazione e rappresentazione della conoscenza*: è necessario definire e sviluppare modelli e linguaggi per la rappresentazione di ontologie e dei mapping che collegano un'ontologia alle sorgenti informative. In quest'ambito, si dovrà risolvere inoltre il problema di estrarre conoscenza da diversi tipi di dati: strutturati, semi-strutturati, testuali e multimediali.
- *Qualità dei dati*: verranno studiati i modelli dei dati più adatti per associare metadati sulla qualità ai dati e all'ontologia del peer, e algoritmi per effettuare query processing e conflict resolution basati sulla qualità.
- *Interoperabilità semantica tra peer*: in quest'ambito dovranno essere studiati algoritmi per la scoperta di mapping fra peer e per la riformulazione di una query di un peer sugli altri peer.
- *Peer Trust*: dovrà essere sviluppato un modello di trust coerente con lo scenario in cui diversi attori possono incontrarsi in un ambiente virtuale e dovono decidere se un partecipante è affidabile.
- *Servizi di ricerca*: tali servizi, utilizzati per scoprire candidati partner, dovranno essere basati su semantica.

[1] http://www.dbgroup.unimo.it/nep4b/

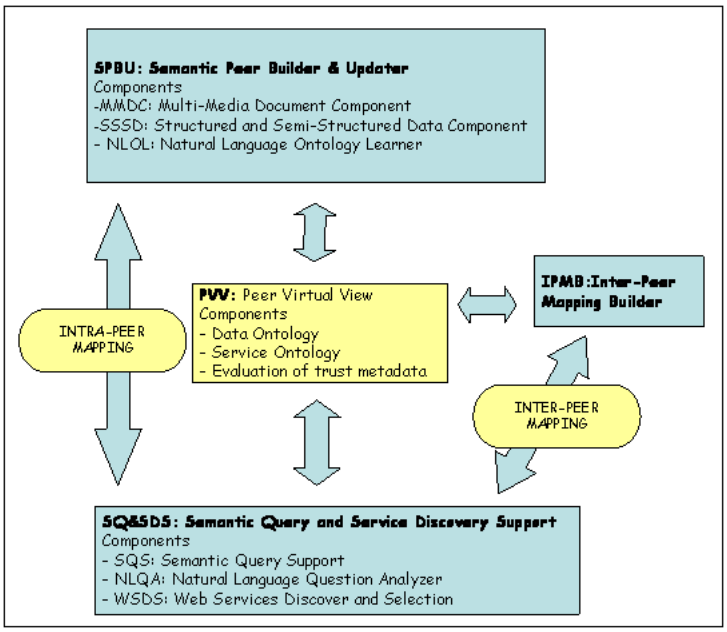


FIGURA 1. Organizzazione di un singolo peer semantico

- *Interfacce in linguaggio naturale*: si dovranno sviluppare strumenti che permettano di formulare query attraverso il linguaggio naturale, al fine di facilitare l'accesso alla conoscenza da parte degli utilizzatori finali.

Dal punto di vista architetturale, il peer è organizzato come mostrato in figura 1. L'elemento principale è rappresentato dalla *Peer Virtual View*, (*PVV*) ovvero l'ontologia che descrive il dominio di quel peer. Essa è costituita dall'ontologia dei dati e dei servizi, più dei metadati riguardo la valutazione della fiducia e della qualità.

Il *Semantic Peer Builder & Updater* (*SPBU*) è un sistema di integrazione delle informazioni basato su una architettura wrapper-mediatore il cui compito è quello di creare, mantenere e arricchire semanticamente la PVV e le associazioni intra-peer con le sorgenti locali. Tale componente prevede moduli specifici per l'estrazione di dati di tipo strutturato e semi-strutturato (*Structured and Semi-Structured Data Component*, *SSSD*), multimediale (*Multi-Media Ontology Component*, *MMOC*) e testuale (*Natural Language Ontology Learner*, *NLOL*). L'*Inter-Peer Mapping Builder* (*IPMB*) è il componente responsabile della scoperta e del mantenimento dei mapping inter-peer relativi alla PVV. Infine, *Semantic Query and Service Discovery Support* (*SQ&SDS*) fornisce le funzionalità per rispondere alle interrogazioni nella rete NeP4B, pianificando la risposta all'interno di un peer semantico e nella rete complessiva.

2. Descrizione della Tesi e Organizzazione del Documento

Il lavoro della presente tesi è stato svolto all'interno del gruppo di ricerca DataBase Group presso il Dipartimento di Ingegneria dell'Informazione dell'Università di Modena e Reggio Emilia. L'attività svolta si inserisce all'interno del progetto NeP4B, precedentemente

descritto. In particolare, nella tesi sono stati affrontati due attività principali. In primo luogo lo studio e la valutazione delle principali caratteristiche degli ecosistemi digitali e del progetto *Digital Business Ecosystem* (*DBE*). In secondo luogo lo studio e l'analisi dei problemi relativi alla qualità dei dati nei sistemi di data integration.

La tesi è stata organizzata nei seguenti capitoli.

Nel *Capitolo 2* verranno introdotte le caratteristiche della *Model Driven Architecture* (*MDA*), uno standard dell'Object Management Group (OMG) la quale fornisce una serie di specifiche per la definizione e trasformazione di modelli di servizi, utile per la progettazione e generazione di software. Tale architettura rappresenta l'approccio utilizzato nel progetto DBE per rappresentare e memorizzare la conoscenza del sistema e per semplificare e automatizzare la progettazione e l'implementazione dei servizi. Nello stesso capitolo vengono inoltre presentati lo standard *Meta Object Facility* (*MOF*), il quale definisce un framework per la gestione integrata di diversi tipi di metadati, e il formato *XML Metadata Interchange* (XMI), il quale consente di mappare i modelli MOF-compliant in documenti XML. Il capitolo conclude con un confronto tra la progettazione di sistemi nell'ambito MDA e database e un confronto fra le meta-architetture del Semantic Web e di MDA, cercando di metterne in luce differenze e similitudini.

Nel *Capitolo 3* viene presentato il progetto DBE. In particolare, vengono introdotte le principali caratteristiche dei digital ecosystem e dei progetti appartenenti all'ononimo cluster di ricerca dell'Unione Europea, del quale DBE è il capostipite. L'architettura funzionale di DBE è presentata nel paragrafo 3, mentre nei successivi paragrafi vengono illustrati i linguaggi utilizzati per rappresentare la conoscenza del sistema.

Lo scopo del *Capitolo 4* è quella di analizzare le caratteristiche del progetto DBE. Pertanto, dopo aver introdotto l'infrastruttura tecnica del sistema, viene illustrato il procedimento di creazione e pubblicazione di un servizio DBE, facendo riferimento ad un semplice servizio realizzato. Vengono inoltre presentate le funzionalità di ricerca semantica ed esecuzione dei servizi. Il capitolo termina con analisi dello stato dell'arte del sistema al momento della sperimentazione, seguita da un'analisi delle possibilità e delle limitazioni riscontrate durante l'utilizzo del sistema.

Nel *Capitolo 5* viene proposto un possibili utilizzo del sistema DBE al fine di creare un servizio di integrazione semantica di dati in un contesto peer-to-peer. Tale capitolo inizia pertanto con uno studio dello stato dell'arte dei sistemi di gestione di dati in ambito P2P, in particolare *Peer Data Management System* (*PDMS*) e *Semantic Overlay Cluster*. Vengono inoltre brevemente introdotte le principali caratteristiche del data management nel progetto NeP4B e infine viene illustro lo usage scenario proposto.

Il *Capitolo 6* offre un'introduzione sui problemi principali da affrontare nella progettazione di modelli per la gestione dei metadati relativi alla qualità delle informazioni. In particolare si introduce il concetto di *data quality* e si analizza il problema della scelta dei giusti criteri per rappresentare le informazioni di qualità.

Nel *Capitolo 7* il problema del data quality viene affrontato relativamente al progetto NeP4B. In particolare viene presentato *DaQuinCIS*, un sistema di data integration quality-driven, e il modello dei dati da esso adottato, ovvero il modello *Data and Data Quality* (*D^2Q*). I due paragrafi successivi offrono un confronto tra le tecniche di query processing e di conflict resolution tra DaQuinCIS e altri importanti lavori presenti in letteratura. Il capitolo termina con una possibile implementazione relazionale della struttura dati definita dal modello D^2Q.

Infine, nel *Capitolo 8* vengono esposte le conclusioni e gli sviluppi futuri.

CAPITOLO 2

La Model Driven Architecture™ (MDA™)

Nel Novembre del 2000, l'*Object Management Group*, meglio conosciuto come OMG[1], ha annunciato la *Model Driven Architecture*, o MDA, ovvero una serie di specifiche per la definizione e trasformazione di modelli di servizi, utile per la progettazione del software. In particolare, MDA (OMG 2003A) definisce un approccio che permette di progettare sistemi IT tenendo separate le specifiche delle funzionalità del sistema dalle specifiche dell'implementazione delle funzionalità su una particolare piattaforma tecnologica. Per questo motivo, MDA definisce un'architettura che fornisce delle guide di riferimento per la strutturazione delle specifiche come modelli.

MDA è supportata da alcuni standard definiti da OMG, tra i quali *Unified Modeling Language* (UML) (OMG 2006B), il linguaggio utilizzato per descrivere molti tipi di artefatti software object-oriented, *Meta-Object Facility* (MOF) (OMG 2005A; OMG 2006A), ovvero un framework generale che permette di strutturare l'informazione su più livelli di astrazione, e *XML Metadata Interchange* (XMI) (OMG 2000; OMG 2005B), un formato basato su XML utilizzato per lo scambio di modelli tra sistemi diversi, che verranno presentati di seguito.

L'approccio definito da MDA e gli standard ad esso correlati, permettono di specificare le funzionalità di un sistema grazie ad un singolo modello, il quale potrà essere realizzato su piattaforme eterogenee, attraverso la definizione di opportuni mapping. Tali standard consentono inoltre di integrare differenti applicazioni collegando esplicitamente i loro modelli, permettendo così l'integrazione e l'interoperabilità delle applicazioni e supportando l'evoluzione del sistema, in quanto le tecnologie delle piattaforme possono cambiare nel tempo.

1. Presentazione di MDA

L'approccio MDA si focalizza sulla separazione delle specifiche delle funzionalità del sistema dall'effettiva implementazione di tali funzionalità. MDA rappresenta lo sforzo di OMG di definire una serie di specifiche che permettano di progettare framework *model-driven*, cioè dove tutte le specifiche del sistema siano date attraverso modelli. E' dunque evidente che tra i concetti principali definiti nelle specifiche di MDA vi sia quello di modello. Di seguito verranno presentati questi concetti e le altre caratteristiche fondamentali di MDA.

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

Sistema, punti di vista, piattaforme e modelli. Il concetto più importante della Model Driven Architecture è quello di *modello* di un sistema, con il quale si intende la descrizione del sistema e del contesto in cui andrà ad operare; spesso un modello è presentato come una combinazione di elementi grafici e di testo (linguaggio naturale o un linguaggio di modellazione formale).

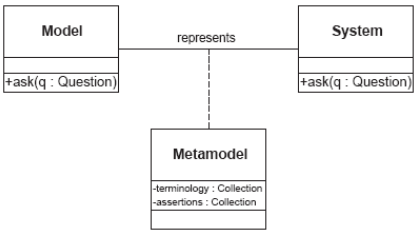


FIGURA 1. Rapporto fra modello, sistema e meta-modello

Il concetto di *sistema* include qualsiasi cosa: un programma, un computer, una combinazione di alcune parti di più sistemi, una federazione di sistemi, persone, un'impresa, etc., mentre l'*architettura* del sistema specifica le parti in cui il sistema stesso è suddiviso e le regole di interazione fra esse. Una *piattaforma* rappresenta invece un insieme di sottosistemi e tecnologie che fornisco un insieme coerente di funzionalità attraverso interfacce che possono essere sfruttate da ogni *applicazione* (ovvero una specifica funzionalità implementata) supportate da tale piattaforma: in questo modo l'applicazione in questione è svincolata dagli aspetti implementativi delle funzionalità offerte dalla piattaforma. Esempi di piattaforme sono CORBA, J2EE, Microsoft .NET e i Web Service.

Un modello è dunque utilizzato per dare una rappresentazione semplificata di un sistema reale (Figura 1). A fronte di un'interrogazione posta su di esso, il modello deve essere in grado di fornire una risposta corretta, ovvero uguale a quella che fornirebbe il sistema.

L'utilità del modello consiste nella sua semplicità rispetto al sistema che rappresenta. Per raggiungere questa caratteristica, molti dettagli del sistema originale sono eliminati e solo alcuni sono implementati nel modello: questa semplificazione, o astrazione, è l'essenza della modellazione.

Ogni volta che è necessario applicare la stessa operazione di modellazione più volte o quando tale operazione deve essere applicata da persone diverse, l'utilizzo di un meta-modello può semplificare il lavoro. In particolare, per *meta-modello* si intende una esplicita specifica di una astrazione. In altre parole un meta-modello definisce il linguaggio utilizzato per rappresentare il modello; un esempio di meta-modello è rappresentato da MOF, descritto nel prossimo paragrafo. Per poter descrivere delle astrazioni, il meta-modello deve definire, come mostrato in figura 1, un insieme di concetti e un insieme di relazioni fra questi concetti. Nello specifico caso di MOF, tali relazioni sono espresse dal formalismo OCL (*Object Constraint Language*) (OMG 2003B), il quale deve essere aggiunto al meta-modello. Si nota che la nozione di meta-modello è fortemente relazionata a quella di ontologia: è possibile estrarre un modello da un sistema, utilizzando uno specifico meta-modello o ontologia.

La tecnica di astrazione definita da MDA per realizzare un modello semplificato del sistema è detta *viewpoint*, o *punto di vista*. Con un punto di vista si vuole catturare certe componenti dell'architettura e delle regole strutturali al fine di evidenziare solo alcune caratteristiche del sistema, eliminando il resto dalla rappresentazione. Ad esempio, nell'ambito della progettazione dei database (BERGAMASCHI *et al.* 1999) si possono distinguere tre punti di vista: quello concettuale, il quale rappresenta la realtà di interesse ad un alto livello di astrazione, quello logico che rappresenta la realtà modellata dal livello concettuale secondo una specifica tecnologia (relazionale, ad oggetti, etc.) e quello fisico il quale rappresenta un'implementazione del modello logico per una piattaforma specifica.

Anche in MDA vengono definiti tre diversi punti di vista: il *computation indipendent viewpoint*, il quale si focalizza sul contesto in cui l'applicazione è inserita e sui suoi requisiti mentre ne nasconde gli aspetti strutturali e implementativi, il *platform indipendent viewpoint* il quale indica le operazioni che il sistema compie senza scendere nei dettagli implementativi di una specifica piattaforma, mentre il *platform specific viewpoint* specializza le specifiche del platform indipendent viewpoint per una piattaforma in particolare.

Sulla base di questi punti di vista, vengono definiti tre distinti modelli esposti nel prossimo paragrafo: computation indipendent model (CIM), platform indipendent model (PIM) e platform specific model (PSM).

I modelli Computation Indipendent, Platform Indipendent e Platform Specific. Un *Computation Indipendent Model* (CIM) rappresenta un modello di un sistema dal punto di vista indipendente dalla computazione; ciò significa che tale modello non deve presentare i dettagli strutturali del sistema ma deve esclusivamente soffermarsi sui requisiti del sistema, in particolare su cosa il sistema deve fare. Un CIM rappresenta in particolare un modello di un dominio.

Un *Platform Indipendent Model* (PIM) è un modello indipendente dalla piattaforma; il suo compito è quello di descrivere l'implementazione delle funzionalità del sistema espresse dal modello CIM tramite un linguaggio che permetta di applicare lo stesso modello a diverse piattaforme.

Un *Platform Specific Model* (PSM) è invece un modello definito per una specifica piattaforma. Tale modello applica dunque le specifiche fornite dal PIM alla piattaforma sulla quale implementare il sistema.

Dalla definizione di CIM emerge che un modello computation indipendent permette di definire le funzionalità del sistema senza specificare i dettagli strutturali e implementativi, oggetto degli altri modelli. Un modello CIM descrive dunque l'ambito in cui il sistema andrà ad operare: per questo motivo i linguaggi che definiscono tali modelli non sono necessariamente legati a concetti e termini tipici dell'informatica ma, al contrario, devono riferirsi ai domini tipici di settori quali il business di interesse, la legislazione vigente, etc.

Inoltre tali linguaggi devono essere comprensibili e utilizzabili da persone esperte dei settori del business definito e non solo da professionisti dell'information technology: un linguaggio CIM può essere dunque anche testuale e basato sul linguaggio naturale e non solo un linguaggio formale ben definito come, ad esempio, UML.

Un esempio di linguaggio CIM è *Business Modeling Language* (*BML*) (DE TOMASI *et al.* 2005A), realizzato nell'ambito del progetto europeo *Digital Business Ecosystem*[2] (*DBE*).

[2]http://www.digital-ecosystem.org

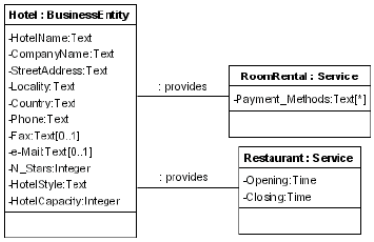


FIGURA 2. Modello CIM (BML) per la descrizione di un hotel e dei suoi servizi

Tale linguaggio permette di definire concetti come *BusinessEntity* per rappresentare un tipo di attività (Hotel, Università, Ospedale, etc.), *Service* per la modellazione di un servizio (prenotazione stanza, lezione, etc.), *Product* per i prodotti e altri ancora.

Si nota che tale linguaggio è utilizzato solo per descrivere a livello concettuale un servizio o una business entity e non per caratterizzarlo computazionalmente. Oltre a questi concetti che riguardano l'organizzazione di una attività, BML permette inoltre di descrivere gli aspetti comportamentali dell'attività, come *BusinessProcess* (ad esempio Booking) e *Transaction* (ad esempio BookingConfirmation), la sua localizzazione geografica, come *LocationType* (ad esempio HotelCity), e concetti relativi ad eventi, come *Event* (ad esempio AccountOpen e AccountClose).

I modelli CIM sono stati introdotti con l'intento di definire completamente il dominio (vengono infatti chiamati domain model) sul quale progettare le proprie applicazioni e per tale motivo è simile al concetto di ontologia. Per la loro definizione, l'utente utilizza come vocabolario, quello tipico del dominio che vuole descrivere. L'esempio sottostante riporta un modello CIM (il linguaggio utilizzato è BML) relativo agli hotel e ai loro servizi; come si nota, i termini utilizzati per descrivere il dominio in questione sono Hotel, Restaurant, HotelName, RoomRental, etc.

L'importanza di modello risiede nel fatto che risolve il problema comunicativo tra chi definisce il dominio e i suoi requisiti e chi dovrà rispettare tali requisiti per l'Implementazione delle funzionalità del sistema.

Per quanto riguarda il modello PIM, esso permette di definire la struttura delle funzionalità del sistema in modo indipendente dalla piattaforma. Questo risultato è spesso ottenuto definendo una virtual machine sulla quale modellare il sistema: in questo modo il modello PIM è definito sulla base di questa piattaforma. Si rendono quindi necessari più mapping tra PIM (macchina virtuale) e le specifiche piattaforme (Java, Web Services, etc.): tali mapping sono quindi alla base per la costruzione dei modelli PSM.

CIM e PIM hanno dunque due obbiettivi diversi: il primo descrive un sistema dal punto di vista delle entità coinvolte e dei servizi offerti (modellazione a livello di business) mentre PIM, definito partendo dalle specifiche date dal modello CIM, permette di focalizzarsi su come le funzionalità del sistema sono strutturate (modellazione del sistema informativo) rispetto ad una macchina virtuale.

Un esempio di linguaggio PIM è rappresentato da *Service Description Language* (SDL) (MONTANARI *et al.* 2005A), sviluppato anch'esso nell'ambito del progetto DBE. Con tale linguaggio i progettisti intendono descrivere, in maniera indipendente dalla piattaforma, l'interfaccia tecnica di un servizio, ovvero l'interfaccia di basso livello comune a tutti gli

utenti. Tale linguaggio permette quindi di definire i servizi del sistema; per definizione si intende il tipo di servizio, la sua interfaccia (ovvero le operazioni che possono essere invocate) e i messaggi che sono generati dallo svolgimento di una determinata operazione (input, output, eccezioni, etc.). Alcuni esempi di classi di questo linguaggio sono *Element*, *Description*, *Interface*, *Operation*, *MessageList* e *SimpleMessage*.

Un modello PSM è una trasformazione di un modello PIM rispetto ad una specifica piattaforma; tale trasformazione rispetta certe regole, definite in base alla piattaforma target. Pertanto, un modello PSM è un modello PIM appartenente ad un livello di astrazione inferiore. Specifici mapping possono essere definiti per Java, WSDL, etc.

Si nota che il modello PSM non corrisponde per forza al codice dell'applicazione; piuttosto esso rappresenta solo l'interfaccia dell'applicazione: il codice è necessario svilupparlo manualmente.

Tale suddivisione dei modelli costituisce l'approccio orizzontale di MDA per la modellazione dei sistemi: infatti tali modelli non vengono ripartiti in diversi livelli gerarchici dell'architettura del sistema (questo è vero almeno per i modelli CIM e PIM) ma sono costruiti allo stesso livello di astrazione (l'approccio verticale è costituito dai 4 livelli dell'architettura di MOF, presentata nel prossimo paragrafo). Quindi, il processo di creazione di un sistema, seguendo l'approccio proposto da MDA, consiste nel rappresentare i requisiti del sistema attraverso un modello CIM, implementarli attraverso un modello indipendente dalla piattaforma, PIM, e, infine, scegliere una o più piattaforme sulle quali implementare i modelli dedicati PSM ed effettuare la conversione da PIM a PSM, come mostrato in figura 3.

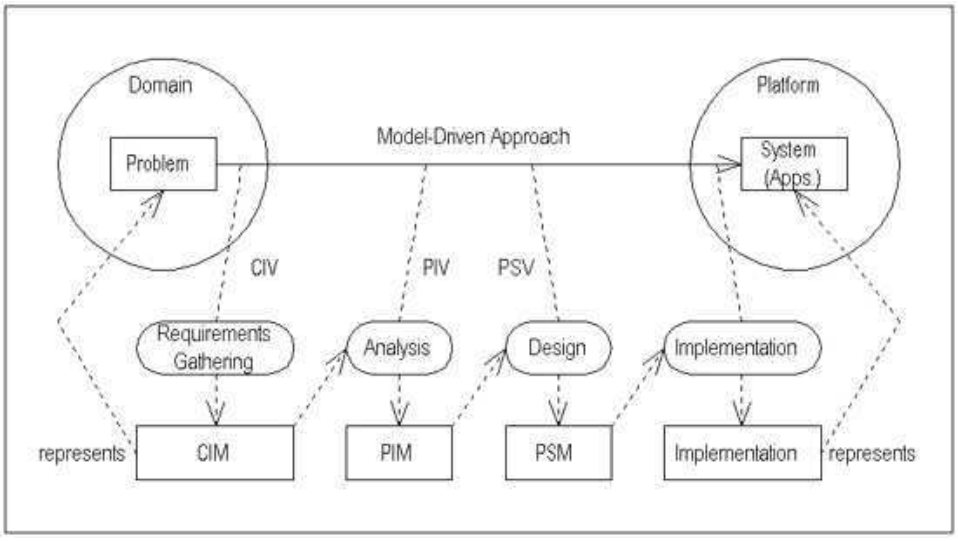


FIGURA 3. Fasi nella progettazione MDA

§

MDA è un insieme di specifiche che, come detto, si basa su altri standard definiti da OMG, in particolare UML e MOF. L'UML è il linguaggio che permette di costruire un modello di un sistema da analizzare o progettare, ovvero permette di dare una rappresentazione astratta di un sistema basata su alcuni concetti fondamentali, come classi e associazioni. I concetti base dell'UML vengono usati anche per descrivere la semantica del linguaggio stesso. Si ha quindi un modello per un linguaggio di modellazione, cioè un meta-modello.

Meta-livello	**Termine MOF**	**Linguaggi**	**Esempi**
M3	Meta-meta-modello	Meta-modello MOF	MOF
M2	Meta-modello	Linguaggi di rappresentazione della conoscenza, di modellazione database e software	OWL, E/R, UML
M1	Metadati	Ontologie, schemi, modelli	Ontologia OWL, Schema E/R, Modello UML
M0	Dati	Sistema reale (istanze di ontologie, schemi e modelli)	Istanze dell'ontologia OWL, dello schema E/R e del modello UML

TABELLA 2. Framework di meta-modellizzazione basato su 4 livelli

MOF rappresenta quel linguaggio che costituisce la tecnologia adottata da OMG per la definizione di una sintassi astratta comune che permetta di definire meta-modelli attraverso dei meta-modelli. La nascita di MOF è stata dettata dal fatto che UML rappresenta solo un possibile meta-modello. A causa del rischio posto dalla presenza di una vasta varietà di meta-modelli diversi e incompatibili, vi è la necessità di avere un framework di meta-modellazione adatto a tutti i meta-modelli utilizzati nella progettazione del software in ambito industriale. La soluzione proposta dalla comunità dell'ingegneria del software è stata quella di fornire un linguaggio per definire meta-modelli, ovvero un meta-meta-modello: questo è il ruolo di MOF.

Il framework di meta-modellazione che prende forma è quindi composto da 4 livelli (tabella 1):

- *Dati* o *Oggetti* (livello M0): tale livello è costituito dai dati che si vogliono rappresentare; l'elemento più ricorrente è l'oggetto, ovvero un'entità del mondo reale. Ne segue che questo livello inferiore descrive uno specifico dominio di informazione.
- *Metadati* o *Modello* (livello M1): definisce un linguaggio per la descrizione di un dominio di informazione, ovvero è costituito da metadati il cui compito è quello di descrivere i dati appartenenti al livello sottostante. Gli oggetti del livello M0 aventi caratteristiche comuni vengono modellati come un'unica classe.

- ❖ *Metamodello* (livello M2): definisce i dati, detti meta-metadati, utilizzati per descrivere la struttura e la semantica del modello; tale descrizione è detta metamodello (linguaggio astratto utilizzato per la definizione di diversi tipi di dati). Tipici concetti che si incontrano in questo livello sono *Classe*, *Attributo* e *Operazione*.
- ❖ *Meta-metamodello* (livello M3): descrive la struttura e la semantica del metamodello; il meta-metamodello rappresenta dunque il linguaggio astratto che permette di definire nuovi tipi di metadati. I concetti presenti in questo livello includono *Meta-Classe*, *Meta-Attributo* e *Meta-Operazione*.

La caratteristica principale della gestione dei metadati in MOF è l'*estendibilità*: l'obbiettivo è quello di creare un framework che supporti qualsiasi tipo di metadati e che permetta anche l'aggiunta di nuovi metadati.

Inoltre, come affermato all'inizio del capitolo, MDA rappresenta il primo importante tentativo di definire un framework per la modellazione di software attraverso il paradigma *model-driven* (BÉZEVIN 2004). L'obbiettivo di questo approccio è quello di utilizzare i modelli in modo attivo nel processo di produzione del software e non come una mera documentazione. Ciò che si vuole fare è implementare tool che utilizzino i modelli del sistema per costruire software.

Si nota inoltre che la progettazione model-driven si distingue da quella classica object-oriented. Infatti se in quest'ultima il principio basilare è *"Tutto è un oggetto"*, nel nuovo paradigma il principio sembra essere *"Tutto è un modello"*.

Tutto è un oggetto	[P1]
Tutto è un modello	[P2]

In base al principio [P1], nelle tecnologie object-oriented è importante che un oggetto sia istanza di una classe la quale può ereditare da una superclasse (figura 4a). Queste due semplici relazioni possono essere chiamate *instanceOf* e *inherits*. Quello che è importante invece nel contesto della model-driven engineering è la possibilità di catturare una vista, o aspetto, del sistema attraverso un modello e che ogni modello sia descritto in termini del suo meta-modello (linguaggio). In questo caso, le due relazioni sono *representedBy* e *conformantTo* (figura 4b). La distinzione tra questi diversi insiemi di relazioni [P1] e [P2] è fondamentale per capire L'Evoluzione tra object-oriented technology e model-driven engineering.

Quindi, l'approccio model-driven permette di rappresentare un sistema attraverso più modelli, ognuno dei quali cattura un aspetto del sistema da modellare. MDA definisce un framework nel quale diversi modelli dello stesso sistema possono essere creati e manipolati, ognuno dei quali può essere definito da un meta-modello diverso. In questo contesto, modelli estratti dallo sistema ma che utilizzano meta-modelli diversi rimangono relazionati. Le specifiche di MDA definiscono infatti altre operazioni eseguibili sui modelli, molte delle quali sono operazioni di trasformazione, argomento del prossimo paragrafo.

§

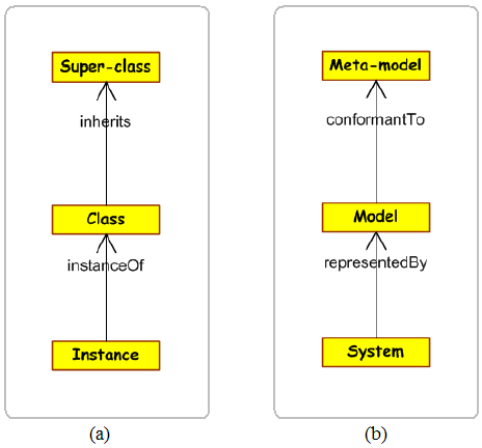


FIGURA 4. Nozioni basilari negli approcci (a) object-oriented e (b) model-driven

Trasformazioni tra modelli. Come si è potuto capire, l'approccio definito da MDA non è basato su un'unica idea. Fra gli obbiettivi che si propone vi sono la separazione dalla descrizione *business-neutral* e l'implementazione specifica per una piattaforma, la possibilità di esprimere differenti aspetti del sistema attraverso linguaggi *domain-specific*, l'istituzione di relazioni tra questi linguaggi in un framework globale e la capacità di esprimere trasformazioni tra di essi.

La trasformazione dei modelli è un altro aspetto centrale dell'approccio definito da MDA. Una RFP (Request for Proposal) (OMG 2002) è attualmente attiva per definire una sorta di "Unified Model Transformation Language". Questo consentirà la trasformazione di un modello *Ma* in un altro modello *Mb*, indipendentemente dal fatto che i corrispondenti meta-modelli *MMa* e *MMb* siano gli stessi o diversi. Si nota inoltre che il programma di trasformazione, composto da una serie di regole come mostrato in figura 5, può essere considerato esso stesso un modello *Mt*. Conseguentemente, esso sarà basato su un meta-modello *MMt*.

E' possibile quindi progettare tool che trasformino automaticamente un modello PIM in uno PSM e successivamente in codice. Questo, oltre ad aumentare l'interoperabilità, avrebbe il vantaggio di facilitare e velocizzare le fasi di implementazione del sistema.

La trasformazione da un modello PIM a un modello PSM avviene in modo automatico, formalizzando i mapping per ogni piattaforma sulla quale il sistema deve operare; in questo modo si ottiene un duplice vantaggio:

- ❖ lo stesso modello PIM può essere applicato a diverse piattaforme semplicemente definendo i mapping tra un modello e l'altro e,
- ❖ quando cambia il contesto è necessario solo modificare il modello PIM, in quanto, attraverso l'uso dei mapping definiti, i modelli PSM sono ricreati automaticamente.

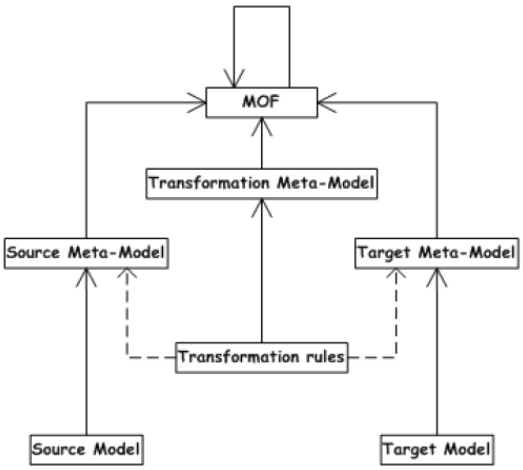


FIGURA 5. Model transformation basata su meta-modello

Infine, sempre nell'ottica di definire sistemi interoperatibili, MDA definisce il concetto di *bridge*, ovvero di ponti tra due PSM, e le regole necessarie per definire tali bridge (in particolare sarà necessario trovare le corrispondenze tra i modelli PSM).

In conclusione, il framework presentato permette di raggiungere diversi obbiettivi tra cui:

- ❖ supporta qualsiasi tipo di modello e paradigma di modellazione;
- ❖ consente di relazionare tipi diversi di metadati;
- ❖ permette di aggiungere nuovi tipi di metamodelli e metadati;
- ❖ supporta lo scambio di modelli e metamodelli tra parti che utilizzano lo stesso meta-metamodello.

2. Meta Object Facility (MOF)

Meta Object Facility (MOF) (OMG 2005A; OMG 2006A) è uno standard definito dall'OMG il cui compito è quello di definire un framework per la gestione dei metadati e un insieme di servizi che consentano lo sviluppo e l'interoperabilità dei sistemi model driven.

In particolare lo standard definisce un linguaggio di modellazione astratto utilizzato per definire altri linguaggi di modellazione relativi a domini specifici. Oltre a questo, lo standard dell'OMG definisce anche un'architettura impostata su 4 meta-livelli (livelli di astrazione), descritta nella sezione precedente e mostrata in figura 6.

§

Di seguito verrà mostrato il modello di MOF. La versione esaminata è la 2.0 (OMG 2006A) la quale estende e migliora la precedente 1.4 (OMG 2005A). La specifica di MOF 2.0 integra e riutilizza la specifica complementare *UML 2.0 Infrastructur*e (OMG 2006B).

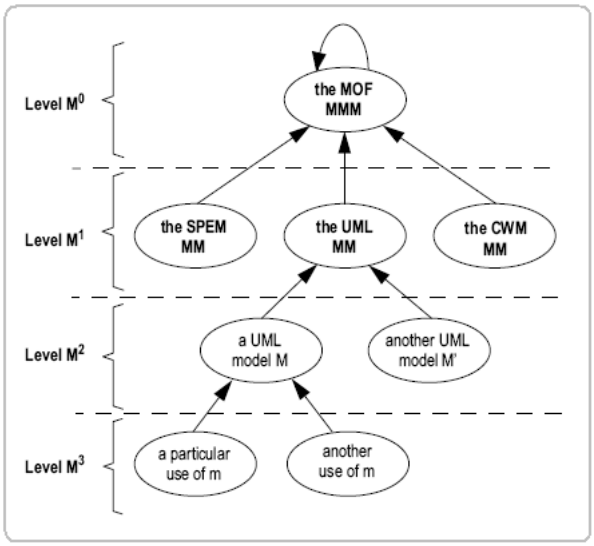


FIGURA 6. Architettura di metadati a 4 livelli definita da MOF

In particolare UML 2.0 fornisce il framework di modellazione e la notazione, mentre MOF 2.0 fornisce il framework per gestire metadati, ovvero per creare, modificare e eliminare metamodelli indipendenti da una particolare tecnologia implementativa. Ciò è ottenuto importando la *Infrastructure Library* di UML 2.0, la quale fornisce i package per definire un meta-linguaggio attraverso il quale implementare meta-modelli. Il modello di MOF 2.0 è costituito principalmente da due package, *Essential MOF* (*EMOF*), il quale definisce le caratteristiche base del modello, e *Complete MOF* (*CMOF*), il quale estende il precedente package. Si nota che sia EMOF che CMOF riutilizzano la Infrastructural Library di UML per facilitare l'utilizzo di tool per lo sviluppo e l'integrazione dei modelli; in particolare tutti i modelli appartenenti alle specifiche MDA condividono lo stesso package Core, come mostrato in figura 7, che può essere quindi definito come il kernel di MDA.

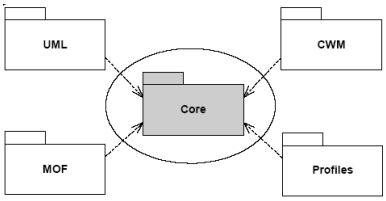


FIGURA 7. Ruolo del package Core in MDA

Questa scelta porta a diversi vantaggi in quanto ogni meta-modello può beneficiare di una sintassi astratta e semantica che sono già state definite. I package EMOF e CMOF importano inoltre altre capacità definite in package separati del modello di MOF i quali includono supporto per identificatori, tipi primitivi aggiuntivi, riflessione e altre ancora. In particolare tali package sono:

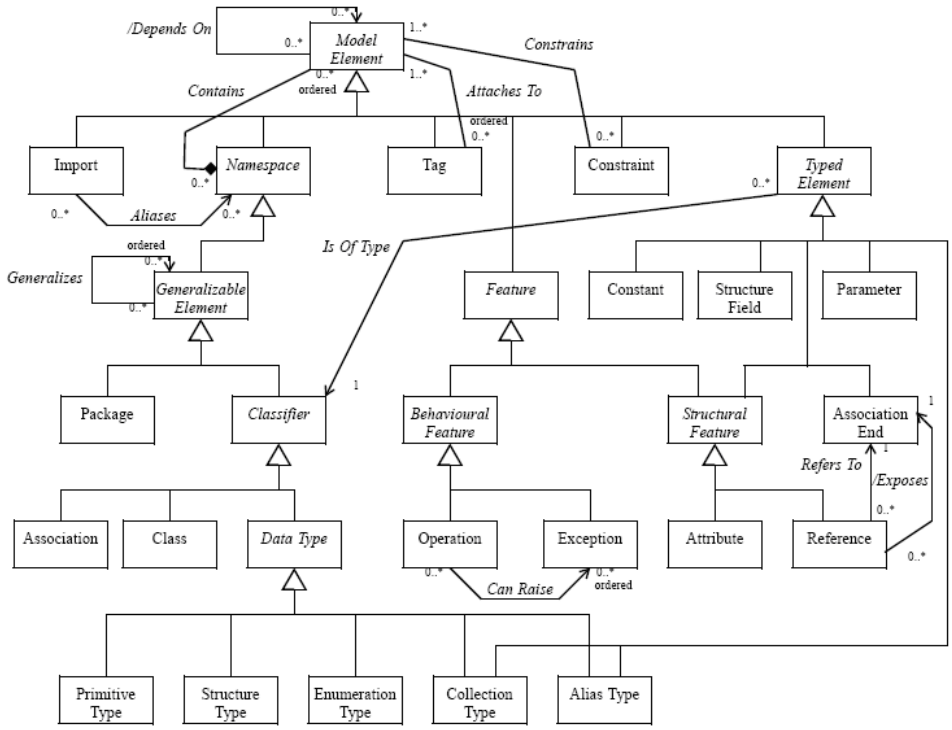


FIGURA 8. Elementi principali del modello MOF

- ❖ *Reflection*: estende un modello con la proprietà di essere self-describing.
- ❖ *Identifiers*: fornisce un'estensione per identificare univocamente degli oggetti di metamodelli senza affidarsi ad un modello dei dati che può essere soggetto a variazioni.
- ❖ *Extension*: utilizzato per estendere gli elementi del modello con coppie nome-valore (tagged value).

Una visione semplificata e generale degli elementi che costituiscono il modello MOF è fornita in figura 8.

Costrutti base del MOF Model. I costrutti base per la definizione di meta-modelli tramite MOF sono le classi, le quali modellano i meta-oggetti, la associazioni, che rappresentano relazioni binarie tra meta-oggetti, i tipi di dato (primitivi, esterni, etc.) e i package, che permettono di raggruppare più classi insieme. In particolare, si possono individuare i seguenti elementi.

Classi. Permettono la rappresentazione degli elementi costitutivi di un modello. Ogni classe può avere due tipi di feature: gli attributi e le operazioni. Gli *attributi*, i quali rappresentano proprietà elementari di una classe, hanno un nome, un tipo ed una molteplicità. Le *operazioni* permettono di accedere a determinati comportamenti associati ad una classe; all'interno della classe sono identificate da un nome univoco e presentano inoltra una lista di parametri (di ingresso e uscita) ognuno con un proprio tipo e molteplicità. Il modello MOF

permette alle classi di essere ereditate da altre classi (specializzazioni/generalizzazione): in questo modo la sotto-classe, oltre che poter definire le proprie feature, eredita tutte le operazioni e attributi della super-classe. Inoltre, una classe può essere dichiarata astratta, ovvero non direttamente istanziabile: tale classe dovrà quindi essere ereditata da una o più classi. Infine, una classe può essere dichiarate come *leaf*, per evitare che altre classi la specializzino, o come *root* per prevenire la creazione di super-classi.

Associazioni. Le associazioni rappresentano un legame logico tra due o più classi. Ogni associazione è composta da due *Association End*, i quali rappresentano le classi collegate dall'associazione. Per ogni associazione è possibile definire il nome dell'associazione e dei due end, il tipo degli end (che deve essere una classe) e la loro molteplicità. Si nota quindi che in MOF, come in UML, non c'è il concetto primitivo di Associazione. Le associazioni, infatti, possono esistere solo nel contesto di due o più classi, cioè di due o più estremi.

Aggregazione. L'aggregazione permette di specificare un particolare tipo di semantica per le associazioni: con una aggregazione si vuole indicare che una istanza di una classe è composta anche dalle istanze della classe associata (relazione *part-of* o *whole-part*). In particolare, MOF associa alle aggregazioni tre tipi diversi di semantica:

- ❖ *Semantica non-aggregate*, la quale indica una relazione debole fra le classi associate, ovvero non sono imposti vincoli sulla molteplicità e le classi possono avere cicli di vita diversi.
- ❖ *Semantica composite*, la quale rappresenta un'aggregazione forte, ovvero dove le istanze delle classi part appartengono esclusivamente alle istanze della classe whole e, tali istanze, presentano lo stesso ciclo di vita.

Riferimenti. Sono un'ulteriore feature delle classi in quanto permettono di rappresentare le associazioni con una notazione *attribute-like*. Come per gli attributi, anche i riferimenti sono composto da un nome, preceduto dal carattere "/" e da un tipo (la classe referenziata) come mostrato in figura 9.

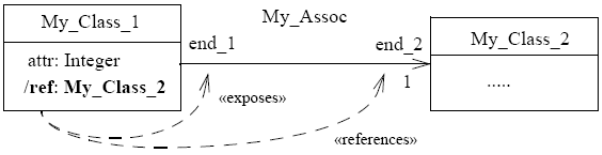


FIGURA 9. Esempio di un riferimento

Data Type. Con MOF è possibile rappresentare tipi di dato non riconducibili ad oggetti. I tipi di dato rappresentabili sono quelli primitivi (interi, stringhe, booleani, etc.) e quelli definiti dagli utenti attraverso opportuni costrutti (enumerazione, strutture, collezioni, etc.).

Package. Gli elementi di un meta-modello possono essere raggruppati utilizzando il concetto di *Package* definito dal modello MOF. I package possono essere generalizzati da uno o più *super-package*, analogamente a quanto avviene per le classi: si nota che ciò implica che la relazione tra sotto-package e super-package è del tipo *IS-A* e non *PART-OF*. I package possono inoltre contenere altri package che a loro volta ne possono contenere altri (*package nesting*): le classi appartenenti ai package interni appartengono anche ai package più esterni. Si nota inoltre che i package innestati non possono essere generalizzati o

essere specializzati e non possono essere importati da altri package. MOF consente anche l'importazione di package in modo da favori il riutilizzo di package già definiti per altri meta-modelli. Una forma particolare di importazione di package è costituita dal *package clustering*, nel quale i package importati vengono raggruppati in cluster. In questo caso si ha che il ciclo di vita di un'istanza di un package del cluster è lo stesso dell'istanza del cluster; ciò significa che quando l'istanza di un cluster è creata, contemporaneamente vengono create le istanze di ogni package appartenente al cluster (lo stesso vale per la rimozione dell'istanza).

Costanti, eccezioni, tag. L'elemento *Constant* è utilizzato per modellare un collegamento tra un nome e un valore, *Exception* per rappresentare le eccezioni che possono incorrere nell'esecuzione di una operazione e *Tag* che permette di associare ad un elemento del meta-modello altri elementi (è composto da nome, ID, collezione dei valori associati al tag e l'insieme degli elementi associati).

UML Infrastructure Library. Il modello MOF riutilizza i principali elementi definiti nel package InfrastructureLibrary di UML (OMG 2006B), il quale a sua volta è scomposto in *Core package* e *Profile package*. Il package Core è il package principale tra quelli che, nel loro insieme, compongono UML in quanto definisce i costrutti base astratti e concreti del meta-modello necessari per lo sviluppo di modelli di livello M1. Si nota inoltre che tale package fornisce i costrutti chiave necessari per la definizione di un meta-modello base e definisce uno scheletro (*backbone*) dell'architettura per supportare costrutti addizionali al linguaggio.

Core è infatti utilizzato per definire la struttura base del meta-modello UML e per tale motivo è stato scelto anche come base del modello di MOF. Come mostrato in figura 10, esso è costituito da 4 sotto-package, *PrimitiveTypes*, *Abstractions*, *Basic* e *Constructs*. Come vedremo nei prossimi paragrafi, il modello EMOF importa esclusivamente i package PrimitiveTypes e Basic in quanto esso vuole fornire solo quei concetti fondamentali per la meta-modellazione, mentre CMOF importa tutta la libraria Core in quanto importa il package Constructs il quale estende Basic e Abstraction.

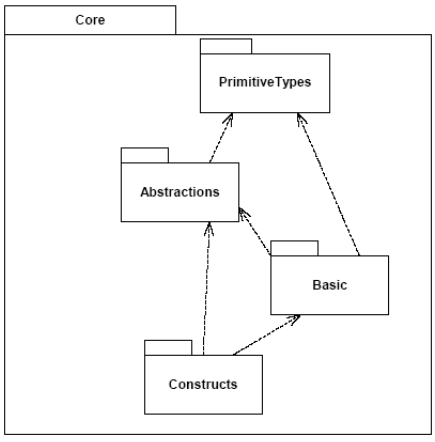


FIGURA 10. Package costituenti *InfrastrucutreLibrary::Core*

I package Reflection, Identifier e Extension. Oltre ad importare i package di Infrastructure Library di UML 2.0, MOF definisce tre package aggiuntivi, ovvero *Reflection*, *Identifiers* e *Extension*, già introdotti in precedenza. Le classi che tali package importano sono mostrate in figura 11 e descritte di seguito.

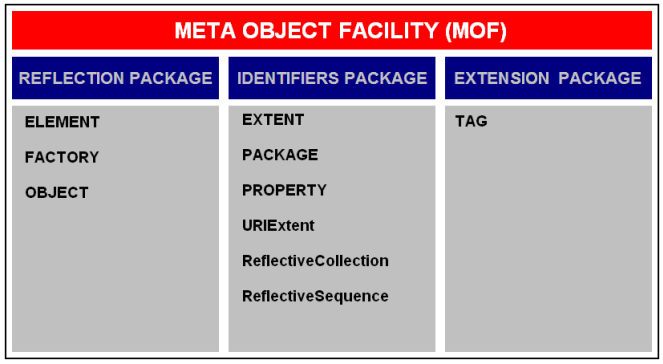


FIGURA 11. Package *Reflection*, *Identifiers* e *Extension*

Il package *Reflection* introduce la proprietà di MOF di essere self-describing. Il suo scopo è quello di fornire un'interfaccia comune a tutte le entità con cui creare, accedere, aggiornare, invocare operazioni e interrogare gli oggetti (di livello M2). La composizione di tale package è mostrata in figura 12. Di seguito vengono descritte le classi che lo compongono:

- *Element*: estende la classe InfrastructureLibrary::Core::Basic::Element con le capacità riflessive; ogni elemento ha una classe che lo descrive (descrive le sue operazioni e proprietà) di cui è istanza. Le operazioni aggiunte permettono di recuperare la classe e il container dell'elemento, di recuperare e definire il valore di una proprietà.
- *Factory*: un Element può essere creato da una Factory; una Factory è un'istanza della classe Factory di MOF. Ogni Factory è associata ad un solo Package e il suo scopo è quello di creare elementi per quel Package. Le operazioni associate a questo elemento permettono di creare un Object fornendo il valore di un oggetto String (`createFromString(dataType:DataType, string:-String):Object`) e di creare una rappresentazione di un Object sottoforma di stringa (`convertToString(dataType:DataType, object:Object):String`). Infine è possibile creare un nuovo Element invocando il metodo `create(metaClass:Class):Element`.
- *Object*: è un supertipo di Element e viene utilizzato per avere un Type che rappresenti sia elementi che data value; Object rappresenta un qualsiasi valore: è l'equivalente di java.lang.Object di Java.

Il package *Identifiers* (figura 14) permette di assegnare degli identificatori univoci agli oggetti appartenenti ad un certo *extent* (contesto). Le classi appartenenti a questo package sono:

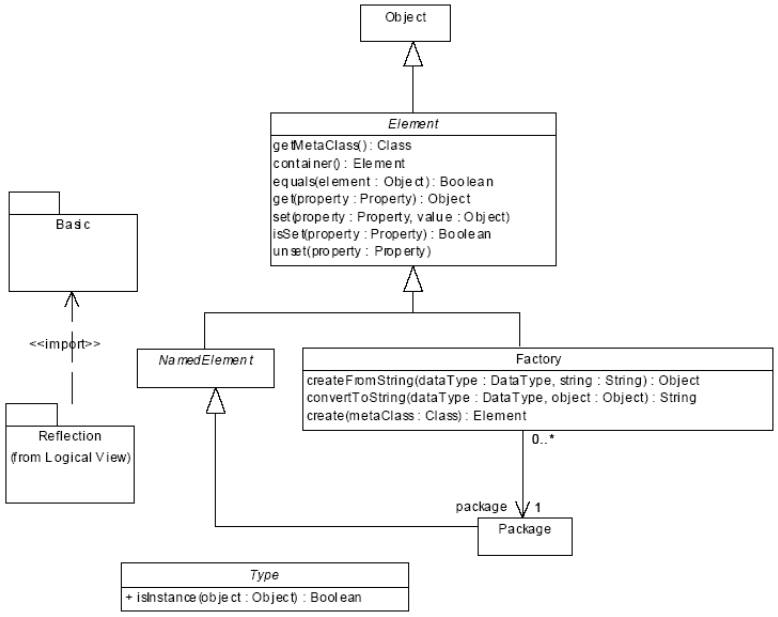


FIGURA 12. Package Reflection

- *Extent*: un extent è un contesto (insieme di Element) nel quale ogni elemento è univocamente identificato; ogni elemento può appartenere a zero o più extent. Vendono definite due operazioni: `elements():ReflectiveSequence` ritorna una ReflectiveSequence degli elementi direttamente referenziati da questo extent e `useContainment():Boolean` la quale ritorna true se l'extent contiene ricorsivamente tutti gli elementi contenuti dai membri di elements().
- *Package*: estende Basic::Package con un URI che può essere utilizzato come identificatore per il package.
- *Property*: in Identifiers, Property è esteso con l'attributo booleano *isID* il quale, se vero, indica che la proprietà può essere utilizzata come identificatore dell'istanza della determinata classe di appartenenza.
- *URIExtent*: tale classe rappresenta un extent che fornisce identificatori URI; le sue operazioni permettono di specificare un identificatore che stabilisce un extent per identificare elementi, di ricavare l'URI di un determinato elemento nell'extent e, viceversa, di ritornare l'elemento dato l'URI.
- *MOF::Common*: è un package che definisce elementi comuni a tutto il modello MOF.
- *ReflectiveCollection*: è una classe riflessiva utilizzata per accedere a proprietà con più valori possibili. Le sue operazioni permettono di gestire tali proprietà multi-valore, in particolare permettono di aggiungere e rimuovere oggetti, come mostrato in figura 13.
- *ReflectiveSequence*: è una sottoclasse di ReflectiveCollection che permette di accedere a proprietà ordinate con più valori possibili.

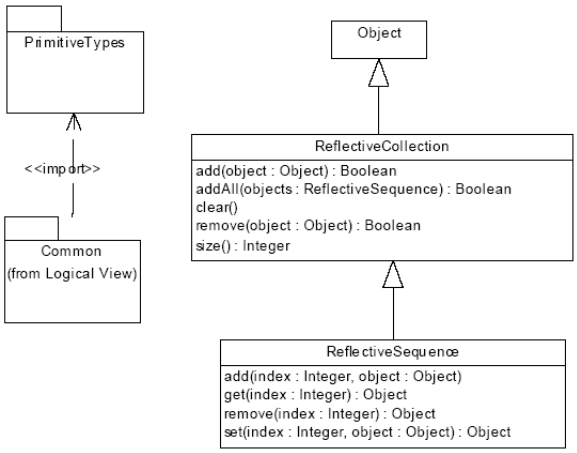


FIGURA 13. Package *MOF::Common*

Il package *Extension* definisce dei meccanismi per associare ad un elemento una collezione di coppie nome-valore (tagged value) per annotare gli elementi stessi con informazioni non presenti nel modello. In tale package viene definita la classe *Tag* la quale rappresenta un singolo pezzo di informazione che può essere associato ad un qualsiasi numero di elementi: in particolare un tag può avere da zero a infiniti elementi associati e un elemento può essere associato a molti tag (figura 15).

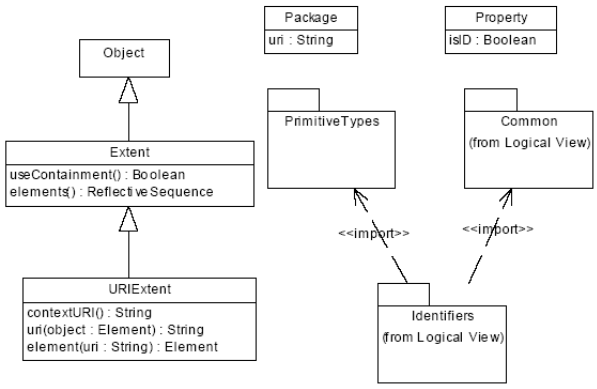


FIGURA 14. Package *Identifiers*

Essential MOF (EMOF) Model. L'obbiettivo primario di *Essential MOF* (EMOF) (OMG 2006A) è quello di consentire la definizione di semplici meta-modelli utilizzando semplici concetti; per questo motivo EMOF fornisce il minimo insieme di elementi necessari per modellare un sistema object-oriented. Per far ciò EMOF fonde il package *Basic* di UML 2.0 Core ed include capacità aggiuntive; inoltre, al fine di permettere la definizione di metamodelli più complessi, EMOF fonde le capacità fornite dai package Reflection, Identifier e Extension (figura 16).

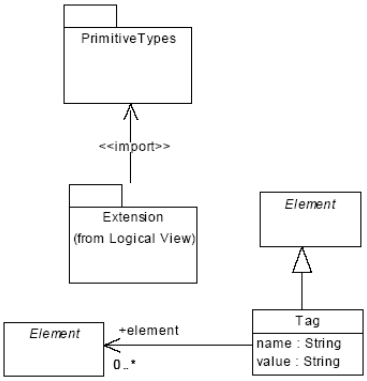


FIGURA 15. Package *Extension*

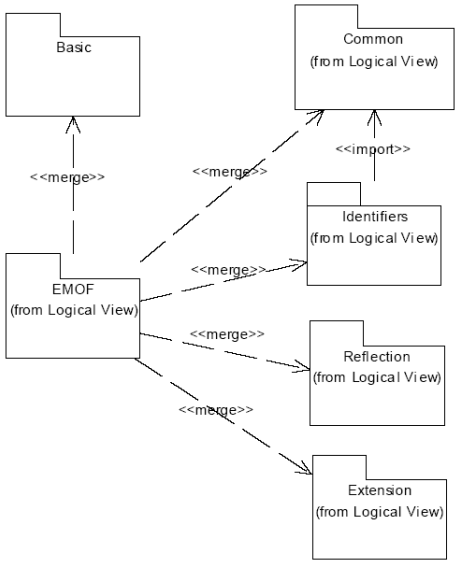


FIGURA 16. Modello Essential MOF (EMOF)

Si nota che EMOF non eredita il package Basic, cioè non definisce semplicemente delle sotto-classi che estendono le classi di Basic con nuovi attributi e operazioni, ma è un modello separato che fonde tale package: ciò è dovuto al fatto che il package Reflection introduce la classe Object come super-classe di Basic::Element la quale richiede la fusione.

Dal package InfrastrucutreLibrary::Core::Basic di UML 2.0 vengono importate le seguenti classi:

- *Class*: rispetto al package di UML 2.0 Infrastructure, EMOF estende la classe Property con l'attributo isID:Boolean il quale permette di indicare se la proprietà

può essere utilizzata come un identificatore per la classe che la contiene (tale attributo è introdotto dal package Identifier).

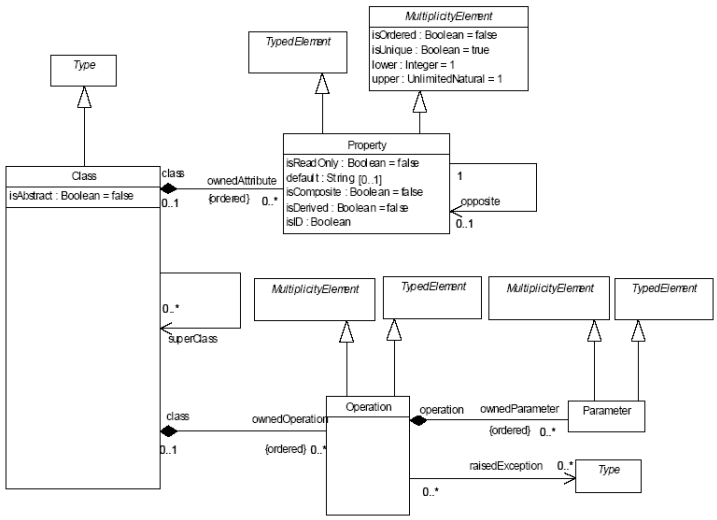


FIGURA 17. EMOF Class

- *DataType*: non presenta sostanziali modifiche rispetto InfrastrucutreLibrary::Core::Basic.
- *Type*: la classe Type viene estesa con l'opeazione `isInstance(element : Object) : Boolean`, la quale ritorna true se l'oggetto element è un'istanza di quel tipo.

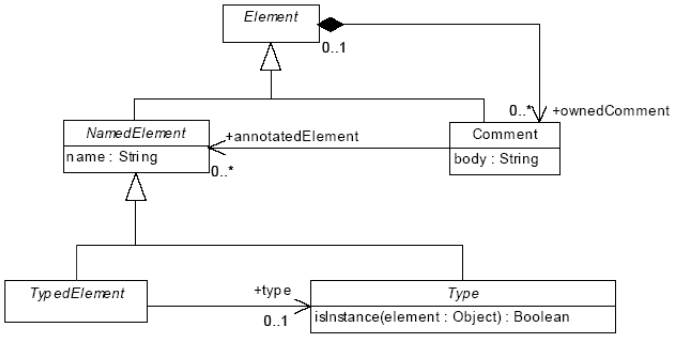


FIGURA 18. Tipi di EMOF

- *Package*: la classe Package è arricchita da Identifier dell'attributo `uri:String`, il quale fornisce un identificatore per il package.

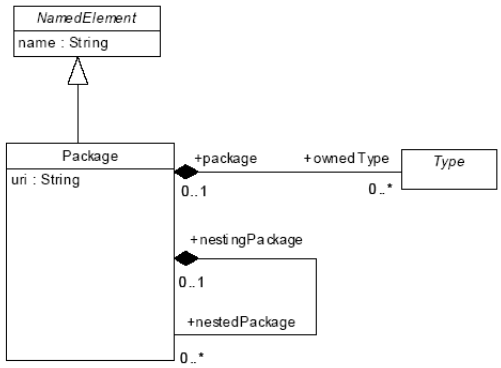


FIGURA 19. Package EMOF

Complete MOF (CMOF) Model. Il modello *Complete MOF* (CMOF) (OMG 2006A) è il meta-modello utilizzato per specificare altri meta-modelli come UML. CMOF non definisce nessuna nuova classe ma fonde assieme package già esistenti: in particolare CMOF è costruito partendo dal package EMOF e da Core::Constructs di UML 2.0 Infrastructure (vedi figura 20). Tale scelta non è casuale: CMOF deve estendere le funzionalità base specificate da EMOF e per tale motivo fonde il package Construct, il quale permette di importare tutto il package Core di Infrastructure Library (al contrario di EMOF che importava solo Basic).

Oltre a questo, CMOF estende il package Reflection con nuove classi; il package CMOF Reflection fonde le operazioni nelle classi Object, Factory e Extent, già esistenti, e aggiunge la classe Link e il datatype Argument. Dalla figura 21 si osserva che le classi definite in tale package sono:

- *Link*: è un'istanza di Association; i suoi attributi permettono di recuperare l'associazione di cui è istanza e i due estremi dell'associazione stessa.
- *Argument*: è un tipo di dato composto da un nome e da un valore utilizzato come parametro di una operazione.
- *Object*: tale classe è estesa con le operazioni delete() per eliminare l'oggetto, `invoke(op:Operation, arguments:Argument[0..*]):Element[0..*]` la quale invoca l'operazione fornita op sull'oggetto passandogli gli argomenti arguments, `isInstanceOfType(type:Class, includeSubtypes:Boolean):Boolean` la quale ritorna vero se l'oggetto è istanza della classe fornita.
- *Element*: anche questa classe è estesa con le stesse operazioni definite per Object.
- *Factory*: sono fornite due nuove operazioni, `createElement(class:Class, arguments: Argument[0..*]):Element` attraverso la quale è possibile creare nuovi elementi e contemporaneamente fornire i valori iniziali delle proprietà (attraverso gli argomenti passati), `createLink(association:Association, firstElement:Object, secondElement:Object):Link` la quale permette di creare un nuovo link e istanziare i suoi attributi.

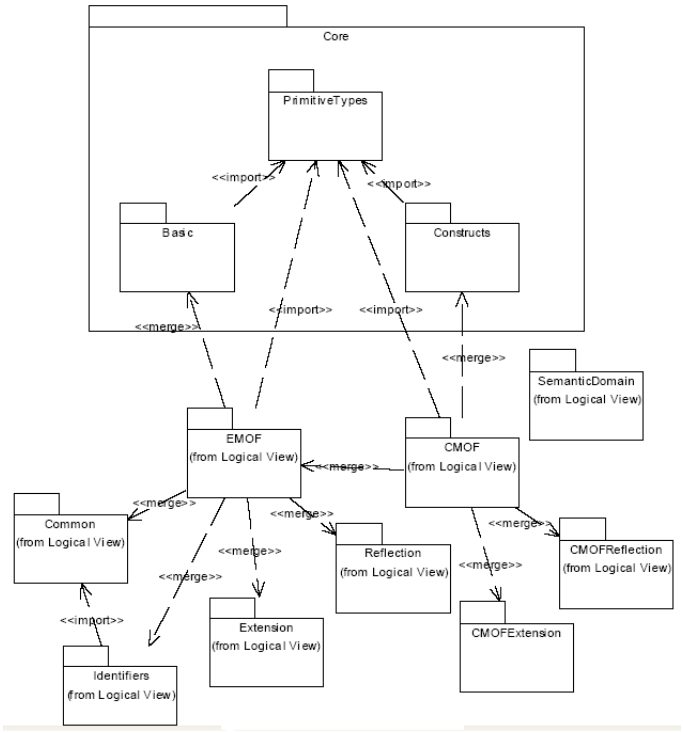


FIGURA 20. Modello Complete CMOF

- *Extent*: quattro sono le nuove operazioni che possono essere eseguite su un elemento Extent, `elementsOfType(type:Class, includeSubtypes: Boolean):Elements[0..*]` la quale ritorna gli elementi nell'extent che sono istanze della classe fornita (e delle sottoclassi se includeSubtypes==true), `linksOfType(type:Association):Links[0..*]` che ritorna invece i link nell'extent che sono istanze di type, `linkedElements(association: Association, endElement:Element, end1ToEnd2Direction: Boolean):Element[0..*]` la quale naviga le associazioni fornite dall'elemento fornito (che è il primo end dell'associazione se end1ToEnd2Direction== true), `linkExists(association:Association, firstElement: Element, secondElement:Element):Boolean` attraverso la quale è possibile verificare l'esistenza di un link per l'associazione fornita tra i due elementi indicati.

Come già detto, CMOF fonde il package *InfrastructureLibrary::Core::Construct* il quale definisce le seguenti sotto-package:

- *Root*: specifica le classi Element, Relationship, DirectRelationship e Comment. Element è l'elemento radice del modello, Relationship è un concetto astratto che specifica una generica relazione tra elementi diversi, DirectRelatonship è una relazione diretta tra un elemento sorgente e un elemento target e Comment rappresenta una annotazione testuale che può essere associata ad un insieme di elementi.

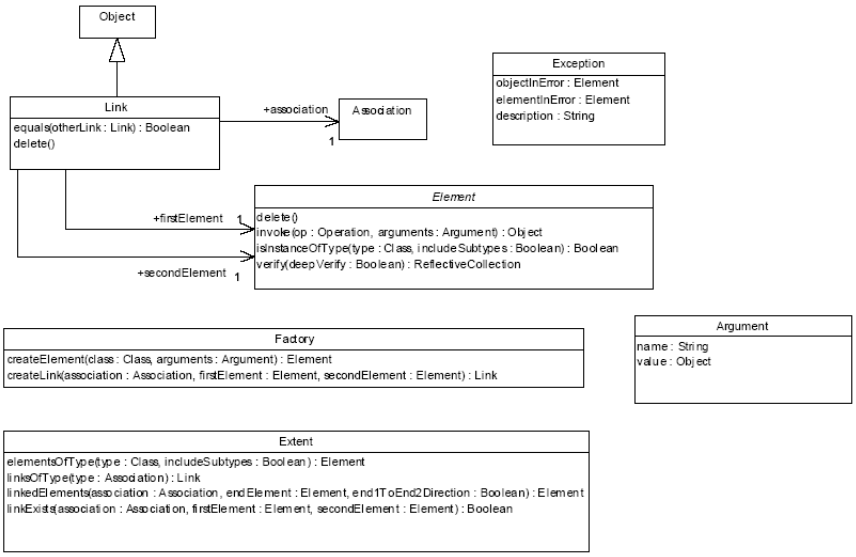


FIGURA 21. Il package CMOFReflection

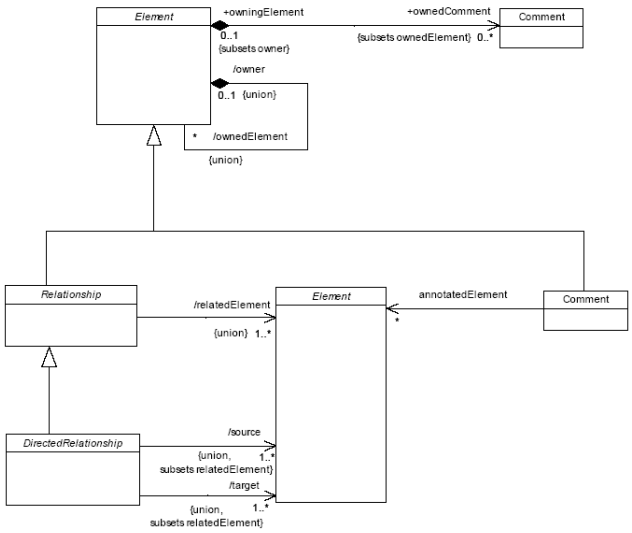


FIGURA 22. Package Root di Core::Constructs

❖ *Expressions*: specifica le classi ValueSpecification, la quale rappresenta uno specifico insieme di istanze (oggetti e valori), Expression, ovvero un albero strutturato di simboli che denotano un insieme di valori quando valutati in un contesto, OpaqueExpression, cioè un'espressione testuale non interpretata che denota un

insieme di valori quando valutati in un contesto. Esempi di espressioni sono xor, else, x+1, mentre un esempio di espressione opaca è a>0.

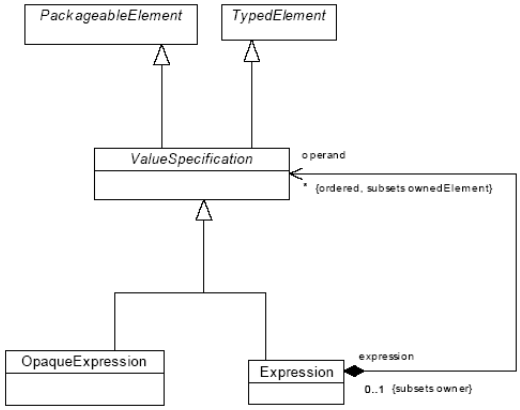


FIGURA 23. Package Expressions di Core::Constructs

- *Classes*: in questo package vengono definite Class, Association e Property (figura 24).
- *Classifiers*: la classe Classifier rappresenta una classificazione di istanze, ovvero descrive un insieme di istanze che hanno caratteristiche in comune, TypedElement, ovvero un elemento al quale è associato un tipo (ad esempio un attributo), un MultiplicityElement ovvero un elemento con specifici attributi per modellare la molteplicità, RedefinableElement il quale rappresenta un elemento (appartenente ad un certo classifier) che può essere ridefinito in modo più specifico o diverso nel contesto di un altro classifier che specializza il precedente, Feature è utilizzata per collegare caratteristiche ad un classifier e StructuralFeature ovvero una feature di un classifier che specifica la struttura delle istanze del calssifier stesso (un attributo, istanza di Property, è un esempio di StructuralFeature).
- *Namespaces*: la classe Namespace specifica un elemento del modello che contiene un insieme di elementi che possono essere identificati dal nome.
- *Constraints*: in tale package si definisce la classe Constraint, ovvero una restrizione espressa in linguaggio naturale o in un linguaggio machine readable la quale permette di specificare parte della semantica di un elemento; viene espressa con una stringa testuale del seguente `formato: constraint ::= { [ <name> : ] <Boolean esxpression> }`.
- *DataTypes*: nel quale si definiscono le classi DataType, Enumeration, EnumerationLiteral e PrimitiveType. Tale package è riportato in figura 28.

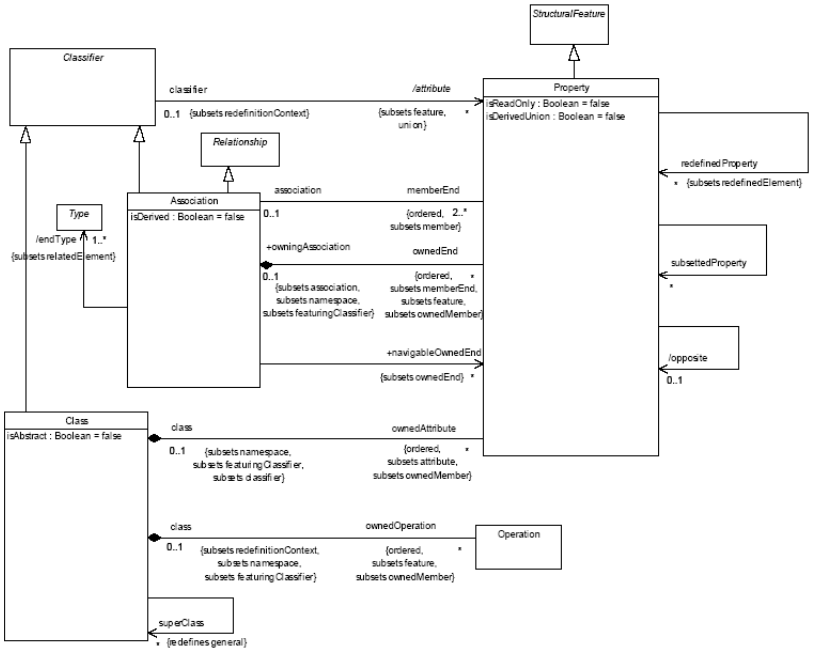


FIGURA 24. Package Classes di Core::Constructs

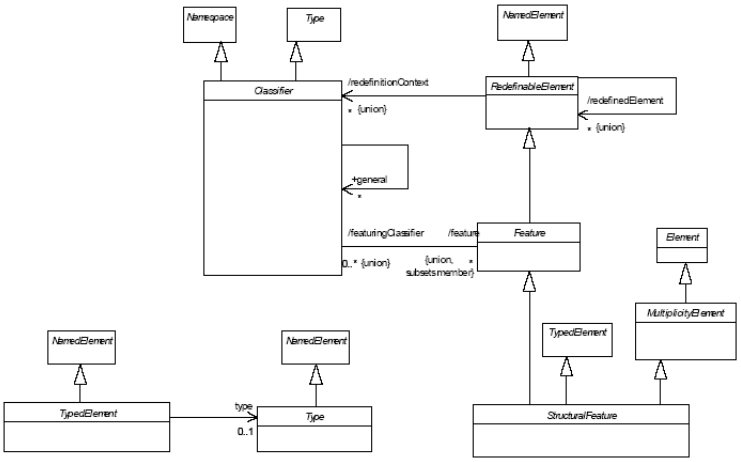


FIGURA 25. Package Classifiers di Core::Construct

❖ *Operations*: definisce le classi BehaviouralFeature, caratteristica di un classificatore che specifica un aspetto del comportamento delle sue istanze (una operazione, istanza di Operation, ne è un esempio), Operation (come specializzazione di BehaviouralFeature) e Parameter, la quale definisce un argomento (di ingresso o uscita) appartenente ad una behavioural feature.

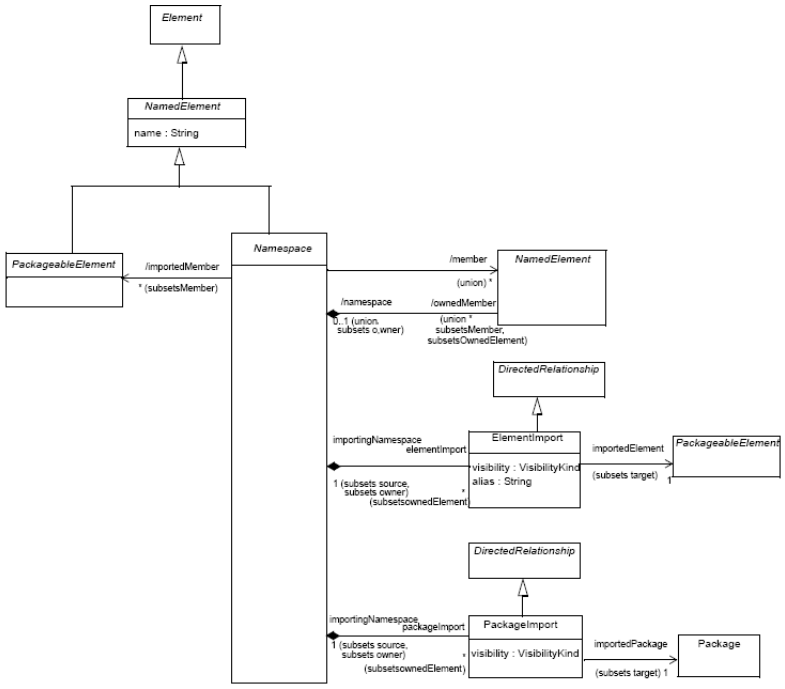


FIGURA 26. Package Namespace di Core::Constructs

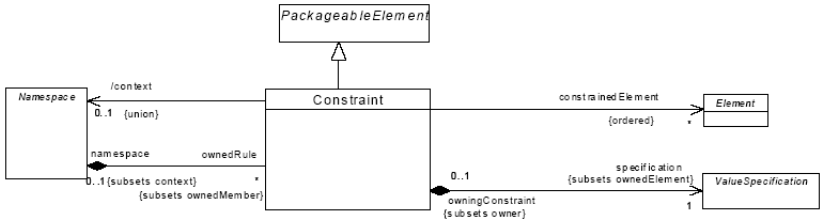


FIGURA 27. Package Constraint di Core::Constructs

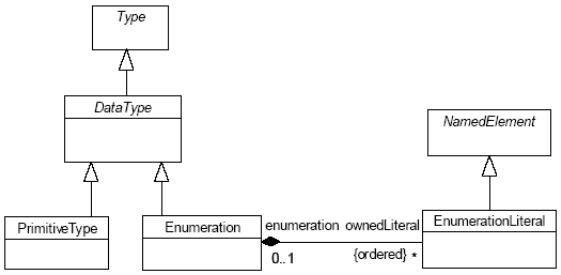


FIGURA 28. Package DataTypes di Core::Constructs

❖ *Packages*: nel quale sono definite le classi Package, utilizzato per raggruppare elementi e per fornire un namespace agli elementi raggruppati, e PackageMerge,

il quale definisce come il contenuto di un package è esteso da quello di un altro package.

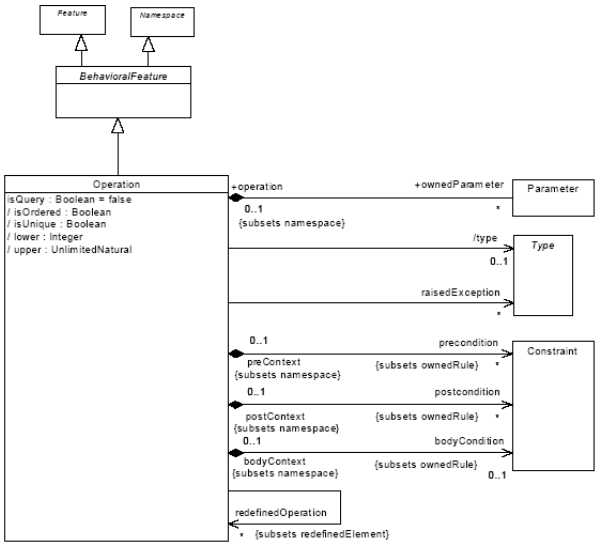


FIGURA 29. Package Operations di Core::Constructs

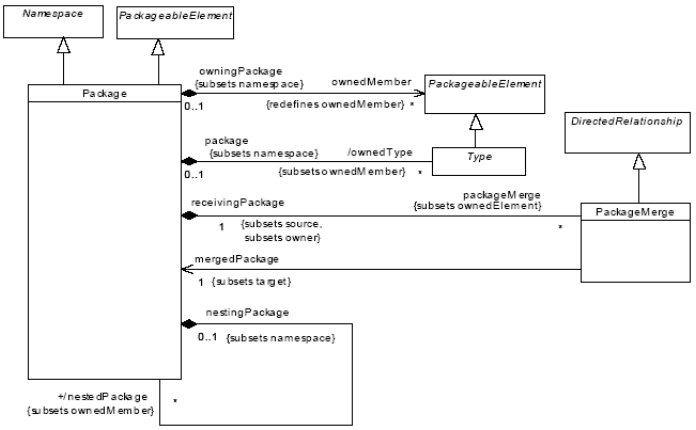


FIGURA 30. Package Packages di Core::Constructs

§

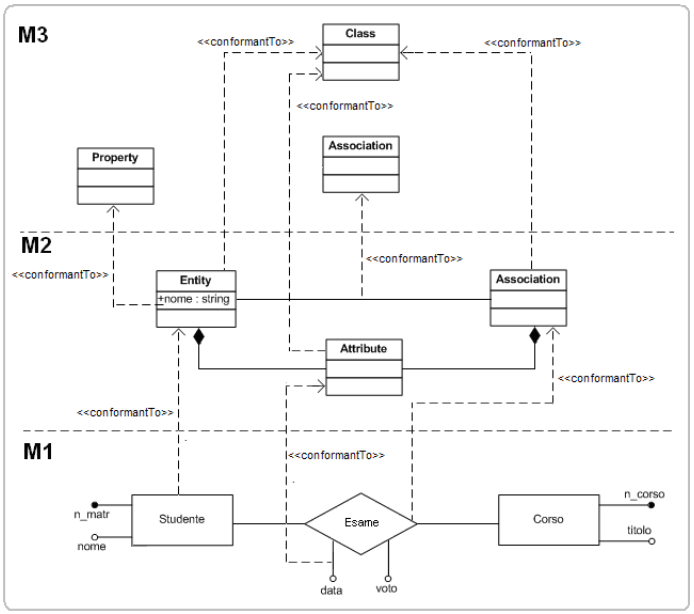


FIGURA 31. Framework MDA dal livello M3 al livello M1

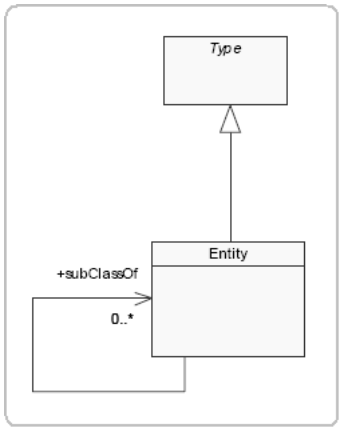


FIGURA 32. Diagramma Entity del modello E/R

Esempio di utilizzo del modello MOF. In conclusione, è stato utilizzato il modello MOF per implementare il modello Entity-Relationship (CHEN *et al.* 1976) in maniera MOF-compliant. In figura 32 è riportato il diagramma nel quale viene definito il concetto di *Entity*, mentre nel diagramma di figura 33 viene definito il concetto di *Relationship*. Infine, le *Key* e gli *Attribute* vengono definiti nel diagramma di figura 34.

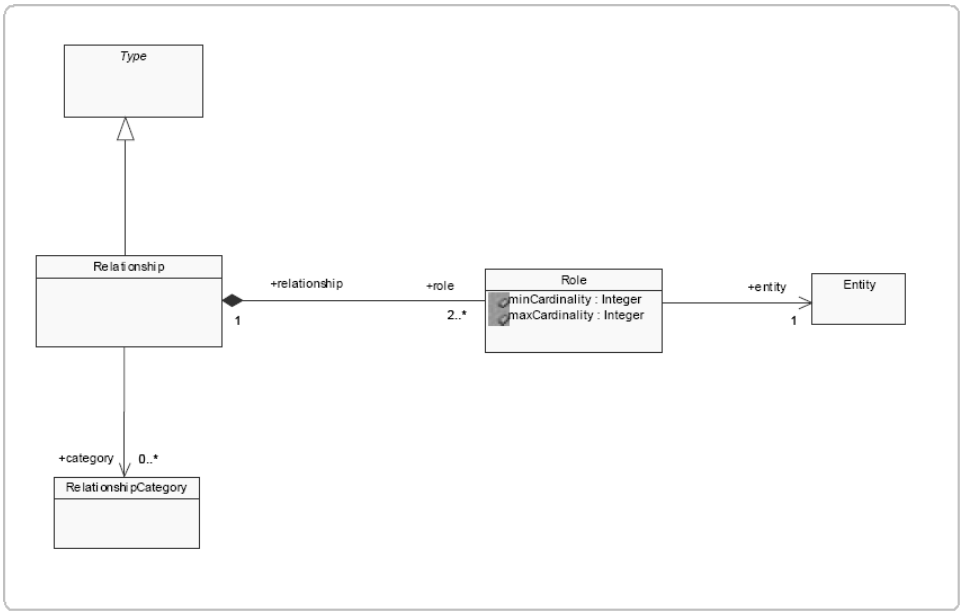


FIGURA 33. Diagramma Relationship del modello E/R

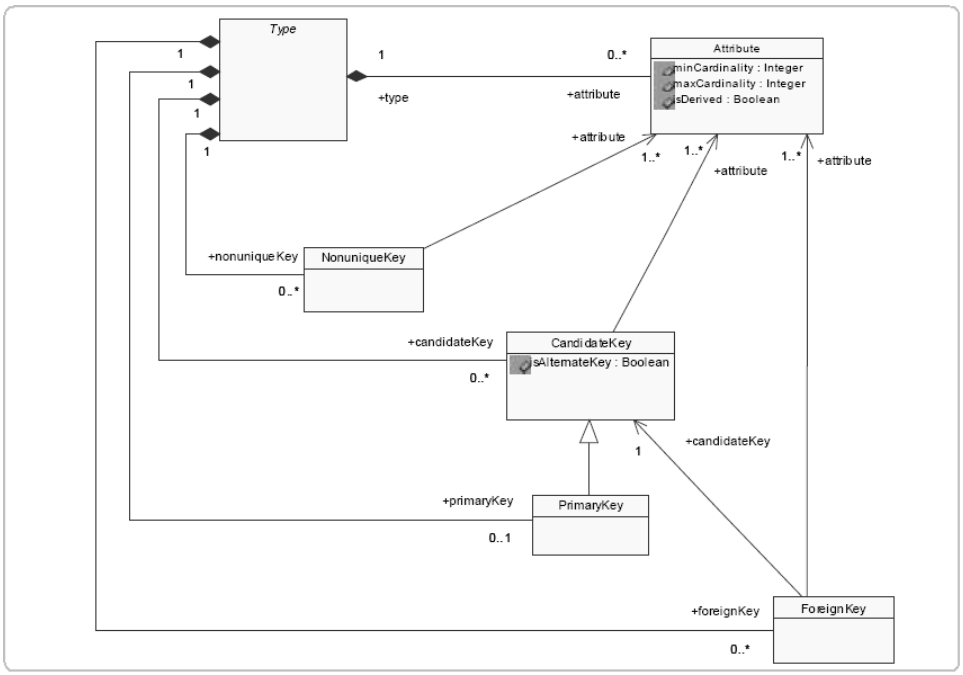


FIGURA 34. Diagramma Keys del modello E/R

Il modello realizzato si pone nel livello M2 dell'architettura MOF; la figura 31 fornisce una rappresentazione generale del framework dal livello M3 al livello M1.

3. XML Metadata Interchange (XMI) Format

XML Metadata Interchange (XMI) (OMG 2005B) è un linguaggio basato su XML, sviluppato sempre da OMG, attraverso il quale i sistemi software possono scambiarsi oggetti, i quali possono essere modelli MOF/UML, schemi di database, classi espresse in un linguaggio di programmazione, ecc.

In pratica, un documento XMI è un documento XML costruito rispettando certe regole; tali regole permettono di descrivere tutti gli elementi che compaiono in un documento XMI, la loro molteplicità e struttura.

Mentre il compito di XML è quello di condividere informazioni tra sistemi diversi, XMI è stato progettato con l'intento di realizzare una soluzione per scambiare modelli (metadati) tra tool di modellazione basati su UML e repository basati su MOF. In questo modo, i tool, per poter inter-operare fra loro, devono solo importare ed esportare dati in formato XMI; in questo modo si elimina dunque la necessità di realizzare diversi moduli import/export per ogni combinazione di tool che interagenti.

Dal momento che l'estensione delle informazioni che possono essere scambiate tra tool è limitato all'insieme delle informazioni che possono essere comprese da entrambi, XMI si basa su MOF il quale fornisce un modo standard per definire meta-modelli. Quindi, una DTD (o XML Schema) di un meta-modello si può ottenere in modo automatico definendo il meta-modello in MOF e applicando successivamente le regole di generazione di XMI.

Lo standard XMI è composto da:

- ❖ Regole per generare XML Document Type Definition[3] (DTD) o XML Schema[4] da meta-modelli MOF-based.
- ❖ Regole per generare documenti XML partendo da metadati MOF-based, e Metadati MOF-based da documenti XML.
- ❖ Direttive di progettazione per le DTD/XML Schema e per i dati XML XMI-related.
- ❖ Mapping UML-DTD e MOF-DTD (o UML-XML-Schema e MOF-XML Schema).

La produzione di documenti XMI consiste in particolare nella produzione di una DTD o di un XML Schema, detti rispettivamente XMI DTD e XMI Schema, le quali permettono di poter verificare, attraverso validazione XML, se un documento XMI sia o meno *well-formed*. Successivamente, altre regole di produzione vengo utilizzate per creare uno specifico documento XMI, conforme allo schema precedentemente creato. Di seguito verranno descritte alcune delle principali caratteristiche di questo standard. Non verranno descritte le regole di produzione degli schemi e dei documenti XMI, per le quali si rimanda a (OMG 2000; OMG 2005B).

§

Nel seguito vedremo le caratteristiche principali dello standard XMI relativamente alla produzione di schemi e documenti XMI.

[3] http://www.w3schools.com/dtd/
[4] http://www.w3.com/XML/Schema/

Principi base. L'organizzazione base di un XML Schema per XMI prevede che le dichiarazioni, i tipi, gli attributi degli elementi XML debbano essere inclusi negli schemi per poter permettere la validazione dei documenti generati. Alcuni di questi elementi XML contengono metadati riguardo i metadati da trasferire; ad esempio, l'identità del meta-modello associato ai metadati, il tool che ha generato i metadati, etc. Le specifiche prevedono inoltre che ogni classe di un meta-modello venga rappresentato nello schema da un elemento XML o da un complexType, il cui nome è il nome della classe. La dichiarazione del tipo classe prevede la dichiarazione degli attributi e dei riferimenti alle association end collegati alla classe. Inoltre, è previsto che ogni XMI Schema contenga un meccanismo per estendere una classe di un meta-modello. Gli elementi extension possono essere inclusi nel modello contenuto di una classe. Tale meccanismo può essere utile nel caso in cui si intenda trasmettere dati riguardo l'estensione di un meta-modello.

Le specifiche di XMI prevedono inoltre che ogni schema XMI debba consistere di:

- L'istruzione indicante la versione di XML, `<? XML version =1.0 ?>`.
- In generale, ogni altra istruzione di processing valida di XML.
- Un elemento XML schema.
- Un elemento XML importato per l'XMI namespace.
- Dichiarazioni per uno specifico meta-modello.

Ogni documento XMI consiste almeno delle seguenti dichiarazioni:

- L'istruzione iniziale di intestazione (con opzionalmente la dichiarazione di codifica).
- Ogni altra istruzione di processing valida di XML.

Modello XMI. Un modello XMI descrive la struttura di un documento XMI. Il modello XMI è una istanza di MOF e permette dunque la descrizione delle informazioni XMI contenute nei documenti XMI.

Il modello XMI è rappresentato dai tre diagrammi della figura 35.

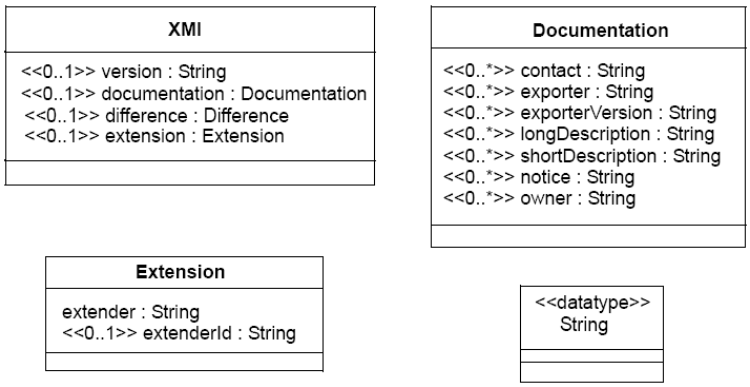


FIGURA 35. Diagrammi del modello XMI

La classe *XMI* rappresenta la classe radice; i suoi attributi sono la versione, documentazione, differenze ed estensioni. La classe *Documentation* contiene molti campi utilizzati per la descrizione del documento (descrizione che non ha scopi computazionali). La classe *Extension* contiene metadati utilizzati per informazioni esterne. Si nota che le informazioni di differenza sono definiti come *addizioni*, *eliminazioni* e *sostituzioni* di oggetti.

In particolare le classi *Add*, *Replace* e *Delete* specificano un insieme di differenze e si riferiscono a oggetti MOF che sono aggiunti o rimossi.

Classe XMI. XMI è l'elemento XML top level per un documento XMI. La sua dichiarazione è riportata in figura 36.

```
<xsd:complexType name="XMI">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:any processContents="strict"/>
 </xsd:choice>
 <xsd:attribute ref="id"/>
 <xsd:attributeGroup ref="IdentityAttribs"/>
 <xsd:attributeGroup ref="LinkAttribs"/>
 <xsd:attribute name="type" type="xsd:QName" use="optional"
                form="qualified"/>
 <xsd:attribute name="version" type="xsd:string" use="required" fixed="2.0"
               form="qualified"/>
</xsd:complexType>

<xsd:element name="XMI" type="XMI"/>
```

FIGURA 36. Dichiarazione di un elemento XMI

L'elemento XMI non deve per forza essere l'elemento radice di un documento XML; l'attributo `xmi:version` denota, oltre la versione di XMI, anche l'inizio delle informazioni XMI. Tale classe presenta inoltre il tag contentType settato come "any" per indicare che qualsiasi elemento XMI può essere rappresentato nel documento.

Classe Extension. Tale classe ha il compito di contenere informazioni estese, ovvero metadati non appartenenti a tale meta-modello; permette dunque di poter aggiungere estensioni al meta-modello di riferimento. Le estensioni sono attributi multivalore della classe XMI, ma possono essere incluse anche in altri locazioni di un documento. La sua dichiarazione è riportata in figura 37.

```
<xsd:complexType name="Extension">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:any processContents="lax"/>
 </xsd:choice>
 <xsd:attribute ref="id"/>
 <xsd:attributeGroup ref="ObjectAttribs"/>
 <xsd:attribute name="extender" type="xsd:string" use="optional"/>
 <xsd:attribute name="extenderID" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="Extension" type="Extension"/>
```

FIGURA 37. Dichiarazione della classe Extension

Si nota che l'attributo *extender* indica il tool che ha effettuato l'estensione.

Classi Add, Replace e Delete. La classe *Add* (figura 38) rappresenta l'aggiunta di un insieme di oggetti al documento descritto o ad altri documenti. L'attributo *position* indica dove posizionare l'oggetto rispetto l'elemento XML, mentre *addition* si riferisce all'insieme di documenti che devono essere aggiunti.

```
<xsd:complexType name="Add">
 <xsd:complexContent>
  <xsd:extension base="Difference">
   <xsd:attribute name="position" type="xsd:string" use="optional"/>
   <xsd:attribute name="addition" type="xsd:IDREFS" use="optional"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Add" type="Add"/>
```

FIGURA 38. Dichiarazione della classe Add

La classe *Replace* (figura 39) rappresenta la rimozione di un insieme di oggetti e l'aggiunta degli oggetti indicati dall'attributo *replacement*.

```
<xsd:complexType name="Replace">
 <xsd:complexContent>
  <xsd:extension base="Difference">
   <xsd:attribute name="position" type="xsd:string" use="optional"/>
   <xsd:attribute name="replacement" type="xsd:IDREFS" use="optional"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Replace" type="Replace"/>
```

FIGURA 39. Dichiarazione della classe Replace

La classe *Delete* (figura 40) rappresenta la rimozione di un insieme di oggetti da questo o da altri documenti.

```
<xsd:complexType name="Delete">
 <xsd:complexContent>
  <xsd:extension base="Difference"/>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Delete" type="Delete"/>
```

FIGURA 40. Dichiarazione della classe Delete

Classe Documentation. Tale classe contiene informazioni riguardo il documento XMI che sta per essere trasmesso come, ad esempio, il proprietario del documento, informazioni di contatto, una descrizione (lunga e corta) del documento, il tool che ha creato il documento e la sua versione, ecc. In figura 41 è riportata la sua dichiarazione.

```
<xsd:complexType name="Documentation">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="contact" type="xsd:string"/>
  <xsd:element name="exporter" type="xsd:string"/>
  <xsd:element name="exporterVersion" type="xsd:string"/>
  <xsd:element name="longDescription" type="xsd:string"/>
  <xsd:element name="shortDescription" type="xsd:string"/>
  <xsd:element name="notice" type="xsd:string"/>
  <xsd:element name="owner" type="xsd:string"/>
  <xsd:element ref="Extension"/>
 </xsd:choice>
 <xsd:attribute ref="id"/>
 <xsd:attributeGroup ref="ObjectAttribs"/>
 <xsd:attribute name="contact" type="xsd:string" use="optional"/>
 <xsd:attribute name="exporter" type="xsd:string" use="optional"/>
 <xsd:attribute name="exporterVersion" type="xsd:string" use="optional"/>
 <xsd:attribute name="longDescription" type="xsd:string" use="optional"/>
 <xsd:attribute name="shortDescription" type="xsd:string" use="optional"/>
 <xsd:attribute name="notice" type="xsd:string" use="optional"/>
 <xsd:attribute name="owner" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:element name="Documentation" type="Documentation"/>
```

FIGURA 41. Dichiarazione della classe Documentation

Attributi di XMI. Vediamo ora i principali attributi utilizzati nella produzione di documenti e schemi XMI.

Attributi di identificazione. Questi attributi sono utilizzati per identificare gli elementi XML in un documento, il che consente anche ad un elemento di poter essere associato ad altri. Tra questi attributi vi sono:

- *id:* permette di identificare un elemento all'interno del documento. La semantica di XML richiede che il valore di tale attributo sia univoco all'interno del documento in cui è dichiarato.
- *label:* può essere utilizzato per fornire un'etichetta di tipo string che identifica un particolare elemento XML.
- *uuid:* l'obbiettivo di questo attributo è quello di fornire un identificatore globale per l'elemento. Il formato che deve rispettare è "id namespace:uuid".

Attributi di linking. Lo scopo di tali attributi è quello di permettere agli elemento XML del documento di riferirsi ad altri elementi. In questo tipo di attributi rientrano:

- *href:* consente di specificare un URI per riferirsi ad un determinato elemento. Ad esempio, questo attributo può essere utilizzato per riferirse a quegli elementi XML il cui valore dell'attributo id è un URI.

- ❖ *idref:* consente ad un elemento XML di riferirsi ad un altro elemento XML appartenente allo stesso documento attraverso il meccanismo IDREF di XML.
- ❖ *uuidref:* permette ad un elemento XML di riferirsi ad un altro elemento XML appartenente allo stesso documento utilizzando il valore dell'attributo uuid dell'elemento di destinazione.

Attributo type. L'attributo *type* è utilizzato per indicare il tipo dell'oggetto da serializzare.

Elementi principali di XMI. Alcuni degli elementi che uno schema XMI-compliant deve dichiarare sono i seguenti:

- ❖ *XMI.header:* contiene gli elementi XML che permettono di identificare il modello, meta-modello, e meta-meta-modello dei metadati serializzati. Opzionalmente è possibile includere nell'header anche un elemento *documentation*.
- ❖ *XMI.content:* tale elemento rappresenta il contenuto del documento, ovvero i metadati da trasferire. In un documento XMI la sua dichiarazione segue quella dell'header.
- ❖ *XMI.extensions:* tale elemento contiene quegli elementi XML contenenti metadati che permettono di estendere il meta-modello.
- ❖ *XMI.model:* identifica il modello rispetto al quale le istanze che che si intendono trasferire sono conformi. Si nota che potrebbero essere indicati più modelli se le istanze sono conformi a più modelli diversi.
- ❖ *XMI.metamodel:* identifica il meta-modello. Come nel caso precedente, è possibile indicare più meta-modelli nel caso in cui il modello sia conforme a più linguaggi.
- ❖ *XMI.metametamodel:* identifica il meta-meta-modello utilizzato per modellare il meta-modello trasferito. Di default, tale elemento è MOF.

4. MDA e Database Design

Anche la progettazione delle basi di dati segue, come MDA, un approccio model-driven. Si possono distinguere in particolare tre passaggi (BERGAMASCHI *et al.* 1999):

- ❖ *Progettazione concettuale*: consiste nella definizione di un modello della realtà di interesse ad un alto livello di astrazione; il modello più utilizzato in questa fase è il modello Entity Relationship (E/R).
- ❖ *Progettazione logica*: in questa fase il modello concettuale viene tradotto in un modello logico dei dati (relazionale, ad oggetti, etc.). Il modello più diffuso è probabilmente il modello relazionale.
- ❖ *Progettazione fisica*: il modello logico viene implementato fisicamente utilizzando il linguaggio SQL.

Tali passaggi corrispondono dunque a livelli di astrazione diversi: con il livello concettuale si vuole rappresentare la semantica di un determinato dominio in modo indipendente dalle specifiche tecnologie, cioè il suo scopo è solo quello di catturare e descrivere in modo comprensibile anche dagli esseri umani i tipi di entità coinvolte e le loro relazioni, con il livello logico la descrizione concettuale è mappata in una specifica tecnologia, ad esempio quella relazionale, mentre con il livello fisico si descrive il livello logico in relazione ad una specifica piattaforma (quindi rispettando i vincoli che essa impone).

Si nota dunque che i modelli logici e fisici sono gli equivalenti in MDA dei modelli PIM e PSM: il modello logico, come quello PIM, è indipendente dalla piattaforma e dunque

deve essere trasformato secondi i vincoli della piattaforma del sistema, generando quindi un modello fisico (che è quindi un PSM), quindi un mapping fra schema logico e fisico corrisponde semplicemente ad una trasformazione tra modelli a diversi livelli di astrazione (cioè tra PIM e PSM). Ciò è vero anche nel caso in cui, invece di avere un sistema con un singolo database, si pensa ad un sistema distribuito che coinvolge più sorgenti informative con schemi eterogenei (BERGAMASCHI *et al.* 2001): in questo caso si dovranno prevedere più mapping fra gli schemi. Quindi anche nei sistemi I^3 si nota la corrispondenza con l'approccio MDA.

Una prima differenza consiste nel fatto che, mentre livello logico e concettuale rappresentano esclusivamente dati e query, i modelli di MDA sono stati pensati per rappresentare tutti i possibili processi di un dato sistema.

Inoltre, si nota che il livello concettuale non trova corrispondenze in MDA (BOUZEGHOUB 2003), a testimonianza del fatto che la suddivisione dei ruoli di ogni livello nei database è più precisa che in MDA. Si potrebbe ipotizzare che un modello concettuale corrisponda ad un modello CIM, in quanto sia un modello CIM che un modello E/R permettono di descrivere un determinato dominio da un punto di vista ontologico, cioè cercando di rappresentare solo la semantica. La differenza consiste nell'approccio progettuale: lo schema concettuale è uno schema logico ad un livello di astrazione maggiore, e lo stesso vale tra schema logico e schema fisico. Quindi, la progettazione di un database prevede prima la generazioni di una visione concettuale della realtà da modellare, la sua traduzione in uno schema logico e infine l'implementazione fisica.

In MDA l'approccio è diverso: CIM non è il livello più astratto di un modello PIM, ma semplicemente una rappresentazione del sistema a livello di business; quindi mentre schema concettuale e logico descrivono le stesse informazioni da livelli di astrazione diversi, CIM e PIM descrivono aspetti diversi dello stesso sistema.

5. MDA e Semantic Web

Il termine *Web Semantico* è stato proposto per la prima volta nel 2001 da Tim Berners-Lee, il quale lo definisce come "un'estensione del Web corrente in cui le informazioni hanno un ben preciso significato e in cui computer e utenti lavorano in cooperazione" (BERNERS-LEE *et al.* 2001). Da allora il termine è stato associato all'idea di un Web nel quale agiscano agenti intelligenti, applicazioni in grado di comprendere il significato dei testi presenti sulla rete e perciò in grado di guidare l'utente direttamente verso l'informazione ricercata, oppure di sostituirsi a lui nello svolgimento di alcune operazioni.

Nella visione di Berners-Lee, il Semantic Web dovrà gestire un ampio spettro di metadati, e dovrà descrivere ontologie che assegnano un significato ai dati che sia anche *machine-understandable*. Quello di ontologia è uno dei concetti principali nei sistemi di rappresentazione della conoscenza e, quindi, assume un ruolo centrale anche nell'ambito del Semantic Web. In letteratura sono presenti varie definizioni di ontologia: quella più diffusa è quella proposta da GRUBER (1993), il quale la definisce come una concettualizzazione formale e condivisa di un dominio particolare, ovvero come l'identificazione degli oggetti che si ipotizza esistano nel mondo e le relazioni fra essi. Altre definizioni sono state date da GUARINO (1998) e da GENESERETH e NILSSON (1987). Le ontologie sono essenziali nei sistemi di knowledge management, nei sistemi ad agenti, di e-commerce, ecc. Queste ontologie devono essere sufficientemente interconnettibli le une alle altre per consentire la creazione di grandi dizionari che possano essere compresi dagli agenti software senza

un'interferenza diretta di un umano. Quindi, il World Wide Web e XML forniranno le ontologie per scopi di interoperabilità e per facilitare la comprensione dei dati da parte delle applicazioni.

Le tecnologie sulle quali il Semantic Web si dovrà fondare dovranno essere sufficientemente "intelligenti" per consentire l'esecuzione di inferenze su enormi quantità di dati rimanendo comunque semplici da utilizzare.

Attualmente, lo strumento adottato per ottenere interoperabilità sintattica è XML[5] (*eXtensible Markup Language*), un meta-linguaggio utilizzato per definire altri linguaggi. Esso descrive una classe di data object, chiamati documenti XML, e descrive parzialmente il comportamento di programmi che processano tali documenti. XML non definisce nè tag nè la grammatica, cioè è un linguaggio completamente estendibile. E' solo richiesto che il documento sia ben-formato in una struttura ad albero. E' inoltre necessario che il documento XML sia valido, ovvero sia conforme al suo XML Schema, il quale definisce la grammatica e l'insieme dei tag della specifica formattazione XML.

Come MDA, anche il Web Semantico si basa su una architettura di metadati composta da più layer (figura 42). In questa architettura, le primitive vengono incrementalmente introdotte dai linguaggi dei livelli inferiori fino a quelli dei livelli superiori; in questo modo, i linguaggi di ogni livello riescono a soddisfare i requisiti di diversi tipi di applicazioni.

Mentre XML e XML Schema consentono una sintassi comune, well-defined e semplice da processare, essi non dicono nulla riguardo la semantica dei dati che descrivono. Ciò significa che altri standard devono essere costruiti nei livelli superiori a XML per descrivere semanticamente i dati. Un primo step in questa direzione è il linguaggio *Resource Description Framework* (*RDF*) (MANOLA & MILLER 2004), un modello generale che risiede a livello di metadati e *RDF Schema* (*RDFS*) (BRICKLEY & GUHA 2004), il linguaggio al livello schema. Il modello dei dati RDF definisce un modello semplice per descrivere le interrelazioni fra le risorse in termini di proprietà e valori. Le proprietà RDF possono essere pensate come gli attributi delle risorse, e in questo senso corrispondono alle tradizionali coppie attributo-valore. Inoltre, le proprietà rappresentano anche le relazioni tra risorse. L'RDF Schema dichiara queste proprietà e fornisce un meccanismo per definire le relazioni tra le proprietà e le altre risorse.

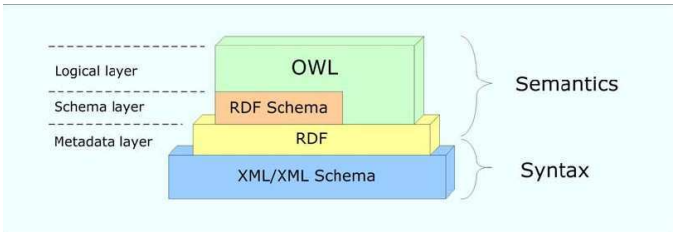


FIGURA 42. I tre layer di metadati del Semantic Web

Per consentire l'esecuzione di inferenze per il Semantic Web, è necessario aggiungere un ulteriore layer sopra RDF(S). Questo è il *logical layer*, il quale introduce i linguaggi per le ontologie, i quali sono basati sull'architettura di meta-modellazione definita dai layer

[5]XML Web Site, http://www.w3.org/XML/

inferiori. Tali linguaggi introducono un insieme arricchito di primitive di modellazione le quali possono essere mappate nelle Logiche Descrittive[6]. Questo consente l'utilizzo di tool che forniscono un generico supporto al ragionamento, in modo indipendente dallo specifico domino. Esempi comuni di linguaggi per ontologie sono OIL, DAML ma, in particolare, OWL.

Il *Web Ontology Language* (*OWL*) (MCGUINNESS & VAN HARMELEN 2004) è un linguaggio di markup semantico pensato per pubblicare e condividere ontologie sul Web. OWL è sviluppato come una estensione di RDF ed è derivato dal linguaggio DAML+OIL.

OWL, inoltre, fornisce una semantica di mondo aperto e consente di importare e unire diverse ontologie. Per poter fornire tali capacità e per supportare il ragionamento in un tempo finito, OWL introduce tre sotto-linguaggi, di espressività crescente: *OWL Full*, *OWL DL* e *OWL Lite*.

OWL Full fornisce la massima espressività ma non fornisce nessuna garanzia dal punto di vista computazionale. La principale caratteristica di OWL Full, in confronto a OWL DL e OWL Lite, è che una classe, che per definizione è una collezione di individui, può essere essa stessa un individuo, come avviene in RDF(S). E' ovvio che questo approccio può portare a modelli che richiedono un tempo infinito per essere computati. OWL DL (Logiche Descrittive) forniscono una espressività massima garantendo comunque la completezza computazionale. Include tutti i costrutti di OWL Full, con dei vincoli aggiuntivi. Il principale vincolo è quello che prevede che le classi non possono essere individui o proprietà, o che le proprietà non possono essere individui o classi. Infine, OWL Lite supporta sostanzialmente classificazioni gerarchiche e semplici vincoli. Si nota infine che OWL Full è una estensione di OWL DL, il quale, a sua volta, è un'estensione di OWL Lite. Pertanto, ogni ontologia OWL Lite è anche un'ontologia OWL DL e OWL Full e ogni ontologia OWL DL è anche una ontologia OWL Full.

§

I due approcci presentano similitudini e differenze. A livello architetturale, PAN e HORROCKS (2001) sottolineano che l'architettura del web semantico è non fissa, al contrario di quella definita da MOF. Ciò che distingue un'architettura non fissa da una fissa è la possibilità di avere, nella prima tipologia, un numero qualsiasi di livelli di classi. Per quanto riguarda il Semantic Web questa proprietà è dovuta all'esistenza di un doppio ruolo per certi elementi di RDFS (come rdfs:subClassOf), imputabile al fatto che RDFS non distingue l'informazione di modellazione nel livello delle ontologie da quella del livello del linguaggio (non distingue, ad esempio, la relazione subClassOf definita dal linguaggio da quella utilizzata in una ontologia). Seppure ciò comporti una maggior compattezza del linguaggio, implica altre sì che la semantica di RDFS sia poco chiara e ciò porta gli autori a concludere che RDFS non sia il linguaggio più adatto per descrivere le ontologie nell'ambito del Semantic Web. Per contro, MDA presenta una architettura fissa, con due livelli di classi ripartiti sul livello M2 (dove si definisce il concetto di Classe) e sul livello M3 (dove si utilizza il concetto di Meta-classe). Questa configurazione ha il vantaggio, secondo Pan e Horrocks, di avere una semantica chiaramente formalizzata e di essere più semplice da capire e da utilizzare.

[6] http://dl.kr.org/

Questa caratteristica di RDF(S) è parzialmente risolta in OWL DL introducendo nuovi elementi di modellazione, come owl:Class, utilizzati per definire le ontologie. In questo caso, rdfs:Class è utilizzato solo per definire owl:Class, owl:ObjectProperty e altre primitive di modellazione e non come elemento per definire ontologie. Al contrario, OWL Full consente un utilizzo non vincolato dei costrutti di RDF(S), pertanto eredita tutti i problemi di RDF(S) stesso.

Si nota inoltre che il concetto di classe in RDF(S) e OWL è diverso da quello definito in MOF e UML. Infatti, owl:Class rappresenta un insieme di individui, chiamato *class extension*, i quali sono istanza della classe. In OWL due classi possono avere lo stesso *class extension* pur rimanendo classi diverse. Inoltre, una istanza di una classe RDF(S) o OWL può appartenere a molti class extension nelle stesso momento, mentre in MOF questa può essere istanza di sono una classe. Infine, le classi RDF(S) e OWL non definiscono direttamente nessun attributo o relazione con altre risorse, le quali sono definite come proprietà, e non esiste nessun concetto simile a quello di operazione (o metodo).

Un'ulteriore differenza, ben conosciuta, è quella che riguarda le associazioni. In RDF(S) le associazioni sono modellate come proprietà (rdf:Property); OWL estende RDF(S) distinguendo due tipi di relazioni: le Object Property (owl:ObjectProperty), le quali corrispondono a relazioni tra classi e le Datatype Property (owl:DatatypeProperty), le quali corrispondono invece al concetto di attributo nei tradizionali linguaggi object-oriented. MOF e UML definiscono il concetto di associazione, ma con delle differenze importanti. Mentre nei linguaggi per il Semantic Web le proprietà sono delle primitive del linguaggio, ciò non è vero per MOF. Ciò significa che in OWL è possibile definire delle proprietà indipendentemente dalle classi che associano, cioè anche se non vi sono classi associate ad esse. Questo non è vero per MOF.

Si possono comunque individuare alcune similitudini. Un possibile parallelismo tra le due architetture è mostrato in tabella 3.

Metalivello	**Termine MOF**	**MOF come modello di rappresentazione di ontologie**	**UML**	**Semantic Web**
M3	Meta-meta-modello	Modello MOF	Modello MOF	-
M2	Meta-modello	Linguaggi di rappresentazione della conoscenza	Meta-modello UML	RDFS, OWL, DAML+OIL, ODL_{I3}
M1	Metadati	Ontologia	Modello UML	Ontologia RDFS, OWL, etc.
M0	dati	Database	Oggetti	Istanze

TABELLA 3. Corrispondenze fra l'architettura definita da MOF e quella definita per il Semantic Web

Si nota in realtà che più di una corrispondenza delle architetture, la tabella mostra il posizionamento dei concetti del Semantic Web nel framework definito da MDA. In particolare, oltre alle ontologie definite al terzo livello dell'architettura del Semantic Web, a livello M1 trovano posto gli schemi definiti utilizzando il linguaggio RDF, mentre lo stesso linguaggio può posizionarsi a livello M2, assieme ai linguaggi di rappresentazione della conoscenza. Infatti, seppure in entrambi gli approcci ad ogni nuovo livello vengono definiti e utilizzati linguaggi nuovi e i rispettivi metadati, per i linguaggi del Semantic web non vale la relazione conformedTo, che lega invece un linguaggio al livello M_i ad un linguaggio al livello M_{i+1} dello stack MOF. Quindi non è possibile definire una corrispondenza univoca tra i livelli delle due architetture, ma, piuttosto, si viene a delineare tra esse una relazione di *ortogonalità*.

La differenza sostanziale tra le tecnologie del Semantic Web e di MDA, che si riflette anche sulla struttura delle architetture, è il loro focus: MOF è interessato alla gestione automatica e integrata dei metadati, mentre lo sforzo della comunità della rappresentazione della conoscenza è rivolto verso la rappresentazione della semantica incorporata nel contenuto dei metadati e la possibilità di effettuare dei ragionamenti automatici su di essa. Questa rappresenta una critica distinzione ma che puntualizza le potenziali aree dove una tecnologia può beneficiare dell'altra. Le tecnologie di MDA vengono utilizzate per sviluppare sistemi aziendali, embedded e real-time su grande scala, senza occuparsi della semantica del dominio che definiscono. Le tecnologie del Semantic Web sono utilizzate principalmente in applicazioni riguardanti risoluzione di termini, e per garantire un'interpretazione del contenuto non ambigua da parte delle macchine, ma poche di esse sono state implementate su una vasta scala.

Questo scenario potrebbe rivestire particolare importanza in tutti quelle applicazioni sensibili al contenuto e al contesto (applicazioni di business intelligence, finanziarie, mediche, di sicurezza, etc) e che richiedono scalabilità. In generale, il modo in cui le applicazioni possano sfruttare le caratteristiche di entrambe le tecnologie è ancora una questione aperta.

Un passo in avanti si potrà fare grazie allo sviluppo dell'*Ontology Definition Metamodel* (*ODM*), un linguaggio basato su MOF per la definizione di ontologie. Infatti, nel 2003, OMG ha rilasciato una Request For Proposal (OMG 2003C) per tale linguaggio; gli obbiettivi che si vogliono raggiungere sono quelli di consentire la definizione di ontologie attraverso tool UML e l'implementazione di tali ontologie in OWL, senza perdita di semantica. Lo scopo è quello di consentire l'interoperabilità tra le tecnologie di rappresentazione della conoscenza e di metadata management. La modalità con cui si vuole raggiungere l'interoperabilità è quella di definire un insieme di meta-modelli, relazionati ma distinti, che rispettino le specifiche di ODM, tra i quali (fidura 42):

- ❖ Un meta-modello di alto livello che rappresenti i concetti di logica descrittiva che devono essere usati principalmente per definire mapping con altri meta-modelli.
- ❖ Un meta-modello che rappresenti la sintassi astratta di RDFS e OWL.
- ❖ Un meta-modello che rappresenti la sintassi astratta di Simple Common Language (SCL), il quale può essere utilizzato come un linguaggio di constraint per ODM o come un linguaggio di rappresentazione della conoscenza.
- ❖ Un meta-modello rappresentante la sintassi astratta di Topic Maps.
- ❖ Un meta-modello rappresentante i concetti chiave del modello entity-relationship.

Oltre ai meta-modelli, di fondamentale importanza in ODM sono i mapping fra meta-modelli e i profili UML, i quali aggiungono una notazione grafica ai meta-modelli.

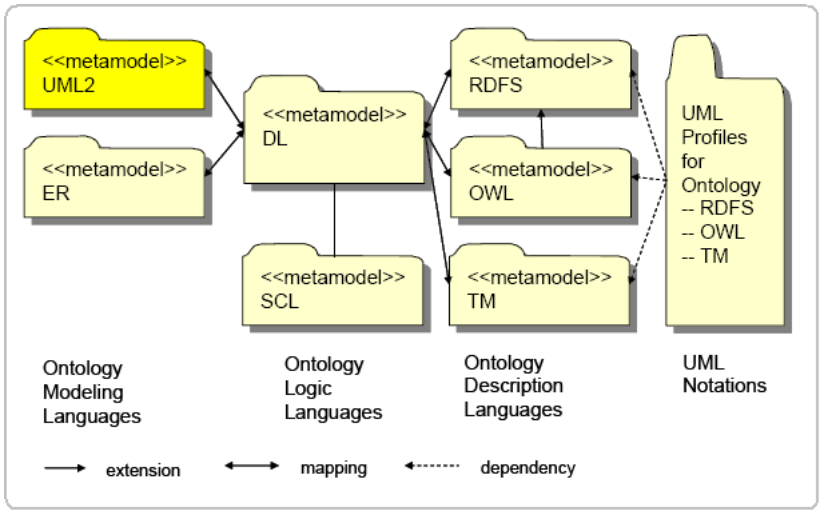


FIGURA 43. Architettura proposta per ODM

Una implementazione del linguaggio ODM è stata realizzata all'interno del progetto DBE, presentato nel prossimo capitolo, con lo scopo di aggiungere semantica ai modelli definiti per la descrizione di business e servizi.

CAPITOLO 3

Digital Business Ecosystem (DBE)

1. Introduzione

Digital Business Ecosystem (*DBE*)[1] è un Integrated Project finanziato dall'Unione Europea il cui scopo è quello di creare un'infrastruttura di prossima generazione che sia in grado di supportare la creazione e l'evoluzione di comunità in un ambiente di business digitale.

Tale progetto, di durata triennale (Novembre 2003- Fine 2006) e appartenente al 6th Framework Programme dell'IST della Commissione Europea, è considerato dalla Commissione Europea come uno dei progetti chiave nell'area dell'ICT for e-business, sottolineato dai sostanziosi finanziamenti ottenuti (10.5 MEuro) e dai numerosi (venti) e importanti partner coinvolti, tra i quali si riconoscono:

- T6, Italia (coordinatrice del progetto)
- London School of Economics and Political Science, Gran Bretagna
- Sun Microsystems Iberica, Spagna
- IBM Belgium - Business Consulting Services, Belgio
- INTEL Ireland, Irlanda
- CENSIS - Centro Studi Investimenti Sociali, Italia
- Soluta.net, Italia
- University of Birmingham - Computer Science Department, Gran Bretagna

L'obbiettivo finale del progetto è quello di costruire un marketspace virtuale che porti i vari attori del sistema, ovvero aziende, rivenditori, clienti, fornitori di servizi commerciali, a stringere rapporti di business attraverso lo scambio di messaggi e all'esecuzione di servizi sulla rete Internet.

In generale, DBE lo si può intendere come una combinazione di un business ecosystem, una rete di clienti e fornitori in un ambiente socio-economico, e di un digital ecosystem, ovvero una infrastruttura digitale auto-organizzata rappresentante un ambiente digitale per organizzazioni collegate in rete che supporta la condivisione di conoscenza.

Da questa definizione si può capire come sia forte la metafora con gli ecosistemi naturali; infatti, uno degli ambiziosi obbiettivi del progetto è quello del *darwinismo digitale*: l'infrastruttura digitale deve essere in grado di evolvere nel tempo per rispondere alle nuove esigenze dei sui membri e per supportare nuovi modelli di business.

Pertanto, la ricerca condotta all'interno del progetto assume un carattere fortemente interdisciplinare: varie attività di ricerca sono svolte nell'ambito della computer science, delle scienze sociali (in particolare economiche) e delle scienze naturali.

DBE è basato sui seguenti principi:

[1] http://www.digital-ecosystem.org

- ❖ Utilizzo di Internet come ambiente dove fornire servizi e fare transazioni.
- ❖ Utilizzo di principi e strumenti open-source (nessuna organizzazione deve dominare l'ecosistema).
- ❖ Utilizzo di applicazioni e servizi in grado di evolvere in base alle necessità degli utenti.

Tra le sue caratteristiche principali vi sono:

- ❖ Basato su un largo insieme di avanzate tecnologie dell'ICT.
- ❖ E' un middleware, una piattaforma dove nuovi modelli di business possono comparire.
- ❖ Fornisce varie funzionalità tecniche (service discovery, aggregator, advisor, evolution feature, ...).

2. Gli Ecosistemi Digitali

Quella degli *ecosistemi digitali*[2], o *digital ecosystem*, rappresenta una importante iniziativa intrapresa dall'Unione Europea nel 2002 (NACHIRA 2002) per meglio supportare l'adozione e lo sviluppo delle tecnologie dell'informazione e della comunicazione da parte delle Piccole e Media Imprese (PMI) europee. Lo scopo di questo concetto è quello di implementare in modo concreto gli obbiettivi definiti nel consiglio europeo di Lisbona del 2000: crescita maggiore, miglioramento della competitività, modernizzazione del modello sociale europeo, tutto nel rispetto delle peculiarità dello sviluppo europeo, basato principalmente su una diffusa rete di PMI.

Nella visione sviluppata dall'Unione Europea, i digital ecosystem rappresentano l'ultima fase dell'adozione di tecnologie per l'informazione e la comunicazione nelle aziende, come mostrato in figura 1.

Le Pmi, nucleo della struttura industriale italiana e europea, a causa delle loro conoscenze focalizzate e delle limitate risorse economiche, si trovano in difficoltà in questa transizione e possono affrontare questa sfida solo imparando a strutturarsi in rete, cooperando e condividendo informazioni e strumenti. Grazie ad una loro maggiore flessibilità e dinamicità potrebbero raggiungere grandi risultati pur rimanendo di piccole dimensioni.

Per realizzare lo scenario descritto che prevede l'aggregazione dinamica dell'offerta e della produzione è necessario raggiungere un successivo stadio di adozione delle tecnologie dell'informazione e comunicazione, descritto come ecosistema digitale.

A seguito di tale evoluzione, la messa in rete delle organizzazioni sfocia nella cooperazione dinamica degli attori sul territorio e la messa a sistema delle risorse crea una comunità che condivide business, conoscenza e infrastrutture. L'ecosistema digitale sfrutta la possibilità di interazione dinamica (cooperazione e competizione) di numerosi e differenti attori (Pmi e grandi imprese, governo e amministrazione locale, istituzioni di formazione, istruzione, innovazione e ricerca) per produrre risultati sistemici in termini di innovazione e sviluppo economico. Lo sviluppo di strumenti e tecnologie scalabili e adattativi forniscono le basi tecnologiche per l'offerta dinamica di servizi aggregati come strumento di integrazione di catene del valore, permettendo la realizzazione di nuovi modelli di business basati sull'associazione dinamica delle imprese dell'ecosistema locale.

[2] http://www.digital-ecosystems.org

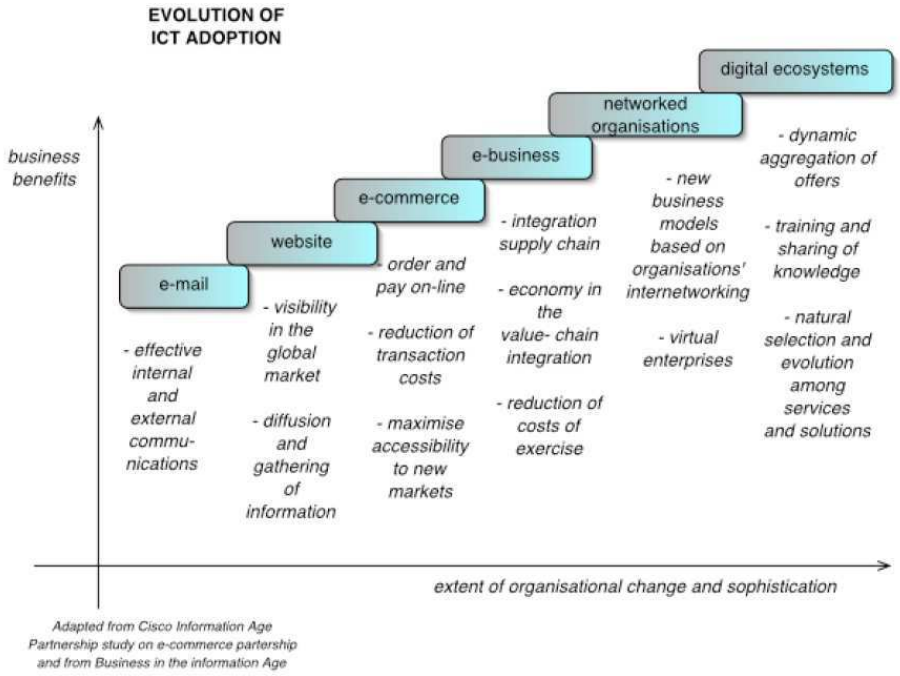


FIGURA 1. Stati evolutivi nell'adozione delle tecnologie dell'informazione e comunicazione

§

In termini generali un ecosistema digitale si può descrivere sfruttando l'analogia con un ecosistema naturale. In particolare, per *ecosistema digitale* (NACHIRA 2002; BRISCOE & DE WILDE 2005), o *digital ecosystem*, si intende un ambiente digitale *self-organising* popolato da specie digitali che possono essere:

- un componente software;
- una applicazione;
- un servizio;
- conoscenza;
- procedure e modelli di business;
- moduli di training;
- contratti, licenze, regolamenti.

In generale, può essere ogni idea o frammento di conoscenza o di opera di ingegno, considerata utile, espressa da un linguaggio (formale o naturale), inviata in rete, che può essere processata (da computer e/o da umani). Questi esseri digitali, come gli esseri viventi interagiscono fra di loro, esprimono un comportamento indipendente e si evolvono secondo leggi di selezione naturale. Le specie meno adatte si assottigliano fino a scomparire: servizi di poco interesse per il mercato vengono sempre meno utilizzati, finché non vengono più supportati e si estinguono.

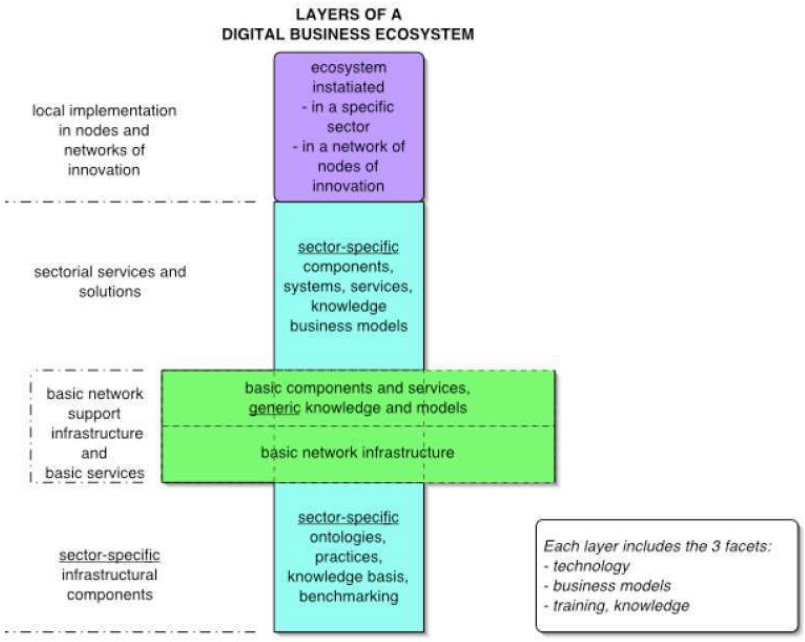


FIGURA 2. Architettura generale di un ecosistema digitale

Nuove specie (servizi digitali, componenti software, ma anche modelli di business, servizi turistici) più evolute appaiono continuamente e decretano l'obsolescenza di altre specie. Gradualmente si creano specie digitali più complesse, originate dalla composizione di specie digitali elementari (servizi, componenti). Come gli ecosistemi naturali, gli ecosistemi digitali devono avere una popolazione sufficiente: una massa critica di specie (e di elementi delle varie specie) per continuare ad esistere. Come in natura esiste una varietà di ecosistemi, dove alcune specie sono in comune, ed altre sono simili ma hanno seguito differenti linee evolutive, altre vivono in un solo ecosistema.

Gli ecosistemi digitali sono composti da tre differenti layer (figura 2), che rappresentano altrettanti livelli di specializzazione:

- ❖ Un ambiente comune di supporto e un'infrastruttura di base generica.
- ❖ Servizi, soluzioni e componenti infrastrutturali specializzati per un settore (ad esempio il turismo) che utilizzano i servizi dell'ambiente di supporto e dell'infrastruttura generica;
- ❖ Una installazione dell'ecosistema applicata ad uno specifico territorio (o in una rete di territori).

Ambiente di supporto e infrastruttura di base. L'ambiente naturale necessario alla proliferazione di specie più complesse (ad esempio un carnivoro) è formato da altre specie (ad esempio un erbivoro) che necessitano di un infrastruttura composta a sua volta da altre specie (l'erba) e necessita di elementi fondamentali (ad esempio acqua, aria). Come la distinzione fra specie di base o d'infrastruttura è arbitraria, altrettanto avviene negli ecosistemi digitali. L'ecosistema digitale si poggia sui servizi di rete pre-esistenti, utilizzando la rete fisica che permette il trasporto. Un ambiente software di supporto generico,

necessariamente distribuito e open source, rappresenta il blocco costitutivo di base dell'ecosistema digitale. L'infrastruttura di base dell'ecosistema include middleware e servizi che implementano protocolli e servizi standard che permettono la comunicazione su rete, protocolli per l'interoperabilità, per i web services. Particolare enfasi è nell'implementazione dei protocolli per il discovery/look-up/join che permettono lo spontaneo e dinamico apparire/scomparire di nuovi servizi/componenti nell'ecosistema e nei sistemi di descrizione ontologica. L'ambiente di supporto di base, oltre all'infrastruttura, offre un insieme di componenti e servizi generici di base non dipendenti dal settore d'applicazione (quali sistemi di pagamento elettronico, sistemi di certificazione, di sicurezza, di trust, ERP, CRM, e-procurement). Inoltre, non vengono trascurati gli aspetti del sapere e delle pratiche di business includendo: strumenti per la condivisione della conoscenza; metodologie di e-learning e training; metodologie, modelli e pratiche per l'integrazione di business. L'infrastruttura di base, i moduli infrastrutturali, i servizi e le metodologie di base sono implementati in una serie di nodi dell'ecosistema, distribuiti sul territorio. Nessun nodo è critico e l'architettura distribuita permette di mantenere l'operatività dell'ecosistema finché vi è un numero sufficiente di nodi attivi. I servizi dell'ambiente di supporto seguono la filosofia dell'ecosistema digitale: l'insieme dei servizi di base è soggetto ad evoluzione; vi sono varie versioni dello stesso tipo di servizio con differenti funzionalità, licenze e costi, prodotti da differenti produttori.

Servizi e soluzioni settoriali. Sull'ambiente di supporto s'innesta lo sviluppo, il riutilizzo o l'adattamento di servizi, componenti e specifiche soluzioni settoriali, relative ad uno specifico settore o catena del valore, quale il turismo. L'ambiente di supporto include:

- ❖ Generici componenti e moduli adattati al settore specifico (es. adattamento di sistemi CRM, sistemi di user profiling, di yield management);
- ❖ Componenti specifici per il settore e realizzati ad hoc (es. sistemi di prenotazione, di aggregazione dell'offerta);
- ❖ Ontologie specifiche che descrivono la semantica di dati e di servizi digitali;
- ❖ Specifici servizi, moduli di formazione, modelli di business specifici del settore;
- ❖ Basi di conoscenza di modelli di business, pratiche e soluzioni.

Implementazione in specifici territori. L'infrastruttura tecnologica, i componenti software, i servizi, sono presenti su un insieme di computer interconnessi. Una soluzione software completa è realizzata assemblando componenti sviluppati in diversi luoghi da produttori differenti. Tale soluzione può essere modificata con la sostituzione di alcuni componenti, ad esempio per renderla adatta all'ambiente locale, o in seguito all'apparire sul mercato (o meglio nella rete degli ecosistemi digitali) di componenti più vantaggiosi in termini di costo o funzionalità. Queste soluzioni facilitano il business e la cooperazione fra imprese appartenenti ad una determinata catena del valore. Queste applicazioni di e-business permettono la creazione di catene del valore distribuite, sia a livello locale, che fra nodi di innovazione distribuiti sul territorio nazionale o europeo, e permettendo l'accesso a nuovi mercati e cooperazioni. Una volta disponibili soluzioni applicative relative ai settori di interesse del territorio, queste vengono utilizzate dalle imprese locali per il loro business e per offrire servizi specifici. Le implementazioni dei servizi digitali specifici faranno quindi riferimento a offerte di servizi di un determinato territorio, o di una rete di territori federati, creando istanze locali degli ecosistemi digitali. Un elemento fondamentale per tale sviluppo e per la messa a sistema delle risorse del territorio, è la presenza di una comunità formata dai principali attori chiave sul territorio: il governo locale e la pubblica amministrazione; la comunità imprenditoriale e in particolare le Pmi; i centri di

ricerca e innovazione e le università. A questa comunità locale (o rete di comunità locali) si aggregano comunità virtuali distribuite (incluse comunità di sviluppatori open source), che svilupperanno strategie, soluzioni tecnologiche, servizi digitali, modelli di business, conoscenze.

§

I progetti di ricerca finanziati dall'Unione Europea inerenti gli ecosistemi digitali sono stati raggruppati in un cluster[3]. Tra questi, oltre a DBE, i principali progetti sono:

- e-NVISION: è un progetto STREP, iniziato nel Gennaio 2006, il cui obbiettivo principale è lo sviluppo e la validazione di una innovativa piattaforma di e-business per le PMI, la quale consente loro: di modellare e adattare nelle loro organizzazioni dei particolari scenari di business richiesti dai loro clienti e fornitori; di integrare tutti le loro applicazioni seguendo l'architettura service-oriented; di incorporare servizi legali, economici e sociali offerti da organizzazioni esterne. Tale piattaforma sarà validata nel settore dell'industria delle costruzioni, prendendo come riferimento diversi casi da quattro Paesi dell'Unione Europea.
- SEAMLESS[4]: si tratta di un progetto STREP, iniziato anch'esso nel gennaio 2006, il cui obbiettivo è quello di progettare, sviluppare e sperimentare un suite avanzata, semantic-based, di servizi ICT che permetta a delle categorie ben definite di piccole imprese di accedere allo spazio di eBusiness creato dal Single Electronic Market of the Enlarged Europe (SEEM) e consentire ad esse di stabilire delle collaborazioni con gli altri partecipanti.
- VISP[5]: iniziato nel Gennaio 2006, questo progetto STREP ha l'obbiettivo di creare una piattaforma software che consenta ad un insieme di PMI di operare come una singola entità per la produzione di soluzioni di Internet Service Provider (ISP) adattate agli specifici bisogni locali.
- LEGAL-IST[6]: si tratta di un progetto SSA il quale studia tre aspetti principali riguardo il software open-source, ovvero la legislazione sul copyright, sui brevetti e sui contratti.
- EFFORT: progetto SSA (in negoziazione) che intende studiare e promuovere il concetto di ecosistema digitale analizzando a fondo le condizioni per una miglio collaborazione all'interno dei cluster di aziende.
- PEARDROP: progetto SSA (in negoziazione) che intende studiare e promuovere il concetto di ecosistema digitale dal punto di vista dello sviluppo regionale.
- CONTRACT: le principali attività di questo progetto STREP (in negoziazione) riguardano la definizione di interazioni elettroniche business-to-business in termini di contratti, stabiliti dinamicamente e gestiti a run-time in un ambiente digitale di business, applicare delle tecniche formali di verifica a collezioni di contratti e applicare tecniche di monitoraggio all'implementazione di tali contratti al fine di costituire le basi per creare fiducia tra i partner.

[3] http://www.digital-ecosystems.org/de/refs/ref_proj.html
[4] http://www.seamless-eu.org/
[5] http://www.visp-project.com/
[6] http://www.legal-ist.org/

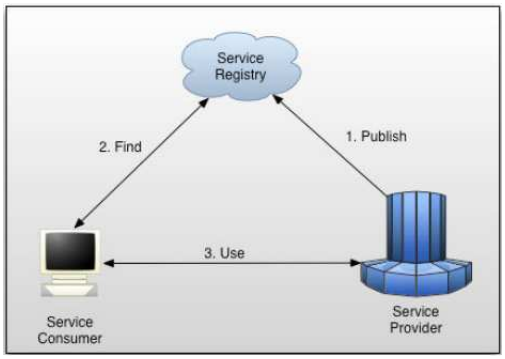


FIGURA 3. Service-Oriented Architecture (SOA)

- ONE: progetto STREP, n negoziazione, che intende sviluppare tecnologie a supporto degli ecosistemi digitali.
- OPAALS[7]: Network of Excellence con lo scopo di studiare un fondamento teoretico per la ricerca sugli ecosistemi digitali, la quale includa tre diversi domini disciplinari: le scienze sociali, dell'informazione e naturali.

3. Architettura di DBE

Lo scopo di questa sezione è quello di presentare l'architettura strutturale e funzionale di DBE. Una delle caratteristiche principali del sistema, è che esso si basa su una architettura peer-to-peer decentralizzata, la quale prevede una knowledge base distribuita sui vari nodi della rete. Inoltre, DBE segue l'approccio Service Oriented per l'invocazione dei servizi offerti nell'ecosistema.

Service Oriented Architecture (SOA). La Service Oriented Architecture è stata proposta dal W3C come l'architettura di riferimento per costruire sistemi informativi Web-based. Tale architettura, indicata con la sigla SOA, si riferisce ad una topologia dove la logica di business dell'applicazione è separata dalla logica di interazione e incapsulata in uno o più componenti software (servizi), che possono essere invocati tramite delle interfacce formali ben definite. Ogni servizio fornisce le sue funzionalità al resto del sistema attraverso la sua interfaccia descritta con un opportuno linguaggio di markup e la comunicazione tra servizi è indipendente dalla piattaforma e dal linguaggio di programmazione utilizzato per implementare il servizio. Questo permette all'architettura di avere un accoppiamento lasco, requisito essenziale per poter classificare una architettura come SOA. I servizi principali definiti da tale architettura e lo schema di interazione fra essi sono mostrati in figura 3.

Il Service Provider è responsabile di definire e implementare un servizio e di pubblicare il servizio in un opportuni Service Registry. Le informazioni che necessitano di essere memorizzate nel registro sono quelle che devono permette al Service Consumer di poter trovare il servizio di cui necessita e la sua locazione.

[7] http://www.opaals.org/

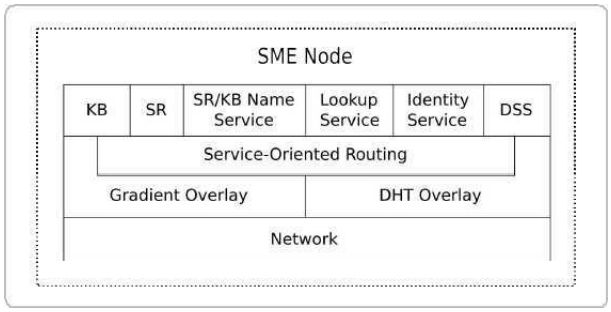


FIGURA 4. Servizi infrastrutturali di DBE costruiti sulla rete P2P

Pertanto, tale architettura prevede che il client interroghi il registro; il registro ritorna, se trovato, il servizio (o una lista di possibili servizi), ovvero l'indirizzo del service provider. A questo punto il Consumer contatta il Provider ottenendo da quest'ultimo l'interfaccia del servizio, necessaria per poter invocare correttamente le funzioni che esso offre.

Architettura P2P di DBE. L'architettura peer-to-peer di DBE è composta da due reti sovrapposte, *overlay network*, dove una è una rete strutturata mentre l'altra è non strutturata.

La overlay network non strutturata, chiamata *gradient topology*, permette la registrazione e la cancellazione di servizi così come la localizzazione delle loro descrizioni (*service manifest*) attraverso un query tool o una ricerca basata su parole chiave. La gradient topology incrementa l'efficienza della knowledge base replicando alcuni dei suoi contenuti sui nodi con prestazioni migliori, in termini di disponibilità del servizio, banda disponibile per la connessione, etc. Questo permette di migliorare la scalabilità del sistema, così come le prestazioni del processo di ricerca. Ne deriva quindi che la principale caratteristica dei questa rete è un maggior carico, in termini di numero di repliche e di query ricevute, è orientato verso quei nodi più performanti.

L'overlay network strutturata è basata sulle *Distributed Hash Table* (*DHT*) che forniscono un mapping efficiente da uno spazio di identificatori (chiavi) agli host. Ad ogni peer è attribuito un identificatore unico (una chiave) e la responsabilità per una parte dello spazio delle chiavi. Tutte le operazioni sulla DHT sono instradate da un peer all'altro, fino a raggiungere la destinazione finale, ovvero il peer responsabile per la chiave richiesta. La rete overlay DHT in DBE è utilizzata per localizzare gli host relativi alle imprese che forniscono i servizi che un client intende utilizzare, basandosi sul loro identificatore (SERVICE_ID) ottenuto dal Service Manifest tramite la gradient topology.

La figura 4 mostra la relazione fra i servizi di infrastruttura di DBE e le overlay network: la Knowledge Base, il Semantic Registry (dove vengono memorizzati i Service Manifest) e i KB/SR Name Service condividono la gradient overlay; Identity Service e Lookup Service sono costruiti sopra la rete DHT.

Tale distribuzione non è casuale. Infatti Knowledge Base e Semantic Registry hanno compiti simili, in quanto entrambi memorizzano modelli di business le descrizioni dei servizi che sono attualmente disponibili nell'ecosistema. Pertanto, ogni client può interrogare tali componenti al fine di ricercare un servizio desiderato. L'informazione ritornata non è la locazione del servizio, ma il suo identificatore univoco. In base a tale identificatore il

servizio di lookup è in grado di individuare l'esatta locazione del peer che ospita il servizio. La divisione dei servizi infrastrutturali sulle due reti overlay risulta quindi semplice da individuare.

§

L'architettura funzionale del sistema (FERRONATO & NEAGLE 2005) è suddivisa in tre principali componenti, ognuno dei quali svolge compiti diversi:

- *Service Factory Environment* (*SFE*)
- *Service Execution Environment* (*ExE*)
- *Evolution Environment* (*EvE*)

La figura 5 mostra i 3 layer sui quali i macro-componenti di DBE sono disposti. Il top layer rappresenta le reali entità della rete, le quali sono costituite da persone, imprese e da beni e servizi. Il middle layer include i due ambienti che sono visibili agli utenti, ovvero SFE e ExE. Dall'immagine si può notare che dati e messaggi sono diretti da SFE a ExE. Il livello inferiore rappresenta l'ambiente EvE il quale implementa le caratteristiche evoluzionali del progetto (*darwinismo digitale*) e che è trasparente agli occhi della comunità. I tre componenti dell'architettura sono illustrati di seguito.

Service Factory Environment (SFE). SFE è quel componente che permette alle aziende (SME o PMI) di rappresentare i propri prodotti e servizi e di renderli disponibili agli altri utenti del sistema. Il processo di creazione richiede la descrizione di tanti aspetti dell'offerta, i quali spaziano dalle business motivation dell'impresa ai parametri delle interfacce software. SFE, oltre a fornire gli strumenti per effettuare la modellazione delle offerte di una organizzazione, cerca di supportare l'utente nella creazione di tali modelli attraverso una percorso di sviluppo, composto da una sequenza di fasi ognuna delle quali coinvolge attori diversi e ha obbiettivi diversi. Tale fasi sono tre, e vanno dalla descrizione dell'azienda e delle sue attività alla generazione del codice necessario per fornire un servizio:

(1) *Business specification*: viene creata una descrizione del servizio dal punto di vista del "business", quindi con termini tipici del dominio in cui il servizio viene fornito. Tale descrizione fornisce dei modelli che corrispondono alla classe CIM dello standard MDA. La fase di business specification è a sua volta suddivisa in *business modelling* e *service modelling*: la prima vuole descrivere tutto ciò che riguarda l'organizzazione e le sue attività, mentre la seconda descrive i servizi offerti dalla SME.
(2) *Interface specification*: una volta modellate le business specification, la PMI deve essere in grado di definire le descrizioni degli e-service che implementano le funzioni di business descritte in BML. Queste specifiche consistono nel modellare l'interfaccia dei servizi, necessaria per accedere alle loro funzionalità. Ciò significa che nella descrizione vengono definiti i messaggi scambiati, l'input e output di ogni messaggio, i tipi di dato degli input e output, etc.
(3) *Coding*: Si tratta dello stadio in cui un servizio viene implementato per essere consumato dalle applicazioni. Il risultato di questo processo è un modello PSM.

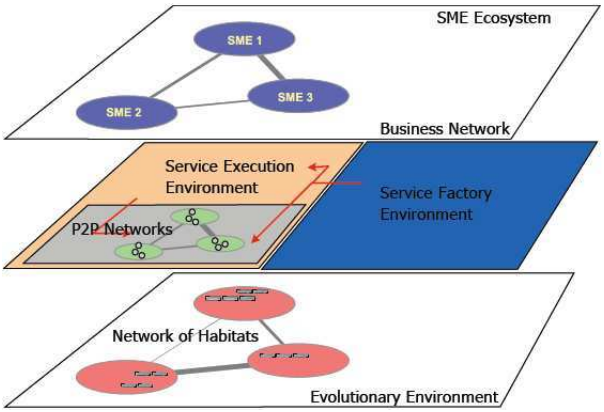


FIGURA 5. Ambienti di DBE organizzati su tre layer

Infine, tutti i modelli e le descrizioni vengono incapsulate in un *Service Manifest* (*SM*) che può essere utilizzato per pubblicare e introdurre l'implementazione all'interno dell'Execution Environment. Questo service manifest contiene una completa rappresentazione del servizio, ovvero include i modelli di business, modelli e descrizione dei servizi e dati relativi ai modelli, tutti rappresentati secondo il formato XMI.

Per supportare queste operazioni, la Service Factory fornisce diversi linguaggi: *Business Modelling Language* (*BML*) e *Semantic Service Language* (*SSL*) sono utilizzati rispettivamente nelle fasi di business e service modelling, mentre *Service Description Language* (*SDL*) è utilizzato per la definizione dell'interfaccia. Ogni documento SDL può essere generato in maniera automatica partendo dal relativo documento SSL. Allo stesso modo, a livello PSM, le implementazioni specifiche vengono create in maniera (semi) automatica partendo dai modelli CIM e PIM: alcuni linguaggi utilizzati per le implementazioni sono Java, WSDL e BPEL4WS. Ciò è reso possibile dal fatto che l'intero framework di modellazione è basato sulle specifiche di Model Driven Architecture, descritte nel capitolo precedente.

L'interazione tra tali linguaggi nel SFE è riportata in figura 6.

In tale figura si nota la presenza di un altro linguaggio, *Ontology Definition Metamodel* (*ODM*), utilizzato per definire le ontologie del sistema; tali ontologie sono utilizzate sia da linguaggi come BML, SSL e SDL per fornire semantica alle descrizione dei servizi e delle organizzazioni, sia per effettuare delle ricerche semantiche di prodotti e servizi all'interno del sistema.Tutti questi linguaggi verranno analizzati successivamente.

Service Execution Environment (ExE). Tale componente permette di negoziare, accedere ed eseguire i servizi pubblicati. L'ExE è implementato per mezzo della rete peer-to-peer descritta precedentemente. Per tale motivo, fornisce degli strumenti per visualizzare tali servizi, sia per dominio di appartenenza o in base al nome a al fornitore del servizio, per cercarli e per invocarli. Un altro importante componente è rappresentato dal *Recommender*, il cui compito è quello di recuperare i servizi che meglio corrispondono alle esigenze (funzionali e computazionali) del cliente.

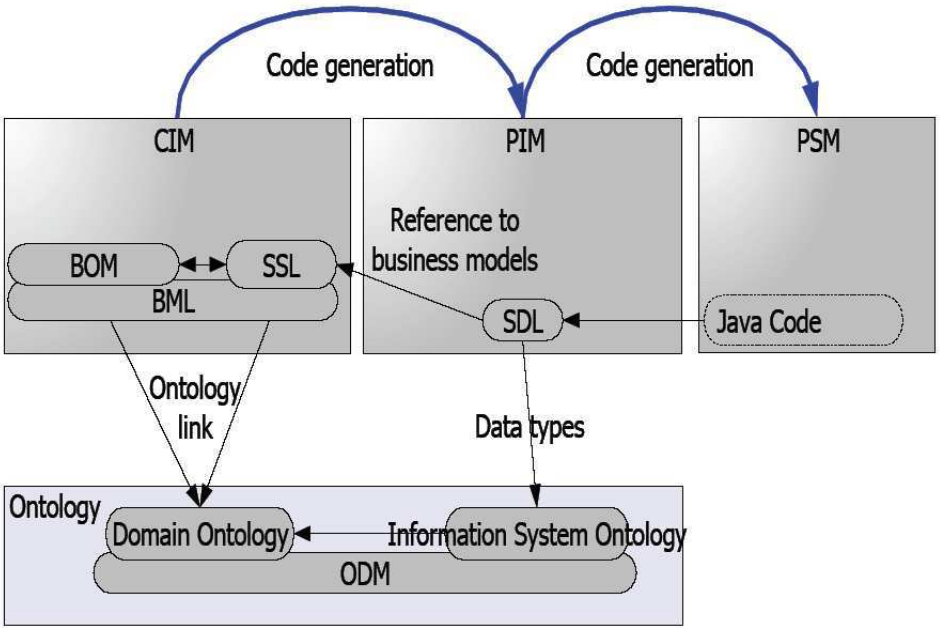


FIGURA 6. Dipendenze funzionali tra i linguaggi in SFE

Indipendentemente dall'esecuzione dei servizi, ExE fornisce dei servizi strutturale e una lista di servizi di base, come:

- Accounting
- Sistema di pagamento
- Information carrier
- Identity Management
- Recommandion
- Reputation management
- Cataloghi dei prodotti

L'Execution Environment è basato sulla tecnologia FADA[8]. La Federated Advanced Directory Architecture (FADA) è una directory di servizi distribuita, la quale appare ai client come un unico virtuale server di lookup. FADA gestisce i proxy per i servizi (Service Proxy). Un proxy è una classe Java che gestisce le comunicazioni con i servizi reali, e che può essere scaricata dai cliente a run-time. I client utilizzano i metodi pubblici sui proxy per accedere ai servizi. Questi metodi sono specificati nelle interfacce Java che i proxy implementano.

Evolutionary Environment (EvE). L'*Evolutionary Environment* (EvE) implementa la parte scientifica e filosofica degli ecosistemi digitali, nella quale i servizi nascono, si evolvono e scompaiono dall'ecosistema.

Gli obbiettivi di EvE sono:

[8]http://fada.sourceforge.net/

- ❖ per ogni utente (SME), EvE vuole creare, in modo automatico, un *service pool*, ovvero un'aggregazione di servizi significativi per quell'utente.
- ❖ utilizzare l'evoluzione dei service pool nel tempo per realizzare delle catene di servizi che si adattino alle esigenze degli utenti.

EvE è dunque quel componente che, combinato con le funzionalità dell'Execution Environment, permette di implementare un ecosistema digitale. Il modello realizzato contiene elementi teorici che provengono dalla ricerca in settori come Multi-Agent System (MAS), Distributed Evolutionary Computing (DEC) e dagli ecosistemi naturali.

Tale modello prevede che ogni PMI sia provvista di un proprio *Habitat*, ovvero la rappresentazione di una PMI e dei servizi forniti o utilizzati dall'impresa stessa. Nell'implementazione attuale, vi è una corrispondenza 1 a 1 tra nodo della rete (e quindi PMI) e habitat. Pertanto, ogni habitat definisce una *agent station*, ovvero quel componente software responsabile dell'esecuzione degli agenti mobili, cioè i servizi del sistema.

Ogni habitat contiene dunque un *agent pool*, ovvero un inseme di servizi e una *evolving population*, costituita da tutte le service chain, create attraverso algoritmi di evolutionary computing, che forniscono una soluzione alle richieste degli utenti. Ogni popolazione, ed ogni organismo (agente) che la compone, è istanziata utilizzando delle fitness function, le quali misurano il tasso di idoneità di un servizio all'interno della catena rispetto alle richieste e al profilo dell'utente.

Gli habitat dell'ecosistema sono interconnessi, come negli ecosistemi biologici, in una rete nella quale i loro servizi vengono distribuiti e dalla quale le PMI possono ricercare i servizi offerte dalle altre organizzazioni. Questa rete permette agli agenti di poter migrare tra un nodo e l'altro, di lavorare in modo autonomo, di aggregarsi ad altri agenti e di comunicare con gli altri elementi del sistema. Una architettura generale dell'evolution environment è fornita in figura 7.

4. Rappresentazione della Conoscenza in DBE

In questa sezione viene introdotto il framework di rappresentazione della knowledge base adottato nel progetto DBE (); i vari modelli e linguaggi specifici utilizzati saranno presentati nelle sezioni successive del capitolo.

Il ruolo principale della Knowledge Base di DBE è quello di mantenere la conoscenza condivisa del Digital Business Ecosystem e di consentirne lo sfruttamento da parte degli habitat che lo costituiscono attraverso vari componenti dell'architettura come il *Semantic Discovery Tool*, utilizzato per la ricerca di modelli e servizi, il *Recommender*, il *Fitness Landscape* dell'Evolution Environment, etc.

Lo sfruttamento di questa conoscenza è fondamentale per consentire l'interoperabilità semantica tra le PMI che partecipano nell'ecosistema digitale.

Il framework rispetta sia requisiti funzionali che tecnici.

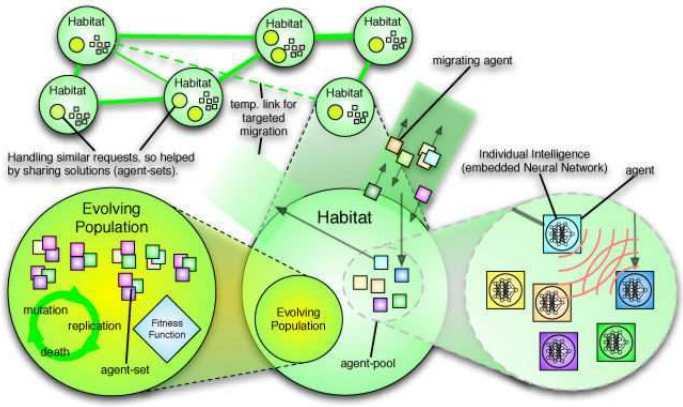


FIGURA 7. Architettura generale dell'evolution environment

Requisiti funzionali. Sono dovuti principalmente al ruolo che la knowledge base occupa nel sistema. La conoscenza di DBE la si può intendere come la "memoria" del sistema, pertanto la KB è il luogo dove ogni abitante memorizza la propria conoscenza, sia pubblica che privata. Questa conoscenza è data dalla descrizione dei modelli di business, delle loro offerte, preferenze, etc. ed è sfruttata all'interno di DBE in due modi: per consentire l'interoperabilità semantica tra diverse PMI e consentire la generazione semi-automatica di software dalle descrizione delle attività delle PMI, cioè dai comportamenti che esse intraprendono nel mondo reale.

- *Interoperabilità semantica*. Per poter attivare delle collaborazioni fra partner diversi, l'interoperabilità semantica fra i sistemi è necessaria. Ciò è possibile se gli utenti del sistema condividono un insieme comune di semantiche, ovvero una comune terminologia, per poter svolgere le loro collaborazioni. A tale scopo, si utilizzano ontologie per descrivere una concettualizzazione dei uno specifico dominio. Attraverso l'utilizzo di ontologie, l'interoperabilità è ottenuta se le varie parti utilizzano lo stesso modello di rappresentazione della conoscenza.
  Al fine di ottenere interoperabilità semantica in DBE, le Business Ontologies e le Service Ontologies sono utilizzate come il meccanismo che permette di catturare la semantica dei modelli di business e dei servizi descritti.
  Oltre a questo, è molto importante che gli utenti possano differenziare se stessi e le proprie offerte dagli altri utenti. Ciò significa che ogni PMI deve essere in grado di descrivere se stessa nel modo che preferisce, ovvero definendo e utilizzando i propri modelli, i quali possono o meno essere conformi a certe ontologie di dominio. Quindi, DBE deve fornire i meccanismi necessari per poter descrivere modelli semantici e le loro istanze.
  Come vedremo, a tale scopo, il framework di rappresentazione di DBE include diversi linguaggi per poter descrivere ontologie di dominio e modelli di business e servizi relativi alle PMI.
- *Generazione semi-automatica del software*. Uno degli obbiettivi principali di DBE è quello di supportare le PMI nel processo di creazione del software e di evoluzione dello stesso. Tali processo devono essere il più automatici possibile. Per tale motivo, la semantica dei modelli di business e dei servizi deve essere

rappresentata in modo che possa facilitare la sua traduzione in modelli di software dai quali il software è generato in un framework costituito da più layer di astrazioni (dai modelli più astratti a modelli più specifici e software-oriented).

Requisiti tecnici. Questi requisiti sono necessari per raggiungere i requisiti funzionali, ma anche per motivi di efficienza dell'intero sistema. Tra questi si riconoscono:

- *Interoperabilità sintattica*: questo requisito è fondamentale sia per ottenere l'integrazione di sistemi diversi, sia per fornire l'infrastruttura di base per ottenere l'interoperabilità semantica e la condivisione di conoscenza tra gli utenti della comunità. In DBE tutta la conoscenza condivisa è codificata in XML. Si nota comunque una importante differenza tra i classici sistemi distribuiti e DBE: nei primi, la comunicazione tra diverse parti può coinvolgere documenti XML conformi ad uno stesso schema; ciò non è vero in DBE, dove ogni PMI può definire nuovi modelli (schemi).
- *Knowledge Integrity Enforcement*: in DBE non c'è un schema predefinito (globale) che definisce il modo in cui la semantica dei business e dei loro servizi sono rappresentati. Quindi, il framework di rappresentazione di DBE fornisce un meccanismo in grado di descrivere in modo uniforme la struttura arbitraria (schemi) dell'informazione da scambiare e l'informazione stessa (semantica). Tale meccanismo non consente solo di esprimere modelli e documenti nello stesso modo, ma facilita inoltre il rispetto dell'integrità dei dati in un framework dove non c'è uno schema globale.

§

Per raggiungere i requisiti funzionali e tecnici, il framework di rappresentazione della conoscenza in DBE è stato sviluppato sulla base dell'architettura MDA-MOF; ciò significa che ogni informazione presente nella knowledge base del sistema è definita attraverso un linguaggio di modellazione basato su MOF. I linguaggi (meta-modelli), definiti all'interno del progetto, che costituiscono tale framework sono:

- *Ontology Definition Metamodel* (*ODM*), utilizzato per la modellazione delle ontologie.
- *Business Modelling Language* (*BML*), il cui compito è quello di rappresentare la semantica di una organizzazione di business.
- *Semantic Service Language* (*SSL*), utilizzato per rappresentare la semantica dei servizi offerti da una organizzazione.
- *Service Composition Metamodel* (*SCM*), necessario per rappresentare la logica di funzionamento di servizi composti.
- *Service Description Language* (*SDL*), utilizzato per modellare l'interfaccia di un servizio.

Un'anteprima del Knowledge Representation Framework è illustrato in figura 8, dove ogni linguaggio è assegnato al rispettivo livello dell'architettura MOF.

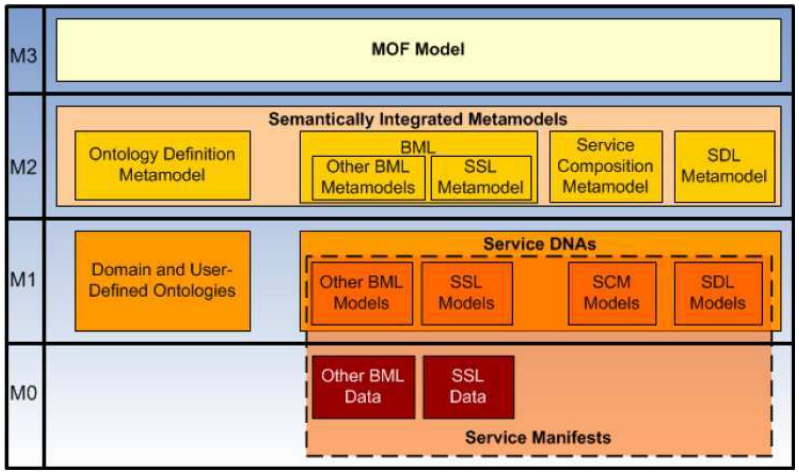


FIGURA 8. Framework di rappresentazione della conoscenza di DBE

Un ruolo chiave all'interno di tale framework è occupata da ODM, il quale permette di definire le ontologie di domino che organizzano semanticamente la conoscenza del sistema e di sfruttare tale conoscenza per la definizione dei modelli di business e per la descrizione dei servizi in modo model-indipendent.

L'insieme dei modelli per la descrizione degli aspetti di una PMI relativi ad uno specifico servizio del mondo reale, i modelli per la descrizione semantica del servizio e i modelli per la descrizione tecnica dell'interfaccia di un servizio costituiscono una entità logica e fisica detta *Service DNA*. Tale entità fornisce un modello completo per descrivere servizi con caratteristiche tecniche e semantiche simili. Infatti, il Service DNA, non descrive un servizio specifico, ma rappresenta un insieme di modelli di livello M1 che descrivono servizi di uno specifico tipo e alcune informazioni riguardo i loro fornitori.

La vera descrizione di un singolo servizio è fornita a livello più basso (M0 dell'architettura MOF), dove ogni PMI fornisce i dati specifici per il proprio servizio e riguardo se stessa. Queste informazioni aggiuntive, insieme al Service DNA, vengono raggruppate in una entità logica e fisica chiamata *Service Manifest* (*SM*). Tali entità vengono pubblicate dai service provider e cercate dai service consumer. Ogni Service Manifest (MONTANARI *et al.* 2005B) è inoltre provvisto di un identificatore univoco, SMID, e contiene le informazioni necessarie per poter individuare il proxy del relativo servizio. Altre informazioni contenute in un SM includono quelle necessarie per la tracciabilità e il versioning del servizio (numero di versione del servizio e il SMID delle versioni precedenti e successive al SM in questione, data di pubblicazione e dell'ultima modifica), l'identificatore della PMI che ha pubblicato il servizio, la disponibilità e lo stato del SM e altre ancora.

MDA e la meta-architettura definita da MOF è utilizzata in DBE come strumento base per soddisfare il requisito di interoperabilità semantica. Un'altra importante ragione che ha influenzato nella scelta di tale architettura è il suo forte orientamento alla produzione di software. MDA infatti è pensata per gestire modelli utilizzati nel processo di sviluppo di software, dove gli sviluppatori possono muoversi da modelli più astratti fino a modelli concreti. Come descritto nel capitolo precedente, le specifiche di MDA definiscono delle regole di trasformazione tra modelli diversi, cioè da modelli CIM a modelli PIM e, a loro

volta, a modelli PSM. DBE sfrutta questa funzionalità, pertanto la conoscenza di DBE può essere sfruttata per generare in modo (semi) automatico il software che implementano i servizi. Questo approccio è fondamentale in DBE per ottenere interoperabilità fra servizi implementati con linguaggi di programmazione diversi. Infatti, è necessario definire i mapping tra un modello (PSM) e il linguaggio di programmazione adottato per poter generare automaticamente il codice del servizio dal modello. Il concetto chiave in questo procedimento è che lo stesso modello può essere utilizzato per creare diverse implementazioni del servizio (una in C, una in Java, etc.).

Un approccio simile è utilizzato per accedere alla conoscenza mantenuta nella KB. Anche in questo caso i progettisti di DBE hanno implementato un meta-modello MOF-based, detto *Query Metamodel Language* (*QML*), che consente un accesso uniforme a tutte le informazioni descritte secondo i linguaggi presentati precedentemente, e in modo indipendente dalla piattaforma, cioè indipendente dal sistema di memorizzazione adottato (database relazionale piuttosto che XML, ad oggetti, etc.). Attraverso gli opportuni mapping, il codice della query nel linguaggio del data management system adottato (cioè un modello PSM) viene generato automaticamente partendo dal relativo modello QML (modello PIM).

5. Rappresentazione delle Ontologie di Dominio

Un ruolo centrale all'interno del framework di rappresentazione della conoscenza in DBE è ricoperto dalle specifiche ontologie di dominio. Tale ontologie descrivono quei concetti e relazioni fra concetti che sono condivisi da un particolare settore di business e rappresentano il punto di riferimento di ogni specifico modello di business o descrizione semantica dei servizi, consentendo la condivisione di conoscenza e l'interoperabilità semantica tra PMI diverse. Ciò è possibile in quanto DBE prevede che gli utenti (le PMI che sfruttano il sistema) modellino e descrivano i loro servizi e business riutilizzando i concetti espressi nelle ontologie, i quali sono largamente accettati dall'intera comunità.

Ontology Definition Metamodel (*ODM*) (CHRISTODOULAKS *et al.* 2005) è il linguaggio sviluppato all'interno del progetto DBE e utilizzato per la rappresentazione di ontologie di domino. Come gli altri linguaggi utilizzati in DBE, anche ODM è inserito nell'architettura MOF; in particolare tale linguaggio occupa il meta-livello M2 in quanto esso fornisce una sintassi per la descrizione di ontologie specifiche di un dominio. Pertanto, le ontologie, ovvero le istanze di ODM, sono inserite nel livello M1. Questa organizzazione è fondamentale per garantire la compatibilità di ODM con OWL: DBE permette infatti di importare delle ontologie definite con il linguaggio OWL-DL.

La necessità di avere un meta-modello per la definizione di ontologie è stata identificata anche dall'OMG stessa, la quale ha rilasciato una specifica Request For Proposal (OBJECT MANAGEMENT GROUP 2003) a questo scopo. Il team di DBE ha sviluppato la propria versione di ODM, in modo consistente alle specifiche dettate dalla RFP, presentata di seguito.

§

Per le specifiche di ODM, come anche per gli altri linguaggi di DBE, vengono utilizzate le seguenti primitive di MOF: *Class*, *Attribute*, *Association*, *DataType* e *Package*. ODM, a livello M2 dello stack MOF, definisce 6 package, presentati di seguito, ognuno dei quali

rappresenta uno specifico aspetto del meta-modello: *Core*, *DataTypes*, *Ontology*, *Classes*, *Properties* e *Individuals*.

Core Package. Tale package definisce le meta-classi astratte utilizzate come superclassi da tutte le altre meta-classi di ODM. La classe radice del package è *Element* e una sua specializzazione è *NamedElement*: ogni elemento di una ontologia deve essere un'istanza di NamedElement, cioè deve essere descritto da un nome, avere un identificatore (ID) e avere associato un NameSpace. *NameSpace* è una classe astratta che definisce un contenitore di più model element. ODM prevede che ogni elemento di una ontologia debba avere il proprio namespace; solo agli elementi istanze della classe Ontology, contenuta nell'ononimo package, è consentito non avere un namespace. Nella versione attuale del modello, solo l'ontologia può essere utilizzata come namespace per i suoi elementi.

Ontology Package. In questo package vengono fornite le classi per modellare le ontologie e le loro relazioni con altri concetti. La classe *Ontology* definisce i vari elementi che compongono un dominio di conoscenza. Una ontologia definisce un namespace per gli elementi che contiene; tali elementi sono concetti (classi), proprietà di tali concetti, relazioni fra concetti e le loro istanze. Le proprietà di una ontologia sono modellate con la classe *OntologyProperty*, la quale permette quindi di relazionare ontologie ad altre ontologie. Ogni istanza di tale classe ha associata un'ontologia come sorgente (domain) e una come destinazione (range). Esempi di OntologyProperty sono *imports*, *priorVersionOf*, *backwardCompatibleWith* e *incompatibleWith*.

Classes Package. Quello di classe il concetto fondamentale di ODM. Per classe si intende un formalismo per il raggruppamento di elementi simili all'interno di una ontologia. Ad ogni classe è associato un insieme di individui chiamato *class extension*; gli individui associati ad una classe sono detti essere istanze della classe. Ogni classe ha un significato intensionale, che è relazionato ma non uguale alla sua extension. Ciò significa che due classi possono avere lo stesso class extension ma continuare ad essere classi diverse. Va notata comunque una differenza sostanziale tra i concetti di classe in MOF e in RDFS. Una classe MOF è una specificazione delle sue istanze, ovvero, prima si definisce la classe e poi la si istanzia. Tutte le istanze di una classe hanno quindi la stessa struttura, le stesse relazioni, constraint, ... Inoltre, le istanze di una classe MOF non possono essere istanze di un'altra classe. Una classe RDFS è invece definita raggruppando gli elementi e gli individui che sono attualmente istanze della classe: in tal caso, l'istanza di una classe può essere istanza di una seconda classe definita dall'ontologia.

Analogamente a OWL, ODM utilizza sei diversi modi per definire una classe:

- ❖ Definizione di una nuova classe con un suo identificatore: è una classe che definisce tutte le sue istanze.
- ❖ Enumerazione di individui che insieme formano le istanze della classe. Questa definizione si ottiene per mezzo dell'associazione *oneOf*; in questo modo, la class extension è costituita da tutti gli elementi che sono associati alla classe tramite *oneOf*.
- ❖ Applicando restrizioni alle proprietà.
- ❖ Intersezione di due o più classi. Questa definizione è ottenuta applicando l'associazione *intersectionOf* tra le classi.
- ❖ Unione di due o più classi. In questo caso, l'associazione utilizzata è *unionOf*.
- ❖ Complemento di una classe. So ottiene utilizzando l'associazione *complementOf*.

ODM permette inoltre di definire degli assiomi per relazionare tra loro diversi concetti. Tali assiomi vengono inseriti all'interno delle descrizioni di classi. ODM fornisce tre diversi tipi di assiomi:

- *subClassOf*
- *equivalentClass*
- *disjointWith*

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

L'assioma di tipo *equivalentClass* permette di dichiarare che due classi hanno esattamente la stessa estensione. Si nota che questo non implica l'uguaglianza intensionale delle classi: due classi che hanno la stessa estensione possono dunque rappresentare concetti diversi.

Infine, l'assioma *disjointWith* viene utilizzato per indicare che l'estensione di una classe non ha membri in comune con l'estensione di un'altra classe, ovvero le estensioni delle due classi sono disgiunte.

Properties Package. Le proprietà di una classe vengono modellate attraverso i costrutti messi a disposizione da questo package. In particolare si definiscono due diversi tipi di proprietà:

- *ObjectProperty*: proprietà che collegano le istanze di una classe con istanze di una seconda classe. E' possibile che una object property sia l'inverso (inverseOf) di un'altra proprietà. Tale associazione tra proprietà è sempre simmetrica: se vale *A inverseOf B* allora vale anche *B inverseOf A*.
- *DatatypeProperty*: proprietà che collegano le istanze di una classe con dei valori specifici (literali). Tali proprietà hanno quindi uno specifico valore come oggetto (range).

ODM prevede la possibilità di relazionare fra loro le proprietà. In particolare, una proprietà può essere definita come equivalente ad un'altra proprietà attraverso l'associazione *equivalenteProperty*. Inoltre, in ODM una proprietà può essere una sotto-proprietà (*subPropertyOf*) di un'altra. Ciò consente la definizione di gerarchie di proprietà.

Esistono inoltre altre due tipi di proprietà, le *annotations* e le *ontology properties*, che verranno illustrate nei paragrafi successivi.

Sulle proprietà possono essere inoltre definite delle restrizioni, le quali vengono modellate attraverso la classe *Restriction*. Anche in questo caso si hanno due tipi di restrizioni:

- *Cardinality Constraint*: è possibile specificare la minima, la massima o la cardinalità esatta dei valori distinti che le istanze di una classe devono avere per rispettare la proprietà.
- *Value Constraint*: vincola il range della proprietà quando applicata alle istanze di una determinata classe.

Per quanto riguarda i vincoli sulla cardinalità, ODM definisce il tipo *minCardinality*, utilizzato per indicare il numero minimo che una proprietà deve avere, *maxCardinality*, il quale viene utilizzato per specificare il massimo numero di valori che una prorpietà può avere e *Cardinality* che indica il numero esatto di elementi che una classe deve avere per la proprietà.

Per effettuare restrizioni sui valori di una proprietà per una data classe ODM, similmente a OWL, mette a disposizione tre diversi costrutti: *allValuesFrom*, *someValueFrom* e *hasValue*. Il costrutto *allValuesFrom* indica che tutti i valori della proprietà specificata devono essere istanze della classe indicata come oggetto, oppure valori indicati nel range. *someValueFrom* 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. Infine, una restrizione del tipo *hasValue* viene utilizzata per indicare che la proprietà deve avere come valore l'istanza o il dato indicati.

Infine, ODM permette di definire degli assiomi sulle proprietà, i quali definiscono caratteristiche delle proprietà. Nella sua più semplice forma, tali assiomi definiscono l'esistenza di una proprietà. I costrutti supportati da ODM per costruire i property axiom sono:

- *subPropertyOf*: permette di definire una gerarchia di proprietà.
- *Domain* e *Range*: Domain indica il soggetto di una proprietà che deve essere una descrizione di classe, mentre Range specifica l'oggetto di una proprietà che può essere descrizione di classe o un valore appartenente ad un certo range.
- *equivalentProperty*: permette di esprimere l'equivalenza tra due proprietà.
- *inverseOf*: indica che una proprietà è l'inverso di un'altra.
- *global cardinality constraint*: ODM definisce tre costrutti per la dichiarazione di restrizioni di cardinalità globali:
  - *FunctionalObjectProperty*: è una object property che può avere solo un unico valore per ogni individuo.
  - *FunctionalDataTypeProperty*: Analogo al caso precedente ma per datatype property. Una functional property, sia essa object o datatype, non può avere più di un valore per lo stesso individuo.
  - *InverseFunctionalProperty*: in una proprietà dichiarata come inversamente funzionale, l'oggetto della proprietà determina in modo univoco il soggetto. Quindi, se una proprietà è inversamente funzionale, allora, 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.
- *logical property characteristic*:
  - *SymmetricProperty*: è una object property tale per cui se la coppia `(x, y)` è un'istanza della proprietà, allora anche la coppia `(y, x)` è istanza della stessa proprietà.
  - *TransitiveProperty*: è una object property tale per cui se le coppie `(x, y)` e `(y, z)` sono istanze della proprietà, allora anche la coppia `(x, z)` è una sua istanza.
  - *DeprecatedObjectProperty* e *DeprecatedDataTypeProperty*: sono proprietà non utilizzate e tipicamente rimpiazzate da altre proprietà.

Individuals Package. Gli individui sono le istanze di una classe. La classe astratta utilizzata per rappresentare tutti gli individui di una ontologia è *Thing*. Tale package definisce tre sottoclassi di Thing:

- *ClassThing*: utilizzata per rappresentare gli individui di una classe.
- *ObjectPropertyThing*: utilizzata per rappresentare le istanze di una ObjectProperty di una classe.

- *DataTypePropertyThing*: rappresenta le istanze di una DatatypeProperty di una classe.

ODM fornisce inoltre tre costruttori per dichiarare fatti relativi all'identità delle istanza:

- *sameAs*: utilizzato per definire l'uguaglianza tra due istanze.
- *differentFrom*: utilizzato per definire che due istanza sono diverse tra loro.
- *allDifferent*: indica che tutte le istanze appartenenti ad un certo insieme sono tutte diverse fra loro.

Annotazioni. ODM introduce un insieme di concetti che permettono di definire costruttori di annotazioni. E' possibile in questo linguaggio effettuare l'annotazione di classi, proprietà, individui e ontologie utilizzando il concetto *AnnotationProperty*. Le istanze di tale classe sono associate con le definizioni di classi, proprietà, individui e ontologie attraverso le associazioni *ClassAnnotation*, *PropertyAnnotation*, *ThingAnnotation* e *OntoAnnotation*. L'*AnnotationObject* di una annotazione (ovvero il suo "valore") deve essere un literale, un URI, o un individuo. In OWL esistono 5 tipi predefiniti di annotazioni (*versionInfo*, *label*, *comment*, *seeAlso*, *isDefinedBy*). ODM non pre-definisce nessuna annotazione: è comunque possibile creare le stesse annotazioni di OWL e altre ancora.

DataTypes Package. ODM permette, attraverso i *data range*, di definire un range di valori da utilizzare per esempio come oggetto per una *datatype property*. Esistono due modi per specificare un data range:

- *Datatype specification*: ODM definisce la classe *DataType*, ovvero la classe di tutti i tipi di dati definiti a livello M1. Specifica uno schema di *data typing* che è appropriato per riferirsi ai tipi di dato di XML Schema. I data type di XML Schema devono essere definiti come istanze della classe DataType do ODM. Ad esempio, il tipo String deve essere definito come istanza di "DataType" con L'URIReference impostato a "http://www.w3.org/2001/XMLSchema#string". I *data values* in ODM sono istanze della classe *Literal*.
- *Enumeration specification*: ODM fornisce la classe *Enumeration* per la definizione di un range di valori. Questi tipi di dato si aggiungono a quelli di XML Schema.

Esempio di ontologia ODM. In questa sezione viene descritto un semplice esempio di ontologia definita utilizzando il linguaggio ODM. L'ontologie, riportata in figura 9, descrive i concetti del settore trasporti, in particolare si descrivono alcuni concetti riguardanti i servizi offerti dal settore.

Ogni concetto è modellato con una classe, rappresentata da un rettangolo di color arancio. Esempi di classi sono "Transport Service", "Car Service" e "Entertainment Service". Tra le classi possono esistere diversi tipi di associazioni: la generalizzazione, indicata dalla freccia bianca, è utilizzata per esprimere le sotto-classi di un concetto. Ad esempio, "Transport Service" è una sotto-classe di "Service". Le frecce verdi indicano invece le *domain* e *range association*. Una domain association ha sempre il verso da una classe ad una object property, mentre la range association ha il verso opposto. Una *object property* indica invece un collegamento fra le istanze di due classi. Un esempio di object property è data della proprietà "hasServices" che collega, ad esempio, la classe "Limousine", la classe dominio, con la classe "Entertainment Service", che rappresenta la classe range.

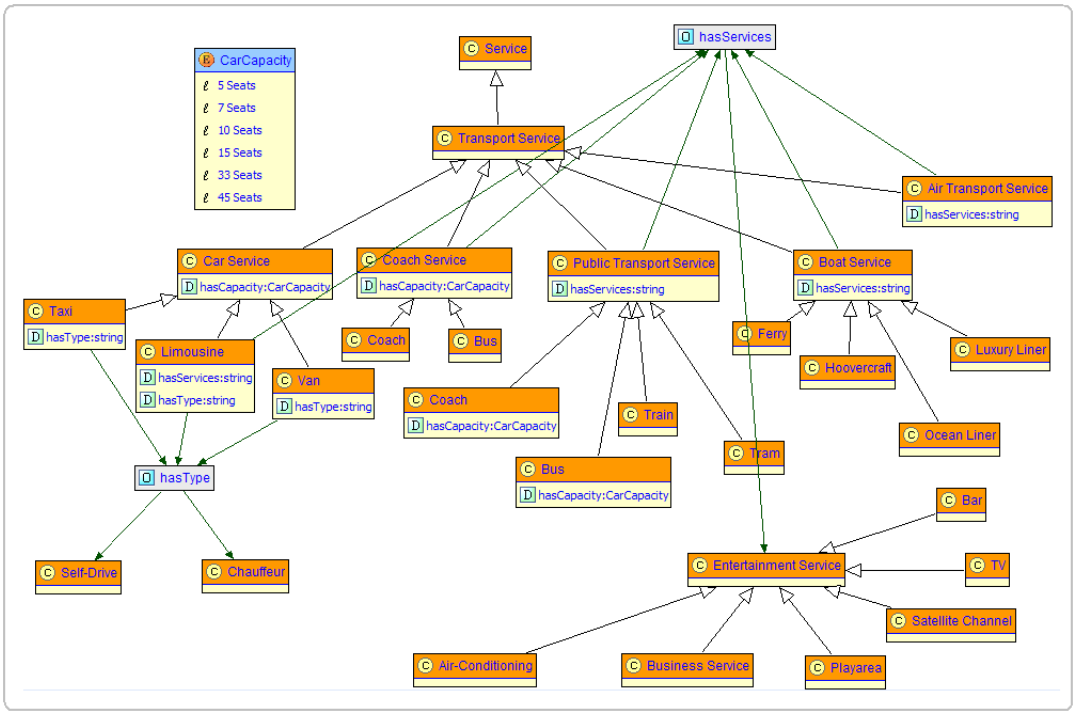


FIGURA 9. Esempio di ontologia ODM: la Transportation Ontology

In ogni classe si possono definire inoltre delle datatype property, le quali sono costituite da un nome e da un tipo. Ad esempio, nella classe “Car Service” è definita una proprietà “hasCapacity” di tipo “CarCapacity”. “CarCapacity” è un esempio di enumerazione, nella quale vengono definiti sei possibili valori (literali).

6. Rappresentazione dei Modelli di Business

Business Modelling Language (*BML*) (DE TOMASI *et al.* 2005A; DE TOMASI *et al.* 2005B) è un linguaggio realizzato all’interno del DBE Project il cui scopo è descrivere le caratteristiche delle PMI e delle attività che esse supportano; in generale è responsabile della descrizione del modello di business. Le informazioni che permette di rappresentare sono, ad esempio, i servizi offerti e richiesti da una azienda, le risorse, i prodotti, il modello di business e, più in generale, tutte quelle informazioni che descrivono la PMI e le sue attività e i suoi prodotti. Lo scopo di questo linguaggio è quello di fornire un framework semantico generale per facilitare le operazioni di business delle aziende del sistema (dalla ricerca dei servizi o prodotti alla stipulazione dei contratti).

In particolare, è costituito da più meta-modelli (livello M2 dell’architettura MOF) che insieme formano la sintassi astratta del linguaggio, necessaria per realizzare i modelli BML (livello M1).

La progettazione di tale framework è stata vincolata al raggiungimento de seguenti obbiettivi:

(1) BML deve essere in grado di rappresentare tutta la conoscenza aziendale attraverso un formalismo che sia machine readable. Inoltre tale linguaggio deve essere comprensibile anche dalla persone coinvolte in una certo business, in quanto esso è espresso in termini di business concept. Deve essere inoltre in grado di descrivere molteplici tipi di informazioni diverse, in particolare tutte le informazioni riguardanti il trading di beni e servizi e quelle necessarie per descrivere l'azienda e le sue attività. Infine, tale linguaggio deve essere indipendente dagli aspetti tecnologici.
(2) BML deve essere sufficientemente espressivo per poter descrivere la SME in qualsiasi scenario di e-Business (in termini di prodotti e servizi offerti, stipulazione dei contratti, modelli di business, organizzazione, ecc.). Il business analyst deve quindi avare a disposizione un linguaggio che gli permetta di descrivere qualsiasi aspetto della conoscenza aziendale.
(3) BML deve essere compatibile con un certo numero di standard, sia open source sia privati, in quanto dovrà essere utilizzato in ambiti molto diversi.
(4) Deve garantire un meccanismo flessibile attraverso il quale i territori e le comunità possano descrivere i loro propri linguaggi e vocabolari in base alle loro necessità. DBE deve inoltre garantire la compatibilità (coerenza) tra questi diversi modelli e vocabolari per garantire l'interazione tra ecosistemi.

Un'altra importante caratteristica di tale linguaggio, così come tutti gli altri linguaggi di DBE, è possibile di esprimere concetti facendo riferimento a quelli definite dalle ontologie di dominio. Questo è uno degli aspetti fondamentali della rappresentazione della conoscenza in DBE, in quanto permette di aggiungere semantica ai modelli utilizzando un unico dizionario condiviso e accettato da tutti i membri del sistema e permette di ottenere interoperabilità semantica tra diverse PMI.

§

Dal punto di vista architetturale, il linguaggio è stato progettato basandosi sull'approccio MOF; pertanto l'architettura è impostata su quattro livelli:

Livello M3: è costituito dal Meta Object Faciliy (MOF), sviluppato e mantenuto da OMG.
Livello M2: specifica il meta-modello BML, ovvero tutti i meta-modelli necessari alla creazione di un vocabolario di un modello di business; questo livello rappresenta quindi la sintassi astratta del linguaggio. Questi meta-modelli sono sviluppati e modificati solo dal personale del progetto.
Livello M1: contiene i modelli di un determinato dominio di business; tali modelli sono sviluppati ricorrendo alla sintassi definita al livello M2. M1 è il livello in cui un business analyst opera per descrivere il modello di business di una azienda. Oltre alla sintassi di BML, questo livello mette a disposizione un insieme di repository di conoscenza riguardo specifici domini di business, i *Business Domain Models* (BDM), per rappresentare tutta la conoscenza di una SME.
Livello M0: in esso sono presenti tutte le istanze di un modello (BML Data).

Si nota infine che BML, insieme a SSL, descritto nella prossima sezione, e a ODM, costituisce il framework *computational indipendent* (CIM) di DBE. Il linguaggio SDL permette invece di creare modelli *platform indipendent* (PIM).

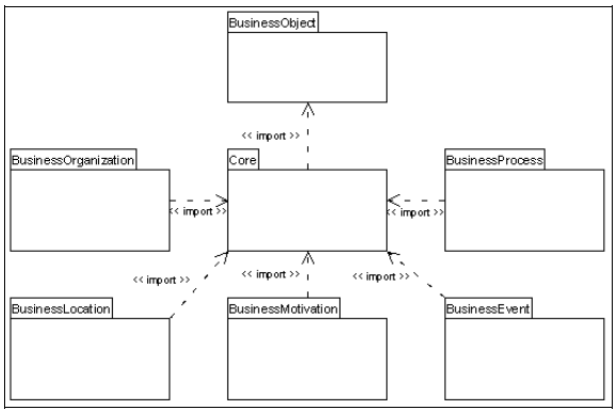


FIGURA 10. Package definiti dal meta-modello BML

§

Il meta-modello del linguaggio è definito nel livello M2 dell'architettura di riferimento: esso fornisce quindi la sintassi astratta del linguaggio ed è composto da 7 package, mostrati in figura 10: *BusinessObject*, *Core*, *BusinessOrganization*, *BusinessProcess*, *BusinessMotivation*, *BusinessEvent* e *BusinessLocation*.

Core Package. Contiene gli elementi base del linguaggio, utilizzati per descrivere i metaconcetti degli altri package. Le classi definite in questo package sono *NodeElement*, *BusinessElement*, *Attribute*, *Association*, *AssociationEnd*, *Source* e *Target*. *BusinessElement* è una classe astratta definita come una generica rappresentazione di un elemento; è costituita da un nome (un valore stringa) e può includere un attributo *ref* utilizzabile per esprimere riferimenti esterni per l'elemento da descrivere; questo meccanismo può essere utilizzato per aggiungere semantica al modello utilizzando delle ontologie esterne.

Ogni classe del meta-modello eredita da questa classe astratta, ad eccezione di Attribute, la classe che permette di modellare le caratteristiche di un business element. Ogni attributo ha un nome, un tipo e una molteplicità. il nome è una stringa, il tipo è un BusinessObject, mentre la molteplicità è un tipo strutturato composto da un limite inferiore e uno superiore (opzionale).

BusinessOrganization. Contiene le classi (meta-concetti) utilizzate per descrivere ogni aspetto dell'organizzazione. Per questo motivo si specificano le entità coinvolte, le risorse, e le loro interazioni. Tale package importa la classe BusinessElement dal Core Package mentre definisce le classi *BusinessEntity*, *Resource*, *Asset*, *BusinessItem*, *Product*, *Service*, *NetworkRole* e *Network*.

Una *BusinessEntity* rappresenta qualcosa di realmente esistente in uno specifico contesto di business; il suo ciclo di vita è indipendente dalle sue attività, funzioni o relazioni. Esempi tipici di di istanze sono "Hotel", "Costumer" e "Department". Tale classe estende la classe astratta BusinessElement; per questo motivo, le sue caratteristiche possono essere descritte da attributi.

Resource è una classe astratta utilizzata per modellare qualsiasi tipo di risorsa disponibile ad una business entity per svolgere le proprie attività. Si distinguono tra due tipi di risorse: *Asset* e *BusinessItem*. Le prime indicano qualcosa di proprio dell'organizzazione in questione, cioè utilizzate esclusivamente da essa, le seconde riguardano le risorse che l'entità usa nelle sue relazioni con il mondo esterno. Pertanto, concetti come "People" e "KnowHow" vengono modellati come istanze di Asset, mentre un esempio di BusinessItem è "Document". Due sottoclassi di BusinessItem sono *Service* e *Product* utilizzate per descrivere entità tangibili o non tangibili prodotte da processi di manufattorizzazione o di fornitura. Anche la classe Resource estende BusinessElement.

La classe *Network* rappresenta il meta-concetto utilizzato per descrivere una forma di collaborazione tra entità diverse. *NetworkRole* ha lo scopo invece di descrivere il tipo di coinvolgimento di una business entity nella rete di collaborazione. Ogni volta che si definisce una Network è necessario indicare almeno un NetworkRole. Anche in questo caso, sia Network che NetworkRole ereditano dalla classe BusinessElement. Se si considera come BusinessEntity la classe "Hotel", allora un esempio di Network è rappresentato da "HotelChain" e di NetworkRole eseguito da "Hotel" è "Member": ciò significa che "Hotel" è membro di una catena di hotel.

Infine, i vari elementi del diagramma possono essere collegati da diversi tipi di associazione, istanze della meta-classe *Association*. Le associazioni possibili sono:

- *owns subunit*: permette di indicare le sotto unità di una BusinessEntity.
- *provides*: collega le BusinessEntity alle Resource che possiede.
- *has role*: associa ad una Network un determinato ruolo.
- *perform*: permette di indicare il ruolo che la BusinessEntity possiede all'interno della Network definita.

BusinessProcess. Riguarda l'aspetto comportamentale delle SME, quindi tale package contiene tutti i meta-concetti necessari per descrivere come le organizzazioni svolgono le loro attività. Ciò significa che BusinessProcess fornisce tutte le informazioni relative all'analisi della definizione di un processo di business.

Da un punto di vista logico, il package BusinessProcess racchiude due aree principali: la *agreement area* e la *behavioural area*. La prima riguarda gli impegni reciproci presi da due PMI, la seconda invece è collegata alle attività lavorative conseguite dalle PMI per raggiungere i propri obbiettivi.

Le cassi che definisce sono *Agreement*, *Contract*, *Commitment*, *Behaviour*, *Role*, *Constraint*, *BusinessProcess*, *Task*, *Transaction*, *BusinessActivity* e *CollaborationActivity*. Inoltre importa Resource, e BusinessEntity dal package BusinessOrganization, BusinessElement da Core e Event da BusinessEvent. Tutte le classi incluse nel package sono un'estensione della classe astratta BusinessElement.

La meta-classe *Behaviour* è definita come il modo in cui una BusinessEntity lavora sotto specifiche condizioni o circostanze o in relazione ad altre BusinessElement. Essa generalizza tre sotto-classi, chiamate *BusinessProcess*, *Transaction* e *Task*. Un *BusinessProcess* descrive un insieme di azioni che portano al raggiungimento di un risultato, che può essere di una singola organizzazione o riguardante più organizzazioni. Ogni BusinessProcess può essere composto da più *Task*: un Task è una *BusinessActivity* se per svolgere il processo non è necessaria l'interazione con altre entità, mentre è una *CollaborationActivity* in caso contrario. Per ognuno di questi tipi di comportamenti, la funzione o la posizione assunta da una entità (PMI) è rappresentata tramite la classe *Role*. Ogni CollaborationActivity può

essere composta da una sequenza di più *Transaction*, ovvero uno scambio di documenti o informazioni all'interno della stessa attività di collaborazione. Ogni transazione è pertanto caratterizzata da un *sender* e da un *receiver*.

Il package permette inoltre di rappresentare leggi che limitano o restringono un Behaviour; la meta-classe utilizzata a questo scopo è *Constraint*. Vi sono diversi modi in cui un Constraint può limitare un Behaviour. Esso può rappresentare una start-condition (condizione che deve essere valida prima dell'esecuzione del comportamento), end-condition (condizione che deve essere valida dopo l'esecuzione del comportamento) o una execution-condition (condizione che deve essere valida durante l'esecuzione del comportamento).

La seconda area logica del package è incentrata sul meta-concetto di *Agreement*. Questa classe astratta modella un accordo tra due o più partner che specifica e regola le condizioni che essi devono rispettare nella collaborazione come, ad esempio, termini di pagamento, di consegna, etc. Due specializzazioni di Agreement sono Commitment e Contract. *Commintmen* è definita come una promessa tra due o più partner di iniziare una collaborazione in futuro. Ogni commitment coinvolge un partecipante, il quale ha uno specifico ruolo (Role). Un *Contract* descrive invece un'insieme di commitment che riguardano i partecipanti di uno specifico processo.

La associazioni definite dal meta-modello sono:

- *performs*: permette di indicare il Role svolto da una BusinessEntity all'interno di un BusinessProcess.
- *involves actor*: permette di indicare i vari Role che partecipano ad un Behaviour.
- *is used in*: permette di indicare, per un dato Behaviour, la Resource utilizzata.
- *produces*: collega sempre un Behaviour ad una Resource; in questo caso si indica però il risultato prodotto dal processo.
- *owns task*: è l'associazione utilizzata per indicare i Task che compongono un BusinessProcess.
- *precedes*: indica una relazione di precedenza tra Bahaviour diversi.
- *needs before*: permette di indicare, per un dato Behaviour, i Constraint che sono start-condition.
- *needs after*: permette di indicare, per un dato Behaviour, i Constraint che sono end-condition.
- *needs during*: permette di indicare, per un dato Behaviour, i Constraint che sono execution-condition.
- *fulfills*: indica i Commitment compiuti da un Behaviour.
- *establishes*: specifica, per un Contract, i Commitment stabiliti.
- *involves partecipant*: permette di indicare i Role che partecipano ad un Contract.
- *contains trasaction*: permette di specificare le Transaction che costituiscono una CollaborationActivity.
- *implies*: permette di indicare degli eventuali eventi (Event) che nascono dall'esecuzione di un certo Agreement (Commitment o Contract).
- *begins when* e *ends when*: indica eventuali Event dai quali dipende l'inizio o il termine di un Behaviour.
- *generates*: indica un eventuale Event generato dall'esecuzione di un Behaviour.

BusinessMotivation. Modella il modo in cui una azienda effettua certe scelte e definisce le sue azioni. Ciò significa che tale package definisce tutte quelle classi necessarie

alla rappresentazione di un piano di business, sottolineando ciò che l'organizzazione vuole raggiungere e ciò di cui ha bisogno per farlo. Importa le classi BusinessElement e BusinessEntity e definisce *MotivationElement*, *Means*, *End*, *Influence* e *Assesment*.

Un *MotivationElement* è la classe astratta che generalizza tutti gli altri meta-concetti del package. Può essere relazionato ad una BusinessEntity permettendo di indicare l'organizzazione che definisce un dato elemento motivazionale. Inoltre, altre associazioni possono essere definite fra elementi motivazionali diversi, dal momento che spesso si influenzano a vicenda.

La classe *End* descrive quello che una organizzazione cerca di ottenere, senza indicare il modo con cui si intende raggiungere questo obbiettivo. Esempi di istanze di tale classe sono rappresentati da "Vision", "Goal" e "Objective".

La classe *Means* rappresenta qualsiasi cosa (strumenti, metodologie, tecniche, capacità, etc.) utilizzata per raggiungere la desiderata End. Non permette tuttavia di indicare gli step necessari per raggiungere l'obbiettivo. Alcuni esempi sono "Mission", "Strategy", "Tactict", "Policy" e "Rule".

Il meta-modello permette inoltre di rappresentare processi o forze che producono effetti senza compiere sforzi tangibili. La classe utilizzata a questo fine è *Influence*. Ad esempio, si potrebbe distinguere tra influenze "External" e "Internal".

L'ultima classe definita dal meta-modello è la classe *Assessment* modella un giudizio riguardo certe Influence che incidono sulla capacità dell'organizzazione di raggiungere i propri End. In altre parole, tale classe esprime una connessione logica tra Influences e Ends e/o Means del piano di business, indicando quali Influences sono rilevanti per quali Ends e Means. Tipici esempi sono rappresentati da "Strength", "Weakness", "Opportunity" e "Threat".

Infine, le associazioni che si possono utilizzare in questo package sono:

- *has*: collega un MotivationElement alla BusinessEntity che lo definisce.
- *is related to*: permette di creare associazioni tra MotivationElement.

BusinessEvent. Fornisce delle classi per la modellazione di eventuali occorrenze in grado di influenzare il comportamento della SME. Definisce esclusivamente la classe *Event*, mentre importa le classi BusinessElement, Agreement e Behaviour: infatti, tale package è strettamente legato al package BusinessProcess.

La classe Eventi, la quale estende BusinessElement, viene definita come una occorrenza che ha degli effetti sul comportamento di una organizzazione. Ciò può avvenire in vari modi: può determinare quando un Behaviour inizia o termina. Inoltre, un Event può essere generato da un dato Behaviour. Allo stesso tempo, un Evento può implicare un accordo tra due parti; in questo caso l'evento sarà connesso ad una istanza della classe Agreement.

Per tale package viene definita la sola associazione *implies*, la quale modella una relazione di precedenza tra due eventi.

BusinessLocation. BML fornisce inoltre costrutti per la modellazione di concetti relativi alla locazione di una organizzazione. Tali costrutti sono forniti dal package BusinessLocation. In particolare, tale package contiene il meta-modello per descrivere la locazione geografica dell'azienda o di sedi collegate, aree geografiche e i loro confini e le relazioni tra esse. Le classi definite in tale package sono *Path* e *LocationType*; importa inoltre le classi BusinessElement e BusinessEntity.

La classe *LocationType* descrive la posizione di qualcosa; in particolare, permette di modellare il sito occupato da una BusinessEntity. La classe *Path* descrive il modo per raggiungere un posto. Ogni path ha una locazione come punto di partenza e un'altra locazione come punto di arrivo. Il meta-modello prevede inoltre la possibilità di aggregare vari path al fine di descrivere percorsi più complessi.

Infine, in tale package vengono definite due nuove associazioni, *is toward* e *is from*. La prima associa ad un Path i LocationType che coinvolge, mentre la seconda associa ad una LocationType i path che permettono di raggiungerla. Il package utilizza anche l'associazione *is in*, la quale permette di indicare la locazione di una BusinessEntity.

BusinessObject. I costruttori di tipi di dato sono contenuti nel package *BusinessObject*. Esso consente agli utenti di creare i tipi di dato di cui necessita per modellare un particolare dominio o business.

Tra le classi definite in tale package si riconoscono:

- *Number*: è il tipo primitivo che permette di rappresentare qualsiasi numero.
- *Text*: tipo primitivo che permette di definire stringhe.
- *Multiplicity*: è un tipo strutturato.
- *URIReference*: consente l'utilizzo di ontologie per l'aggiunta di semantica ai modelli.
- *BusinessType*: rappresenta un tipo di dato generico che un utente può istanziare per definire i suoi specifici attributi a livello M1. Esso può possedere certe Properties.
- *Property*: rappresenta un campo posseduto da un BusinessType, al fine di istanziare business object definiti dall'utente.
- *Primitive*: è definito come il building block base per esprimere lo stato di un attributo.
- *Enumeration*: rappresenta un business object i cui valori sono gli elementi di un insieme finito di literali.
- *EnumerationLiteral*: è il label utilizzato per definire un valore di una Enumeration.

Esempio di modello BML. Di seguito viene mostrato un esempio di un Business Organization di un Business Process. Il modello implementato descrive alcuni dei concetti inerenti un gruppo di ricerca universitario e le sue attività.

In figura 11 è mostrato il modello relativo alla descrizione dell'organizzazione, effettuata utilizzando i concetti del package *Organization*. Dall'immagine si può notare che sono definite tre BusinessEntity, "ResearchGroup", "Professor" e "Student", ognuna delle quali con i propri attributi. Si nota che oltre ai tipi primitivi, come *string*, *int*, *date*, *anyURI*, etc., vi sono anche altri tipi di dato, come "Professor", "Address" o "Department": questi tipo di dato sono dati dai concetti definiti da certe ontologie disponibili per il sistema. Il modello definisce inoltre servizi ("Teaching", "Research" e "Consultancy") e Asset ("Staff", "Labs" e "Competency").

Il modello che descrive i processi di business è mostrato in figura 12. Il modello descrive i ruoli e i task in cui si articola un corso universitario ("ToughtCourse") fornito dal gruppo di ricerca in questione ("ResearchGroup"). Come si nota, tale processo si sviluppa all'interno del servizio "Teaching" ed è composto da tre task: "Registration", "Lesson" e "Exam".

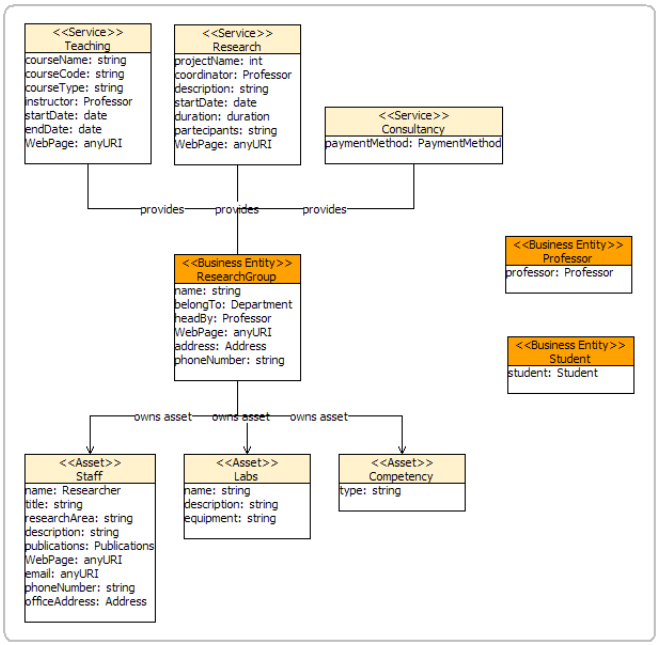


FIGURA 11. Esempio di BML: descrizione dell'organizzazione

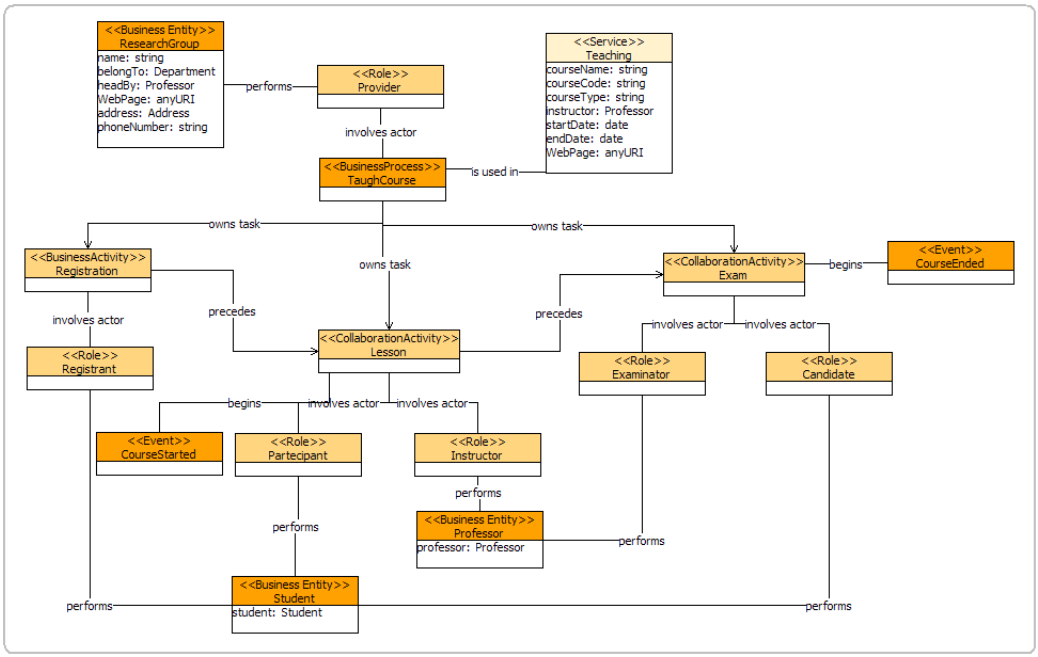


FIGURA 12. Esempio di BML: descrizione dei processi

La prima è modellata come una BusinessActivity perchè coinvolge solo una business entity ("Student"), mentre gli altri due task sono CollaborationActivity in quanto la loro esecuzione prevede l'interazione di più parti. Si nota infine che l'attività "Exam" genera la conclusione del corso.

7. Definizione di Servizi Semantici

In questa sezione viene presentato il linguaggio *Semantic Service Language* (*SSL*). Tale linguaggio (CHRISTODOULAKS *et al.* 2005; DE TOMASI *et al.* 2005B) è stato sviluppato con lo scopo di definire i modelli di descrizione semantica dei servizi che una organizzazione mette a disposizione della comunità: le istanze di questi modelli sono le *Semantic Service Description* (*SSD*), le quali vengono memorizzate nella knowledge base come parte del Service Manifest ed essere quindi pubblicate nell'ecosistema. Tali descrizioni semantiche rappresentano una delle principali informazioni utilizzate dal *Service Discovery*.

SSL è parte integrante del framework di rappresentazione della conoscenza in DBE, pertanto adotta anch'esso la stessa architettura di meta-modellazione definita da MOF.

Questo linguaggio è parte di BML ma, a differenza di quest'ultimo, il suo ruolo è quello di descrivere semanticamente un servizio, dal punto di vista più esterno possibile (quello dei clienti).

Si nota che, in DBE, la descrizione di un servizio include sia quella concettuale che quella tecnica. In particolare, per ogni servizio in DBE è necessario specificare:

- Cos'è il servizio: è una descrizione a livello di business; in tale fase è necessario fornire a quale settore il servizio appartiene, a quale area geografica si applica, etc. Questo è il compito svolto da SSL.
- Come lavora: questa descrizione riguarda i servizi composti; deve essere indicata la composizione di tali servizi composti e la logica di comunicazione, in particolare la sequenza di messaggi che devono essere scambiati. Il linguaggio che svolge tali compiti è *Business Process Execution Language* (*BPEL*).
- Come viene utilizzato: vengono descritti i dettagli tecnici del servizio come il formato dei messaggi, le operazioni disponibili, le eccezioni, ecc. Questa descrizione è effettuata utilizzando *Service Description Language* (*SDL*).

SSL permette inoltre di rappresentare diversi profili di uno stesso servizio: un profilo definisce come un servizio deve essere descritto per un scopo particolare. SSL infine supporta tre tipi di semantiche:

- Le caratteristiche speciali di un servizio: si possono riferire a caratteristiche di business quali i metodi di pagamento supportati, le caratteristiche di risorse o prodotti, così come tutte le altre informazioni che possono influenzare la decisione di un candidato cliente (ad esempio, la descrizione di informazioni metereologiche per servizi di prenotazione di residenze di villeggiatura). Nel caso in cui tali informazioni siano state descritte attraverso un modello di business, SSL offre la possibilità di importare queste informazioni nel modello SSL. Inoltre, SSL integra ODM per consentire ai modelli SSL di utilizzare i concetti tipici di un certo dominio, in questo modo aumenta anche l'interoperabilità.

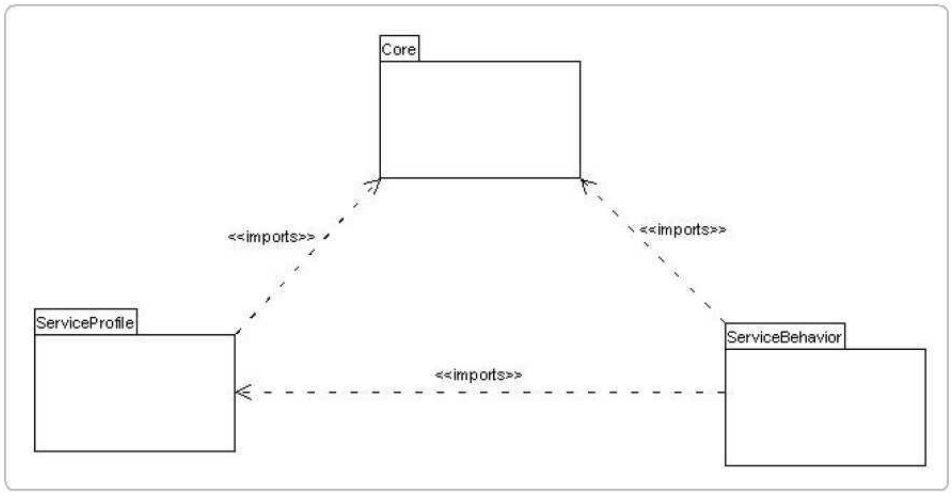


FIGURA 13. Meta-modello di SSL

- Per poter consumare un servizio, è necessaria un'interazione tra le due parti coinvolte durante la quale si scambiano risorse o informazioni. SSL permette di specificare l'interazione tra fornitore e cliente con un alto livello di astrazione. Ciò significa che le parti coinvolte non devono descrive il formato dei messaggi ma solo le informazioni o le risorse che devono essere scambiate. Per far ciò, SSL definisce la nozione di funzionalità del servizio, per poter descrivere le unità logiche di una interazione a livello di business tra fornitore e consumatore. In fine, con SSL è possibile specificare delle condizioni che devono essere rispettate per poter iniziare un'interazione. Queste informazioni saranno prese in considerazione dal compilatore SDL per la creazione automatica dell'interfaccia tecnica dei servizi.
- Permette di specificare quali sono i responsabili (individui o organizzazioni) di un servizio (contact information).

§

Come mostrato in figura 13, il meta-modello di SSL è definito da 3 package:

- *Core*
- *ServiceProfile*
- *ServiceBehaviour*

Core Package. Come per i linguaggi descritti in precedenza, tale package fornisce gli elementi base sui quali tutti gli altri elementi di SSL sono definiti. Tra questi elementi vi sono classificatori, attributi, tipi complessi e semplici, associazioni, ecc. Inoltre, tale package definisce gli elementi utilizzati per l'integrazione di elementi esterni (in particolare provenienti da BML e ODM).

La classe astratta *Element* è la radice del meta-modello: ogni concetto del modello è una specializzazione di Element, ad eccezione dei L'Iterali. In esse vengono definiti i seguenti attributi, che quindi vengono ereditati da tutte le classi che specializzano Element:

- *namespace*: ogni modello SSL può fornire un namespace per i suoi elementi. La nozione di namespace in SSL è la stessa di quella nel linguaggio XML.
- *name*: specifica il nome dell'elemento
- *uri*: è l'identificatore univoco dell'elemento; è costruito concatenando il namespace con il name.
- *documentation*: permette di fornire una descrizione testuale dell'elemento.

La classe *ExternalConceptReference* è utilizzata per l'integrazione di concetti esterni a SSL; ogni classe importata da BML o ODM deve quindi essere espressa in termini di SSL. L'attributo uriReference permette di fornire l'URI del concetto importato.

Un elemento chiave del meta-modello è rappresentato da *Classifier*, il quale rappresenta un elemento astratto utilizzato per la classificazione di altri elementi. Un classifier è composto da un insieme ordinato di proprietà, modellata dalla classe *Property*. Una proprietà rappresenta una caratteristica funzionale di un classifier. Ogni proprietà prevede la possibilità di indicare la molteplicità attraverso un limite inferiore e uno superiore. Ogni Property è una specializzazione di *TypedElement*: per tale motivo, per ogni proprietà è necessario definire il suo tipo.

SSL consente di descrivere associazioni fra classifier; questo è realizzato tramite la primitiva *Association*. In SSL tutte le associazioni definiscono delle relazioni binarie; inoltre, ogni associazione è un gruppo di due endpoint, gli *AssociationEnd*. Esistono due tipi di AssociationEnd: *SourceEnd* e *TargetEnd*. Il primo è relativo al classifier che la cui definizione include il classifier associato, mentre il secondo è relativo al classifier che costituisce una caratteristica per l'altro classifier. Un'altra caratteristica di un classifier è data dalla classe *Reference*; essa consente ad un sistema (ad esempio un query e/o un inference engine) di avere diretta conoscenza dei classifier associati ad una source.

Tale package definisce anche i tipi di dato del meta-modello. La super-classe astratta è rappresentata da *Type*. Le sue specializzazioni includono i tipi primitivi (*PrimitiveType*) come String, Integer, Float, Boolean, Double e Long, il tipo enumerazione (*Enumeration*), il tipo concetto (*Concept*), il quale può essere utilizzato per riferirsi a entità del mondo reale (ad esempio per descrivere le risorse fornite da un servizio), a informazioni relative al servizio (come i modi in cui il servizio è fornito) e a qualsialsi altri tipi di informazioni semantiche e il MultiplicityType, modellato tramite il tipo struttura del linguaggio MOF (la molteplicità è infatti costituita da due valori).

ServiceProfile Package. Questo package fornisce le primitive che permetto di modellare la struttura semantica di un servizio. Con il termine "struttura semantica" si intendono le informazioni che descrivono cosa il servizio riguarda, non come consumarlo.

La class *Service* è la primitiva utilizzata per descrivere semanticamente una classe di offerte (tipicamente servizi). E' possibile che per un determinato servizio siano forniti più modelli semantici, in quanto il servizio può assumere diverse forme, in base alla locazione geografica o al tipo di clienti. Si consideri ad esempio un proprietario di un hotel che vuole descrivere il suo servizio di prenotazione in due diversi modi: uno per clienti business (B2B) e uno per gli altri clienti (B2C). Queste due descrizioni del servizio si devono basare su modelli diversi, in quanto possono avere caratteristiche diverse.

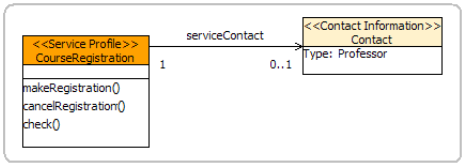


FIGURA 14. Esempio di modello SSL: il servizio “CourseRegistration”

Un Service può specificare un riferimento ad un concetto esterno attraverso l’attributo *refersTo*; un secondo attributo, *serviceKind*, permette di indicare se il servizio è del tipo business-to-business, business-to-consumer o misto. Inoltre, ogni Service deve appartenere ad almeno un *Business Domain* (ad esempio, “Finanza” o “Turismo”): in questo modo, l’utente che crea il servizio può essere assistito da ontologie specifiche per quel dominio.

I Service possono essere *Intangible* o *Tangible Service*. I primi modellano quei servizi che offrono “valore” come, ad esempio, un search engine per il web, mentre i secondi forniscono ai clienti delle risorse tangibili (ad esempio il noleggio di automobili).

SSL permette inoltre di modellare risorse di valore economico, le quali possono essere convertite in liquidi, attraverso la primitiva BusinessResource. Essa può essere utilizzata per modellare:

- ❖ Risorse tangibili come edifici, veicoli, computer, etc.
- ❖ Risorse intangibili come conoscenze, persone, brevetti, etc.
- ❖ Prodotti.

La classe *ServiceParamater* permette di modellare altre informazioni relative alla semantica del servizio che possono facilitare la visibilità del servizio. In particolare, questo tipo di informazioni non riguarda le risorse che partecipano alla fornitura del servizio, ma piuttosto modellano imposizioni dovute dal consumo del sevizio come, ad esempio, i tipi di contratto supportati, il tipo di pagamenti accettati, etc.

Infine, tale package definisce la classe *ContactInformation*, la quale modella un contatto per il servizio, spesso riferente alla persona o all’organizzazione responsabile del servizio stesso.

ServiceBehaviour Package. *ServiceBehaviour* definisce le interazioni a livello di business che devono essere soddisfatte per consumare un servizio. In particolare, questo package permette una descrizione di alto livello delle interazioni e dei messaggi che devono essere scambiati in un processo di business.

In particolare, la classe *BusinessFunction* modella le funzioni, eseguite dal provider del servizio, che il cliente può invocare per fruire del servizio; ad esempio, per un servizio di prenotazione di una stanza, una funzione potrebbe essere “Make Reservation” o “Cancel Reservation”.

La classe *BusinessMessage* descrive invece un messaggio che può essere scambiato tra le parti, durante la fornitura di un servizio; ad esempio, un ipotetico cliente di un ristorante potrebbe chiedere il menù del giorno. Ogni BusinessMessage è un TypedElement, che significa che ogni messaggio deve essere provvisto di un opportuno tipo. Si nota che

Esempio di un modello SSL. In figura 14 è riportato un semplice modello SSL il quale descrive il servizio di registrazione ai corsi universitari offerti dal gruppo di ricerca definito nella sezione precedente. In questo caso, il servizio, il cui nome è "CourseRegistration", definisce due BusinessFunction, "makeRegistration", la quale permette di effettuare la registrazione al corso, e "cancelRegistration" che permette di rimuoverla. Per ogni funzione sono definiti dei BusinessMessage, non visibili dalla figura, sia di input che di output. Ad esempio, la funzione "makeRegistration" prevede come parametri di ingresso il codice del corso e dello studente, mentre come uscita prevede il codice di registrazione. Per il servizio viene specificata ContactInformation di tipi "Professor": tale concetto è preso da una opportuna ontologia di dominio definita all'interno del sistema.

8. Descrizione dei Servizi

Service Description Language (*SDL*) (MONTANARI *et al.* 2005A) è il linguaggio, anch'esso sviluppato nell'ambito del progetto europeo Digital Business Ecosystem, utilizzato per descrivere i servizi in modo indipendente dalla piattaforma. A differenza di BML e SSL, SDL fornisce una descrizione tecnica dei servizi, ovvero l'interfaccia (di basso livello) che il servizio offre a tutti i client: in particolare, permette di definire il formato dei messaggi, le operazioni disponibili, le eccezioni, etc.

Le sue caratteristiche sono:

- ❖ Indipendente dal middleware.
- ❖ Permette la composizione automatica dei servizi.
- ❖ Utilizza ontologie di supporto alla definizione della semantica dei servizi e dei dati.
- ❖ Utilizza il meta-linguaggio MOF per la definizione del meta-modello ed è rappresentato con un UML Class Diagram.

Tra le sue principali funzionalità si riconoscono:

- ❖ Definisce l'interfaccia del servizio, ovvero la sequenza di messaggi che devono essere inviati e ricevuti. I messaggi possono essere annotati rispetto ad una ontologia per poter definire il loro significato.
- ❖ Definisce la semantica del servizio e di tutti i messaggi scambiati.
- ❖ Deve fornire un supporto alla composizione dei servizi.
- ❖ Consente il mapping tra messaggi che non presentano lo stesso formato ma contengono le stesse informazioni; il mapping deve essere "guidato" da una ontologia.
- ❖ Non fornisce meccanismi di ereditarietà: un servizio può estenderne un altro solo attraverso un processo di copia del servizio da estendere e modifica della copia.

SDL è inoltre un documento XMI basato sul meta-meta-modello MOF; inoltre deve essere indipendente dalla tecnologia e dalla piattaforma, ma, attraverso opportuni tool, deve consentire la generazione automatica di interfacce per una specifica tecnologia o protocollo (ad esempio SOAP, WSDL, Java, ecc.).

Tale linguaggio è molto simile, da un punto di vista logico, a WSDL ma presenta qualche caratteristica aggiuntiva: è indipendente dal middleware ed è semanticamente arricchito. Infatti, seppure WSDL possa essere considerato indipendente dalla piattaforma, in quanto non dipende dall'implementazione dei Web Services, nell'ambiente di DBE esso non è indipendente dal middleware perchè è pensato apposta per descrivere Web Service.

§

Di seguito vengono descritti gli elementi che costituiscono il linguaggio.

Element. E' la classe radice del meta-modello. Essa definisce un solo attributo, *ElName*, il quale permette di specificare il nome dell'elemento.

SemanticElement. Tutti gli elementi di SDL derivano da tale classe. Ciò significa che ogni elemento di un modello SDL può contenere un riferimento ad una ontologia, in quanto questo è considerato il metodo più semplice per introdurre semantica. L'attributo utilizzato a tale scopo è chiamato *ontologyReference*. SemanticElement specializza la classe Element, pertanto eredita da essa l'attributo ElName.

Definitions. E' la radice di una Service Description in quanto contiene le definizioni di *Type*, *Message* e *Interface*. In particolare, una definizione di un servizio è una composizione di tipi, messaggi e interfacce; un oggetto può appartenere ad una sola definizione.

Interface. Interface rappresenta un tipo astratto di Service; essa è un raggruppamento logico di operazioni e può essere annotata con una ontologia. Le operazioni elementari che una interfaccia supporta sono definite dalla classe *Operation*. Anche in questo caso, ogni singola operazione può essere parte di solo una Interface. DBE prevede che una Interface venga generata automaticamente da una istanza della classe Service del meta-modello SSL. Ad esempio, una Interface "CourseRegistration" (figura 14) sarà prodotta sulla base del ServiceProfile "CourseRegistration" mostrato in figura 15. Tale interfaccia è definita come una composizione delle tre operazioni "cancelRegistration", "makeRegistration" e "check".

Interface Operation. *Operation* è una sequenza di messaggi relativi ad una singola azione di un Service; in particolare ogni operazione definisce i messaggi di input, output e di eccezioni. Per quanto riguarda le eccezioni, ne esistono di due tipi: Technical e Functional. Le prime riguardano guasti tecnici (il database è down), mentre le seconde occorrono al verificarsi di guasti per "ragioni di business" (ad esempio, per il servizio CourseRegistration, non è più possibile registrarsi al corso).

Un esempio è costituito dall'operazione "makeRegistration" dell'esempio riportato in figura 15a. Per tale operazione è possibile definire il riferimento alla lista dei messaggi di input, chiamata "makeRegistrationRequest", la lista dei messaggi di output, detta "makeRegistrationResponse" e le eccezioni tecniche e funzionali.

Si nota che l'esempio di figura 15 è estremamente semplice. Infatti in SDL la granularità di Operation è più particolareggiata che in SSL. Infatti, mentre in SSL sono espresse solo le operazioni semanticamente significative per descrivere il servizio, SDL permette di esprimere anche tutte le operazioni "tecniche" non utili alla caratterizzazione semantica del servizio ma, piuttosto, alla sua esecuzione. Ad esempio, per un servizio di prenotazione di una stanza di hotel, "RoomReservation", SSL definisce solo l'operazione "makeReservation" e la lista dei messaggi scambiati, ad esempio, come input, "RoomType" e "date". Con SDL è possibile inoltre definire il metodo "getRoomTypeList" che ritorna la lista di tutti i tipi di stanze disponibili in quell'hotel.

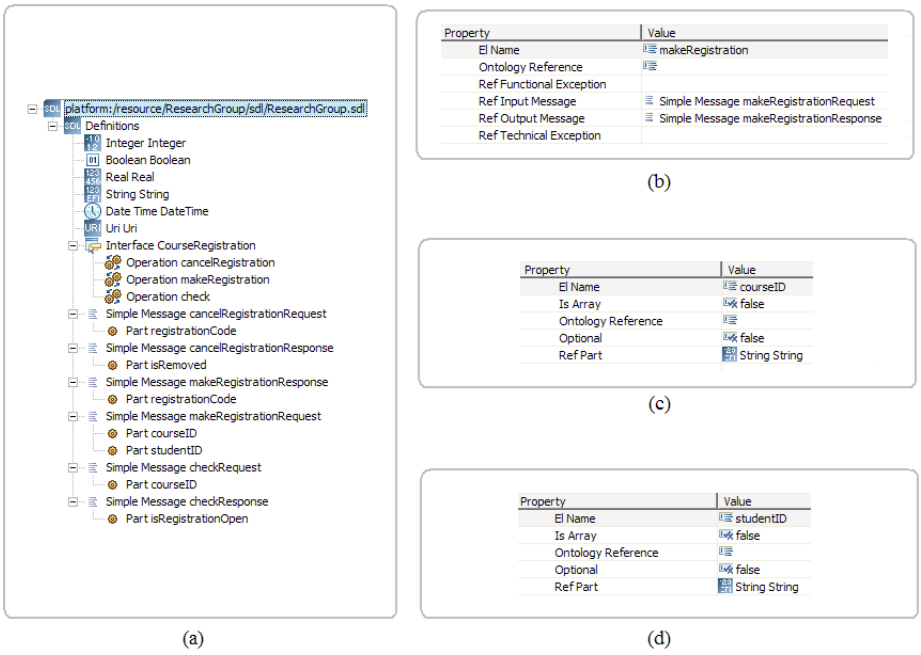


FIGURA 15. Esempio SDL: CourseRegistration

SimpleMessage. Un messaggio è l'unità base di comunicazione tra un Service e un Client e viceversa. Un *SimpleMessage* è annotato con una ontologia ed è composto da componenti *Part*, i quali rappresentano parti elementari (opzionali) di un messaggio. Ogni Part specificato deve essere provvisto di un tipo di dato.

Un esempio di SimpleMessage è rappresentato da "makeRegistrationRequest" (figura 15b), il quale rappresenta il messaggio di input per l'operazione "makeRegistration". Tale messaggio prevede due oggetti Part, "courseID" rappresentante il codice del corso e "studentID", l'identificativo dello studente che intende registrarsi. Per entrambi gli oggetti, il tipo di dato definito è String (figure 15c-d).

MessageList. Tale classe, la quale è una aggregazione di SimpleMessage, è stata introdotta per realizzare i concetti di eccezioni Technical e Functional. Quando una eccezione è generata, tale lista è utilizzata per decidere come procedere nell'esecuzione del servizio. Ad esempio, se un proxy non è disponibile, potrebbe essere utile che L'Eccezione tecnica contenesse il ProxyID, mentre se è la rete ad essere down sarebbe utile conoscere il numero delle connessioni tentate e l'URL dell'entry node. In questo modo è possibile capire il tipo di errore verificatosi e prendere i giusti provvedimenti.

Type. Esistono due tipi di *Type*: *ComplexType* e *SimpleType*. I primi sono composti da una lista ordinata di parti, dove ogni parte può essere un elemento di un particolare tipo (semplice o complesso), mentre i secondi sono concetti astratti (SDLBoolean, SDLInteger, SDLReal, SDLDateTime, SDLUri e SDLString).

Esempio di documento SDL. L'esempio descritto durante la presentazione del linguaggio è riportato in figura 15. In figura 15a sono riportati i vari elementi che costituiscono il documento SDL; la figura 15b riporta le proprietà dell'operazione "makeRegistration", la 15c quelle della parte "courseID", mentre la 15d raffigura le proprietà di "studentID".

9. Composizione di Servizi

Oltre alla definizione delle interfacce, la seconda descrizione tecnica dei servizi riguarda la logica interna di esecuzione e comunicazione dei servizi composti, effettuata tramite il *Service Composition Meta-model* (*SCM*) (MCKITTERICK *et al.* 2005). L'obbiettivo è quello di sviluppare un linguaggio che sia compatibile con *Business Process Execution Language For Web Services* (*BPEL4WS*) (MICROSOFT *et al.* 2003), il quale è il linguaggio maggiormente accettato per i web services. Allo stato attuale però le differenze tra i due linguaggi sono talmente che minime che l'editor e l'engine sviluppati per la definizione di modelli SCM e l'esecuzione dei servizi composti sono editor e engine BPEL.

BPEL è un linguaggio che definisce un modello, incentrato sui processi, per specificare il comportamento dei processi di business basati sull'interazione dei processi con i partner.

BPEL è basato su precedenti linguaggi come WSFL, progettato da IBM, e XLANG, definito da Microsoft. L'ultima versione, BPEL4WS 1.1, è modellato su alcune tecnologie del Web, come XML Schema 1.0, XPath 1.0, WSDL 1.1.

BPEL fornisce sia costrutti presi dai linguaggi di programmazione (come switch, while, ecc.) sia graph-based. La notazione di BPEL include controllo di flusso, variabili, esecuzione concorrente, input e output, gestione degli errori.

Di seguito viene data una breve descrizione del linguaggio.

§

Descrizione di BPEL. La parte principale di un modello BPEL è l'elemento *Process*. Ogni processo specifica l'ordine con cui i servizi sono invocati. Tutti i modelli BPEL devono definire almeno un Process, il quale può consistere dei seguenti elementi:

- *Partner links*: descrivono le relazioni tra due servizi a livello di interfaccia. Tali link forniscono un collegamento ai servizi da invocare e ai client che hanno invocato il processo.
- *Partners*: descrive ogni parte che partecipa al processo, inclusi i servizi invocati e e i client del processo.
- *Variables*: forniscono un mezzo per contenere i messaggi che costituiscono lo stato del processo.
- *Correlation Sets*: rappresentano un insieme di proprietà che identificano univocamente il processo di business.
- *Fault handlers*: permettono di definire un insieme di attività per la gestione delle eccezioni e di eventuali errori.
- *Compensation handlers*: forniscono un wrapper per una attività di compensazione la quale può descrivere come compensare il processo di business.

- *Event handlers*: permettono di definire un insieme di attività per la gestione degli eventi.
- *Parent Activity*: rappresenta una singola attività BPEL, tipicamente un contenitore per altre attività (ad esempio una sequenza). Questa attività iniziale può contenere un insieme estensibile di attività le quali forniscono la struttura e la logica del processo di business.

Le *Activities* sono i building block di un processo BPEL. Le *Basic Activities* rappresentano i costrutti base utilizzati per definire semplici comportamenti quali ricevere un messaggio (*Receive*), invocare un servizio (*Invoke*), generare una risposta ad una operazione (*Reply*), assegnare un valore alle variabili (*Assign*) e altre ancora. Le *Structured Activities* sono simili ai costrutti condizionali e di loop dei linguaggi di programmazione e sono utilizzati per combinare le attività base di un processo. Esempi di tali attività sono *Sequence*, la quale contiene una o più attività da eseguire sequenzialmente secondo l'ordine in cui sono inserite in lista e *Switch*, la quale supporta comportamenti come "switch" e "case" dei linguaggi di programmazione. Le *Additional Activities* permettono di definire la portata delle variabili (*Scope*) e di gestire attività anomale. Le attività sono collegate tramite link, che possono essere espliciti o impliciti.

Infine, BPEL distingue tra processi sincroni e asincroni. Un processo è sincrono quando blocca un client fino a quando il processo non termina, asincrono in caso contrario.

10. Il Linguaggio di Interrogazione

In questa sezione viene presentato il *Query Meta-model Language* (*QML*) (KAZASIS *et al.* 2005; KOTOPOULOS *et al.* 2006), ovvero il linguaggio utilizzato in DBE per l'accesso ai dati. Dal momento che lo scopo di QML è quello di consentire la formulazione di query e di vincoli sui livelli M3, M2 e M1 dello stack di MOF, le sue metaclassi sono state implementate come specializzazione degli elementi del meta-modello MOF; ciò significa che QML è un'istanza di MOF e quindi si trova a livello M2 nella sua architettura.

QML è basato sul *Object Constraint Language 2.0* (BOLDSOFT *et al.* 2003), il quale è utilizzato come base formale del suo meta-modello. Come Object Constraint Language (OCL), i package principali definiti nel meta-modello sono *Expressions Package* e *Types Package*, come mostrato in figura X, i quali definiscono le espressioni e i tipi di QML. Questi due package formano il *Core Package* di QML, il quale utilizza il meta-modello MOF. Si nota che OCL è basato sul package core di UML. Per poter quindi interrogare e applicare vincoli sui modelli MOF, il meta-modello QML ridefinisce OCL allineando le meta-classi di UML a quelle di MOF. Inoltre, dal momento che lo standard OCL non fornisce una sintassi, astratta o concreta, per esprimere delle query, il suo meta-modello è stato arricchito con un'appropriata espressione e con una meta-classe helper.

In particolare, una query viene trattata come una espressione OCL (*constraint*) su un oggetto definito in un meta-modello (*context*); gli oggetti ritornati sono del tipo del contesto e qualsiasi altro oggetto del meta-modello che può essere direttamente referenziato dal contesto (*result type*).

Altre due estensioni sono state effettuate al linguaggio OCL:

- il package Context Declarations
- operatori booleani fuzzy

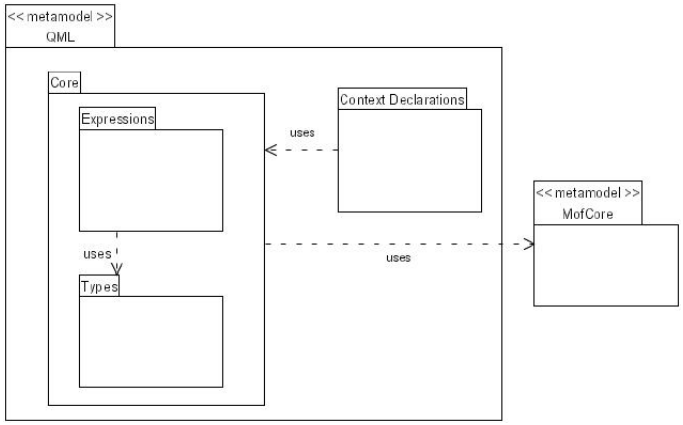


FIGURA 16. Package del linguaggio QML

Mentre le *Context Declaration* non sono utilizzate in OCL in quanto i vincoli sono attaccati direttamente agli elementi a cui si riferiscono, in QML è necessario rappresentare l'informazione di dove e come una *OclExpression* è applicata. In particolare, le informazioni espresse riguardano il contesto dell'espressione, il suo tipo (*invariant constraint*, *operation*, *attribute definition*) e ogni altra informazione necessaria per ogni tipo. Tale package definisce un insieme di meta-classi helper, di cui la principale è la *QueryContextDecl*, pertanto esso non appartiene al package Core del meta-modello QML.

La seconda estensione è stata necessaria in quanto QML deve supportare matching approssimato e deve consentire l'ordinamento dei risultati in base ad un valore di rilevanza.

§

Il meta-modello SSL, come mostrato in figura 16, è composto dai package *Expressions*, *Types* e *Context Declarations*, i quali vengono descritti di seguito.

Package Expressions. La struttura base di tale package è costituita da tre classi: *OclExpression*, *PropertyCallExp* e *VariableExp*.

OclExpression. Una *OclExpression* è un'espressione che può essere valutata in un dato ambiente. Essa è una super-classe astratta ed è la classe top-level del package. Ogni OclExpression ha un tipo che può essere determinato staticamente analizzando l'espressione e il suo contesto. La valutazione di un'espressione ritorna un *valore*; se il valore è booleano, l'espressione può essere utilizzata per come vincolo e query. L'ambiente di una OclExpression definisce che elementi del modello sono visibili e possono essere utilizzati in una espressione. La classe utilizzata per definire l'ambiente è *ModelElement*.

PropertyCallExp. Espressione che si riferisce ad una proprietà (operazione, attributo, associazione e iteratori predefiniti per collezioni) ed è riferita da una source expression. Il valore del suo risultato è la valutazione della corrispondente proprietà. PropertyCallExp

è una meta-classe astratta e la sua source expression è l'istanza che performa la property call.

ModelPropertyCallExp. Generalizza tutte le property call che si riferiscono a *Feature* o *AssociationEnd* nel meta-modello MOF. Tale meta-classe è specializzata in tre sotto-classi. AssociationEndCallExp è un riferimento ad un AssociationEnd definito in un modello MOF. E' utilizzato per determinare gli oggetti collegati ad un oggetto target da una associazione. AttributeCallExp è un riferimento ad un attribuito di un Classifier definito in un modello MOF; tale espressione ha il compito di valutare il valore dell'attributo. Infine, OperationCallExp si riferisce ad una Operation definita in un Classifier. L'espressione può contenere un lista di argomenti se l'operazione ha dei parametri; in questo caso, il numero e il tipo degli argomenti deve corrispondere ai parametri.

LoopExp. E' una espressione che rappresenta un loop costruito su una collezione. Ha una variabile che rappresenta l'elemento della collezione durante l'iterazione. L'espressione *body* viene valutata per ogni elemento nella collezione. Ha due sottoclassi, IterateExp e IteratorExp. Una *IterateExp* è una espressione che valuta la sua espressione *body* per ogni elemento di una collezione. In particolare, valuta la sua body expression per ogni elemento della sua collezione *source* . Il valore valutato della body expression in ogni step dell'iterazione diventa il nuovo valore della variabile *result* per il prossimo step. *IteratorExp* rappresenta tutte le operazioni predefinite che si possono eseguire su una collezione, come select, collect, reject, forAll, exists, etc.

IfExp. Il risultato di una espressione IfExp è una delle due alternative; tale valore dipende dalla valutazione di una *condition*.

LetExp. E' un particolare tipo di espressione che definisce una nuova variabile con un valore iniziale. Tale variabile non può cambiare il suo valore; il valore è sempre dato dalla valutazione dell'espressione iniziale. La variabile è visibile nell'espressione *in*.

LiteralExp. E' una espressione la quale non ha argomenti che producano un valore.In generale, il valore *result* è identico al simbolo dell'espressione. Tale classe include elementi come l'intero "1" o una qualsiasi stringa.

VariableExp. Espressione che si riferisce ad una variabile. Esempi di tali espressioni sono riferimenti alle variabili *self* e *result* o alle variabili definite dall'espressione Let. Le variabili vengono dichiarate attraverso la classe VariableDeclaration, la quale associa ad esse un tipo.

Package Types. QML è un linguaggio tipato, pertanto ogni espressione ha un tipo il quale può essere esplicitamente dichiarato o derivato staticamente. Tale package è analogo al corrispondente nel meta-modello OCL con la differenza che gli elementi del modello UML sono tradotti nei corrispondenti elementi di MOF. La classi principali definite sono *Classifier*, la classe astratta che rappresenta l'elemento radice del package, e *DataType*, una specializzazione di Classifier. I tipi definiti dal modello sono:

- Strutture (classe *TupleType*)
- Tipi primitivi (classe *PrimitiveType*)
- Collezioni (classe *CollectionType*)

TupleType viene utilizzata per rappresentare strutture di tipi diversi, senza limitazioni sul tipo di dati che può essere incluso in essa.

I tipi collezione (o CollectionType) vengono utilizzati per rappresentare collezioni di un particolare tipo di dato. Le collezioni possono essere di quattro tipi differenti:

- *bag*
- *set*
- *ordered set*
- *sequence*

Il tipo *bag* (*BagType*) permette di creare un insieme non ordinato di elementi, dove ogni elemento può comparire diverse volte. Il tipo set (SetType) rappresenta un insieme non ordinato di elementi, dove ogni elemento può comparire solo una volta. Se si desidera una collezione ordinata di elementi dove elementi distinti compaiono una sola volta, allora il tipo da utilizzare è *ordered set* (*OrderedSetType*). Il tipo *sequence* (*SequenceType*) infine è un tipo collezione il quale descrive una lista di elementi ordinati rispetto la loro posizione nella lista, dove ogni elemento distinto può comparire più volte.

I *tipi primitivi* (*PrimitiveType*) principali definiti nel package sono: *Integer*, *Real*, *String* e *Boolean*.

QML prevede inoltre un *VoidType* il quale specifica un tipo che è conforme a tutti gli altri tipi.

Package Context Declarations. La sintassi concreta delle *Context Declaration* è spiegata nelle specifiche di OCL2.0 (Sezione 12.13). Per esprimere l'idea di query come un vincolo (constraint) su un elemento è stata aggiunta la meta-classe *QueryContextDecl*.

Tale classe rappresenta una query identificata da un nome *simpleName*. La query ritorna un insieme di oggetti del tipo *result* (i quali possono essere un qualsiasi *Classifier*) per i quali i vincoli espressi nella *bodyExpression* sono rispettati. QueryContextDecl ha inoltre un *pathName*, ovvero una sequenza di stringhe che rappresenta il contesto della query, ovvero il tipo dell'oggetto al quale i vincoli della bodyExpression sono riferiti. QML prevede infine la possibilità di associare più contesti ad una QueryContextDecl: in questo modo è possibile definire query che combinano informazioni semantiche da più meta-modelli diversi.

Operatori Booleani Fuzzy. Uno degli obbiettivi di QML è quello di fornire un framework per effettuare ricerche approssimate. Per questo motivo il meta-modello di QML è stato esteso in modo da supportare espressioni con operatori booleani fuzzy. In particolare, il package OCLBoolean di OCL Basis Library è stato esteso con tre nuovi operatori:

- *fAND*
- *fOR*
- *fNOT*

Tali operatori sono utilizzati per la operazioni fuzzy di congiunzione, disgiunzione e negazione rispettivamente. A tali operatori corrispondono delle funzioni fuzzy, rispettivamente f_{AND}, f_{OR} e f_{NOT}, realizzate all'interno del Code Generator Module come dichiarazioni XQuery.

§

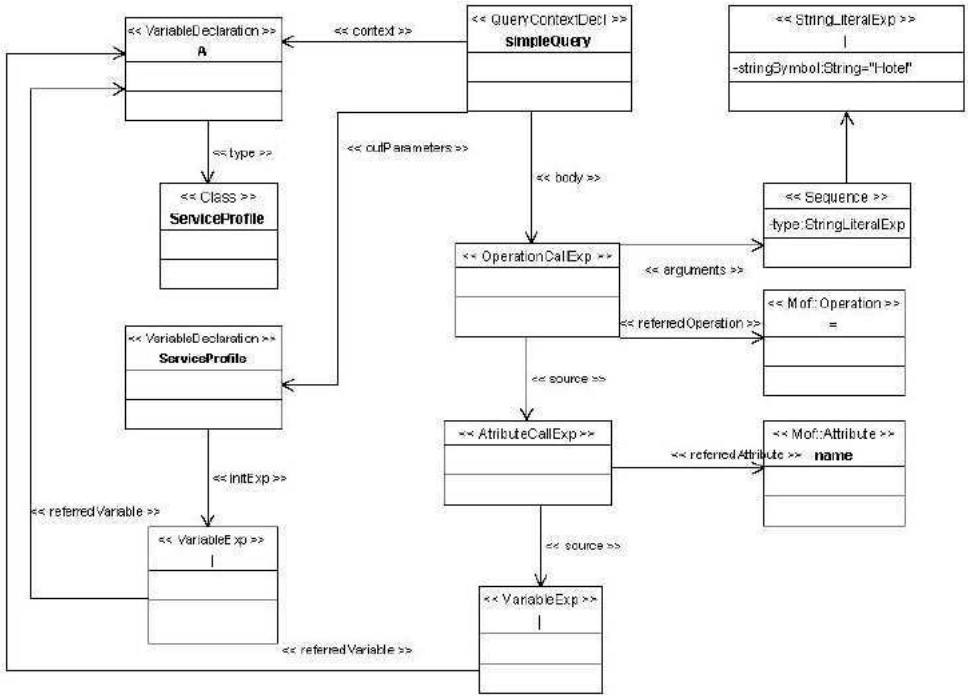


FIGURA 17. Modello QML relativo alla query dell'esempio 10.1

EXAMPLE 10.1. Consideriamo il meta-modello *Semantic Service Language* (*SSL*), il quale permette di descrivere semanticamente un servizio, e consideriamo la seguente query, posta su tale meta-modello, la quale recupera tutti i Service Profile il cui nome è uguale a "Hotel":

```
Context A: SSL::ServiceProfile simpleQuery
A.name = "Hotel"
out RServiceProfile := A
```

In figura 17 è rappresentato il modello QML relativo alla query `simpleQuery`. La semantica della query, in termini del meta-modello QML può essere spiegata nel seguente modo. La query, definita in un oggetto *QueryContextDecl*, ha nome "simpleQuery". Il suo contesto, chiamato "A", ha come tipo la classe MOF "ServiceProfile" del meta-modello "SSL". Il parametro di output della query è "RServiceProfile", al quale è assegnato un valore dalla variabile "A". Si nota che il tipo del parametro di output è assegnato in modo automatica (in questo caso è la classe "ServicePackage") al parametro e che, nel caso in cui la query preveda più argomenti come risultato, il tipo della query viene calcolato come un tipo struttura (TupleType). Il corpo dell'espressione è la *OperationCallExp* che si riferisce all'operazione "=" del tipo primitivo String. In questa query l'OperationCallExp è costituito da un solo argomento, la *StringLiteralExp* "Hotel". Inoltre, l'OperationCallExp ha una sorgente di navigazione, la *navigation source*, al quale è applicato. Tale sorgente è la *AttributeCallExp*, la quale si riferisce all'attributo "name" della classe ServiceProfile. La sorgente dell'AttributeCallExp è una *VariableExp* che si riferisce alla variabile "A", la quale rappresenta il contesto della query.

EXAMPLE 10.2. L'esempio precedente mostra come interrogare un modello. Questo esempio mostra invece come, attraverso QML, sia possibile esprimere query sulle istanze dei modelli, ovvero sui dati presenti a livello M0 dello stack definito da MOF. La query seguente cerca tutti i ServiceProfile "Hotel" il cui ServiceAttribute "HotelName" ha valore "Hilton" presenti a "Roma".

```
Context A: HotelModel#Hotel instanceQuery
A.HotelName = "Hilton" and
A.HotelAddress.City = "Rome"
out Hotels := A
```

Si nota che nell'interrogazione di dati a livello M0 la query è espressa in termini di uno specifico modello (in questo caso il modello "Hotel"); ciò significa che non si è in grado di recuperare l'hotel "Hilton" di "Roma" se tale hotel è espresso in termini di un altro modello, ad esempio "HotelModel2", il quale presenta una struttura diversa. Al contrario, questo problema non è rilevante per la ricerca di modelli in quanto si assume che i meta-modelli cambino con basse frequenze.

CAPITOLO 4

Creazione, Ricerca ed Esecuzione di Servizi DBE

1. Infrastruttura Tecnica

In questa sezione vengono illustrati i componenti dell'architettura di DBE e descritte le loro funzionalità. L'analisi riportata fa riferimento alla versione del sistema rilasciata nel periodo in cui questa tesi è stata scritta. Altre versioni sono previste nei mesi successivi alla stesura di tale tesi.

Nel capitolo precedente sono stati descritti i tre componenti funzionali dell'architettura di DBE, ovvero il Service Factory, l'Execution Environment e l'Evolution Environment. Per ognuno di questi componenti sono stati progettati e sviluppati software che ne implementano le funzionalità; tali software sono il DBE Studio, il quale rappresenta un Service Factory, il DBE servent, ovvero l'execution environment e EveNet, il quale implementa l'evolution environment.

DBE Studio. DBE Studio è l'integrated development environment per DBE; allo stato attuale è un insieme di editor, tool e wizard che consentono la creazione, l'analisi e il deployment dei servizi nell'execution environment. Esso è costruito attualmente sulla piattaforma Eclipse[1] ed è rilasciato sotto la Eclipse Public License.

La versione analizzata è la 0.2.0 rilasciata nel mese di Febbraio 2006. Recentemente (fine Novembre 2006) è stata rilasciata la versione 0.3.0.

Al fine di supportare le operazioni di creazione e deployment dei servizi e dei modelli di business, DBE Studio offre diversi tool organizzati in diverse *perspective*, tra cui:

- *Business Analysis*: fornisce gli strumenti per modellare i servizi e i modelli di business, ovvero, tale perspective contiene gli editor BML e SSL, implementati come plug-in per Eclipse. Permette inoltre di cercare modelli già creati e di riutilizzarli nel proprio progetto, attraverso la funzione di import. I modelli possono pertanto essere memorizzati e recuperati dal Semantic Registry. Tale perspective fornisce anche il tool BML Data Editor, il quale permette di istanziare un determinato modello BML, cioè permette di creare un documento che si posiziona a livello M0 dell'architettura di MOF.
- *Ontology Analysis*: gestisce tutte le funzionalità per la gestione delle ontologie, ovvero fornisce l'editor ODM, permette di memorizzare e caricare ontologie dalla knowledge base.
- *Semantic Discovery*: Fornisce un'interfaccia grafica e gli strumenti per cercare nelle KB dei peer, modelli di business o di servizi già creati. I tool creati permettono di creare template per modelli BML, SSL e SDL e di sfruttare tali template per generare delle vere e proprie query.

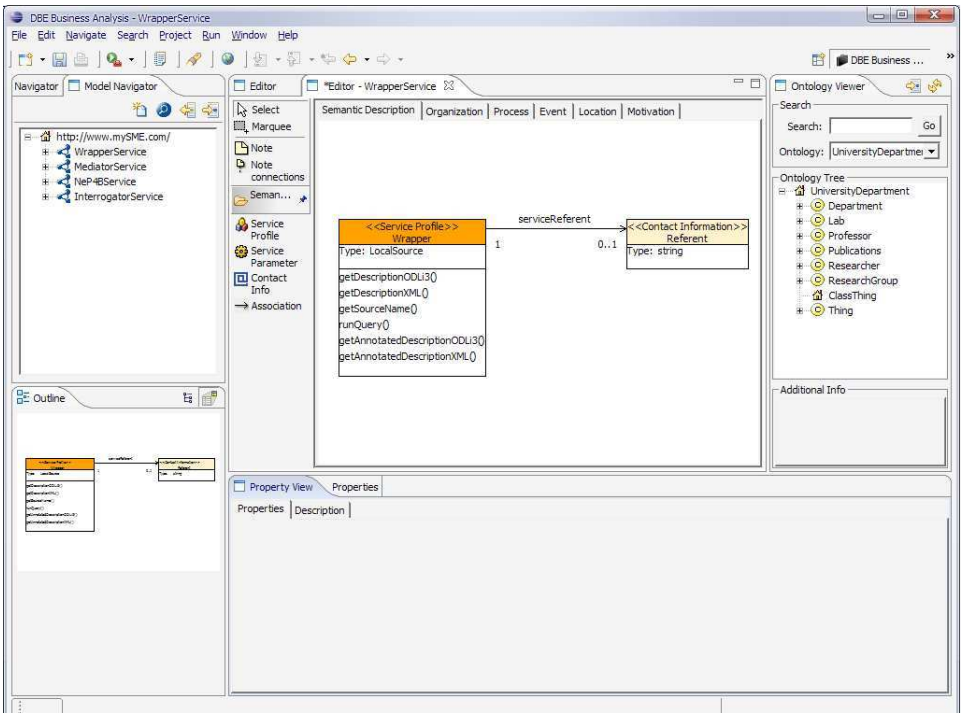


FIGURA 1. Screenshot della Business Analysis Perspective di DBE Studio

- *Service Composition*: permette di creare servizi composti; implementa pertanto l'editor BPEL.
- *Service Development*: è la perspective da utilizzare per l'implementazione del servizio. Contiene diversi tool, tra i quali un editor SDL, che permette di definire l'interfaccia tecnica del servizio. Contiene inoltre il plug-in SDL2WSDL, il quale converte un documento SDL in un documento WSDL. La generazione degli stub e degli skeleton avviene per mezzo del plug-ing WSDL2Java. Un ulteriore plug-ing, chiamato Service Manifest Creator, consente di recuperare dal Semantic Registry i modelli BML e SDL relativi al servizio che si sta implementando e, da essi, di creare il Service Manifest. Infine, tale perspective contiene anche un editor Java, utilizzato per creare la logica di business del servizio.
- *Service Publishing*: permette di pubblicare un servizio nell'ecosistema. In questa fase, tutti i file necessari per eseguire un servizio vengono archiviati in un file DAR (DBE Archive). Esso rappresenta un file compresso contenente l'implementazione del servizio, l'implementazione dell'eventuale interfaccia utente, eventuali informazioni di workflow, il service manifest e un file di configurazione. Terminata la creazione del file DAR, è possibile pubblicare il proprio servizio attraverso il plug-in Service Exporter.

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

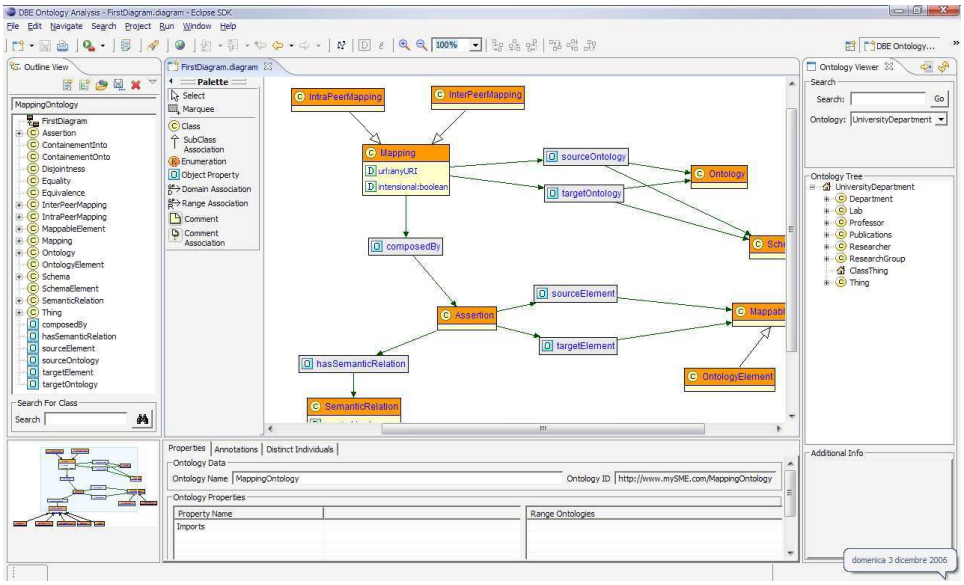


FIGURA 2. Screenshot della Ontology Analysis Perspective di DBE Studio

In figura 1 è riportata l'interfaccia di DBE Studio relativa alla business analysis perspective. La parte centrale dello screenshot contiene il BML Editor: si possono notare infatti i vari package del linguaggio BML (Organization, Process, Event, Location e Motivation) più la Semantic Description per la modellazione SSL dei servizi. La parte destra dello screenshot contiene l'Ontology Viewer, un plug-in che fornisce la visualizzazione (ad albero) delle ontologie memorizzate nella KB, pertanto è molto dipendente dal plug-in KB-Toolkit. La parte destra contiene invece un Model Navigator, il quale visualizza tutti i modelli di business memorizzati nella KB e il Navigator, che visualizza invece i progetti DBE attivi. Infine, la Property View è un componente di supporto al BML Editor.

In figura 2 è invece riporta la perspective relativa all'ontology analysis. Le parti centrale e inferiore sono costituite dall'Ontology Editor, mentre la parte destra contiene sempre l'Ontology Viewer. L'Outline View fornisce anch'esso una visione ad albero di tutti i concetti descritti nell'ontologia e fornisce anche gli strumenti per creare nuove ontologie, memorizzare e caricare ontologie dalla KB ed eliminare ontologie presenti. Nelle versioni successive di DBE Studio sono anche forniti gli strumenti per importare ed esportare le ontologie ODM in formato OWL-DL.

La figura 3 rappresenta invece lo screenshot relativo alla Service Discovery Analysis. Essa contiene i tool per generare query ed eseguire query. In particolare, tale perspective permette di generare dei Query Template relativi ai modelli BML, SSL e SDL i quali forniscono lo scheletro di una query: ogni template permette di specificare quali elementi del meta-modello (per BML, BusinessEntity, Service, Product, ecc.) interrogare. Le query sono create sulla base di tali template specificando i valori che il modello recuperati dovranno rispettare. La query mostrata in figura 3 è stata costruita sulla base di un template che prevede l'interrogazione di una Business Entity e di un suo attributo. La query create cerca quindi una BusinessEntity chiamata "ResearchGroup" con un attributo "name".

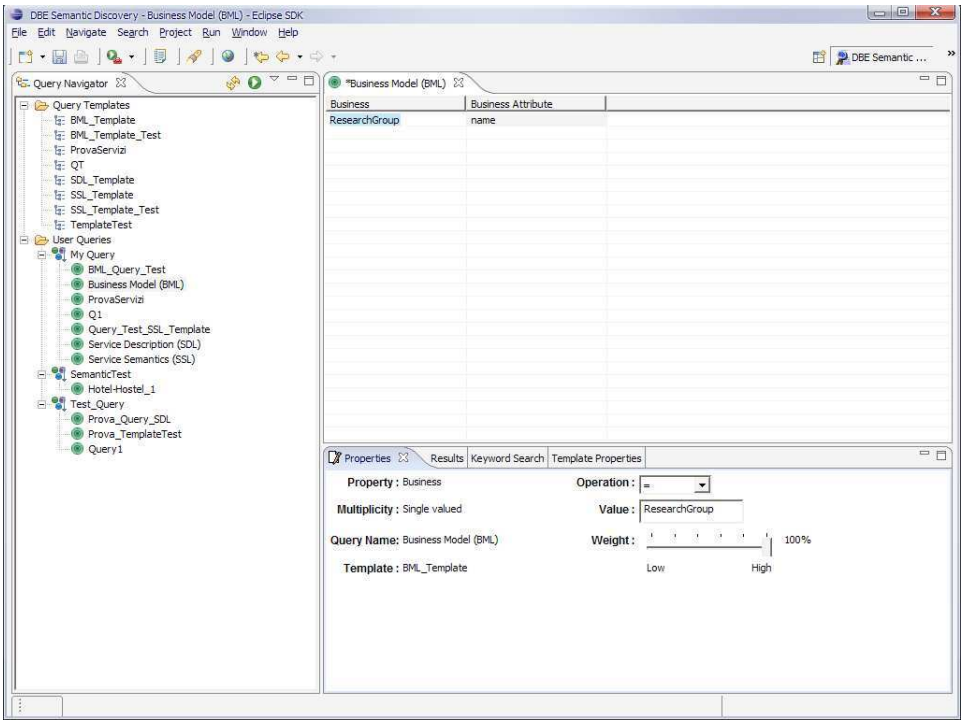


FIGURA 3. Semantic Discovery perspective di DBE Studio

Ogni elemento della query è valutato rispetto una operazione, che attualmente sono “=”, “like” e “<>”; queste operazioni permettono di esprimere che i modelli ritornati devono contenere elementi uguali, simili o diversi a quelli specificati nella query. I risultati di tale query vengono mostrati nel tab “Results”. Oltre a questo tipo di query, è possibile effettuare ricerche *keyword-based*, le quali vengono eseguite nel tab “Keyword Search”. Si nota infine che questo servizio interroga la knowledge base al fine di recuperare modelli BML, SSL e SDL.

Il servizio di semantic discovery è disponibile anche per il DBE Portal, un componente del sistema che verrà presentato al termine di questa sezione. In questo caso il servizio permette di ricercare servizi piuttosto che modelli; per questo motivo il servizio non interroga la Knowledge Base ma il Semantic Registry.

DBE ServENT. Il ServENT, il cui nome deriva da “SERVer” e “cliENT”, è quel componente software di DBE che implementa le funzionalità previste per l’Execution Environment. Ogni ServENT ha pertanto un doppio ruolo: funziona da server per l’esecuzione dei servizi che ospita e, allo stesso tempo, funziona da client per accedere e localizzare altri servizi da eseguire.

Durante questa tesi è stata valutata principalmente la versione 0.2.0 del servent, ma anche le versioni 0.2.1 e 0.2.2. Recentemente (metà e fine novembre) sono state rilasciate le versioni 0.3.0, 0.3.5, 0.3.7 e 0.3.8.

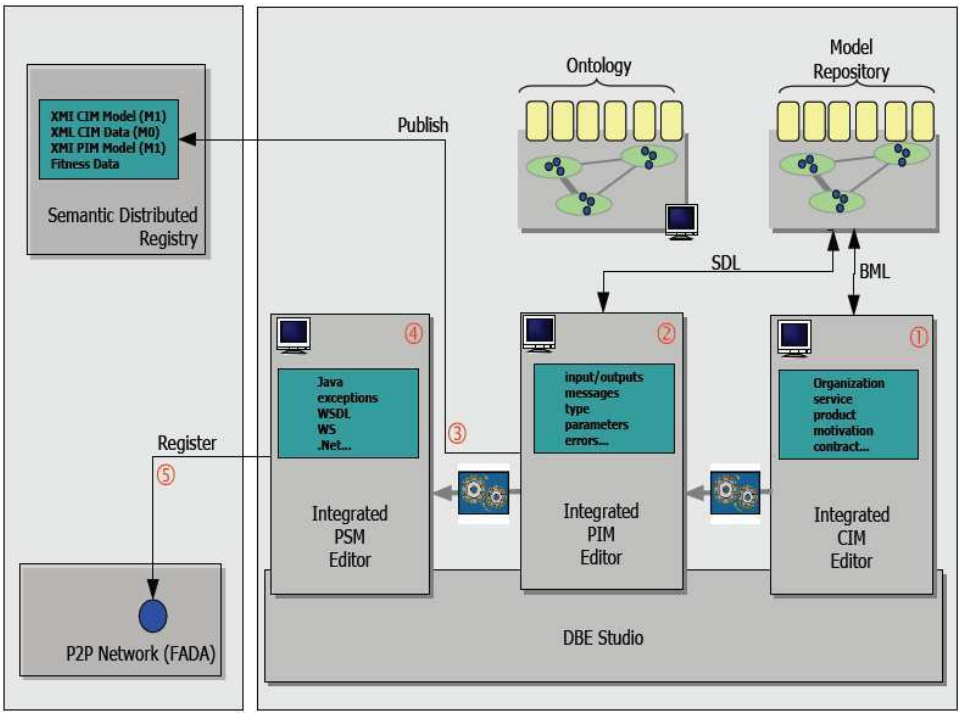


FIGURA 4. Interazione tra DBE Studio e DBE ServENT

Come già affermato nel capitolo precedente, ogni ServENT è interconnesso agli altri attraverso FADA[2], ovvero un servizio di directory P2P nel quale vengono registrati tutti i servizi DBE pubblicati. I tool forniti dal ServENT permettono di eseguire le seguenti operazioni:

- *Service Registration e Deployment*: terminata la progettazione del servizio per mezzo dei tool del DBE Studio, esso viene registrato nel "sistema nervoso" di DBE (la rete P2P definita da FADA) insieme al suo service manifest. L'interazione tra DBE Studio e DBE ServENT è mostrata in figura 4. Il primo passo consiste nella creazione del modello BML; successivamente (passo 2) viene creato il modello SDL il service manifest del servizio viene pubblicato nel semantic registry (passo 3). Il service proxy viene generato nel quarto step e pubblicato in FADA nel quinto. Dall'immagine si nota che i passi 3 e 5 interagiscono con il servent.
- *Service Search e Retrieval*: Per eseguire un servizio, la PMI deve prima trovare il servizio corretto. Il tool di ricerca, chiamato *Recommender*, permette di effettuare ricerche basandosi su un insieme di criteri che includono parole chiave, attributi e descrizione semantica dei servizi. Esso ritorna una lista di servizi che soddisfano i criteri specificati dall'utente. Selezionando uno di questi servizi, il service proxy viene scaricato nel servent del richiedente ed utilizzato per invocare i metodi opportuni.

[2] http://fada.sourceforge.net/

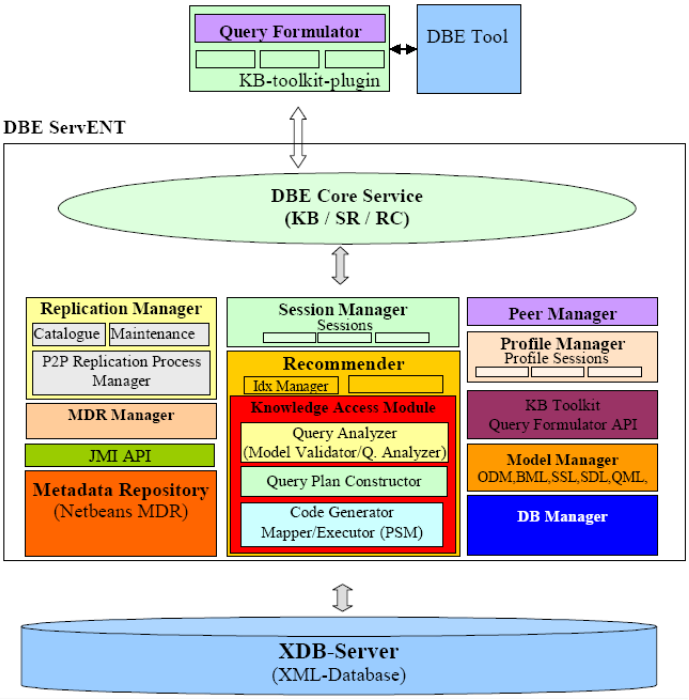


FIGURA 5. Componenti del DBE ServENT

- *Service Execution*: l'esecuzione di un servizio è basata sul proxy del servizio stesso, il quale nasconde al client tutte le problematiche relative alla distribuzione, in particolare alla locazione reale del servizio: per ogni richiesta ricevuta, il proxy invoca il servizio di back-end attraverso il servent. Lo scopo infatti del ServENT è quello di far apparire agli utenti del sistema che tutti i servizi siano localizzati nel nodo locale, anche quando non lo sono.

Il servent è inoltre *protocol agnostic*, caratteristica che è garantita dall'utilizzo dell'Abstract Protocol Adapter (APA). La comunicazione nella versione analizzata del ServENT è basata sul protocollo SOAP e anche sulla serializzazione di oggetti su HTTP.

Si nota inoltre che il ServENT fornisce inoltre l'integrazione, trasparente all'occhio dell'utilizzatore finale, con l'Evolution Environment e con alcuni servizi di base come, accounting, logging, sicurezza, pagamento e altri ancora.

Ogni servent è implementato come un insieme di tool e servizi (figura 5). In particolare, ogni servent fornisce una serie di *Core Service*, tra i quali vi sono il *Knowledge Base Service*, il *Semantic Registry Service*, *FADA* e il *Distributed Storage System* (DSS). Tra i tool principali si riconoscono invece il *Recommender*, il *KB Toolkit* e il *Metadata Repository*.

Metadata Repository (MDR). Tutta la conoscenza mantenuta dalla knowledge base rappresentata dai meta-modelli e dai modelli MOF-based è memorizzata nel repository MDR. Anche il meta-modello QML e le sue istanze (query) sono memorizzate in MDR. MDR fornisce varie interfacce per interagire con gli altri componenti del sistema. Il Knowledge Access Module utilizza tre interfacce:

- *IRepository*, che include operazioni per caricare e salvare modelli e meta-modelli nel formato XMI e operazioni per generare interfacce JMI partendo dai meta-modelli.
- *IQML*, che rappresenta un insieme di interfacce JMI che consentono l'accesso alle istanze del meta-modello QML in modo trasparente.
- *IReflective*, ovvero un insieme di interfacce riflessive JMI che consentono l'accesso ai modelli senza la generazione di interfacce specifiche per i meta-modelli.

Knowledge Base (KB) e Semantic Registry (SR) Service. Questi servizi, anch'essi basati su JMI, esportano le funzionalità della knowledge base mettendole a disposizione degli altri componenti di DBE. Queste funzionalità forniscono supporto per la memorizzazione, l'interrogazione e il recupero di modelli e dati. Nel Service Factory Environment viene utilizzato il KB Service, mentre nell'Execution Environment è utilizzato il SR Service.

Query Formulator. Il Query Formulator è quel componente che si interfaccia direttamente con l'utente e consente la formulazione di query consistenti rispetto il meta-modello QML. Nella versione attuale, tale componente permette di visualizzare le primitive dei meta-modelli implementati in DBE (BML, SSL e SDL) e di porre dei vincoli su tali primitive, che devono essere soddisfatti dai modelli o dalle istanze richieste. Si nota che esso non è parte del ServENT, ma interagisce direttamente con esso.

Recommender. Il *Recommender* è utilizzato in DBE per personalizzare l'accesso alla conoscenza del sistema da parte degli utenti. Esso deve essere in grado di soddisfare entrambi gli approcci classici di information retrieval per fornire accesso alla conoscenza, basati su una rischiesta esplicita dell'utente con l'utilizzo di certi criteri e, in aggiunta, il ruolo di servizio autonomo, per filtrare la conoscenza di DBE e fornire all'utente utili raccomandazioni. Il responsabile di queste funzionalità è il *Recommender Service*. Le preferenze degli utenti e i requisiti necessari per le procedure di filtraggio e raccomandazione sono fornite dagli *user profile*. Questa informazione viene memorizzata negli *User Profile Model* (*UPM*). La parte principale di un profilo è quella che mantiene le informazioni sulle preferenze dell'utente nella forma di vincoli (*constraint*) riferiti ad uno specifico utente. Tutti i vincoli sono formulati tramite espressioni OCL. Si nota che il Recommender Service sfrutta il modulo di accesso per recuperare le informazioni di interesse; i componenti di tale modulo sono illustrati di seguito.

Query Analyzer. Questo modulo analizza una espressione QML in modo da interpretarla e validarla rispetto la semantica definita dal meta-modello dell'espressione. Tale componente utilizza l'interfaccia IQML per accedere alle espressioni QML. L'interfaccia IReflective è invece utilizzata per accedere all'informazione relativa al meta-modello, memorizzata in MDR, per poter effettuare la valutazione della semantica della query.

Query Execution Plan Constructor. Il compito di questo modulo è quello di valutare le espressioni QML per generare una rappresentazione ad albero sintattico. Questo modulo ha due obbiettivi principali. Il primo è quello di nascondere agli occhi dell'execution engine (nella versione attuale si tratta di un XQuery executor) la complessità del meta-modello QML. Il secondo obbiettivo, non ancora implementato attualmente, è quello di supportare l'ottimizzazione delle query e di semplificare il più possibile le espressioni. Il processo di creazione dell'albero sintattico è basato sull'applicazione di certe regole agli elementi del modello QML, ovvero agli elementi della query.

Code Generation. QML è utilizzato per porre una query rispetto uno specifico meta-modello. Le istanze di questo meta-modello possono essere memorizzate in una qualche memoria

permanente (relazionale, XML, ecc.). Per poter rispondere all'interrogazione posta secondo il meta-modello QML, la query deve essere tradotta nell'appropriato linguaggio (code) per interrogare i dati memorizzati sul data management system utilizzato (ad esempio, SQL per DBMS, XQuery per basi di dati XML, ecc.). Tale componente dunque prende in input l'albero sintattico e produce il codice da eseguire sul data management system. Per generare il codice, è necessario eseguire un mapping tra i concetti nel meta-modello e gli elementi dello schema utilizzato per catturare questa informazione per lo specifico sistema di data management. Nella versione attuale, le query formulate vengono memorizzate nel repository MDR e il codice generato è XQuery, in quanto il sistema utilizzato per la memorizzazione persistente dei dati è un database XML.

Query Executor. Le espressioni prodotte dal Code Generator sono inviate all'executor, il quale è responsabile per l'esecuzione della query nell'ordine corretto. Analogamente al caso precedente, l'esecutore è un XQuery engine.

EveNet. Rappresenta il componente software che implementa le funzionalità dell'Evolution Enviroment. Tale software è concepito e implementato come un sistema decentralizzato, in modo da garantire l'affidabilità del servizio fornito da EvE e che non vi sia nessun controllo centralizzato per quanto riguarda l'evoluzione dei servizi.

Rispetto agli altri componenti del sistema, EveNet è trasparente, ovvero il suo funzionamento non è direttamente percettibile dagli utilizzatori del sistema. Infatti, EveNet riceve informazioni dall'execution environment e in base ad esse aggiorna la service chain per quella determinata PMI. In particolare, EveNet deve poter accedere alla knowledge base e al semantic registry per raccogliere più informazioni possibili riguardo i servizi.

Al momento di questa analisi nessuna versione di EveNet era disponibile. La prima versione è stata rilasciata all'inizio di Dicembre 2006.

DBE Portal e DBE Desktop. Il *DBE Portal* è un sito web che fornisce un accesso rapido a DBE. Esso è responsabile di fornire le applicazioni di DBE, ovvero rappresenta una piattaforma che consente di scaricare il DBE Studio, ServENT, e il DBE Desktop. Il cuore del DBE Portal è rappresentato da un Content Management System il quale permette di organizzare e gestire il contenuto del portale. Inoltre, esso permette agli utenti di cercare i servizi utilizzando quindi un normale Web Browser, di invocarli attraverso il protocollo HTTP e di eseguirli. Tutta la computazione è eseguita a lato server (portal) mentre i risultati dell'esecuzione sono mostrati nel client (browser). Questo approccio consente quindi ad ogni utente con una semplice connessione ad Internet e con un normale browser di eseguire servizi offerti nell'ecosistema di DBE, senza dover installare tutto il software di DBE. Il portal è composto da tre diverse aree funzionali:

- *Information Functionality*: consentono di accedere ai documenti di DBE.
- *Application/Service Functionality*: permettono di cercare, visualizzare ed eseguire servizi. Forniscono inoltre servizi di White directory (informazioni di contatto e indirizzo), Yellow directory (catalogazione dei servizi in base al tipo di business) e Green directory (informazioni tecniche riguardo i servizi).
- *Infrastructural Functionality*: registrazione a DBE, accesso alle applicazioni i DBE, elenco dei nodi della rete, statistiche sul sistema (numero di nodi, topologia e dimensione della rete, ecc.).

Durante lo svolgimento della tesi tale componente era in fase di sviluppo. La prima versione è stata rilasciata alla fine di Novembre 2006 ed è stata inclusa nella versione 0.3.8 del ServENT.

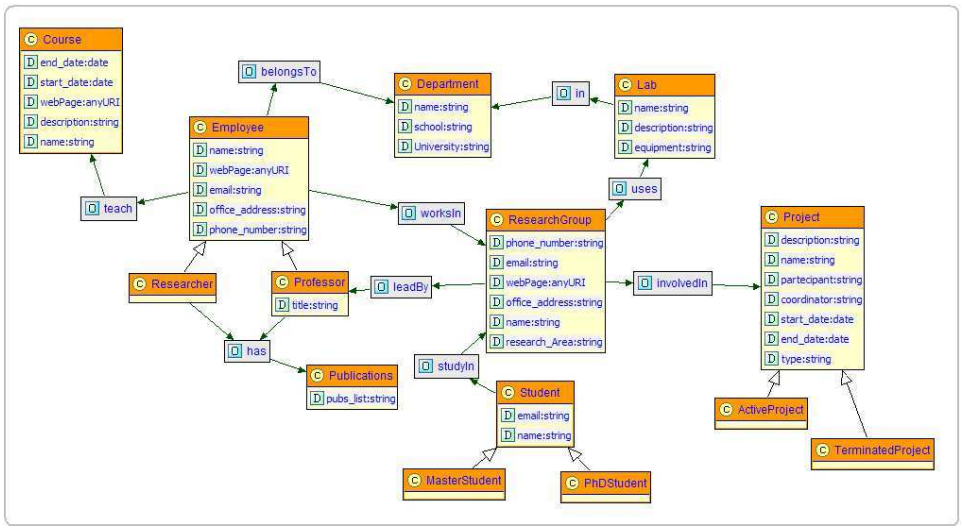


FIGURA 6. Ontologia OntologyDepartment

Il *DBE Desktop* è un tool che consente di gestire i servizi utilizzati da una PMI. In particolare esso fornisce delle funzionalità di monitoraggio delle transazioni in atto.

2. Creazione e Deployment di Modelli e Servizi: un Caso di Studio

Di seguito si mostra il procedimento da seguire per la modellazione di servizi in DBE. Lo strumento utilizzato a tale scopo è DBE Studio, descritto nella sezione precedente. Questo procedimento è stato seguito per creare un semplice servizio, chiamato Course Registration, che permette ad uno studente di iscriversi ad un corso universitario. Sono stati inoltre creati modelli e ontologie per permette la realizzazione di tale servizio. In particolare, si è modellato un gruppo di ricerca universitario, il quale è il fornitore del servizio creato. Come vedremo, la modellazione e realizzazione dei servizi è composta da diversi step.

Creazione delle ontologie

Le ontologie in DBE assumo un doppio ruolo: forniscono agli sviluppatori una “guida” per creare i modelli di business e rappresentano un’importante strumento di typing per gli attributi dei modelli sviluppati. Tale fase non è strettamente necessaria in quanto è possibile che tra le ontologie presenti nel sistema vi siano già quelle necessarie per la modellazione dei propri servizi. Tuttavia è molto utile in quanto permette di definire tutti i concetti che verranno utilizzati nei modelli BML e SSL e, pertanto, permette di semplificare la fase di modellazione dei business e dei servizi. Lo strumento da utilizzare è l’Ontology Editor, il quale fornisce gli strumenti per caricare le ontologie dalla KB, modificarle, e anche di crearne di nuove. Per la descrizione dei concetti inerenti un gruppo di ricerca, è stata creata l’ontologia mostrata in figura 6, detta *DepartmentOntology*.

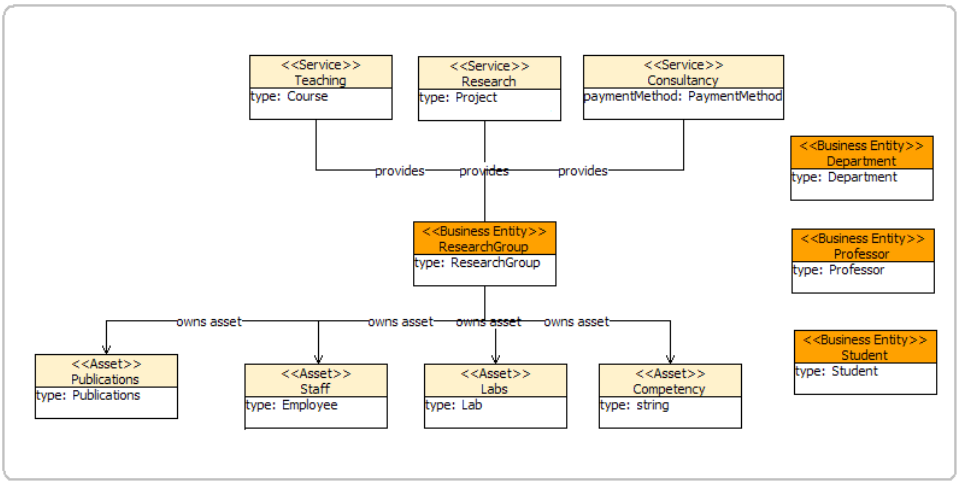


FIGURA 7. Organization Diagram di ResearchGroup

In tale ontologia vengono definiti alcuni dei principali concetti riguardanti il mondo accademico. La classe `ResearchGroup` descrive un gruppo di ricerca, ognuno dei quali presenta un nome, un'area di ricerca, un indirizzo, ecc. Ogni gruppo di ricerca è guidato (`leadBy`) da un `Professor`, può essere coinvolto in più `Project` e utilizza per le proprie attività dei laboratori (`Lab`). `Professor` e `Researcher` appartengono ad un `Department` e possono insegnare più `Course`. L'ontologia descrive anche i concetti di `Publications` e `Student`.

Creazione dei modelli BML e SSL. La creazione di tali modelli avviene nella Business Analysis perspective di DBE Studio. Il BML Editor costituisce un ambiente grafico per la modellazione di modelli BML e SSL e fornisce tutte le primitive necessarie alla modellazione. La creazione del modello di business può prevedere diverse fasi, le quali catturano caratteristiche diverse del business da modellare. Ad esempio, le primitive per la descrizione generale dell'organizzazione, delle sue normali attività, risorse, prodotti e servizi vengono fornite dal package *BusinessOrganization* di BML e possono essere modellate nella *Organization* perspective del BML Editor. In particolare, l'editor prevede una specifica perspective per ogni package definito nel meta-modello di BML e ogni perspective fornisce le primitive di modellazione definiti dal package corrispondente. Pertanto, le altre perspective dell'editor sono *Process*, *Event*, *Location* e *Motivation*. Il BML Editor prevede inoltre una ulteriore perspective, la *Semantic Description*, la quale fornisce le primitive necessarie per creare modelli SSL.

Per creare un qualsiasi elemento di un modello è necessario selezionare dall'editor il corrispondente comando. Ad esempio, facendo riferimento alla figura 1, per modellare un Service Profile selezione il comando "Service Profile" del BML Editor. Un questo modo si ottiene un elemento vuoto che è possibile modificare attraverso la Property View (nella parte inferiore di DBE Studio). In generale, ogni perspective di Business Analysis fornisce i comandi relative alle primitive del package. Ad esempio, la Semantic Description fornisce i comandi per generare i *Service Profile*, i *Service Parameter*, le *Contact Information* e le *Association*.

In figura 7 è riportato il modello BML relativo al package Organization del servizio realizzato. Come si vede, esso sfrutta i concetti dell'ontologia DepartmentOntology come tipi di dato per i concetti del modello.

Un gruppo di ricerca è modellato come una BusinessEntity; esso prevede un solo attributo, `type`, il cui tipo è rappresentato appunto dalla classe `ResearchGroup` definita nell'ontologia creata. Ciò significa che ogni `ResearchGroup` è descritto da tutti gli attributi previsti dalla classe dell'ontologia, ovvero `name`, `research_area`, ecc.

Tale procedimento è stato seguito per modellare anche gli altri concetti espressi nel modello. Le pubblicazioni, le persone che coinvolgono lo staff del gruppo, i laboratori impiegati per la attività di ricerca sono stati modellati come Asset. E' stato inoltre introdotto un ulteriore asset che descrive le competenza (`Competency`) del gruppo di ricerca. Un ResearchGroup inoltre fornisce tre tipi di servizi: `Teaching`, `Research` e `Consultancy`. Quest'ultimo Service è descritto da un attributo, `paymentMethod`, che specifica il tipo di pagamento accettato per un servizio di consulenza. La classe `PaymentMethod` è stata presa da un'ontologia già presente nella KB; i tipi di pagamenti previsti sono contanti, carta di credito, bonifica bancario, ecc.

Come detto, la realizzazione del modello SSL relativo al servizio è realizzato nella Semantic Description della business analysis perspective. Il servizio realizzato (figura 8), CourseRegistration, prevede tre operazioni principali: `makeRegistration()`, che permette ad uno studente di registrarsi ad un corso, `cancelRegistration()`, utilizzata per rimuovere l'iscrizione e `check()` che consente di verificare se per un dato corso sono consentite le iscrizioni.

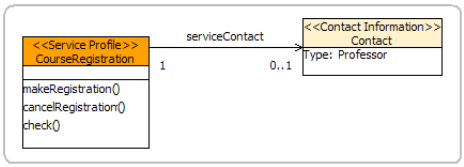


FIGURA 8. Semantic Description del servizio realizzato

L'inserimento dei parametri di input e output per ogni operazione viene effettuato nella Property View di DBE Studio (figura 9). Per ogni parametro è possibile specificare il tipo di dato che, anche in questo caso, può essere una classe definita da una ontologia.

Aggiunta dei dati. E' possibile istanziare i modelli BML e SSL creati, o importanti, attraverso il BML Data Editor, mostrato in figura 10. Questo tool recupera i modelli direttamente dalla KB; pertanto, prima è necessario individuare il modello da istanziare tra quelli presenti nella KB. Successivamente, il data editor permette di istanziare ogni attributo definito negli elementi dei modelli. Anche i dati creati sono successivamente memorizzati nella KB in formato XMI, esattamente come tutti gli altri modelli definiti.

Dalla figura 10 si osserva che l'editor consente di inserire i dati relativi sia alle data property (name, email, ecc.) che alle object property (leadBy, uses, ecc.) derivate dall'utilizzo della classe ResearchGroup come tipo di dato della relativa BusinessEntity.

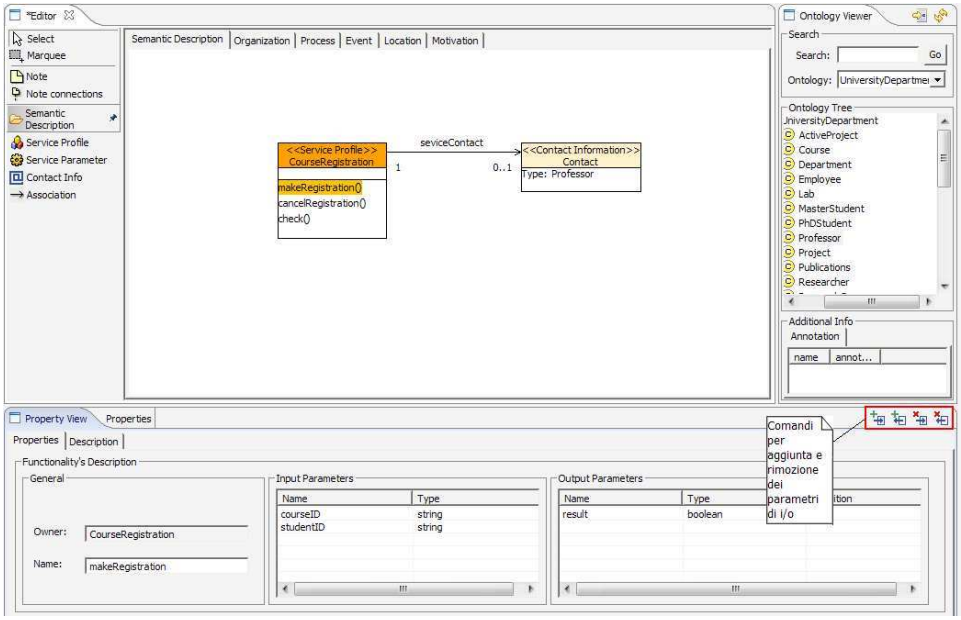


FIGURA 9. Inserimento dei parametri di input/output per le operazioni nei modelli SSL

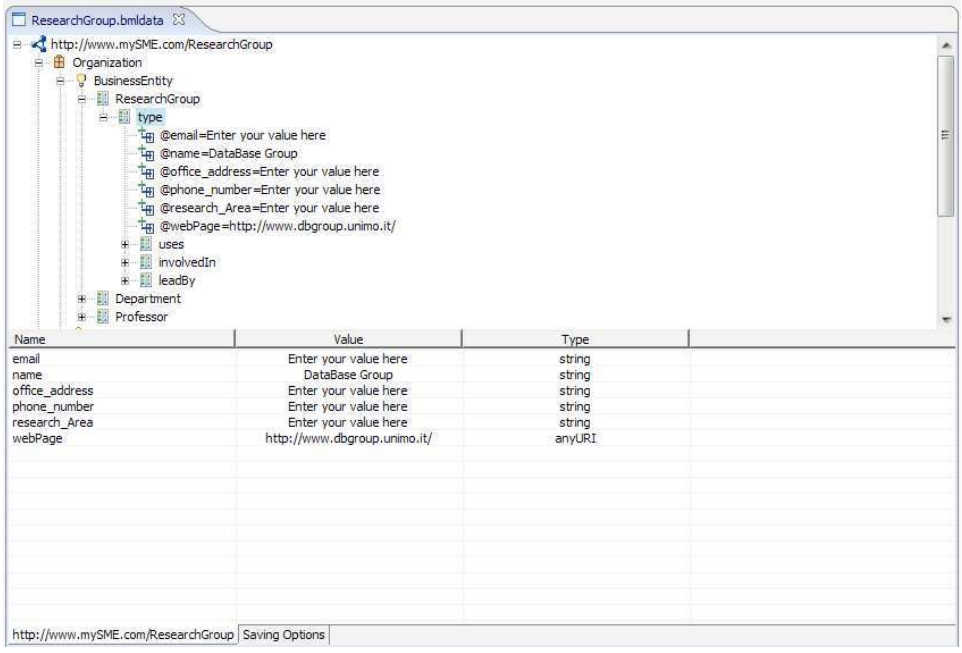


FIGURA 10. Screenshot relativo al BML Data Editor

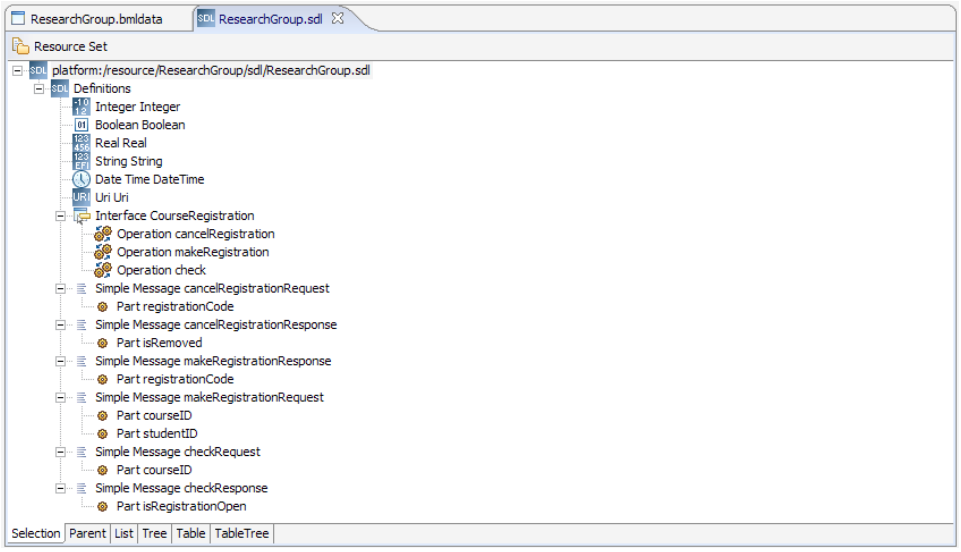


FIGURA 11. Modello SDL relativo al servizio CourseRegistration

Generazione della Service Description. Il passo successivo consiste nel generare la descrizione SDL del servizio partendo dal modello BML. In particolare, il tool SSL2SDL consente di tradurre la parte di descrizione semantica di un modello BML (SSL), memorizzato in formato XMI, nella rispettiva interfaccia SDL. Spesso, come descritto nel capitolo precedente, le operazioni definite nel modello SSL non sono sufficiente per creare un servizio completo; esse servono principalmente infatti per caratterizzare semanticamente il servizio. Dal momento che lo scopo di SDL è quello di descrivere il servizio in modo tecnico, si rende necessario aggiungere nuove operazioni a quelle definite nel modello SSL. Questo è reso possibile dall'SDL Editor, il quale fornisce un'interfaccia grafica dove la descrizione SDL viene strutturata ad albero, come mostrato in figura 11.

Come si nota dalla figura 11, SSL2SDL effettua i seguenti mapping fra modelli:

- Un *ServiceProfile* di SSL viene trasformato in una *Definition* di SDL.
- La *ServiceFunctionality* è tradotta in una *Interface*.
- Le *Operation* di SSL vengono tradotte in *Operation* di SDL, le quali sono parti costituenti dell'interfaccia.
- I parametri di input e output sono mappati in *SimpleMessage*.

Generazione dell'Adapter e del codice del servizio. Dal modello SDL è possibile creare in modo (semi) automatico la classe Java che implementa l'adpater del servizio. Tale operazione è effettuata dal tool SDL2Java.In particolare tale tool crea due classi Java partendo dal file SDL: una è rappresentata dall'adapter, mentre la seconda è l'interfaccia astratta del servizio.

Il codice che implementa il servizio va inserito nella classe Adapter. Spesso, i servizi implementati servono per pubblicare in DBE dei servizi legacy posseduti dalla PMI.

```
/** Generated with SDL2Java Compiler    */
package org.dbe.demos.courseregistration;

import javax.xml.rpc.holders.BooleanHolder;

public class CourseRegistrationAdapter implements CourseRegistration, Adapter {
    /*---------Service Methods--------*/
    CourseRegistrationImpl impl = new CourseRegistrationImpl();

    public void cancelRegistration(String registrationCode,
            BooleanHolder isRemoved) throws java.rmi.RemoteException {
        // write your code here
        isRemoved.value = impl.cancelRegistration(registrationCode);
    };

    public void makeRegistration(String courseID, String studentID,
            StringHolder registrationCode) throws java.rmi.RemoteException {
        // write your code here
        registrationCode.value = impl.makeRegistration(courseID, studentID);
    };

    public void check(String courseID, BooleanHolder isRegistrationOpen)
            throws java.rmi.RemoteException {
        // write your code here
        isRegistrationOpen.value = impl.check(courseID);
    };

    /*---------Adapter Methods--------*/public void init(ServiceContext context) {
        // write your code here
    }

    public void destroy() {
        // write your code here
    }
}
```

FIGURA 12. Classe CourseRegistrationAdapter del servizio CourseRegistration

In questo caso, la classe che implementa l'adapter deve contenere il codice per richiamare le funzioni dell'applicazione legacy: tale classe pertanto fa da ponte tra i sistemi legacy e l'ecosistema definito da DBE.

Nel nostro caso, l'implementazione vera e propria del servizio è stata realizzata in una seconda classe chiamata CourseRegistrationImpl.java. La classe CourseRegistrationAdapter.java (l'adapter) definisce pertanto un oggetto di tipo CourseRegistrationImpl sul quale invoca gli opportuni metodi. L'adapter del servizio realizzato è mostrata in figura 12. Come si nota, tale classe implementa la CourseRegistration.java, che rappresenta l'interfaccia del servizio, e ne definisce i tre metodi.

Creazione del Service Manifest. Per poter creare il service manifest del servizio, DBE offre uno specifico tool il quale fornisce un'interfaccia grafica di supporto alla creazione (figura 13). L'identificatore del service manifest (SMID) viene creato in modo automatico dal sistema. L'utente ha la possibilità di inserire il nome del service manifest (SMName), una descrizione e la versione. Il Service Manifest Creator permette inoltre di recuperare i modelli (BML, SDL e BML Data) per poterli includere nel SM; la scelta e l'aggiunta dei modelli BML e SDL avviene nel ServiceDNA tab (non mostrato in figura) mentre quella relativa a BML Data avviene nel ServiceManifest tab (vedi figura).

Pubblicazione del servizio. La pubblicazione del servizio è competenza del DBE Service Exporter. Questo tool, mostrato in figura 14, consente di selezionare il servizio realizzato, di aggiungergli un nome e una breve descrizione.

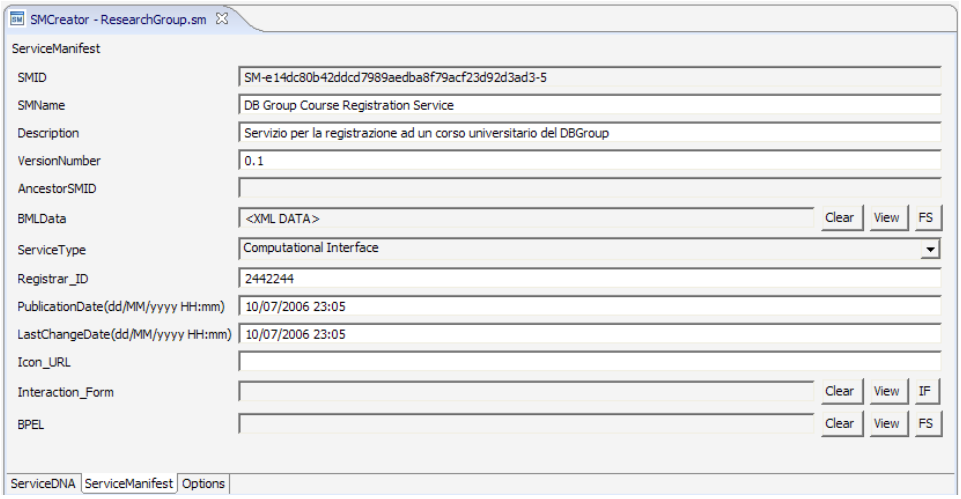


FIGURA 13. Service Manifest Creator

La pubblicazione del servizio nell'ecosistema necessita anche di indicare la classe che implementa l'adapter del servizio e, opzionalmente, un'interfaccia (che può essere una Applet o una Swing Java o un'interfaccia Laszlo). Terminata questa fase, il servizio è accessibile da chiunque utente nella rete.

3. Ricerca di Modelli e Servizi

Come affermato nella prima sezione del capitolo, la ricerca di modelli e servizi in DBE è eseguita del Recommender, attraverso il Knowledge Access Module, avvalendosi anche dei servizi di Knowledge Base e Semantic Regisrty.

In particolare, se l'ambiente nel quale il recommender viene invocato è il Service Factory Environment (DBE Studio), il servizio che accede alle funzionalità del modulo di accesso è il KB Service, il quale permette di recuperare i modelli di business memorizzati nella KB; quando l'ambiente è l'Execution Environment (ServENT), è il SR Service che utilizza le funzionalità del modulo per accedere ai Service Manifest.

Viene ora descritto il meccanismo di query processing peer-to-peer adottato da DBE. Due sono i problemi che tale meccanismo risolve: il routing delle query nella rete e il processing di una query che arriva da un altro peer sul proprio peer.

Il routing della query è basato sull'utilizzo di *indici semantici* memorizzati in ogni peer. Tali indici sono utilizzati per identificare quei peer che possono fornire una risposta significativa ad una query. In questo modo è possibile inviare la query direttamente a questi nodi. Questa soluzione offre due vantaggi principali: evita il flooding della query nella rete, alleggerendo quindi il traffico sulla rete, e permette di ottenere un numero maggiore di risposte significative. In particolare, il meccanismo di flooding è utilizzato principalmente per scoprire nuovi nodi nella rete e pertanto viene utilizzato nel bootstrap del sistema. Sulla base delle risposte ottenute dalla query, attraverso un meccanismo di machine learning vengono costruiti gli opportuni indici semantici. Tali indici hanno la seguente struttura:

Path	Ontology Concept	Rank	Node ID

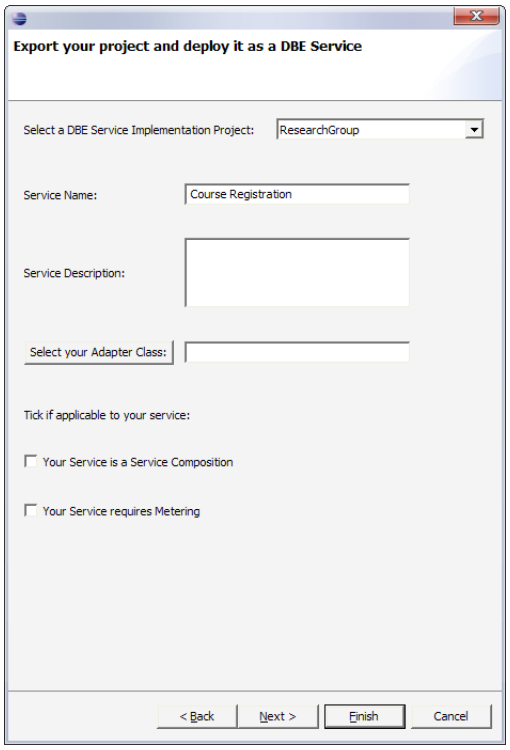


FIGURA 14. DBE Service Exporter

dove `Path` indica il percorso espresso dalla query, `Ontology Concept` è il concetto collegato al path, `Rank` è un valore da 0 a 1 che misura la similarità fra i concetti nei due peer e `Node ID` è l'identificatore del peer target. La creazione degli indici è basata su una procedura di apprendimento continua. Il risultato di questa procedura è l'acquisizione di conoscenza del contenuto di nodi diversi.

Quindi, le query vengono indirizzate in base agli indici semantici. Quando un peer riceve una query posta su un altro nodo, nasce il problema di processare la query rispetto la conoscenza locale. Questo problema è causato da possibili inconsistenze tra il modello rispetto al quale la query è posta e quello rispetto al quale deve essere valutata. Una query è composta principalmente da due parti: un elemento del modello e un valore. DBE prevede anche un terzo componente, ovvero il concetto dell'ontologia associata all'elemento del modello. A tale riguardo, la cosa importante da notare consiste nel fatto che tali concetti sono presi dalle ontologie di dominio, accessibili a tutti i nodi della rete. In base a questa assunzione vi sono due possibilità: il nodo sul quale la query è stata formulata e il nodo al quale la query è stata posta usano le stesse ontologie di dominio oppure usano ontologie diverse ma appartenenti allo stesso dominio. In entrambi i casi il nodo che ha ricevuto la query deve analizzarla per identificare tutti i concetti che contiene. Se i concetti sono originati dalla stessa ontologia allora il matching tra gli elemento dei due modelli diventa semplice. Nel secondo caso invece viene applicato un meccanismo che identifica possibili matching tra i concetti della query e quelli del modello locale.

Si nota infine che il meccanismo di routing basato su indici semantici consente di creare dei collegamenti semantici tra i nodi che non corrispondono necessariamente a quelli della rete fisica: si può pertanto affermare che l'implementazione attuale del Recommender è basata sul concetto di *Semantic Overlay Network* (LOSER *et al.* 2003).

4. Esecuzione dei Servizi

Per eseguire un servizio è necessario recuperare dalla rete FADA il suo proxy. Attraverso l'interfaccia del servizio, il proxy è in grado di invocare in servizio di back-end, attraverso i servizi offerti dal ServENT. In questa invocazione il sistema di accounting viene sia dal servent del client che da quello della PMI server. Il sistema di accounting è in grado in questo modo di registrare diversi parametri, quali l'identità dei due utenti coinvolti, il servizio, l'operazione invocata e altri ancora. I risultati ritornati da una invocazione sono passati dal server al client passando per il sistema di accountig. Successivamente tali risultati vengono visualizzati nell'interfaccia a lato client. Un'illustrazione dettagliata della fase di esecuzione di un servizio è riportata in figura 15: si nota che il ServENT fornisce l'abilità di caricare il service proxy a runtime e di fornire un'interfaccia SOAP per la comunicazione con l'applicazione consumer.

5. Valutazione del Sistema

NeP4B e DBE condividono molte caratteristiche comuni, tra cui:

- ❖ Focus sulle PMI: si vogliono creare un'architettura e dei tool dell'ICT che consentano alle PMI di competere nel mercato globale.
- ❖ Interazioni fra imprese basate su Internet.
- ❖ Architettura peer-to-peer del sistema.
- ❖ Condivisione di conoscenza fra le imprese connesse.
- ❖ Sviluppo e utilizzo di servizi web semantici.

E' pertanto interessante valutare possibili punti di incontro fra tali progetti. In particolare, l'infrastruttura di DBE si basa su un modello P2P puro; questa è molto appropriato per le PMI in quanto, ad esempio, non necessitano in questo modo di un indirizzo IP statico. Inoltre, dal momento che alcune parti di DBE sono basate su tecnologie testate (ad esempio l'Execution Environment è basato su FADA), questo fa sì che DBE fornisca una buona infrastruttura per creare e testare i futuri DBE Service. Questo scenario assume una valenza maggiore considerando che DBE utilizza alcuni linguaggi standard e fornisce (o fornirà) dei tool per generare ed eseguire codice scritto con tali linguaggi. In particolare, DBE utilizza BPEL per la composizione di servizi e implementa un BPEL Engine per la loro esecuzione. Inoltre, DBE utilizza ODM per la definizione delle ontologie di dominio che, seppure non sia un linguaggio standard (è stato implementato all'interno del progetto), è strettamente correlato a OWL-DL: come mostrato in (CHRISTODOULAKS *et al.* 2005), fra i due esiste quasi un mapping 1 a 1. Questa caratteristica ha permesso l'implementazione di tool che consentano di importare ed esportare ontologie nei formati ODM e OWL-DL fra DBE e i sistemi esterni.

Infine, all'interno del progetto DBE sono state sviluppate diverse GUI utilizzate per la modellazione di business e servizi, e di diversi strumenti di sviluppo.

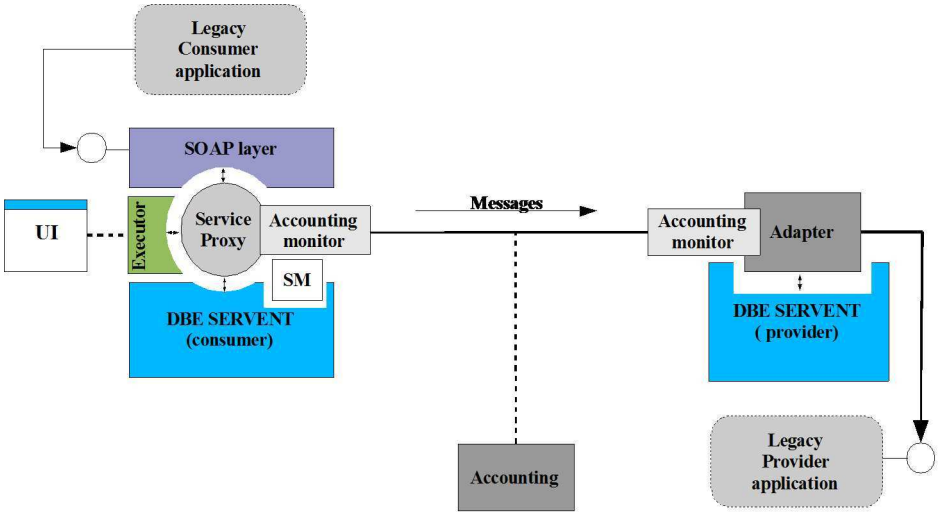


FIGURA 15. Esecuzione di un DBE Service

Per contro, vi sono una serie di limitazioni legate a tool e ai servizi implementati in DBE che restringono l'applicabilità dei questi stessi strumenti in contesti diversi quali appunto NeP4B. Infatti, a causa del fatto che al momento dell'analisi tale progetto è in corso, la quasi totalità dei tool e dei servizi si trova in fase di progettazione o di sviluppo e, pertanto, le versioni rilasciate non sono molto stabili. Dal momento che la conclusione del progetto è prevista per Dicembre 2006-Gennaio 2007, le versioni definitive dei vari tool saranno disponibili solo dopo questo periodo.

Ne consegue che cercare di valutare il sistema in quelli che dovrebbero essere i suoi obbiettivi e caratteristiche finali non è possibile.

Diversi problemi si sono incontrati durante l'utilizzo di DBE. In primo luogo, non esiste uno strumento integrato che permetta di creare servizi, cercali, scaricarli ed eseguirli ma, tuttora, per svolgere queste operazioni è necessario installare diversi strumenti, in particolare il DBE Studio e il ServENT. Inoltre, tali componenti non integrano un supporto di memorizzazione permanente, pertanto è necessario installare ed eseguire separatamente tale componente (che attualmente è costituito da XBD, un database per dati XML) rispetto agli altri. Inoltre, l'installazione del ServENT non è completamente automatica ma è necessario che l'utilizzatore editi in modo opportuno i file di configurazione del servent e di fada, questo anche nell'ultima versione rilasciata. Ne consegue che spesso vi sono dei problemi di integrazione e comunicazione fra i vari tool, in particolare fra DBE Studio e la knowledge base, il che compromette il corretto funzionamento del componente stesso.

Tale sistema non è ancora stato testato al di fuori dei circoli accademici e degli sviluppatori stessi e solo il ServENT è stato presentato alle regioni pilota. Pertanto, non si è in grado di valutare l'effettiva scalabilità del sistema. Questo implica anche che l'ecosistema che DBE si prefigge di realizzare al momento consiste di pochi utenti. Come conseguenza si ha che la creazione dei servizi implica prima un notevole sforzo di modellazione delle ontologie e dei modelli di business più adeguati in quanto è difficile trovarne di utili nell'ecosistema.

Inoltre, molti dei servizi previsti per la versione finale non sono ancora stati rilasciati ma sono in fase di sviluppo. Ad esempio, i servizi di accesso, autorizzazione e autenticazione

non sono disponibili, così come quello di accounting. Una prima versione del DBE Portal è stata inclusa solo nelle ultime versioni del servent (rilasciate in Novembre 2006), pertanto essa presenta solo alcune delle funzionalità finali. Lo stesso discorso vale per EveNet, la cui prima versione, la 0.1.0, è stata rilasciata all'inizio di Dicembre e non è ancora stata opportunamente testata. Analogamente, il BPEL Engine, necessario per eseguire servizi composti, è stato rilasciato solo di recente (versione 0.3.1 del servent), pertanto necessita di essere testato. In generale, tutti i tool sviluppati nel progetto, anche quelli più datati, sono in perenne aggiornamento e pertanto risulta difficile valutarne le caratteristiche. In particolare, il Recommender è uno di quei servizi che si trova sempre in fase di sviluppo: è pertanto difficile riuscire a valutare in modo corretto il servizio di Semantic Discovery.

Allo stato attuale, le implementazioni non sono aperte. DBE ha adottato un approccio Open Source, ma le tecnologie di base sono disponibili solo per Java, utilizzando FADA come rete P2P di base e RMI come protocollo. Quindi, solo le applicazioni Java possono accedere direttamente alle infrastrutture di DBE, mentre rimangono esclusi Web Service implementati attraverso .NET o attraverso altre piattaforme.

Per quanto riguarda i linguaggi utilizzati in DBE, la maggior parte di essi sono stati creati ad-hoc (come BML e SSL), mentre altri derivano da standard ma con cambiamenti (come SDL). Ne consegue che il loro utilizzo al di fuori di DBE è limitato o nullo.

Infine, va notata anche la difficoltà nel trovare delle informazioni tecniche dettagliate riguardo i componenti del sistemi, ma anche questo è da ricollegare principalmente al fatto che il progetto è ancora in fase di sviluppo.

§

Altri limiti relativi agli strumenti sviluppati in DBE sono stati notati durante la fase di modellazione dei servizi. Il principale riguarda la mancanza, nel linguaggio BML, di tipi di dati che definscano delle relazioni uno a molti, come le liste. Questo implica che è possibile inserire, facendo riferimento al modello BML riportato in figura 7, un solo "Course", o un solo "Project", quando si andrà ad istanziare tale modello. Inoltre, le ontologie in DBE vengono utilizzate solo per definire nuovi tipi di dato per gli attributi degli elementi nei modelli definiti. Questo rappresenta una limitazione alla condivisione di conoscenza. Tale approccio, inoltre, differisce da quello di NeP4B dove le ontologie sono direttamente utilizzate per descrivere il dominio di un semantic peer.

In generale, dal punto di vista della semantica, esistono notevoli differenze fra i due approcci. Entrambi utilizzano metadati per descrivere semanticamente i contenuti dei peer, ma NeP4B, contrariamente a quanto accade per DBE, prevede di fare ampio uso di mapping, sia intra- che inter-peer. Questo approccio consente:

- ❖ interoperabilità fra diversi attori attraverso la trasformazione del modello dei dati;
- ❖ trasformazione della struttura dei dati tra diversi peer;
- ❖ possibilità di sviluppare tecniche avante di query processing semantico.

Al contrario, DBE presenta un approccio semantico "leggero": il query processing non sfrutta mapping fra ontologie e lo stesso ODM risulta incompleto per esprimere tali mapping. Infatti, esso fornisce solo costrutti utili per esprimere relazioni estensionali, come

`subClassOf` e `equivalentClass`, ma non definisce costrutti per esprimere relazioni intensionali, quali ipernimia, iponimia e sinonimia. Inoltre, rispetto alla RFP di OMG, l'Ontology Definition Metamodel sviluppato nell'ambito del progetto DBE prevede un solo meta-modello. In particolare, non viene definito il meta-modello per le logiche descrittive, limitando la possibilità di effettuare inferenze sulle ontologie. Tale linguaggio prevede inoltre semplici meccanismi per integrare diverse ontologie (`imports`) e per gestire la loro dinamicità (`priorVersionOf` e `versionInfo`).

CAPITOLO 5

Data Integration negli Ecosistemi Digitali

1. Gestione dei Dati in Ambito P2P

Una delle caratteristiche principali di un ecosistema digitale è la necessità di gestire risorse computazionali e database eterogenei disseminati in una rete peer-to-peer i cui nodi sono altamente dinamici. Imprese (in particolare PMI), organizzazioni, pubbliche amministrazioni e, in generale, ogni tipo di comunità interconnessa necessita di accedere a sistemi informativi distribuiti integrati, i quali continuano a crescere di numero, dimensioni e complessità. Ogni entità dell'ecosistema deve poter accedere ad ogni sorgente informativa e interoperare con altre entità in modo efficiente dal punto di vista delle performance e dei costi. Pertanto, le tematiche di data integration in tali sistemi assumono una rilevanza particolare.

Le attuali reti P2P supportano solo un insieme limitato di metadati. Recentemente sono emerse diverse proposte che vogliono superare questo limite. In (BERNSTEIN *et al.* 2002) si propone una classe di reti P2P detta *schema-based* che combina l'approccio P2P con quello dell'integrazione dei dati e del semantic web. Tali reti sono costruite su peer che usano ontologie per descrivere il loro contenuto e le relazioni semantiche tra concetti di diverse ontologie dei peer. I *Peer Data Management Systems* (*PD MS*) (HALEVY *et al.* 2003) rappresentano un esempio di tali sistemi. In essi ogni nodo può essere una sorgente di dati, un mediatore, o entrambi. Il nodo mediatore è il responsabile dell'integrazione semantica di un insieme di sorgenti informative per derivare lo schema globale delle informazioni acquisite. Questo è l'approccio che sarà seguito nel progetto NeP4B.

Un altro approccio è quello che riguarda il raggruppamento semantico e l'organizzazione dei contenuti nelle reti P2P (THEOTOKIS & SPINELLIS 2004). Vari schemi di raggruppamento sono stati presentati come, quello semantico basato sul contenuto, quello basato sulla località o sulla distanza nella rete. L'approccio *semantic overlay clustering* (LOSER *et al.* 2003) ha l'obbiettivo di creare livelli logici sopra la rete fisica, tramite l'accoppiamento semantico delle informazioni fornite dai peer per raggruppare i nodi basati su super-peer. Questo è l'approccio seguito nel progetto DBE.

Nel seguito vengono introdotte le caratteristiche degli approcci PDMS e semantic overlay cluster.

§

I PDMS sono quei sistemi che si propongono come punto di incontro tra il mondo dei database e quello delle reti P2P. Tali sistemi intendono fornire un'architettura decentralizzata e facilmente estendibile per la gestione dei dati, all'interno della quale ad ogni utente sia

concesso in ogni istante di agire con la massima libertà sui propri contenuti e allo stesso tempo di accedere a contenuti che risiedono su altri peer. I partecipanti ad un sistema di questo tipo sono infatti entità connesse in reti autonome, indipendenti e verosimilmente eterogenee che decidono di condividere il proprio contenuto informativo con gli altri componenti del sistema stesso.

Caratteristiche. In particolare, si tratta di sistemi che si occupano di integrare semanticamente sorgenti informative eterogenee ma, a differenza di quanto accade per i classici sistemi impiegati per questi scopi, sono basati su un'architettura P2P e sono quindi completamente decentralizzati e distribuiti. Inoltre, i partecipanti a questi sistemi sono nodi di uguale importanza e non hanno ruoli predefiniti. Tali nodi sono anche autonomi e indipendenti e risulta quindi necessario per un PDMS non solo essere in grado di gestire la loro dinamicità ma anche la loro *eterogeneità*, in quanto, rispetto ai tradizionali sistemi P2P per la condivisione di contenuti, i PDMS si propongono di fornire *tecniche più ricche per la gestione dei dati*, che permettano di reppresentarne la semantica ed eseguire su di essi interrogazioni complesse.

Questi sistemi possono essere quindi considerati come una sorta di evoluzione dei classici sistemi per l'integrazione dei dati, in cui il singolo schema logico comune viene sostituito da una serie di *mapping semantici* tra i nodi, ovvero da un insieme di relazioni che permettono ad ogni nodo di esprimere le corrispondenze che esistono tra il suo contenuto informativo e quello di alcuni degli altri nodi presenti nella rete. Non impiegando uno schema logico globale, permettono di evitare le onerose attività iniziali di progettazione necessarie alla sua creazione e semplificano notevolmente tutte le operazioni di aggiornamento dovute all'evoluzione del sistema. Ogni peer può inoltre effettuare interrogazioni sull'intero sistema semplicemente utilizzando il proprio schema, senza essere costretto ad adottarne uno globale.

I PDMS si presentano poi come sistemi particolarmente interessanti per la realtà del Semantic Web, che si propone di condividere a livello idealmente universale dati semanticamente arricchiti: la quantità di ontologie e schemi che vengono in questo contesto coinvolti è talmente grande da rendere assolutamente inadeguate l'adozione di strutture centralizzate che devono essere definite a priori.

Problematiche. La realizzazione di un sistema per la gestione dei dati in ambito P2P comporta numerose difficoltà che derivano principalmente dalle carenza che tali architetture mostrano in materia di capacità di amministrazione dei contenuti e della loro semantica. Si possono individuare quattro macro-problematiche progettuali fondamentali:

- ❖ RAPPRESENTAZIONE DEI DATI. Le sorgenti informative che partecipano al sistema sono entità autonome e indipendenti, è perciò verosimile supporre che esse adottino modalità diverse per rappresentare i propri dati ed occorre quindi definire dei linguaggi comuni. Molti *ontology languages* sono presenti in lettura, tra cui ODL_{I^3} (BERGAMASCHI *et al.* 1998), *OWL* (MCGUINNESS & VAN HARMELEN 2004), *RDF* (MANOLA & MILLER 2004), *UML* (OMG 2006B) e *ODM* (DE GIACOMO *et al.* 2004; OMG 2003C). I requisiti che un linguaggio completo per la descrizione di ontologie deve soddisfare sono i seguenti.
  - Il linguaggio deve esprimere la conoscenza di dominio. Esistono molti tipi diversi di possibili informazioni riguardo un dominio, che possono essere catalogate come: concetti/classi/entità, attributi/slot, facet, tassonomie, relazioni, assiomi, istanze/individi/fatti, regole. Inoltre, un ontology language

deve includere operatori per gestire le istanze e deve fornire primitive per la definizione di relazioni di equivalenza e non equivalenza e di restrizioni sulle proprietà. Infine, devono permettere la rappresentazione di conoscenza intensionale ed estensionale.
- Devono fornire qualche meccanismo di inferenza. Ciò significa stabilire come la struttura statica rappresentata nella conoscenza del dominio può essere sfruttata per eseguire dei processi di ragionamento.
- Devono fornire meccanismi che permettano il riutilizzo e l'integrazione di ontologie già sviluppate. Inolte, devono fornire primitive per il mapping di concetti appartenenti a ontologie diverse.
- Dal momento che le ontologie possono evolvere nel tempo, il linguaggio deve fornire meccanismi per gestire tale dinamicità.
- Al fine di facilitare il processo di modellazione e di comprensione delle ontologie, i linguaggi devono avere un supporto grafico.
- Le ontologie possono essere descritte con lingue diverse, pertanto il linguaggio deve fornire un supporto al multilinguismo, ad esempio attraverso la lessicalizzazione dei termini o l'annotazione del termine con un'ontologia lessicale di riferimento (ad esempio WordNet).

❖ CONNESSIONE DEI DATI. A causa dell'eterogeneità delle sorgenti informative che compongono il sistema, occorre un meccanismo di integrazione decentralizzato che permetta ai peer di condividere dati e cooperare. Nella maggior parte dei casi si ricorre all'impiego di collegamenti semantici locali, detti *mapping*, che vengono stabiliti tra i vari peer. Importanti decisioni relative a tali collegamenti riguardano il loro processo di creazione, i formalismi/linguaggi che vengono impiegati per rappresentarli e la scelta dei criteri che permettono di stabilire quali peer collegare tra loro.
Per quanto riguarda il linguaggi di mapping, la maggior parte di essi rappresenta i mapping in due diversi modi: come *tuple* o come *regole*. Nel primo tipi di linguaggi, il mapping è rappresentato come una tupla di 5 elementi, $\langle$`id, e, e', R, n`$\rangle$, dove `id` è l'identificatore della corrispondenza, `e` è un elemento appartenente ad una ontologia `o`, `e'` è l'elemento appartenente ad una ontologia `o'` diversa dalla prima, `R` è la relazione che lega i due elementi `e` `n` denota un indice di confidenza per questo mapping.
Il secondo tipo di linguaggi considera un insieme di espressioni, di un particolare linguaggio `L`, con variabili incluse in esse. Le corrispondenze tra concetti sono pertanto espresse attraverso clausole come quella sottostante, nella quale le variabili nella parte sinistra sono quantificate universalmente sull'intera formula e quelle nella parte destra sono quantificate esistenzialmente:

$$\forall \overline{x_f}(f \implies \exists \overline{x_g} g)$$

❖ RISOLUZIONE DELLE INTERROGAZIONI. Il sistema deve risolvere le interrogazioni fornendo tutte le informazioni utili contenute all'interno dei diversi nodi ed occorre quindi che le query inizialmente poste ad un peer vengano opportunamente riformulate per essere propagate anche agli altri. Al termine del processo occorre poi fondere i risultati in un'unica risposta da presentare al richiedente. Tutte queste operazioni risultano particolarmente critiche per le prestazioni del sistema e la sua efficienza poichè la maggior parte delle informazioni ad esse necessarie devono essere desunte a tempo di esecuzione.

Esistono due tipi principali di processi di riformulazione, ovvero l'*unfolding* e il *rewriting*. L'utilizzo di un algoritmo piuttosto che un altro dipende dalla direzione del mapping disponibile. Ad esempio, dati due peer P_2 e P_1, se il mapping definisce P_1 come una query su P_2 ($M_{1,2}$), allora le riformulazione consiste in un processo di unfolding. L'unfolding dei mapping è un processo simile a quelli di riformulazione delle query nei sistemi di information integration GAV-based (BERGAMASCHI *et al.* 2001). Nel caso in cui il mapping sia definito nella direzione opposta, cioè $M_{2,1}$, allora la query deve essere riscritta utilizzando tale mapping. Tale processo è simile a quello adottato dai sistemi di data integration basati su un approccio LAV.

- RICERCA DEI DATI. Siccome un'architettura P2P non prevede l'impiego di alcun controllore globale, occorre disporre di un meccanismo decentralizzato che permetta la localizzazione e il recupero dei dati. Si tratta di una problematica strettamente collegata alla risoluzione delle interrogazioni ma riguarda trasversalmente molti altri aspetti, come la collocazione dei dati e la gestione dell'evoluzione del sistema. Alcuni dei lavori presenti in letteratura ricercano i documenti in base al loro identificatore (id-based). Molti altri lavori utilizzano tecniche di information retrieval e machine learning: in particolare utilizzano misure riguardo statistiche sulle parole chiavi, sulla probabilità delle parole chiavi di comparire in un documento, ecc.

I vari sistemi esistenti attualmente, brevemente introdotti di seguito, propongono soluzioni progettuali differenti a questi quesiti, alcune delle quali sono completamente innovative, mentre altre derivano da ambiti diversi e possono però essere adattate al contesto P2P.

Panoramica sui sistemi esistenti. Attualmente esistono diversi progetti di ricerca che si concentrano sullo sviluppo di sistemi per la gestione di dati in ambito P2P e nel seguito viene presentata una breve descrizione di alcuni di questi progetti.

Piazza. *Piazza* (HALEVY *et al.* 2003; HALEVY *et al.* 2004) è un progetto condotto dall'Università di Washington con lo scopo di realizzare un risistema distribuito facilmente estendibile per la condivisione decentralizzata e scalabile dei dati di un vasto numero di sorgenti eterogenee.

L'architettura di Piazza è costituita da una serie di nodi peer connessi in rete, ognuno dei quali può essere una sorgente informativa, un mediatore o entrambi. Ogni peer ha un proprio schema (peer schema) che rappresenta il dominio di interesse contenente le relazioni (peer relation) che connettono tra loro gli elementi dello schema. Ogni peer può inoltre (non obbligatoriamente) fornire al sistema dei dati effettivi, memorizzati in forma di stored relation, che li collegano al peer schema corrispondente e mantiene alcuni mapping semantici locali che specificano le corrispondenza tra il suo schema e quello dei nodi vicini.

Il meccanismo di risoluzione delle query prevede la presenza su ogni nodo di due moduli principali:

- *Query riformulation engine*, che, presa in ingresso una query posta ad un determinato peer, impiega i mapping semantici disponibili per ottenere in uscita una serie di query equivalenti da sottoporre ad altri nodi;
- *Query evaluation engine*, che si occupa invece dell'effettiva valutazione e risoluzione delle query.

Ogni interrogazione che viene posta ad un nodo del sistema viene infatti riformulata sulla base dei suoi mapping semantici per essere trasferita dal nodo iniziale ai suoi vicini ed essere presso di loro valutata. Tale processo si ripete ricorsivamente e in questo modo è possibile ottenere una risposta contenente tutti i dati del sistema provenienti dai peer collegati da un processo semantico che viene infine ridiretta al nodo richiedente.

Hyperion. Si tratta di un progetto (ARENAS *et al.* 2003) condotto dalle Università di Toronto e Ottawa con lo scopo di realizzare un'architettura per un PDMS basata su una rete di nodi peer in grado di coordinarsi a runtime nell'esecuzione delle classiche operazioni tipiche dei DBMS.

Alla base del progetto sta infatti l'idea di concepire il PDMS come un convenzionale DBMS al quale viene aggiunto un nuovo strato software (P2P layer) che ha il compito di implementare le funzionalità che sono richieste dai peer per condividere e coordinare i dati senza compromettere in alcun modo la loro autonomia. Questo strato si occupa di stabilire e gestire a runtime e in maniera semi-automatica, le relazioni tra i peer, costruendo in tal modo una rete P2P logica.

La condivisione dai dati è attuata mediante l'impiego di mapping table e mapping expression, che permettono di definire le corrispondenze che esistono tra valori memorizzati in peer differenti.

Dal punto di vista architetturale, il sistema risulta composto da una serie di peer che sono database relazionali locali e che stabiliscono tra di loro dei legami in modo da definire una rete P2P. Ogni peer a sua volta è composto da un'interfaccia (*P2P API*) che consente agli utilizzatori di accedere al sistema, un *DBMS* dove vengono contenuti i dati locali, la mapping table e le mapping expression, e un *P2P layer*.

Un P2P layer risulta composto da tre moduli:

- *Acquaintance Manager* (*AM*). Permette di creare in modo semi-automatico i mapping tra peer, espressi attraverso mapping table e mapping expression.
- *Query Manager* (*QM*). Gestisce l'esecuzione delle query che sono poste al sistema dai suoi utilizzatori. Nel sistema si distinguono due tipi di query: *query locali*, risolte considerando solo i dati del peer locale, e *query globali*, che devono essere riformulate e inviate ai nodi peer vicini.
- *Rule Manager* (*RM*). Garantisce il rispetto delle politiche di consistenza definite tra i peer per gestire le possibili evoluzioni e modifiche del sistema.

Edutella. Si tratta di un progetto (NEJDL *et al.* 2003) condotto dalle Università di Hannover Karlshure con lo scopo di sviluppare un'infrastruttura in grado di agevolare l'interoperabilità e la condivisione delle risorse nelle reti P2P. L'idea base è di combinare reti P2P e metadati che descrivono le risorse amministrate dai singoli peer, in modo da consentire ad applicazioni eterogenee di interagire e cooperare, secondo i principi propri del Semantic Web.

L'architettura di Edutella è basata sull'approccio a super-peer ed è costruita su RDF. I componenti previsti sono:

- *Edutella Services*. Si preoccupa di definire i formati e i protocolli da impiegare nello scambio dei dati.
- *Edutella Peers*. Implementa tutti i servizi necessari al funzionamento del sistema, tra i quali:

- *Annotation Service*. Si occupa del processo di annotazione di tutto il materiale memorizzato nel sistema e fornisce a tale scopo un meccanismo di visualizzazione dei vari documenti che permette di estrarne i metadati.
- *Mapping Service*. Si occupa di attuare le traduzioni tra vocabolari differenti di metadati necessarie per consentire l'interoperabilità tra i vari peer.
- *Mediation Service*. Si occupa di fornire viste coerenti per dati che appartengono a sorgenti diverse, risolvendo i problemi legati all'eventuale presenza di informazioni ridondanti e in conflitto.

❖ *Query Service*. Definisce e standardizza i metodi per porre le interrogazioni e per risolverle.

Edutella presenta una architettura parzialmente centralizzata in quanto i nodi super-peer si comportano da sistemi a mediatore per il sottoinsieme di peer a loro collegati. Questi super-peer si occupano di integrare i metadati e fondere i dati provenienti dalle varie sorgenti risolvendo gli eventuali conflitti, ma agiscono anche da centri di registrazione e distribuzione delle query. In base alle registrazioni effettuate, un super-peer è in grado di selezionare i peer rilevanti per rispondere ad una query.

Sewasie. Si tratta di un progetto europeo (BERGAMASCHI *et al.* 2003; BENEVENTANO *et al.* 2005) coordinato dall'Università di Modena e Reggio Emilia il cui scopo è quello di studiare le problematiche di gestione della conoscenza e di ricerca delle informazioni in un contesto P2P basato su sorgenti informative web.

L'approccio proposto combina quello schema-based con quello super-peer, formando una rete schema-based super-peer organizzata in una architettura che prevede due livelli: quello inferiore, detto *peer level*, e quello superiore, chiamato *super-peer level*. In particolare:

❖ Un *peer* contiene un sistema di integrazione dei dati responsabile dell'integrazione di sorgenti informative eterogenee in una *ontologia* composta da un *Global Virtual View* (GVV) annotata e da *mapping* (di primo livello) verso gli schemi delle sorgenti.

❖ Un *super-peer* contiene anch'esso un sistema di integrazione dei dati, il quale integra le GVV dei suoi peer in una *ontologia* composta da una GVV delle GVV dei peer integrati e da mapping a tali GVV.

In specifico, l'architettura di SEWASIE (figura 1) è basata su diversi agenti, presentati di seguito:

❖ *SEWASIE Information Node* (*SINode*). Sono sistemi a mediatori, basati su MOMIS, che forniscono una vista integrata di sorgenti informative eterogenee. Ogni SINode è descritto da una ontologia definita in linguaggio ODL_{I^3} ed è composto da:
- *Virtual Data Store* (*VDS*): rappresenta una vista virtuale di tutte le informazioni gestite dal SINode e consiste delle sorgenti informative gestite, dei wrapper e del metadata repository.
- *Ontology Builder* (*OB*): fornisce le funzionalità per creare e mantenere la GVV che descrive il SINode e i mapping fra GVV e sorgenti locali.
- *Query Manager* (*QM*): è l'insieme di funzioni che permettono di riformulare la query (*unfolding*) posta sulla GVV in più query valide per le sorgenti informative, di porre tali query ai wrapper delle rispettive sorgenti, collezionare i risultati, effettuare qualche ulteriore filtraggio sull'insieme dei risultati e, infine, presentare la risposta all'agente richiedente.

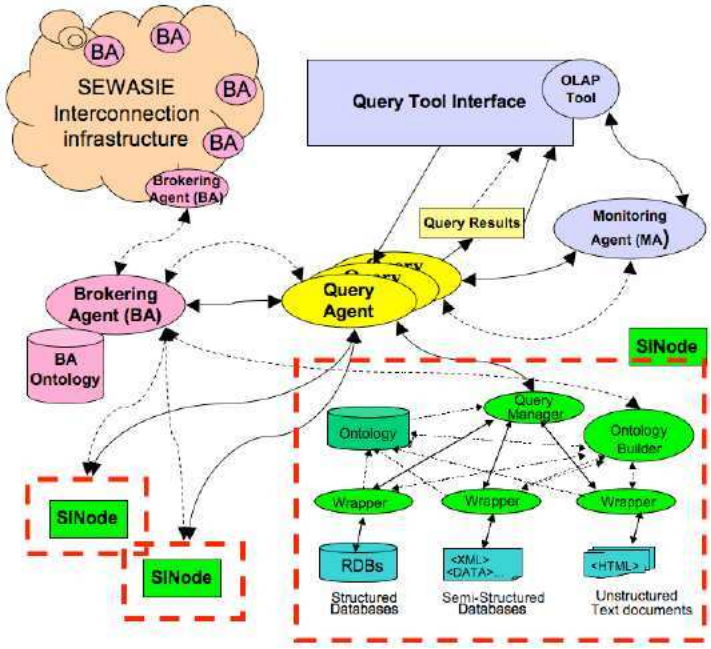


FIGURA 1. Architettura ad agenti di SEWASIE

- *Brokering Agent* (*BA*). Sono nodi peer di reti che condividono conoscenza e metadati. La conoscenza di ogni BA è rappresentata come un'ontologia (*BA ontology*), la quale rappresenta una vista integrata di più GVV, e da mapping che relazionano i concetti nelle GVV con quelli della BA ontology. Sia l'ontologia che i mapping sono espressi per mezzo di un linguaggio formale, che anche in questo caso è rappresentato da ODL_{I^3}.
- *Monitory Agents* (*MA*). Sono componenti utilizzati per memorizzare, in un repository permanente, una risposta ad una certa query. MA è quindi responsabile di ripetere periodicamente la query per osservare eventuali cambiamenti nel risultato. Nel caso in cui certi documenti risultano modificati, MA ha il compito di avvisare l'utente.
- *Querying Agent* (*QA*). Si occupa della gestione delle query poste al sistema dagli utenti. In particolare, tale agente, data una query in ingresso, è in grado di: porre la query e le parti rilevanti dell'ontologia dell'utente al sistema; definire il query plan, riscrivere la query per uno specifico SINode, e fondere i risultati provenienti da SINode diversi; processare l'informazione data dal BA e identificare i SINode da accedere per rispondere alla query; riportare al richiedente la risposta alla query, contenente sia i dati che i metadati.

Come si è potuto notare, avendo due livelli di mapping, nella fase di query processing è necessario effettuare due riformulazioni delle query, ovvero la riformulazione rispetto la BA ontology e la riformulazione rispetto la GVV del SINode. Tali processi sono simili e sono costituiti da:

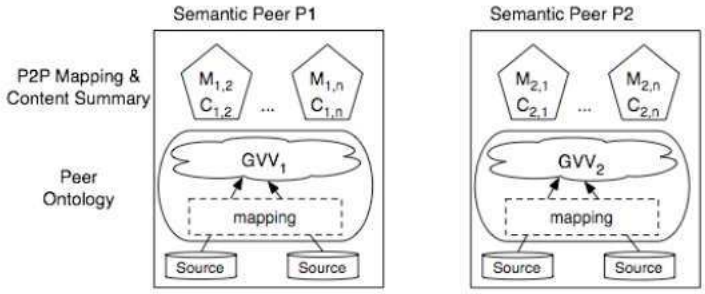


FIGURA 2. Rete di peer semantici in WISDOM

- *Query expansion*: la query posta su una ontologia (BA ontology o SINode ontology) viene espansa in modo da considerare tutti i vincoli espliciti ed impliciti nell'ontologia.
- *Query unfolding*: gli atomi nella query espansa sono riformulati, in termini di unfolding, in base ai mapping definiti tra l'ontologia sulla quale la query è stata posta e le ontologie di livello inferiore (SINode ontolgoy e lo schema delle sorgenti locali).

Wisdom. Anche WISDOM (BERGAMASCHI *et al.* 2005) è un progetto coordinato dal DataBase Group dell'Università di Modena e Reggio Emilia il cui obbiettivo è quello di studiare, sviluppare e sperimentare metodi e tecniche per cercare e interrogare sorgenti informative disponibili nel Web, che si assume siano sia indipendenti le une dalle altre, ovvero che presentano delle eterogeneità a diversi livelli di astrazione (di formato, di logica e di semantica). Ciò che caratterizza L'Approccio è l'adozione di una architettura distribuita basata sul paradigma P2P e sull'adozione di *ontologie di dominio*. Come per SEWASIE, l'integrazione delle sorgenti è separata in due livelli: il livello inferiore opera e gestisce una integrazione forte del contenuto delle sorgenti informative formando un *semantic peer* descritto da una *ontologia*; a livello superiore si effettua una integrazione meno stretta tra le informazione offerte dai semantic peer della rete.

In generale, WISDOM rappresenta un evoluzione architetturale rispetto il progetto SEWASIE in quanto possiede le seguenti caratteristiche:

- l'estrazione dei dati da Web Site di grandi dimensioni attraverso opportuni wrapper;
- si fornisce una rete di peer semantici attraverso una integrazione lasca dell'informazione fornita dai peer stessi;
- permette di scoprire i mapping fra le ontologie dei semantic peer;
- query processing peer-to-peer eseguito tramite i mapping inter-peer.

Come SEWASIE, anche l'architettura di WISDOM segue l'approccio di rete super-peer schema-based. Una rete di peer semantici (figura 2) si viene a costituire per mezzo dei mapping inter-peer: ciò consente di recuperare informazioni anche al di fuori del peer che ha ricevuto la query.

Pertanto, uno dei componenti principali di questa architettura è rappresentato dal *Semantic Peer*. Ogni semantic peer contiene un sistema di integrazione intelligente delle informazioni, basato sul sistema MOMIS, per integrare sorgenti di dati eterogenee, che nel contesto di WISDOM si assume siano sorgenti Web. Il risultato di questa operazione è la GVV del peer. Per poter accedere alle sorgenti informativi, WISDOM prevede l'impiego di *wrapper* associati alle sorgenti i quali forniscono uno schema logico attraverso il quale i livelli superiori dell'architettura possono porre le loro query. Per una singola pagina, un wrapper consiste di un programma che estrae i dati dalla pagina HTML e li organizza in un formato più strutturato, come XML. Per un sito Web, un wrapper offre una vista uniforme e strutturata dei dati pubblicati nel sito.

Al fine di costruire la rete i peer semantici, è necessario associare un'ontologia ad ogni peer. Tale ontologia viene costruita partendo dalla GVV del peer; in particolare, si arriva ad una *peer ontology* arricchendo la GVV di nuovi concetti (classi, attributi, relazioni), integrandola con ontologie già esistenti o modificando i suoi concetti al fine di elevarli ad un livello di astrazione maggiore.

Una volta costruite le ontologie, esse vengono interconnesse per mezzo di *peer-to-peer mapping*: un mapping semantico peer-to-peer,$M_{i,j}$, è una relazione tra l'ontologia Ont_j del peer P_j e l'ontologia Ont_i del peer P_i. Attraverso tali mapping P2P, una query ricevuta da un peer può essere idealmente estesa ad ogni peer per il quale viene definito un mapping. Sempre in via ipotetica potrebbe comunque non essere conveniente propagare la query nella maniera appena descritta, soprattutto se il peer target ha una limitata estensione per i concetti inclusi nella query. Pertanto, ad ogni mapping viene associato un *content summary*, il quale fornisce informazioni quantitative riguardo l'estensione dei concetti coinvolti nella definizione del mapping. Sulla base di tale informazione, si è quindi in grado di valutare la convenienza della propagazione al peer target del mapping.

§

In questa parte verrano introdotte le caratteristiche dei sistemi P2P che effettuano un raggruppamento semantico delle informazioni, ed in particolare dei semantic overlay cluster.

La caratteristica di questo approccio è quella di creare delle "peer communities", simili alle comunità reali tra umani, nelle quali l'appartenenza dipende dalle relazioni tra peer che condividono interessi comuni. Queste comunità possono essere utilizzate per effettuare ricerche più efficienti e produrre risultati di qualità migliore. Il raggruppamento è spesso implementato attraverso un insieme di attributi: le comunità sono costruite tra peer che condividono attributi simili.

L'approccio semantic overlay clustering, basato su una rete parzialmente centralizzata di tipi super-peer (LOSER *et al.* 2003) ha lo scopo di creare un livello logico sopra la rete fisica, raggruppando i peer con informazioni semanticamente correlate in cluster. Pertanto ogni cluster contiene un insieme di peer associati ai loro corrispondenti super-peer.

Un esempio di questo tipo di approccio è presentato di seguito.

Stanford Peers. Presso l'Università di Stanford è stato sviluppato un progetto (CRESPO & GARCIA-MOLINA 2003) che propone un'organizzazione flessibile e innovativa per le reti dei sistemi P2P in cui le connessioni tra i nodi sono influenzate da loro contenuto. In particolare questo sistema prevede che i collegamenti vengano stabiliti tra i nodi che contengono dati semanticamente simili: in questo modo si vengono a formare più *Semantic Overlay Network* (*SON*). All'interno delle singole SON viene poi definita un'organizzazione gerarchica sulla base di come si specializzano i vari argomenti. Il risultato è quindi una rete di super-peer in cu è possibile indirizzare la query solamente verso le SON che trattano argomenti a loro correlati, aumentando in tal modo la probabilità di ottenere i dati desiderati velocemente e riducendo allo stesso tempo il carico sui nodi che hanno contenuti non correlati.

In tale sistema, il processo di costruzione delle SON si articola in tre fasi:

- Si sceglie, sulla base dei dati effettivamente distribuiti nel sistema, la gerarchia da impiegare per classificare i concetti che definiscono le varie SON.
- Ogni nodo che si connette al sistema esegue un *document classifier*, che associa un concetto della gerarchia ad ognuno dei suoi documenti.
- Infine il nodo esegue un *node classifier* che, sulla base dei risultati prodotti dal document classifier, lo assegna alle specifiche SON.

Per quanto riguarda il processo di risoluzione delle query, il sistema prevede che esse vengano sottoposte ad un opportuno classificatore in modo da individuare le SON da contattare. A questo punto i nodi appropriati possono essere selezionati dai super-peer delle singole SON.

2. Data Management in NeP4B

Analogamente a molti progetti presentati in precedenza, l'approccio seguito per il progetto NeP4B è quello PDMS, dove ogni peer semantico non è una semplice sorgente di dati ma è un mediatore, che integra sorgenti di dati eterogeni includendo documenti testuali e multimediali. L'insieme dei dati integrati dal peer sono messi a disposizione della rete P2P tramite un'ontologia, la *Peer Virtual View* (*PVV*), che rappresenta il dominio di interesse del peer.

Il problema dell'interoperabilità semantica dei peer viene risolto attraverso la definizione di *mapping peer-to-peer*, ovvero mapping definiti fra concetti appartenenti a PVV diversi; atraverso tali mapping sarà infatti possibile effettuare riformulare la query di un peer sugli altri peer.

In generale, le caratteristiche che si vogliono sviluppare riguardo la gestione dei dati nel sistema NeP4B sono riassunte di seguito:

- Approccio schema-based su una architettura P2P.
- Approccio PDMS: ogni peer rappresenta un mediatore, ovvero integra sorgenti di dati eterogenei (peer semantico).
- Utilizzo di ontologie di dominio per la descrizione delle informazioni gestite da un peer semantico.
- Definizione di mapping peer-to-peer per connettere concetti appartenenti a peer diversi.

- ❖ Query processing peer-to-peer basato sulla riformulazione delle query; tale riformulazione può essere eseguita in termini di unfolding o rewriting, in base alla direzione dei mapping inter-peer.
- ❖ Utilizzo di un vasto insieme di linguaggi per catturare la conoscenza delle PMI; tra di essi i principali sono *XML*, ODL_{I^3}, *OWL*, *RDF* e *WSMO*.

3. DBE Usage Scenario per il Progetto NeP4B

In questo scenario i tool offerti da DBE e la sua infrastruttura P2P di comunicazione vengono utilizzati per sviluppare gli opportuni servizi al fine di creare un sistema di data integration basato su un'architettura peer-to-peer, dove ogni peer costituisce un peer semantico.

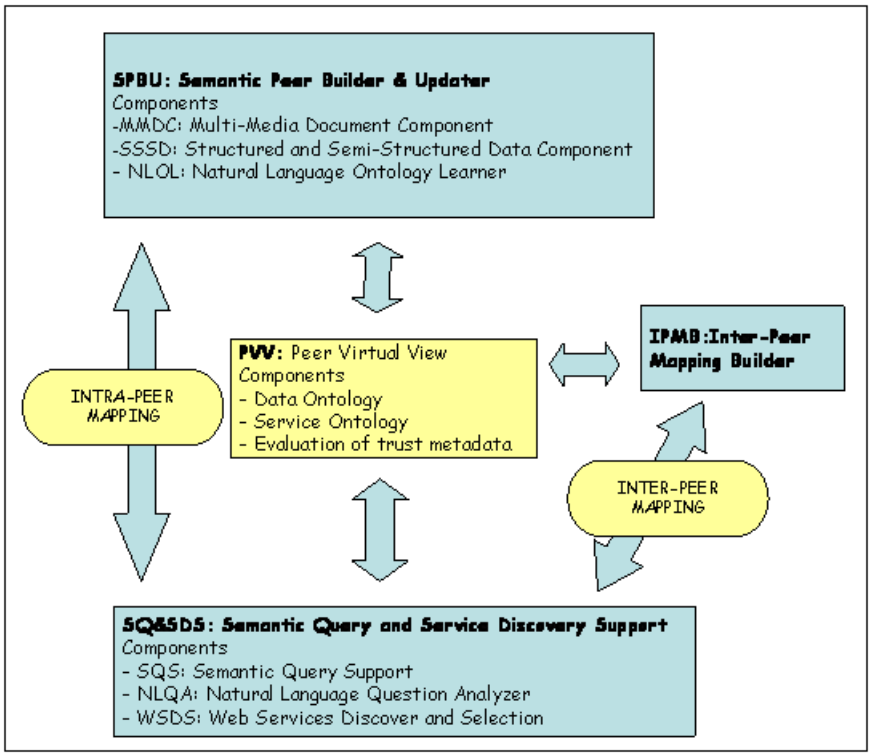


FIGURA 3. Organizzazione di un singolo peer semantico in NeP4B

Per poter descrivere in modo corretto lo scenario, è necessario identificare gli attori coinvolti. In base all'organizzazione di un singolo peer di NeP4B, riportata in figura 3, è possibile identificare i seguenti soggetti:

- ❖ *Utente generico*: rappresenta chiunque richieda le funzionalità offerte dal sistema. Può essere dunque una applicazione software, una PMI o una singola persona.

- ❖ *Sorgenti locali*: rappesentano le sorgenti informative che ogni PMI possiede. Tutte le sorgenti sono descritte da schemi, i quali possono presentare eterogeneità.
- ❖ *Semantic Peer Builder & Updater* (*SPBU*): rappresenta quel componente che estrapola conoscenza delle sorgenti locali e le integra creando un peer semantico. Inoltre, tale soggetto (che è un componente architetturale di NeP4B) ha anche il compito di mantenere aggiornato l'ontologia che descrive il peer semantico
- ❖ *Peer semantici*: rappresentano un'integrazione di due o più sorgenti locali. Ogni peer è descritto da una ontologia, che in NeP4B prende il nome di *Peer Virtual View* (*PVV*), definita con in linguaggio comune agli altri peer, e da un insieme di mapping (intra-peer) tra i concetti (globali) della PVV e i corrispondenti concetti (locali) dello schema delle sorgenti. Si nota che non necessariamente un peer semantico corrisponde ad una PMI.
- ❖ *Inter-Peer Mapping Builder* (*IPMB*): rappresenta quell'attore responsabile della definizione e della scoperta di associazioni fra diversi peer semantici, ovvero fra elementi appartenenti a PVV distinte.
- ❖ *Semantic Query Support* (*SQS*): è quel componente che fornisce le funzionalità per rispondere all'interrogazioni poste su un peer semantico.

Il sistema deve essere in grado di gestire diversi casi di'uso, tra cui: creazione e mantenimento dei peer semantici, definizione e/o scoperta di associazioni semantiche tra peer (mapping inter-peer) ed esecuzione di query che sfruttino tali associazioni per poter seguire dei percorsi semantici per riformulare le interrogazioni in un peer in interrogazioni ai suoi vicini.

Creazione dei peer semantici. Gli attori principali coinvolti in tale fase sono le sorgenti locali e SPBU. In particolare, per ogni sorgente locale, SPBU deve essere in grado di recuperarne lo schema e di identificare le corrispondeze semantiche tra elementi appartenenti a schemi diversi (mapping intra-peer). Il processo di creazione di un peer semantico è composto dei seguenti 5 step:

(1) SPBU contatta la sorgente locale da integrare, richiedendole lo schema che ne descrive il contenuto;
(2) La sorgente estrae lo schema. Tale schema può essere memorizzato localmente oppure deve essere estratto dalla sorgente; in tal caso è necessario anche tradurlo secondo il formalismo adottato da SPBU per la definizione delle ontologie.
(3) Successivamente, la sorgente risponde a SPBU inviandole la descrizione del proprio contenuto.
(4) SPBU cerca le corrispondenze semantiche fra diversi schemi o, meglio, inserisce i concetti dello schema ricevuto all'interno del common thesaurus. Conseguentemente viene generata la PVV.
(5) Tale ontologia viene memorizzata in un opportuno repository.

Il sistema deve inoltre prevedere azioni di aggiornamento della PVV; è infatti possibile che gli schemi delle sorgenti locali subiscano delle modifiche. In tal caso SPBU deve prontamente effettuare i dovuti aggiornamenti sui mapping inter-peer e, conseguentemente, sulla struttura della PVV.

Definizione/scoperta di mapping inter-peer. In questa fase, un peer deve compiere tutte le azioni che gli permettono di scoprire nuove relazioni semantica con altri peer. Ogni relazione scoperte con un nuovo peer permette di definire un mapping tra i concetti delle

ontologie che ne descrivono i contenuti. Ogni nuovo mapping permette dunque di definire per un peer semantico un nuovo “vicino”. Pertanto, le operazioni da intraprendere per definire nuovi mapping fra peer sono le seguenti:

(1) Il peer semantico contatta l’IPMB fornendogli la propria PVV.
(2) IPMB ricerca altre ontologie (PVV) di altri peer presenti nel sistema.
(3) Per ogni PVV trovata, IPMB estrapola i possibili mapping con la PVV del peer richiedente.
(4) Successivamente IPMB comunica al peer richiedente i nuovi mapping creati.
(5) Il peer semantico aggiorna i suoi mapping in base alle notifiche ricevute da IPMB.

Esecuzione delle query. In questa fase i mapping inter-peer vengono sfruttati per eseguire una query, inizialmente posta su un peer, anche sui peer vicini ad esso. Infatti, in NeP4B quando un utente sottomette una interrogazione, il componenete SQS del peer s’incarica di risolverla nella rete semantica; più precisamente la richiesta è risolta all’interno del suo nodo e poi propagata alla rete. Infine i risultati vengono raccolti e mostrati all’utente. La cosa importante da notare in questa fase è che per mezzo delle associazioni semantiche tra peer, una richiesta su un peer può ottenere dati rilevanti da qualunque peer accessibile nella rete. Pertanto, si distinguono le seguenti fasi operative:

(1) Un utente generico, interfacciato ad un peer semantico, pone una query al sistema.
(2) Il peer semantico contatta il SQS inviandogli la query e i mapping, sia inter-peer che intra-peer.
(3) SQS riformula la query rispetto i mapping intra-peer ed invia le interrogazioni riformulate alle rispettive sorgenti locali. Nel frattempo riformula la query secondo i mapping con i peer vicini e gli invia le query. Infine, SQS si pone in attesa di ricevere i risultati.
(4) Le sorgenti locali e i peer vicini ricevono le query e le eseguono localmente. Per quanto riguarda le query poste su altri peer, esse vengono computate seguendo i passi 1-4 ora descritti. In questo modo la query originaria viene propagata nella rete anche a quei nodi non direttamente collegati a quello iniziale.
(5) I risultati vengono man mano raccolti e organizzati ed infine inviati all’utente richiedente.

La figura 4 mostra una schematizzazione di tali casi d’uso.

§

Viene ora illustrata l’implementazione dello scenario con DBE. L’idea è quella di progettare dei servizi DBE che implementino le funzionalità richieste dal sistema, ossia di creazione dei peer semantici, dei mapping inter-peer e di query processing sui nodi della rete.

Specifiche dei requisiti. Di seguito vengono discussi i requisiti per ogni caso d’uso presentato in precedenza.

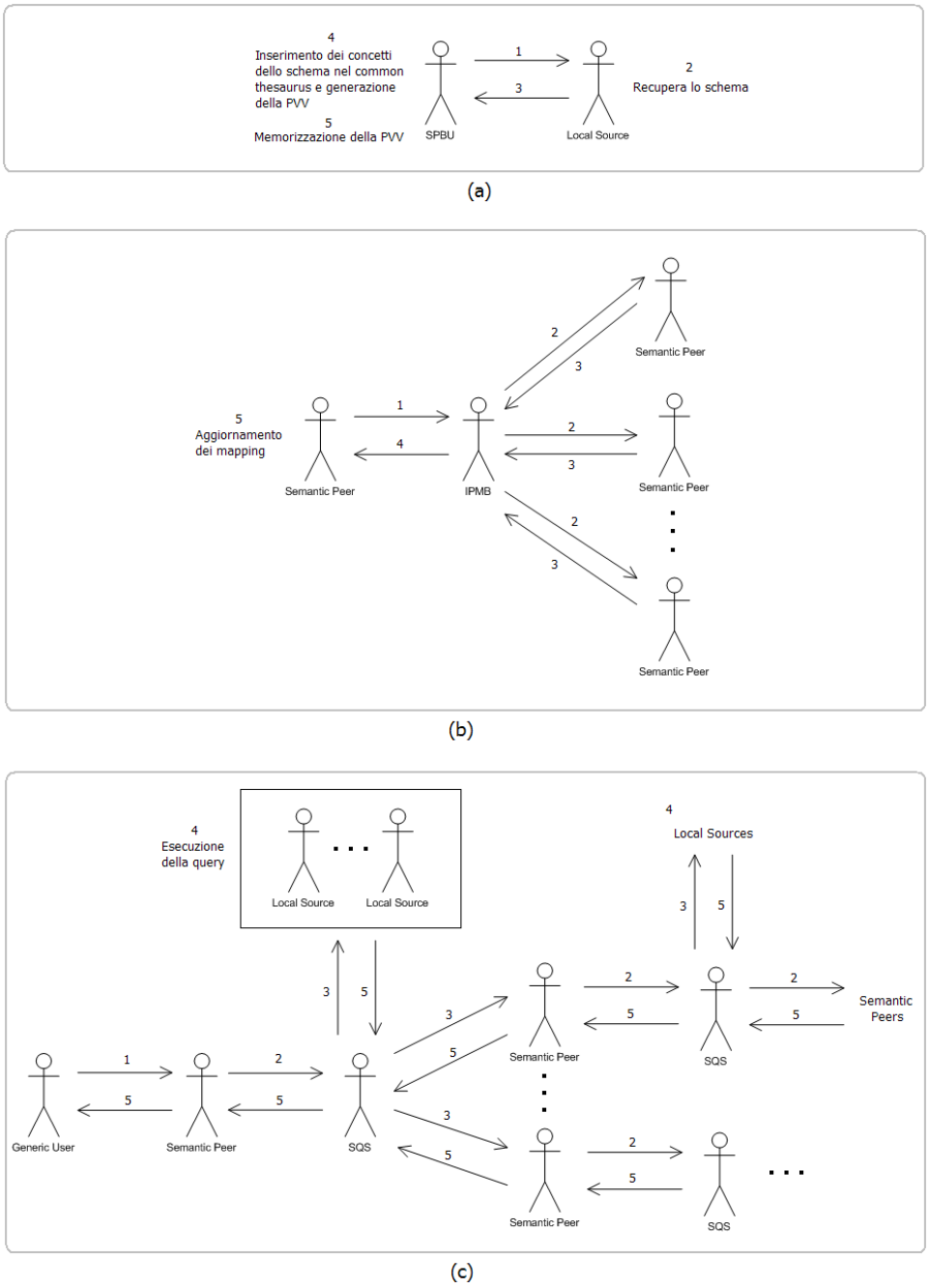


FIGURA 4. Schematizzazione delle funzionalità relative alla (a) creazione dei peer semantici, alla (b) scoperta dei peer-to-peer mapping e al (c) query processing

Creazione dei peer semantici.

(1) SPBU deve essere in grado di cercare ed accedere alle sorgenti dati.
(2) La sorgente deve essere in grado di estrarre lo schema che la descrive, tradurlo nel formato utilizzato da SPBU e inviarlo al richiedente (SPBU).
(3) La sorgente deve anche essere in grado di annotare il proprio schema con i concetti offerti dall'ontologia lessicale WordNet.
(4) SPBU deve essere in grado di memorizzare la PVV.
(5) SPBU deve mantere un collegamento con le sorgenti locali che integra.

Scoperta di mapping inter-peer.

(1) SPBU deve essere in grado di contattare IPMB inviandogli la propria PVV.
(2) A sua volta, IPMB deve essere in grado di cercare altre GVV all'interno della rete.
(3) IPMB deve essere in grado di memorizzare i mapping trovati e di mantenere un collegamento ai peer semantici collegati ad esso.
(4) Per poter esprimere i mapping è necessario un linguaggio capace di modellare le relazioni semantiche che possono intercorrere tra concetti appartenenti a ontologie diverse.

Esecuzione delle query.

(1) Ogni peer semantico deve riuscire a contattare le sorgenti locali per fornigli la query da processare.
(2) Le sorgenti locali devono tradurre la query ricevute nel linguaggio della sorgente stessa, eseguire la query e ritornarla al peer richiedente.
(3) I peer semantici devono poter contattare i SQS inviandogli la query e i mapping.
(4) SQS deve essere in grado di tradurre le query in base ai mapping inter-peer e intra-peer.
(5) SQS deve riuscire a contattare altri peer semantici per inviargli le query tradotte; si nota pertanto che SQS deve conoscere la locazione dei peer vicini.
(6) Infine, SQS deve essere in grado di raccogliere i risultati e inviarli al richiedente.

Servizi

All'interno di DBE, lo scenario descritto può essere implementato attraverso un insieme di servizi diversi che interagiscono fra loro. La scelta e il tipo di servizi da progettare dipende dalle specifiche delle funzionalità che il sistema offre e dai requisiti che tali funzionalità devono rispettare.

La funzionalità di creazione di peer semantici sono svolte dal sistema *MOMIS*. Tale sistema inoltre rispetta tutte i requisiti specificati per questa operazione. In particolare permette, attraverso degli opportuni *wrapper*, di collegarsi alle sorgenti locali, di estrarre i loro schemi, convertirli nel formato ODL_{I^3} e annotarli rispetto l'ontologia lessicale WordNet. Inoltre, grazie al mediatore, integra gli schemi ricevuti in un unico schema globale, che rappresenta lo schema del peer semantico. Quindi tale sistema potrebbe rappresentare il servizio che permette di creare e gestire peer semantici.

Si nota comunque che tale sistema deve essere implementato in modo da poter accettare connessioni dall'esterno per poter riceve query; inoltre esso deve essere in grado di stabilire connessioni con altri servizi per poter definire dei mapping e per poter esportare le query poste sul nodo locale.

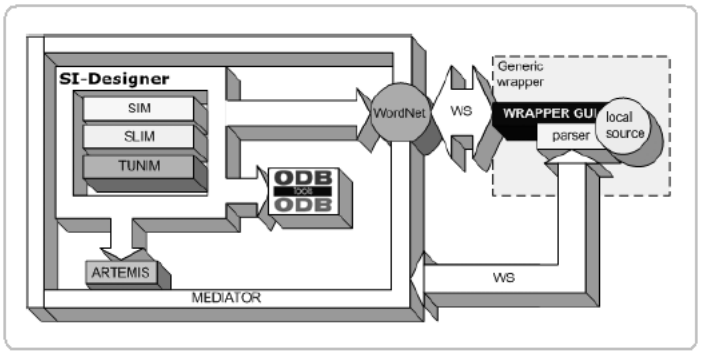


FIGURA 5. Architettura di MOMIS basata su Web Service

Pertanto, per l'implementazione di tale componente si sceglie di utilizzare l'architettura di MOMIS basata su Web Service descritta da BENETTI *et al.* (2004). Tale architettura prevede il disaccoppiamento dei componenti del sistema, pubblicandoli come web services XML. In particolare, ogni nodo è incapsulato in un wrapper concepito come un web service che rende disponibile le informazioni per il mediatore, concepito anch'esso come un web service per il wrapper. I web services sono implementati tramite il protocollo SOAP; in tal modo, l'architettura di MOMIS (figura 5) permette lo scambio di informazioni tra wrapper e mediatori attraverso una piattaforma client-server basata su SOAP. Ogni wrapper rende disponibile la descrizione ODL_{I^3} della sorgente sottostante, mettendo a disposizione come un servizio SOAP il metodo appropriato (`getDescription`), il quale ritorna lo schema ODL_{I^3} come una stringa XML. Al momento, i formati supportati sono relativi a dati relazionali, object-oriented e XML. Inoltre, il wrapper permette l'annotazione con WordNet direttamente a livello di sorgente, interagendo con il dizionario di WordNet acceduto come un web service messo a disposizione dal mediatore. Naturalmente, il wrapper dovrà fornire il client SOAP che invochi i metodi necessari per cercare i significati all'interno dell'ontologia di WordNet. Tramite il metodo `getAnnotatedDescription`, il mediatore è in grado di recuperare dal wrapper lo schema annotato della sorgente. Il mediatore è quindi in grado di valutare le relazioni lessicali tra i termini e inserirle all'interno del common thesaurus.

Si può pertanto sfruttare questa architettura di MOMIS e adattarla a DBE; in questa fase i web service definiti nell'architettura, ovvero il mediatore e i wrapper, devono essere trasformati in DBE Service. Questa operazione è relativamente semplice: è necessario importare nel DBE Studio l'interfaccia WSDL dei web service e generare, attraverso l'opportuno tool, la corrispondente interfaccia SDL. Successivamente, dall'interfaccia SDL si genera la classe java che implementa l'adapter e, in tale classe, vado ad inserire il client del web service.

Quindi i servizi relativi alla fase di creazione di creazione dei peer semantici sono due, il **wrapper** e il **mediatore**, entrambi modellati come DBE services. Il wrapper fornisce un metodo di accesso alle sorgenti uniforme, permettendo di recuperarne lo schema e di eseguire query su di esse. Il mediatore, per poter integrare schemi di sorgenti eterogenee in un'unica PVV, deve essere composto delle seguenti funzionalità, offerte come DBE services:

- *LocalMappingSystem*, il quale genera e gestisce i mapping locali (o intra-peer), cioè tra PVV e le descrizioni delle sorgenti informative.
- *LocalQueryManager*, il quale offre le funzionalità per gestire le query poste sul peer semantico (da un utente o da un servizio). In particolare tale componente dovrà riformulare tali query per mezzo dei mapping inter-peer e porre le query hai wrapper che gestiscono le rispettive sorgenti.
- *WNService*, che permette di cercare lemmi all'interno dell'ontologia lessicale WordNet e di recuperarne il significato.
- *Repository*, necessario per memorizzare la PVV e i mapping inter-peer.

Ogni azienda del sistema che voglia creare un peer semantico dovrà implementare un DBE service per il mediatore e uno o più DBE service per i wrapper (un wrapper per ogni sorgente che si intende integrare). In alternativa un peer può anche essere solo un wrapper, e quindi rendere disponibile a servizi esterni i propri dati e lo schema della propria sorgente. Inoltre un peer può implementare esclusivamente un mediatore e integrare sorgenti che appartengono ad altri nodi, interagendo con i loro wrapper.

Il mediatore classico del sistema MOMIS dovrà essere esteso per permettere la definizione di mapping inter-peer e la propagazione delle query non solo alle sorgenti locali ma anche ai nodi vicini. Due ulteriori servizi sono necessari: il **P2PMappingSystems**, il quale permette di estrapolare i mapping peer-to-peer (o inter-peer) e il **P2PQueryManager**, che permette di propagare le query sui nodi vicini.

Il *P2PMappingSystem* è un DBE service che è in grado di agire sia come client che come server per un mediatore. Infatti un mediatore può contattare il rispettivo mapper al fine di trovare mapping con altre PVV. In tal caso, il mediatore contatta il servizio attraverso un opportuno metodo (`findMapping`). A questo punto il P2PMappingSystem cerca altri servizi mediatori nella rete; attraverso un metodo messo a disposizione dal mediatore (`getDescription`), esso è in grado di recuperare l'ontologia di dominio di quel peer semantico e di ricavare i mapping tra gli elementi delle due PVV. Si nota che questa implementazione dei servizi permette di separare i compiti: il mediatore è responsabile della generazione e del mantenimento della PVV, mentre il P2PMappingSystem è responsabile per i mapping inter-peer. Tali mapping dovranno essere memorizzati in un opportuno repository persistente.

Il *P2PQueryManager* è quel servizio responsabile del query processing nel sistema realizzato. Si nota che il processo di query processing prevede due fasi distinte: l'interrogazione del nodo locale e l'interrogazione dei nodi vicini. Per quanto riguarda la prima fase, essa è già realizzata dal LocalQueryManager, un componente del mediatore; il P2PQueryManager è pertanto il responsabile della seconda fase. Ogni volta che una query viene posta al sistema, viene invocato un opportuno metodo offerto dal mediatore per porre la query sul peer locale il quale, a sua volta, pone la richiesta al P2PQueryManager per propagarla ai peer vicini. Il mediatore sfrutterà le mapping table per riformulare la query in più query locali e sfrutterà le funzionalità offerte dai wrapper per porre la query sulla sorgente. Per quanto riguarda l'esecuzione delle interrogazioni sui nodi vicini, il P2PQueryManager deve poter trasformare la query rispetto gli schemi degli altri peer. Per far ciò deve conoscere i mapping inter-peer gestiti dal P2PMappingSystem. Pertanto, il P2PQueryManager funge da client del P2PMappingSystem, invocando il metodo per recuperare i mapping. Una volta recuperato i mapping inter-peer, è in grado di trasformare le query. Per poterla inviare ai peer vicini, il P2PQueryManager invia la query ai mediatori di tali peer e si mette in attesa delle risposte. Dal momento che il progetto NeP4B prevede

che la query iniziale possa subire più step di riformulazione, la query che arriva ad un peer semantico, oltre che ad essere valutata localmente, attraverso il mediatore, viene valutata anche dal suo P2PQueryManager. Tale processo può essere ripetuto ricorsivamente più volte.

Modellazione dei servizi e delle ontologie. In DBE i servizi vengono modellati attraverso i linguaggi BML e SSL. SDL rappresenta invece l'interfaccia del servizio necessaria per poter invocare i suoi metodi.

Come detto nel paragrafo precedente, ogni SME che intenda fornire i propri dati in uno scenario come quello descritto in precedenza, deve implementare i 4 servizi necessari (o solamente alcuni di essi).

Pertanto sono stati sviluppati i modelli BML e SSL dei servizi che dovranno essere utilizzati dalle aziende per creare le istanze dei propri servizi. In particolare, le PMI dovranno esclusivamente modificare i dati relativi a tali modelli (livello M0 dell'architettura MOF), inserendo quelli di proprio interesse. In questo modo è possibile effettuare più facilmente delle ricerche semantiche di altri servizi simili implementati da altre organizzazioni. Inoltre, oltre alla modellazione dei servizi, è necessaria anche la modellazione di ontologie per descrivere il sistema in questone. L'importanza di tali ontologie nasce dal fatto che le loro classi costituiscono un'importante meccanismo di typing per gli elementi dei modelli BML e SSL. In tale sezione vengono dunque descritti i modelli realizzati.

L'ontologia in figura 6 descrive i mapping che intercorrono fra le ontologie; tale schema è stato sviluppato sulla base del metamodello per ontology mapping descritto in (HAASE & BROKMANS 2006) ed importa i concetti di *Ontology*, *Schema*, *OntologyElement* e *SchemaElement* dall'ontologia *OntologyConcept*, descritta successivamente nel paragrafo (figura 5). La classe centrale dell'ontologia è la classe `Mapping`, la quale presenta due attributi, un uri, che consente di identificare univocamente il mapping, e un attributo booleano, `intensional`, che consente di indicare se il mapping in questione è intensionale o estensionale. Rispetto al meta-modello presentato da Haase e Brokmans (2006) alla classe `Mapping` sono state aggiunte le due sottoclassi `IntraPeerMapping` e `InterPeerMapping` per poter rappresentare i due diversi casi. ogni mapping è composto da più `Assertion` ed ha sempre una `sourceOntology` e una `targetOntology`. Sono state `aggiunte` delle restrizioni sulle proprietà `sourceOntology` e `targetOntology` per rappresentare il fatto che le ontologie sorgenti e destinazione degli `IntraPeerMapping` sono costituite da schemi (classe `Schema`), mentre quelle degli `InterPeerMapping` sono costitutite solo da elementi della classe `Ontology`. ODM infatti fornisce dei costrutti che permettono di implementare restrizioni sul range della proprietà quando applicata ad una particolare classe; tali costrutti, che prendono il nome di *value constraint*, sono di tre tipi diversi: *allValuesFrom*, *someValuesFrom* e *hasValue*. In questo caso è stato utilizzato il tipo allValuesFrom, il quale vincola il range dell'extent di una proprietà. In particolare, le seguenti restrizioni sono poste:

Classe `IntraPeerMapping`.

- `sourceOntology allValuesFrom Schema`
- `targetOntology allValuesFrom Schema`

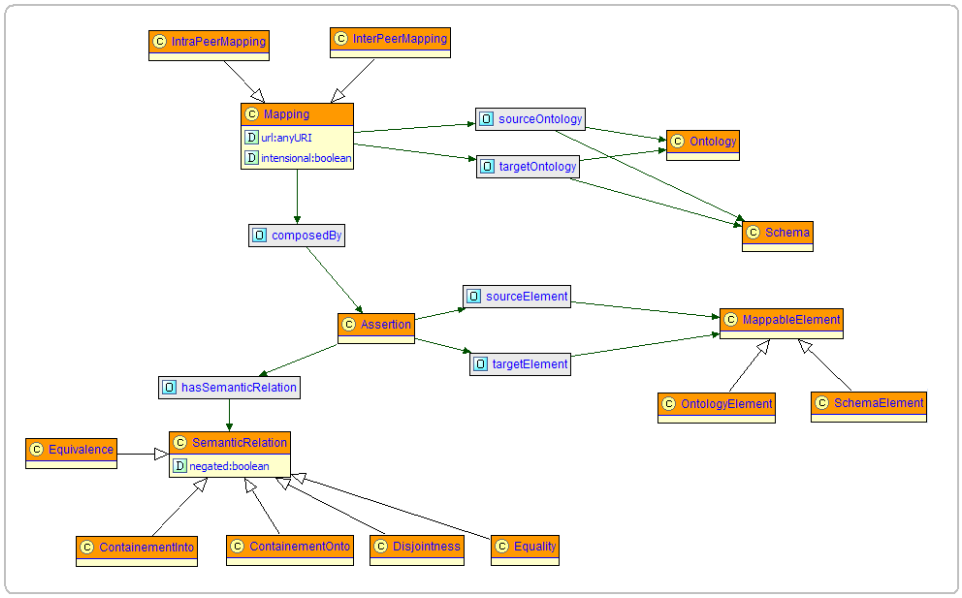


FIGURA 6. Ontologia che descrive i mapping

Classe `InterPeerMapping`.

- `sourceOntology allValuesFrom Ontology`
- `targetOntology allValuesFrom Ontology`

Inoltre, vengono poste le seguenti restrizioni sulla cardinalità rispetto le stesse proprietà per la classe `Mapping` (chee vengono quindi ereditate dalle sue due sottoclassi):

- `sourceOntology Cardinality 1`
- `targetOntology Cardinality 1`

le quali indicano che un mapping ha una sola ontologia sorgente e una sola ontologia di destinazione.

Di seguito vengono indicate le altre restrizioni definite nell'ontologia (tra parentesi viene indicata la classe rispetto la quale la restrizione è definita):

- (`Mapping`) `composedBy minCardinality 1`; `composedBy maxCardinality *`
  Inidicano che ogni mapping è costituito almeno da una asserzione.
- (`IntraPeerMapping`) `sourceElement allValuesFrom SchemaElement`
  Indica che tutti gli elementi sorgente di una asserzione appartenente ad un mapping intra-peer sono degli `SchemaElement`.
- (`IntraPeerMapping`) `targetElement allValuesFrom SchemaElement`
  Indica che tutti gli elementi destinazione di una asserzione appartenente ad un mapping intra-peer sono degli `SchemaElement`.

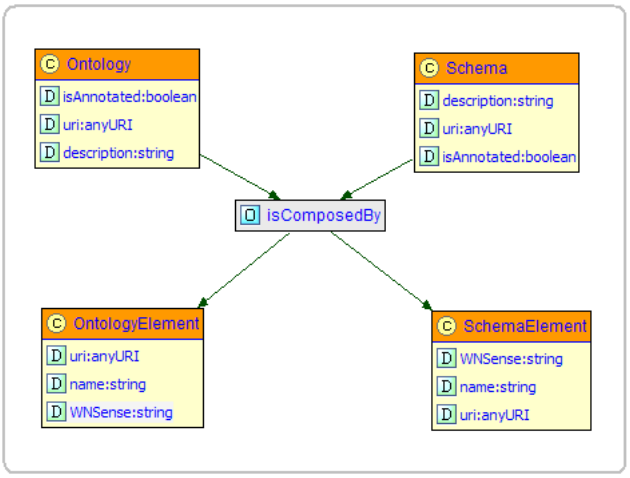


FIGURA 7. Schema dell'ontologia OntologyConcept

- (`InterPeerMapping`) `sourceElement allValuesFrom OntologyElement`
  Indica che tutti gli elementi sorgente di una asserzione appartenente ad un mapping inter-peer sono degli `OntologyElement`.
- (`InterPeerMapping`) `targetElement allValuesFrom OntologyElement`
  Indica che tutti gli elementi destinazione di una asserzione appartenente ad un mapping inter-peer sono degli `OntologyElement`.
- (`Assertion`) `sourceElement Cardinality 1`; `targetElement Cardinality 1`
  Indicano che una asserzione è costituita da un `MappableElement` sorgente ed uno di destinazione.
- (`Assertion`) `hasSemanticRelation Cardinality 1`
  Indica che una asserzione è costituita da una sengola relazione semantica.

Le classi `Ontology`, `OntologyElement`, `Schema` e `SchemaElement` sono descritte dall'ontologia *SemanticPeerNetwork* mostrata in figura 7.

Si nota che i concetti `Ontology` e `Schema` sono descritti dagli stessi attributi. La differenza tra i due è il loro diverso contesto: `Ontology` si riferisce alla descrizione di un peer semantico, mentre `Schema` rappresenta la descrizione di una sorgente locale. Tra i loro attributi vi sono `uri`, che permette di identificare un'ontologia (o uno schema), `description`, ovvero una stringa rappresentate lo schema dell'ontologie (o schema) e `isAnnotated`, il quale indica se la descrizione è stata annotata rispetto a WordNet. Analogamente, anche `OntologyElement` e `SchemaElement` sono descritti dagli stessi attributi: un uri, il nome dell'elemento (`name`), e l'indirizzo del WordNet sense che annota l'elemento (`WNSense`). La proprietà `isComposedBy` relaziona le ontologie e gli schemi agli elementi che le compongono. Attraverso le seguenti restrizioni si impone che una ontologia sia composta da soli `OntologyElement` e che uno schema ammetta solo elementi del tipo `SchemaElement`:

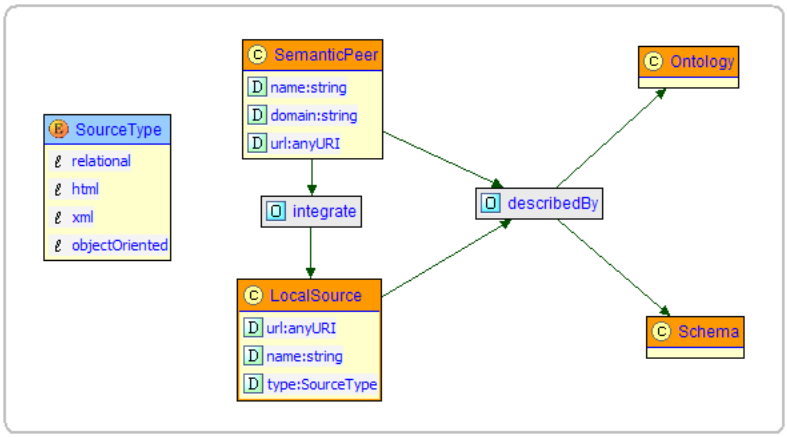


FIGURA 8. Ontologia che descrivi i tipi di nodi della rete

- (`Ontology`) `isComposedBy allValuesFrom OntologyElement`
- (`Schema`) `isComposedBy allValuesFromSchemaElement`

I concetti di semantic peer e local source sono descritti nell'ontologia di figura 8, la quale importa l'ontologia precedente per riutilizzare le classi `Ontology` e `Schema`.

E' in questa ontologia che si definisce che un `SemanticPeer` e descritto (`describedBy`) da una `Ontology` mentre una `LocalSource` è descritta da uno `Schema`. Oltre alla proprietà `describedBy` è necessario definire su di essa le due seguenti restrizioni:

- (`SemanticPeer`) `describedBy allValuesFrom Ontology`
- (`LocalSource`) `describedBy allValuesFrom Schema`

Inoltre, ogni peer e sorgente sono descritti esattamente da una ontoogia e da uno schema rispettivamente:

- (`SemanticPeer`) `describedBy Cardinality 1`
- (`LocalSource`) `describedBy Cardinality 1`

L'ontologia descrive inoltre il fatto che ogni semantic peer integra più sorgenti locali in un'unica vista rappresentata appunto dall'ontologia. Le seguenti restrizioni sono poste sulla proprietà integrate:

- (`SemanticPeer`) `integrate minCardinality 1; integrate maxCardinality *`
  Si esprime che ogni semantic peer deve integrare almento una sorgente locale.
- (`LocalSource`) `integrate Cardinality *`
  Indica che una sorgente locale può appartenere a nessun, a uno o a più semantic peer.

Gli attributi `url` permettono di identificare la localizzazione del semantic peer e della sorgente locale. L'attributo `domain` di `SemanticPeer` è utilizzato per descrivere brevemente il dominio del peer. Infine, la `LocalSource` prevede un attributo `type`; questo è un tipo enumerazione che permette di indicare il tipo della sorgente (relazionale, pagina html, object-oriented o un documento XML).

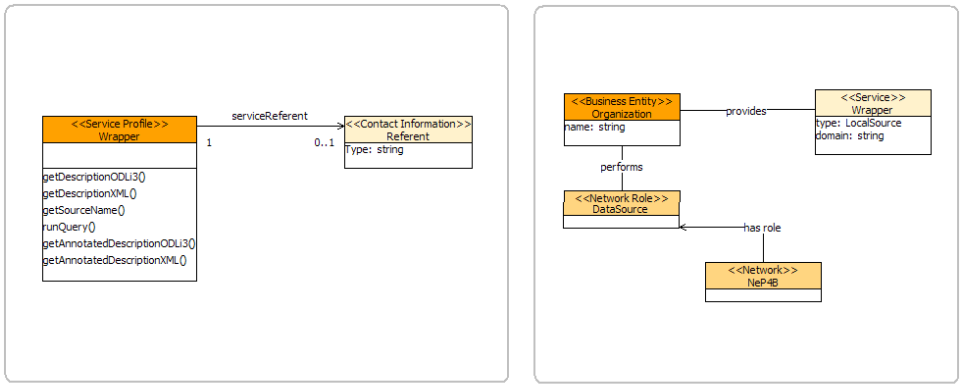


FIGURA 9. Modelli (a) SSL e (b) BML relativi al wrapper

Vediamo ora la modellazione dei servizi. Per la descrizione dei servizi e dei business model DBE mette a disposizione due linguaggi, SSL e BML rispettivamente. Per lo scopo che ci si è prefissi, la parte rilevante di modellazione avviene tramite il linguaggio SSL. Per quanto riguarda la modellazione BML, si utilizzano solamente i costrutti dell'Organization Package: questi modelli sono utilizzati principalmente per fornire un supporto nella ricerca di servizi di wrapping, mediazione, di mapping e di interrogazione forniti da altre PMI.

In figura 9 sono riportati i modelli SSL e BML del wrapper. Per quanto riguarda la descrizione semantica del servizio (figura 9a), sono definiti diverse operazioni eseguibili dal wrapper, le principali delle quali permettono di recuperare i modelli della sorgente collegata al wrapper e di eseguire localmente una query. Il wrapper esegue tre attività principali. In primo luogo, dopo la connessione con la sorgente locale, recupera il suo schema, lo traduce in formato ODL_{I^3} e lo memorizza localmente; in questo modo, quando un mediatore chiede di recuperare tale schema attraverso i metodi `getDescriptionODLi3()` e `getDescriptionXML()`, il wrapper deve solo leggere lo schema memorizzato. La seconda fase è quella di annotazione dello schema rispetto l'ontologia WordNet. Per compiere questa operazione il wrapper necessita di collegarsi al mediatore, in quanto il database lessicale è gestito da tale componente; questa operazione sarà discussa successivamente. Il terzo compito principale di un wrapper è quello di porre alla sorgente una query, inviatagli dal mediatore attraverso l'operazione `runQuery()`, e di ritornare i risultati.

In particolare, le operazioni `getDescriptionODLi3()` e `getDescriptionXML()` ritornano rispettivamente lo schema della sorgente in formato ODL_{I^3} e XML. Pertanto entrambi prevedono un parametro in uscita che è rappresentato da un oggetto `Schema` per il primo e una stringa per il secondo. I metodi `getAnnotatedDescriptionODLi3()` e `getAnnotatedDesciptionXML()` sono analoghi ai precedenti con la distinzione che il valore di ritorno è la descrizione della sorgente annotata rispetto a WordNet. Il metodo `getSourceName()` ritorna una stringa contenente il nome della sorgente. Infine il metodo `runQuery()` accetta in ingresso un stringa (la query) e restituisce in uscita un stringa riportante i risultati dell'esecuzione della query.

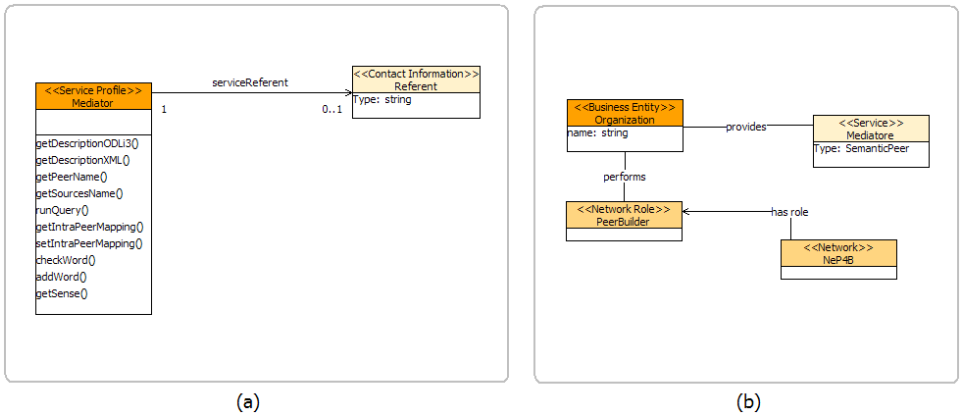


FIGURA 10. Modelli (a) SSL e (b) BML relativi al mediatore

Il servizio modellato in figura 9a prevede inoltre un informazione relativa ad un contatto per il servizio; questa caratteristica non è rilevante ai fini dell'implementazione del servizio ma è stata comunque aggiunta, così come in ogni altro servizio modellato.

In figura 9b è invece riportato il modello BML. Con tale modello si indica un'organizzazione di nome `name` che fornisce un servizio di wrapping; pertanto, nel contesto `NeP4B` (Network), tale organizzazione assume il ruolo di `DataSource` (Network Role), in quanto mette a disposizione la propria conoscenza agli utenti della rete. Come si nota dalla figura, il servizio Wrapper presenta un attributo `type` di tipo `LocalSource`: in questo modo è possibile aggiungere semantica al modello implementato in quanto `LocalSource` è la classe definita all'interno dell'ontologia SemanticPeerNetwork. Questo consente di definire per quel wrapper il suo nome, URL e il tipo di sorgente a cui è connesso (relazionale, XML, etc.). Dal momento che ogni azienda che intende fornire servizi di wrapping per consentire l'accesso alle proprie sorgenti dati deve istanziare tale modello BML, esso rappresenta una importante riferimento per ricercare altri servizi simili a questo. Ad esempio, un impresa che intenda cercare sorgenti dati relative al dominio meccanico, potrebbe porre al sistema una query del tipo "Cerca tutte le organizzazioni che forniscono un servizio Wrapper relativo al dominio della meccanica". A questo scopo, l'annotazione con l'ontologia SemanticPeerNetwork è di fondamentale importanza, in quanto permette di indentificare meglio la corrispondenza semantica tra le diverse istanze del servizio. I modelli BML relativi agli altri servizi sono stati realizzati seguendo lo stesso schema di quello relativo al wrapper.

I modelli relativi al mediatore sono riportati in figura 10a-b.

Il modello SSL di figura 10a rappresenta il profilo del servizio. Tale profilo contiene tutti le operazioni che determinano il comportamento del mediatore e che sarrano svolte dai suoi componenti. I due metodi `getDescriptionODLi3()` e `getDescriptionXML` permetto ad un servizio quale il P2PMappingSystem di accedere al Repository del mediatore e di recuperare la PVV del peer semantico; il valore di ritorno dei due metodi è rappresentato da un oggetto della classe `Ontology` (derivato dall'ontologia OntologyConcept) per il primo, mentre è una stringa per il secondo metodo. Il repository è acceduto anche per recuperare o memorizzare i mapping con le sorgenti locali: in tal cosa vengono forniti

i metodi `getIntraPeerMapping()` e `setIntraPeerMapping()`. Di fondamentale importanza è il metodo `runQuery()` il quale permette di porre una query al peer semantico: questa funzione è utilizzata principalmente dal P2PQueryManager il quale è il responsabile della propagazione delle query tra i peer che compongono il sistema. Ogni query ricevuta da un peer è computata dal LocalQueryManager, il componente del mediatore che si incarica di riformulare la query (in termini di unfolding) per ogni sorgente gestita dal mediatore, di porre le query ai rispettivi wrapper, di organizzare e di presentare i risultati ottenuti al richiedente. Pertanto tale motodo accetta una stringa in ingresso, rappresentante la query, e restituisce i risultati della query, ancora in formato stringa. Infine, i metodi `checkWord()`, `addWord()` e `getSense()` servono per poter interagire con il WNService del mediatore, ovvero quel componente che permette ai wrapper di annotare gli schemi delle sorgenti locali. Il metodo `checkWord()` prende in ingresso una stringa che rappresenta un vocabolo e restituisce la lista dei significati che tale vocabolo assume. Si nota che in questo caso, come anche per la funzione getIntraMapping(), sarebbe stato utile avere un costrutto lista come tipo del valore di ritorno; le specifiche di BML non prevedono comunque un costrutto di questo tipo pertanto si è scelto di utilizzare le stringhe per rappresentare il tipo di tali attributi. Questo, in realtà, è un problema che si incontra spesso nella modellazione con BML e che verrà discusso al termine di questo capitolo. Il metodo `addWord()` permette di aggiungere un *sense* per un determinato vocabolo nel caso in cui questo non sia già presente nel databese di WordNet. Infine, `getSense()` permette di selezionare un particolare sense per un vocabolo: a questo fine, tale operazione ritorna l'indirizzo logico del significato nella base di dati di WordNet; tale indirizzo viene utilizzato per annotare l'elemento della sorgente locale.

Per quanto riguarda il modello BML, rappresentato in figura 10b, esso ha la stessa struttura di quello relativo al wrapper, con la differenza che il tipo del servizio è modellato come un `SemanticPeer` e che il ruolo dell'organizzazione nella rete `NeP4B` è quello di `PeerBuilder`.

Il servizio P2PMappingSystem è descritto in figura 11. Il suo comportamento è definito dalle seguenti operazioni:

- `findMapping()`: è il metodo invocato dal mediatore per ordinare al P2PMappingSystem di scoprire i mapping peer-to-peer con le ontologie di dominio associate ad altri peer. In ingresso, tale metodo prevende l'ontologia del peer semantico, pertanto, il tipo di tale parametro è rappresentato dalla classe Ontology dell'ontologia OntologyConcept. Il valore di ritorno è un insieme di mapping con altre ontologie. Anche in questo caso sarebbe stato utile un costrutto lista per il tipo del valore di ritorno (una lista di mapping); in mancanza di tale possibilità il tipo di tale parametro è il tipo string. Un P2PMappingSystem che ha ricevuto una richiesta di questo tipo ricerca nella rete eventuali ontologie simili alla PVV del peer richiedente. Nel corso del progetto NeP4B dovranno essere studiati e sperimentati vari metodi per scoprire mapping inter-peer fra i nodi della rete. Nello scenario che si sta descrivendo, la ricerca di ontologie di dominio può sfruttare il servizio di Semantic Discovery offerto da DBE. In particolare, P2PMappingSystem può porre delle query all'engine di DBE per cercare servizi altri servizi mediatore dai quali recuperare le ontologie. In particolare, P2PMappingSystem può interrogare i modelli BML relativi ai servizi di mediazione richiedendo , ad esempio, "Tutte le organizzazioni che offrono servizi Mediator con dominio relativo alla meccanica"; inoltre potrebbe utilizzare nella

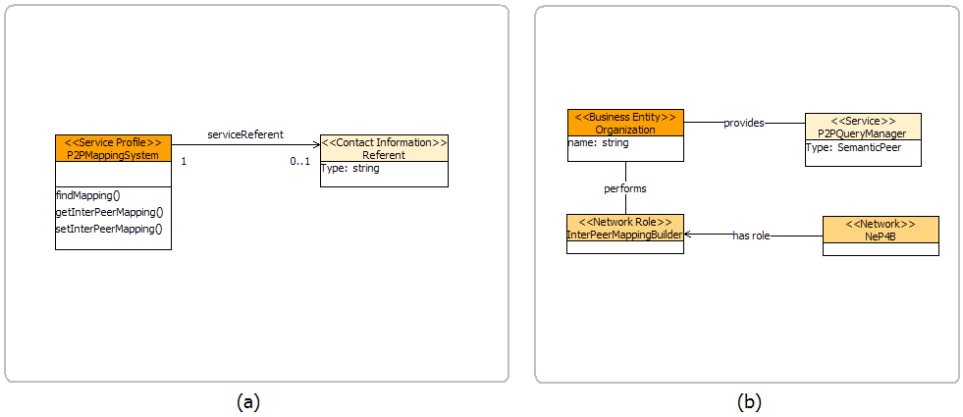


FIGURA 11. Modelli (a) SSL e (b) BML relativi al mapper

query tutti i concetti definiti dal modello BML in quanto si assume che ogni organizzazione che fornisce tali servizi utilizzi esattamente i modelli descritti in questa sezione. Una volta recuperati i servizi disponibili, il P2PMappingSystem può recuperare le PVV dei peer attraverso il metodo `getDescriptionODLi3()` o getDescriptionXML() forniti dal loro mediatore.

- ❖ `getInterPeerMapping()`: consente ad un servizio P2PQueryManager di recuperare i mapping peer-to-peer per poter riformulare le query. Tale metodo prevede un parametro di ritorno (di tipo stringa), ovvero la lista dei mapping di quel peer semantico.
- ❖ `setInterPeerMapping()`: consente di memorizzare i mapping inter-peer nel repository del servizio; tale metodo accetta in input la lista dei mapping da memorizzare.

Il modello BML di tale servizio, riportato in figura 11b, definisce che l'organizzazione fornisce servizi P2PMappingSystem e che partecipa alla rete `NeP4B` con il ruolo di `Inter-PeerMappingBuilder`.

L'ultimo servizio è il P2PQueryManager, i cui modelli sono riportati in figura 12. Tale servizio, invocato dal mediatore relativo al proprio peer, consente di propagare la query in arrivo sul peer a tutti i peer per i quali è definito un mapping. Esso definisce una sola funzione, `runQuery()`, che prevede in ingresso la query posta su peer e in uscita l'insieme dei risultati ottenuti interrogando i peer delle altre organizzazioni. Le azioni che tale servizio dovrà compiere sono, principalmente, il recupero dei mapping peer-to-peer attraverso la funzione `getInterPeerMapping()` del servizio P2PMappingSystem, la riformulazione della query ricevuta in input rispetto tali mapping, e l'invio delle query riformulate ai rispettivi peer. Questa ultima funzione viene eseguita invocando il metodo `runQuery()` messo a disposizione dai mediatori.

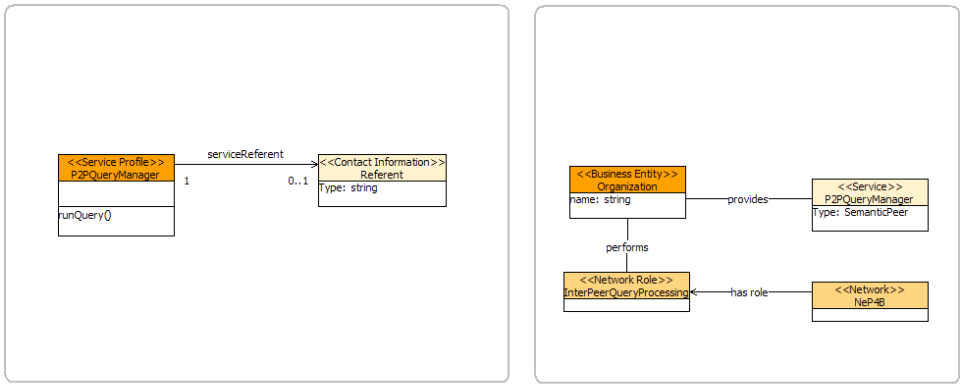


FIGURA 12. Modelli (a) SSL e (b) BML relativi all'interrogator

Il modello BML di tale servizio è analogo a quello di tutti gli altri servizi descritti in precedenza: le uniche differenze riguardano il ruolo dell'organizzazione, che è di `InterPeerQueryProcessing`, e il servizio offerto, che è appunto il `P2PQueryManager`.

CAPITOLO 6

Data Quality: Introduzione al Problema

1. Introduzione al concetto di Data Quality

Il termine *data quality*, o *qualità dei dati*, è tipicamente utilizzato per riferirsi ad un insieme di caratteristiche che i dati devono possedere come, ad esempio, il grado di correttezza o di aggiornamento. Tale problema riveste una sempre maggiore importanza in ambito industriale, in quanto una bassa qualità dei dati può portare a conseguenze economiche negative (WANG & STRONG 1996).

Le conseguenze di una scarsa qualità dei dati sono spesso verificate ogni giorno nelle proprie attività, ma difficilemente vengono apportate le dovute correzioni, dove necessario. Ad esempio, il ricevimento di due mail identiche generate automaticamente indica probabilmente un problema di duplicazione nei record del database. Analogamente, la recapitazione errata di una lettera viene spesso attribuita all'inefficienza del servizio postale; in realtà, in molti casi questo tipo di errori hanno una causa data-related, ad esempio un errore nella digitazione dell'indirizzo, originato nel database degli indirizzi.

La qualità dei dati assume quindi un ruolo molto importante nel garantire l'efficienza e l'efficacia delle operazioni e dei servizi svolti da una organizzazione. Un studio sulla qualità dei dati effettuato dal Data Warehousing Institute (REF) ha valutato in 600 bilioni di dollari all'anno il costo che i problemi di qualità causano nelle aziende degli Stati Uniti. Altri esempi indicanti l'importanza della data quality nei processi aziendali sono riportati di seguito:

- *Costumer matching*. I sistemi informativi delle organizzazioni private e pubbliche possono essere visti come il risultato di un insieme di attività indipendenti e scarsamente controllate che producono database caratterizzati, molto spesso, da informazioni sovrapposte o coincidenti. Nelle organizzazioni private, è frequente il caso in cui le informazioni riguardo clienti e/o fornitori vengano mantenute in diversi repository, aggiornati con frequenze e procedure diverse; il risultato di questo tipo di situazione è la presenza di informazioni duplicate e inconsistenti.
- *Organization fusion*. Quando organizzazioni diverse, o unità distinte della stessa organizzazione, si fondono, è necessario integrare i loro legacy system. Tale integrazione richiede compatibilità e interoperabilità ad ogni livello del sistema informativo, con il livello del database che deve garantire interoperabilità fisica e semantica.

§

La qualità dei dati è una tematica affrontata in diversi ambiti, in particolare in ambito statistico, gestionale e informatico, e in molti altri settori scientifici. L'attività degli statistici si concentra spesso sulle proprietà delle stime statistiche, la qualità delle decisioni prese di fronte all'incertezza, l'adattamento dei modelli statistici. Utilizzando concetti come errore standard, distorsione, bontà di adattamento ed errore nei test di verifica d'ipotesi, sono state costruite varie metodologie per la stima e l'analisi nei quali la qualità dei dati gioca un ruolo centrale. Uno dei lavori principali in tale ambito è quello di FELLEGI e SUNTER (1969) i quali hanno elaborato una teoria matematica per considerare i duplicati nei dati statistici. In ambito gestionale, l'interesse dei ricercatori, nato agli inizi degli anni '80, è rivolto alla gestione e alla risoluzione dei problemi della qualità nei sistemi di produzione di dati (BALLOU & PAZER 1985). In ambito informatico, i ricercatori hanno iniziato ad occuparsi del problema del data quality agli inizi degli anni '90; in questo settore, la ricerca si focalizza sullo sviluppo di metodi per definire, misurare e migliorare la qualità dei dati elettronici memorizzati nei database, data warehouse e nei sistemi legacy. Molti studi sono stati effettuati nel contesto di singoli sistemi informativi dove sono state proposte metodologie per modellare quality requirement nei database relazionali (WANG *et al.* 1993) e per migliorare la process quality (SHANKARANARAYAN *et al.* 2000). Recentemente, molta attenzione è riposta nel problema del data quality relativo a sistemi multi-database e nei sistemi di integrazione di dati (DE GIACOMO 2004; LENZERINI 2004), come data warehouse (JARKE *et al.* 1999) e Cooperative Information Systems (MECELLA *et al.* 2002). L'integrazione di grandi moli di dati, acquisite da multiple sorgenti con rappresentazioni eterogenee delle informazioni, comporta, oltre a conflitti di natura semantica, dovuti dai diversi modelli delle sorgenti, dei conflitti a livello d'istanza, dovuti alle inconsistenze fra i dati. In questo contesto, il data quality è dunque studiato come soluzione ai problemi di data cleansing e di object fusion (HERNANDEZ & STOLFO 1998; NEILING *et al.* 2003).

Anche all'interno della computer science, non vi è un accordo comune fra i ricercatori riguardo la definizione di qualità dei dati. In (KAHN *et al.* 2002) l'information (o data) quality è definita come una scienza inesatta in termini di assessments e benchmarks. Inoltre, la data quality è stata definita da WANG e STRONG (1996) come "fitness for use", con una evidente enfasi sulla sua natura soggettiva. Un'altra definizione è quella di distanza tra i dati rappresentati nei sistemi informativi e gli stessi dati nel mondo reale (ORR 1998; WAND & WANG 1996); tale definizione può essere vista come una definizione operazionale, anche se è piuttosto complesso valutare la data quality sulla base di un confronto con gli oggetti reali.

In generale, la data quality può essere collegata ad un insieme di "dimensioni", come l'accuratezza o la completezza, che sono tipicamente definite come proprietà o caratteristiche della qualità. Come vedremo in seguito, anche nella scelta delle dimensione e nella loro definizione non vi è un accordo comune fra gli addetti ai lavori.

§

Da quanto detto, si evince che il concetto di data quality presenta molte faccie, così come nella sua definizione possono essere coivolte molte dimensioni. I problemi relativi alla qualità dei dati aumentano se si considera l'eterogeneità dei tipi di dato che possono essere coivolti e dei sistemi informativi.

Data Quality e Tipi di Dato. Al fine di rappresentare gli oggetti del mondo reale, molti tipi di dato sono stati creati e molte diverse classificazioni di tali tipi sono state elaborate. In generale, la definizione e la progettazione di un sistema che gestisca la qualità dei dati può subire modifiche a seconda del tipo di dati che si deve gestire.

Tra le varie classificazioni, la principale è quella che distingue tre tipi diversi di dati:

- *Strutturati*, quando ogni data element ha una struttura fissa (ad esempio, le tabelle del modello relazionale).
- *Semi-strutturati*, quando i dati hanno una struttura con un certo grado di flessibilità (ad esempio XML).
- *Non strutturati*, quando non è definita nessuna specifica struttura, come nel caso del linguaggio naturale.

In questo caso, le dimensioni e le tecniche per data quality devono essere adattate per i tre tipi di dati.

Una seconda classificazione è quella che prevede la ripartizione dei tipi di dato in elementari e aggregati. I tipi *elementari* sono dati rappresentanti fenomeni atomici del mondo reale (età, codice fiscale), mentre quelli *aggregati* sono ottenuti da una collezione di dati elementari applicando qualche funzione di aggregazione (come il numero medio di studenti frequentanti i corsi di laurea in ingegneria). Questa classificazione è utile per distinguere diversi livelli di severità nel misurare la qualità. Ad esempio, l'accuracy di un attributo `Sex` (dato elementare) cambia radicalmente se è impostato come F (female) invece di M (male); al contrario, se l'età di una singola persona è memorizzata come 25 invece di 35, l'accuracy dell'età media della popolazione di milioni di abitanti non viene compromessa.

Data Quality e Tipi di Sistemi Informativi. Analogamente al caso precedente, l'impatto del data quality dipende anche dal sistema informativo che gestisce i dati. Si possono distinguere i seguenti tipi: Monolitico, Distribuito, Data Warehouse, Cooperativo e Peer-to-Peer.

- In un sistema informativo *monolitico*, la logica di presentazione, applicazione e di data management sono uniti in un singolo nodo computazionale. Pur essendo estremamente rigidi, essi forniscono diversi vantaggi alle organizzazioni, quali la riduzione dei costi dovuta alla omogeneità delle soluzione e alla centralizzazione della loro gestione. In tali sistemi i dati hanno un formato comune e la gestione della qualità dei dati risulta agevolata dalla centralizzazione delle procedure.
- Un *data warehouse* (*DW*) è un insieme centralizzato di dati collezionati da sorgenti diverse, progettato per supportare operazioni di decision making. Il problama più critico per i DW riguarda la pulizia e l'integrazione delle diverse sorgenti dati.
- Un *sistema informativo distribuito* rilassa la struttura rigida dei sistemi monolitici, consentendo la distribuzione delle risorse e delle applicazioni su nodi appartenenti ad una rete distribuita geograficamente. Il vantaggio di questo tipo di archtettura è la separazione su più tier delle logiche di presentazione, applicazione e del data management. Sorgono però problemi di data management riconducibili alla gestione non centralizzata degli stessi.
- Un *sistema informativo cooperaivo* (*cooperative information system - CIS*) può essere definito come un sistema informativo su larga scala che interconnette vari sistemi informativi di diverse organizzazioni; pertanto, in tali sistemi l'integrazione delle sorgenti dati è un aspetto rilevante.

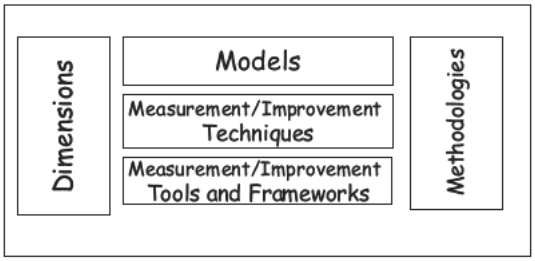


FIGURA 1. Argomenti di ricerca principali inerenti la qualità dei dati

Per quanto riguarda la relazione fra CIS e data quality, se da un lato è possibile avvalersi della cooperazione fra sistemi per scegliere le sorgenti più qualitative (e quindi migliorare la qualità complessiva del sistema), dall'altro la mancanza di un controllo centralizzato del data flow può causare un abbassamento progressivo della qualità.

- Infine, un sistema informativo *peer-to-peer* può essere caratterizzato da un vario insieme di caratteristiche, quali: l'alta autonomia dei peer, l'alta eterogeneità, assenza di un controllo e di un database centralizzato, assenza di una vista globale del sistema. Chiaramente, in questo scenario la gestione del data quality è estremamente complessa in quanto non esiste nessuna obbligazione ai peer riguardo la qualità dei loro dati e perchè è impossile eseguire dei controlli centralizzati.

§

A causa della sempre crescente che la qualità dei dati assume all'interno di organizzazioni private e pubbliche, della sua natura non ben definibile, della varietà dei tipi di dato e di sistemi informativi, la qualità dei dati è investigata in diverse discipline. I topic di ricerca principali sono mostrati in figura 1.

Un primo aspetto riguarda la scelta delle *dimensioni* utilizzate per misurare la qualità delle informazioni. Tale problema verrà analizzato nella prossima sezione.

Di fondamentale importanza è la scelta del *modello* (o dei modelli) utilizzati per rappresentare i dati, il loro schema; tali modelli devono essere estesi per poter rappresentare le dimensioni e altri aspetti relativi al data quality. Questo aspetto verrà affrontato nel prossimo capitolo.

Tecniche, corrispondenti ad algoritmi, procedure knowledge-based, e processi di learning che forniscono una soluzione alle data quality activity. Esempi di tali attività sono: identificare se due record di database diversi rappresentano lo stesso oggetto del mondo reale e trovare la sorgente più affidabile per una specifica informazione.

Le *metodologie* forniscono delle linee guida per scegliere il processo di misurazione e miglioramento di data quality più efficace all'interno di uno specifico sistema informativo.

FIGURA 2. Esempio di record

2. Il problema della Scelta delle Dimensioni

Capita spesso che quando si pensa alla qualità dei dati, ci si riferisca esclusivamente all'accuratezza delle informazioni. Infatti, i dati sono considerati di scarsa qualità se sono stati memorizzato in maniera errata. In realtà, la qualità dei dati non è univocamente identificabile con l'accuratezza. Altre dimensioni, tutte significative, come completezza o consistenza, devono essere definite per poter caratterizzare in modo completo la qualità dei dati.

Si consideri ad esempio il record `Citizen`, con campi `Name`, `Sex` e `Email`, riportato in figura 2.

Se Name ha valore "Mke", mentre il valore corretto rispetto il dizionario inglese è "Mike", allora l'accuratezza del dato memorizzato è bassa. Un esempio di bassa completezza è fornito considerando il campo `Email`; un valore null per tale campo può assumere diversi significati, come *(i)* il cittadino in questione non ha un indirizzo e-mail, e quindi il campo è inapplicabile (cioè non ha nessun impatto sulla completezza), oppure *(ii)* lo specifico cittadino possiede un indirizzo e-mail ma questo non è stato memorizzato: in quest'ultimo caso la completezza assume un valore basso.

La definizione delle dimensioni che caratterizzano la qualità dei dati non è sempre così semplice come nell'esempio precedente. Vi sono due problemi principali:

- ❖ In letteratura, non c'è accordo sull'insieme di dimensioni caratterizzanti il data quality. Sono state avanzate molto proposte, ma nessuna è riuscita ad emergere rispetto le altre e a imporsi come standard *de facto*.
- ❖ Anche se alcune dimensioni sono universalmente considerate come indispensabili, potrebbe non esserci accordo comune sul loro significato. Nelle varie proposte avanzate, lo stesso nome è utilizzato per indicare caratteristiche con diversa semantica.

Nel seguito di questa sezione, si analizzeranno le dimensioni e le loro definizioni in alcuni lavori proposti in letterata; tali lavori saranno poi confrontati per valutare quali dimensioni sono considerate indispensabili per caratterizzare la qualità delle informazioni.

§

Proposte di dimensioni per data quality. Diverse proposte sono state effettuate negli ultimi dieci anni; di seguito verranno brevemente illustrate alcune fra le principali iniziative.

- ❖ (WANG & WAND 1996). Tale proposta è basata su un modello formale per la definizione dei sistemi informativi. Le dimensioni per data quality sono definite

considerando delle funzioni di mapping dagli oggetti del mondo reale a quelli memorizzati nel sistema informativo. Ad esempio, la non accuratezza dei dati significa che il sistema informativo rappresenta uno stato del mondo reale diverso da quello che dovrebbe rappresentare, mentre la dimensione completezza è definita come un mapping mancante tra lo stato del mondo reale e lo stato del sistema informativo. In tale proposta vengono identificate e definite 5 dimensioni principali, ovvero *accuracy*, *completeness*, *consistency*, *timeliness* e *reliability*.

- (WANG & STRONG 1996). Tale proposta, già accennata in precedenza, deriva da uno studio empirico e definisce le dimensioni dal punto di vista del consumatore. Tali dimensioni sono state infatti selezionate intervistando direttamente i consumatori dei dati: partendo da un insieme di 179 dimensioni, ne sono state selezionate 15.
- (REDMAN 1996). In tale proposta, le dimensione sono state raggruppate in tre diverse categorie, corrispondenti al punto di vista concettuale dei dati, al valore dei dati e al loro formato. Un totale di 17 dimensioni sono state definite: 5 per la prima categoria, 4 per la seconda e 8 per l'ultima categoria.
- (JARKE *et al.* 1999). Tale proposta è stata avanzata all'interno del progetto europeo DWQ, Foundations of Data Warehouse Quality. L'obbiettivo del progetto è quello di guidare le attività di progettazione dei data qarehouse. In questo contesto, sono state proposte specifiche dimensioni, classificate in base ai ruoli assunti dagli utenti all'interno del sistema. In particolare, si hanno 6 dimensioni per la categoria "Design and Administration Quality", 6 dimensioni per "Software Implementation Quality", 5 per "Data Usage Quality" e altre 5 dimensioni per "Data Stored Quality".
- (BOVEE *et al.* 2001). Seguendo il concetto di qualità come "fitness for use", espresso in (WANG & STRONG 1996), tale proposta include 4 dimensioni e alcune sotto-dimensioni. I dati sono "fit for use", o "adatti per l'utilizzo", quando: 1) sono in grado di dare informazioni cercate (*accessibility*); 2) sono comprensibili (*interpretability*); 3) sono applicabili al dominio di interesse (*relevance*); 4) quando si ritiene che siano credibili (*credibility*).
- (NAUMANN *et al.* 1999). Si propone un algoritmo che seleziona le sorgenti da interrogare in base alla valutazione della qualità delle sorenti stesse e delle specifiche query. Vengono definite un totale di 12 dimensioni ripartite in 3 categorie: *source-specific*, riguardanti la qualità di una sorgente informativa; *QCA-specific*, inerenti la qualità di specifiche query computabili da una sorgente; *attribute-specific*, le quali valutano la qualità di una sorgente in termini della sua abilità a fornire gli attributi di una specifica query.
- (NAUMANN 2002). In questo lavoro vengono proposti delle dimensioni specifiche per Web Information System integrati. Le 21 dimensioni definite sono ripartite in 4 categorie: *content-related*, riguardanti i dati attuali che sono stati recuperati; *technical*, relative ad aspetti collegati alla sorgente informativa, alla rete e all'utente; *intellectual*, collegati ad aspetti soggettivi delle sorgenti dei dati; *instantiation-related*, riguardanti la presentazione dei dati.

Confronto fra le varie proposte. Un confronto fra le varie alternative è mostrato in tabella 1. In particolare, la tabella mostra come le diverse proposte usano gli stessi nomi per dimensioni con significati diversi (D) o simili (D%). La lettera S è utilizzata per indicare che lo stesso nome è utilizzato per indicare lo stesso concetto: in questo modo si è in grado di osservare le proposte che utilizzano le stesse dimensioni.

	WANG & WAND 1996	WAND & STRONG 1996	REDMAN 1996	JARKE *et al.* 1999	BOVEE *et al.* 2001	NAUMANN *et al.* 1999	NAUMANN 2002
Accuracy	S	S	S	S	S	S	S
Completeness	S	D	S	D%	S	S	S
Consistency	S		D%	S	S		
Representational Consistency		S	S			D	S
Timeliness	S	S		S	S	S	S
Currency	S		S	S	S		
Volatility	S			S	S		
Interpretability		S	D%	D%	S		S
Ease of Udersta-ning/Understandability		S				S	S
Reliability	D			D		D	
Credibility				D	D		
Believability		S					S
Reputation		S				S	S
Objectivity		S					S
Relevancy/Relevance		S	S		D%	D	S
Accessibility		S			S		
Security/Access Security		S					S
Value-added		S					S
Concise representation		S					S
Appropiate amount of data/amount of data		D	D			D	D
Availability				S		S	S
Portability			D	D			
Responsiveness/Response Time				S		S	S

TABELLA 1. Confronto fra gli insiemi di dimensioni nelle diverse proposte

Dalla tabella 1 si deduce che *accuracy* e *completeness* sono le uniche dimensioni definite da tutti le proposte analizzate. Oltre a tali dimensioni, anche quelle relative alla consistenza (*consistency* e *representational consistency*) e al tempo (*timeliness*, *currency* e *volatility*) sono state prese in considerazione da tutte le proposte. In particolare, la consistenza è tipicamente considerate a livello di istanza (*consistency*) o di formato (*representational consistency*). Le dimensioni relative al tempo sono principalmente rappresentate dalla dimensione *timeliness*. Anche interpretability è considerata in molti lavori, sia a livello di formato che di schema.

Ogni altra dimensione rimanente è inclusa solo in una minoranza delle proposte. In alcuni casi vi è una totale disaccordo sulla definizione di una dimensione, come per la *reliability*.

In particolare, per ogni dimensione riportata in tabella 1 si possono fare le seguenti considerazioni:

- ACCURACY. E' definita in tutte le proposte come il grado di correttezza di un valore quando confrontato con uno di riferimento.
- COMPLETENESS. Anche questa dimensione ha una definizione uniforme intutte le proposte. Essa viene definita come il grado di presenza dei dati in una data collezione. Una definizione diversa è data in (WANG & STRONG 1996): essa viene infatti vista da un punto di vista *user-dependent*, come il livello con cui i dati sono in grado di gestire il processo in questione in modo sufficientemente ampio e profonodo. In (JARKE *et al.* 1999) viene data una definizione simile, ma anche per il livello di schema (intensionale), oltre per quello estensionale (di istanza).
- CONSISTENCY. E' definita in tutti i lavori, fatta eccezione per WANG e STRONG (1996), NAUMANN *et al.* (1999) e NAUMANN (2002), i quali considerano esclusivamente la representational consistency. In tutte le definizone essa rappresenta la consistenza fra diversi valori di dati come, ad esempio, tra il sesso di una persona e il suo nome. Si nota che solo REDMAN (1996) dà una definizione di consistency sia a livello concettuale, di istanza e di formato.
- REPRESENTATIONAL CONSISTENCY. Per (WANG & STRONG 1996) e (NAUMANN 2002) rappresenta il grado con cui i dati sono sempre rappresentati nello stesso formato. In (NAUMANN *et al.* 1999) essa viene definita da un punto di vista diverso: rappresenta il tempo, in secondi, di consumo del wrapper dovuto all'esecuzione di una query; più la presentazione della sorgente è consistente, meno lavoro dovrà compiere il wrapper per il parsing, traduzione, ecc.
- TIMELINESS, CURRENCY E VOLATILITY. Sono tutte dimensioni relative al tempo. *Timeliness* è definita come il grado con cui i dati sono temporalmente validi per il loro utilizzo. Questa dimensione può essere espressa in termini di *currency*, ovvero quanto i dati sono recenti (o frequenza di aggiornamento dei dati), e *volatility*, la quale misura per quanto tempo i dati rimangono validi. In (REDMAN 1996) viene definita solo la currency, mentre in (WANG & STRONG 1996), (NAUMANN *et al.* 1999) e (NAUMANN 2002) solo timeliness.
- INTERPRETABILITY. Riguarda il formato con cui i dati sono specificati e la chiarezza (non ambiguità) della definizione dei dati. Tale dimensione è pertanto molto simile a *Understandability/Ease of understanding*, ma sia (WANG & STRONG 1996) che (NAUMANN 2002) le considerano dimensioni distinte. Inoltre, in (REDMAN 1996) vengono proposte due diverse dimensioni: *interpretability* e *clarity of definition*. Infine, anche in (JARKE *et al.* 1999) si effettua una distinzione, che è quella tra l'interpretability dei dati e degli schemi.
- RELIABILITY, CREDIBILITY, BELIEVABILITY, REPUTATION E OBJECTIVITY. Si tratta di dimesioni strettamente relazionate. In (WAND & WANG 1996) la *reliability* indica se i dati possono essere considerati affidabili per una determinata informazione. Un definizione completamente diversa è fornita in (JARKE *et al.* 1999), dove è definita come la frequenza di fallimenti di un sistema (fault tolerance). In NAUMANN *et al.* (1999), la reliability esprime invece un giudizio sull'accuratezza dei metodi con cui i dati sono stati prodotti.

Per quanto riguarda la dimensione *credibility*, in (JARKE *et al.* 1999) essa rappresenta la credibiltà della sorgente che fornisce l'informazione. In (BOVEE *et al.* 2001) la credibility misura quanto l'informazione è accurata, completa, consistente.
Believability, *Reputation* e *Objectivity* sono definite in modo simile in (WANG & STRONG 1996) e in (NAUMANN 2002). Believability è il grado di accettazione delle informazioni come veritiere, reali e credibili; reputation rappresenta il grado con cui i dati sono veritieri o garantiti in termini della loro sorgente o del loro contenuto; infine, objectivity misura il livello di imparzialità dei dati. Si nota che anche in (NAUMANN *et al.* 1999) viene fornita una definizione di reputation, simile alla precedente; in particolare, con la reputation gi esprime un giudizio sulla sorgente basato sulla preferenza personale dell'utente o sulle sue esperienze professionali.

- RELEVANCY/RELEVANCE. Tale dimensione è presente nella maggior parte delle proposte. La *relevancy* misura quanto i dati sono rilevanti per un determinato processo o operazione. Una definizione leggermente diversa è data da (BOVEE *et al.* 2001), nella quale i dati sono rilevanti se sono recenti e se soddisfano i criteri imposti dall'utente. In (NAUMANN *et al.* 1999) la definizione di relevancy è data in funzione delle query che una sorgente può computare; in particolare la rilevanza è definita come la percentuale di oggetti del mondo reale rappresentati nella sorgente informativa.
- ACCESSIBILITY. Esprime il grado di disponibilità dei dati e la facilità/velocità di recupero degli stessi. Per tale dimensione vi è un sostaziale accordo sulla sua definizione, così come per *Access Security* o *Security*, che implica che l'accesso ai dati sia effettuato in maniera sicura.
- VALUE-ADDED E CONCISE REPRESENTATION. Sono due dimensioni definite in (WANG & STRONG 1996) e (NAUMANN 2002). *Value-added* è misura i benefici che i dati forniscono all'utente. *Concise representation* misura invece il grado di compattezza dei dati.
- APPROPRIATE AMOUNT OF DATA. In (WANG & STRONG 1996) misura il livello di appropriatezza della quantità o del volume dei dati. Una definizione più specifica viene fornita in (Naumann 2002), il quale la definisce come la dimensione del risultato di una query, misurata in byte. In (Naumann et al. 1999) l'*amount of data* è definito come il numero di attributi nella risposta che non sono stati specificati nella query. Infine, una ulteriore definizione, diverse dalla precedenti, è fornita da REDMAN (1996), il quale il nome "appropriate amount of data" è utilizzato per riferirsi all'apropriatezza del formato dei dati.
- AVAILABILITY. E' definita in (JARKE *et al.* 1999), (NAUMANN *et al.* 1999) e (NAUMANN 2002) come come la disponibilità di una sorgente o di un sistema.
- PORTABILITY. REDMAN (1996) la definisce come l'indipendenza del sistema dal formato dei dati; per JARKE *et al.* (1999) rappresenta invece l'indipendenza del sistema dal software che gestisce i dati.
- RESPONSIVENESS/RESPONSE TIME. E' definita, in tutte le proposte che la considerano, come l'intervallo di tempo che intercorre tra la sottomissione di una query e la sua risposta.

In conclusione a questa analisi, si può affermare che le dimensioni condivise dalla maggior parte delle proposte sono quelle di accuracy, completeness, consistency, timeliness e interpretability. Pertanto, è possibile elaborare una definizione di base per data quality, che

includa le caratteristiche considerate principali dalle varie proposte: *si definisce la qualità dei dati come un insieme di dimensioni, tra le quali accuracy, completeness, consistency, timeliness e interpretability*.

§

Seppure l'utilizzo e la definizione delle dimesioni di accuracy, completeness, consistency, timeliness e interpretability sono condivise dalla maggioranza delle proposte avanzate, non è detto che tali dimesioni da sole permettano di caratterizare la data quality per un determinato sistema informativo. La scelta delle rimanenti dimensioni è un processo critico che deve considerare diversi requisiti e condizioni. Ad esempio, come già accennato precedentemente, tale scelta dipenda dal tipo di dati e dallo specifico sistema informativo. Si propone pertanto la seguente classificazione per le dimensioni, al fine di fornire una linea guida per la scelta delle dimensioni migliori per il proprio sistema. Inparticolare, le proposte analizzate precedentemente sono state raggruppate in base a 4 criteri:

- *Approccio nella definizione delle dimensioni*. Tale criterio considera il processo seguito per la definizione dell'insieme delle dimensioni. Si considerano tre tipi di approcci: *Teoretico*, *Empirico* e *Intuitivo*. L'approccio teoretico base la definizione sulla proposta di un modello formale o una teoria. L'approccio empirico consiste nel costruire le dimensioni partendo da esperimenti, interviste o questionari. Nell'approccio intuitivo le dimensioni sono definite semplicemente in base al buon senso.
- *Punto di vista di modellazione*. In questo caso si considera la prospettiva sui dati adottata per definire le dimensioni. Vi sono tre possibili prospettive: *Schema*, *Istanza* e *Formato*. Schema si riferisce alla prospettiva intensionale, come la definizione di una tabella in un modello relazionale; la prospettiva istanza è relativa all'estensione dei dati, ovvero i valori attuali; il formato riguarda invece la rappresentazione dei dati.
- *Punto di vista di misurazione*. Questo criterio considera il modo con cui i dati sono analizzati al fine di valutare la loro qualità. Anche in questo caso si distinguono tre diverse viste: *Processo*, *Sistema* e *Prodotto*. La prima prospettiva considera il modo in cui i processi che producono i dati influiscono sul data quality. La prospettiva sistema considera che l'intero sistema informativo influenzi la qualità dei dati. Ad esempio, se si considera un sistema informativo distribuito, bisogna porre attenzione alle dimensioni relative al tempo e a come i loro valori sono influenzati dallo scambio di dati tra nodi diversi. Infine, la prospettiva prodotto riguarda in modo specifico i dati e la qualità percepita dall'utente. Questo implica che i dati vengano visti come un prodotto da rilasciare ai consumatori e tipicamente include dimensioni soggettive come interpretability o understandability.
- *Dipendenza da un contesto*. Con questo criterio si vuole sottolineare il fatto che certe proposte sono state elaborate per essere applicate a contesti specifici.

  In tal caso, la classificazione prevede due valori distinti: *Yes*, indica che le dimensioni sono relative ad un contesto specifico, *No* indica che la proposta è general-purpose.

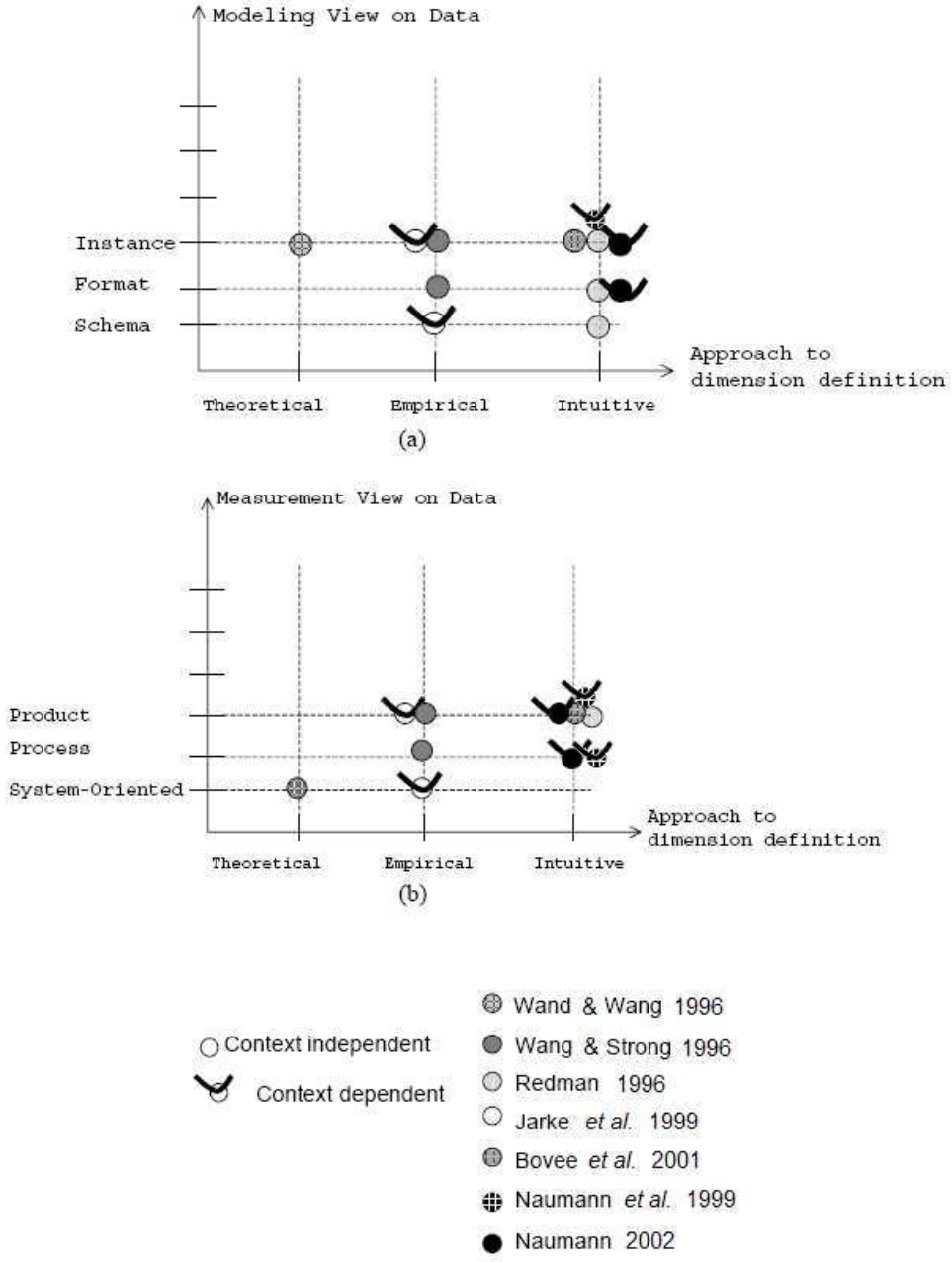


FIGURA 3. Una classificazione delle proposte di dimensioni per data quality

La posizione delle varie proposte rispetto le classificazione descritta è riportata in figura 3. La figura 3a mostra la posizione di ogni proposta rispetto il piano *Approccio nelle definizione delle dimensioni - Punto di vista di modellazione*; si nota che la sola proposta che copre tutti i punti di vista di modellazione sui dati è quella di REDMAN (1996). La proposta

di JARKE *et al.* (1999) è specifica per il contesto dei data warehouse, mentre (NAUMANN *et al.* 1999) e (NAUMANN 2002) per sistemi di integrazione delle informazioni (l'ultima delle due in particolare è specifica per web integration system). In figura 3b è riportata la ripartizione delle proposte nel piano *Approccio nelle definizione delle dimensioni - Punto di vista di misurazione*. In questo caso, nessuna proposta copre tutti i punti di vista di misurazione sui dati.

I vari *approcci alla definizione delle dimensioni* danno infomazioni utili per la scelta del migliore insieme di dimensioni. Un approccio intuitivo può essere suficiente se è richiesta solo una introduzione generale al problema del data quality; quando invece la qualità delle informazioni è alla base di decisioni strategiche per il sistema, allora sono più adeguati approcci teoretici o emipirici.

I diversi *punti di vista di modellazione sui dati* aiutano a capire il giusto livello di astrazione da adottare. Ad esempio, se si considerano informazioni non strutturate, come documenti testuali, le dimensioni che si concentrano sullo schema non sono utili. Se invece si considerano immagini, le dimensioni specifiche per il formato sono le più appropriate.

Per quanto riguarda i *punti di vista di misurazione*, quello di processo consente di focalizzarsi sul miglioramento della qualità all'interno di una organizzazione, investingando sulle cause di bassa qualità. Il punto di vista di sistema diventa importante quando un particolare sistema informativo supporta una organizzazione. Il punto di vista di prodotto è particolarmente significativo quando i dati sono il prodotto reale dell'azienda, come nel caso delle pubblica amministrazioni, il cui compito principale è quello di gestire dati riguardo cittadini e aziende.

In fine, anche la *dipendenza da un contesto* è di significativa importanza, in quanto se un insieme di dimensioni è progettato per un certo dominio applicativo, allora esso rappresenta la miglior scelta per quel dominio.

3. Conclusioni

In questo capitolo è stato introdotto il concetto di data quality. Come si è potuto capire, si tratta di una area multidisciplinare in quanto i dati, in diversi formati e per diversi scopi, sono utilizzati in molte operazioni della vita quotidiana o nelle attività lavorative.

Si è inoltre visto come tale concetto sia di non semplice definizione, in quanto diverse definizioni sono formulate per ogni disciplina che studia tale problema. Anche dal punto di vista informatico, trattato in questa tesi, entrano in gioco diversi fattori che impediscono un accordo comune sul concetto di qualità. Uno dei problemi principali riguarda la definizione delle dimensioni utilizzate per valutare la qualità dei dati. Come visto, tale scelta dipende principalmente dal tipo dei dati in esame e dal tipo di sistema informativo. Al fine di guidare nella scelta della definizione di qualità dei dati maggiormente adeguata alle proprie esigenze, sono state comparate alcune proposte di dimensioni ed è stata elaborata una classificazione di tali dimensioni. Dal confronto è risultato che alcune dimensioni, quali l'accuratezza, la completezza, la consistenza e quelle relative al tempo, sono tra le più utilizzate e sono sempre definite in modo simile nelle varie proposte.

CAPITOLO 7

Data Quality in NeP4B

1. Introduzione

A causa della natura aperta dei sistemi P2P, chiunque può entrare a far parte di una rete di peer. Ciò comporta due problemi diversi, ma correlati fra loro: la gestione della credibilità e la qualità dei dati. La gestione della credibilità, già indicata da Daswani come uno dei problemi aperti dei sistemi P2P (DASWANI *et al.* 2003), consente ai peer di identificare altri peer di cui si fidano e interagire solo con tali peer. Ogni sorgente dati può in principio influenzare il risultato finale integrato e dunque deve essere controllata: ad esempio, è necessario proteggere il sistema dalla diffusione di informazioni che provengono da sorgenti non affidabili. I meccanismi di credibilità sono strettamente legati alla qualità dei dati disponibili ai peer. I dati sono spesso affetti da problemi di qualità, che, come visto precedentemente, vanno da una scarsa accuratezza a livello sintattico (i dati sono errati, ad esempio a causa di errori commessi nella fase di raccolta delle informazioni), alla presenza di vari formati di dati o di inconsistenze, sia in una singola sorgente sia in diverse sorgenti, sino a problemi relativi alla loro diffusione.

Data la natura aperta di tali reti, nessun tipo di restrizione può essere applicata sulla qualità dei dati condivisi dai peer. Perciò, è essenziale che il controllo della qualità dei dati venga applicato sia attraverso i meccanismi di credibilità sopra descritti, che attraverso adeguate misure che prendano adeguatamente in considerazione la qualità nel processo di interrogazione. In molti scenari applicativi è infatti necessario che chiunque acceda alle informazioni abbia la possibilità di recuperare dati di elevata qualità o, se possbile, ai dati migliori disponibili nella rete.

§

Analogamente al caso dei sistemi P2P sopra descritto, uno degli obbiettivi del progetto NeP4B è quello di poter fornire un supporto efficace all'accesso delle sorgenti di informazione, in un contesto dove i peer possono evolvere nel tempo; tale accesso deve tenere conto anche della qualità dei dati, necessaria per fornire risposte migliori alle query.

A questo fine, all'interno del WP2, verrà studiato il modello dati più adatto per associare metadati sulla qualità ai dati e all'ontologia del peer; in particolare, si vuole definire una metodologia che, partendo da metriche di qualità, valutate tramite techniche statistiche e di machine learning esistenti, produca un modello adatto a rappresentare l'asociazione tra le metriche di qualità e le istanze dati e l'ontologia del peer.

In NeP4B, le funzioni del data quality sono molteplici e riguardano il query processing, trust management e object fusion.

Data Quality e Query Processing. Uno degli obbiettivi del progetto è quello di studiare soluzioni per il processing efficace ed efficiente di query sull'ontologia di dominio, attraverso la riscruttura verso le sorgenti informative. In NeP4B si deve quindi permettere agli utenti di specificare determinati vincoli di qualità nelle query e il query manager deve ritornare solo i dati che rispettano tali vincoli: gli algoritmi che verranno sviluppati devono dunque supportare un query processing di tipo *quality-aware*.

Data Quality e Object Fusion. Il data quality verrà utilizzato come supporto alle tecniche di integrazione dei dati e risoluzione delle inconsistenze. Infatti, in un contesto peer-to-peer, è molto probabile avere differenti istanze dello stesso oggetto, le queli possono essere inconsistenti tra loro, che dovranno essere identificate per raggruppare le informazioni riguardanti uno stesso oggetto presente in diversi repository. Il data quality può quindi essere utilizzato come strategia di risoluzione delle inconsistenza, che in NeP4B avviene a tempo di query.

Data Quality e Peer Trust Management. Tra gli obbiettivi economici e organizzativi del progetto vi è quello di sviluppare un modello di trust coerente con lo scenario in cui diversi attori possono incontrarsi in un ambiente virtuale e devono decidere se un partecipante è affidabile. I metadati di qualità associati alle istanze di dato e all'ontologia dei peer possono servire come fondamento per sviluppare tecniche di reputazione che consentano ai peer di esprimere le loro opinioni su altri peer.

§

La qualità dei dati nelle sorgenti integrate è dunque un problema rilevante: alcuni recenti metodi, discussi di seguito, che si occupano dei problemi della qualità delle informazioni, per esempio risolvendo i conflitti a livello d'istanza o incorporando le misure di qualità nel processo di selezione delle sorgenti, sono stati proposti per le tradizionali architetture centralizzate wrapper-mediator. Questi metodi devono essere estesi al contesto P2P. Inoltre, lo stato dell'arte sulla pulizia dei dati, che include schema matching, data scrubbing, riconoscimento dei duplicati e data fusion, è eseguito per applicazioni specializzate che sono costose da installare, configurare e manutenere, dunque sono necessarie nuove strategie più leggere per la pulizia dei dati.

Il problema della qualità dei dati, come visto, è affrontato con approcci diversi e tutt'ora non vi è un accordo comune su quale di questi sia il migliore. Tutti i ricercatori impegnati in questo campo concordano comunque sul fatto che le scelte progettuali dipendono dal contesto e dal sistema in questione.

Come visto, uno dei primi problemi è quello di definire un modello dei dati che permetta di associare metadati della qualità alle istanze di dato e alle ontologie dei peer. Questo problema non può prescindere dallo studio e dalla definizione delle dimensioni e metriche utilizzate per modellare i valori di qualità; è inoltre necessario scegliere un opportuno formato per lo scambio di tali informazioni.

Nel seguito del capitolo si cercherà dunque di definire il problema del data quality relativamente al progetto NeP4B, considerando le aree nel quale esso interviene e il contesto in cui dovrà operare. Verranno pertanto analizzati e confrontati alcuni dei sistemi attualmente esistenti. Infine, verrà proposta una implementazione relazionale per il sistema MOMIS del modello D^2Q proposto all'interno del progetto DaQuinCIS (SCANNAPIECO *et al.* 2004).

2. Il progetto DaQuinCIS

Nell'ambito dei Cooperative Information System (CIS), ovvero di sistemi informativi distibuiti su vasta scala che interconnettono vari sistemi di diverse e autonome organizzazioni distribuite geograficamente e che condividono gli stessi obbiettivi, il problema della gestione della qualità dei dati è affrontato nel progetto *Methodologies and Tool for Data Quality inside Cooperative Information Systems*[1] (DaQuinCIS).

DaQuinCIS è un progetto nazionale co-finanziato dal MURST e che vede la partecipazione del Dipartimento di Informatica e Sistemisticha dell'Università di Roma "La Sapienza", del Dipartimento di Elettronica e Informazione del Politecnico di Milano e del Dipartimento di Informatica Sistemistica e Comunicazione dell'Università di Milano Bicocca. Gli obbiettivi di tale progetto sono principalmente quelli di definire un framework integrato che includa una metodologia e un'architettura distribuita per il monitoraggio e il miglioramento della qualità dei dati nei CIS.

Nel seguito della sezione verranno presentati l'architettura e i componenti principali di DaQuinCIS e il modello dei dati utilizzato per la reppresentazione dei dati e dei relativi dati di qualità, chiamato *Data and Data Quality* (D^2Q) model.

§

ARCHITETTURA DI DAQUINCIS. L'architettura proposta per la gestione dei dati di qualità consente la diffusione dei dati e la relativa qualità e il miglioramento della qualità dati all'interno del sistema. Ogni organizzazione offre servizi alle altre organizzazioni sul proprio gateway e altri servizi specifici per il proprio sistema di back-end. Per questo motivo il gateway si interfaccia internamente ed esternamente attraverso servizi. Altri servizi sono offerti dall'infrastruttura di comunicazione. In tale architettura, i servizi sono tutti identici nel senso che sono istanze dello stesso software; inoltre sono "peer", ovvero includono funzionalità sia di client che di server, utilizzate a seconda degli obbiettivi da perseguire. L'architettura generale del sistema è riportata in figura 1.

L'architettura prevede l'utilizzo di un modello comune a tutte le organizzazioni per l'esportazione dei dati e della relativa qualità, il modello *Data and Data Quality* (D^2Q). Tale modello include la definizione di (i) costrutti per rappresentare i dati, (ii) un insieme comune di proprietà dei dati di qualità, (iii) i costrutti per rappresentare tali informazioni e (iv) le associazioni tra dati e la rispettiva qualità. Il modello D^2Q sarà descritto nel prossino paragrafo.

Oltre al gateway e al modello D^2Q, l'architettura di DaQuinCIS prevede altri importanti componenti, descritti di seguito.

- QUALITY FACTORY. E' quel servizio implementato in ogni organizzazione della rete il cui compito è quello di valutare la qualità dei dati dell'organizzazione. Essendo distribuito in ogni nodo della rete, si evita che la valutazione della qualità sia affidata ad un'unica entità centralizzata. Oltre ai vantaggi di efficienza e all'eliminazione di un singolo punto di rottura, in questo modo è anche possibile, per ogni organizzazione, modificare gli algoritmi di calcolo della qualità.

[1] http://www.dis.uniroma1.it/~dq/

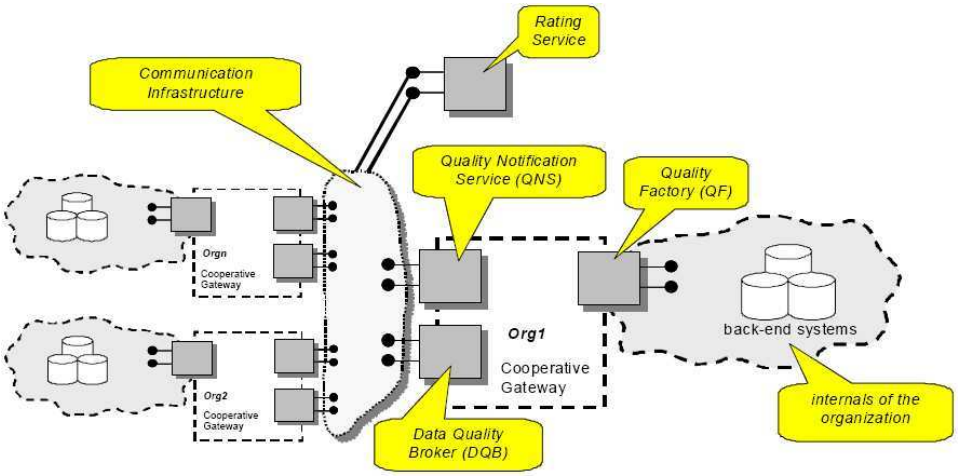


FIGURA 1. Architettura di DaQuinCIS

L'architettura del Quality Factory (figura 2a) è composta da quattro moduli principali: *Quality Analizer*, *Quality Assessment*, *Monitoring* e *Data Quality Certificate*. Il Quality Analyzer ricevere le richieste da parte degli utenti e identifica in base ad esse i dati da recuperare e le dimensioni da valutare mentre il processo di misurazione della qualità è eseguito dal Quality Assessment. Il modulo Monitoring è utilizzato nella valutazione off-line: esso contiene tutte le definizioni delle regole degli eventi che attivano la valutazione.. Infine, il Data Quality Certificate garantisce che i risultati di una richiesta siano conformi ai vincoli di data quality specificati. DaQuinCIS fornisce due possibili modalità per calcolare la qualità dei dati, valutazione *on-line* e *off-line*, permettendo alle organizzazioni di poter scegliere la loro.

- QUALITY KNOWLEDGE REPOSITORY. Consiste di una knowledge base utilizzata dagli altri componenti per eseguire le loro funzioni. In particolare esso mantiene la conoscenza interschema, la quale permette di rappresentare le relazioni tra schemi che consentono di individuare le equivalenze intensionali ed estensionali tra dati in differenti organizzazioni. Inoltre, in esso vengono mantenute informazioni statistiche relative ai dati di qualità forniti in passato dalle organizzazione, al fine di poter sviluppare servizi di rating per valutare la fiducia in una sorgente.
- DATA QUALITY BROKER. Il compito di questo componente è quello di porre le richieste di informazioni degli utenti alle altre organizzazioni cooperanti; tali richieste possono includere un insieme di requisiti di qualità che i dati desiderati devono soddisfare. Tale compito è chiamata funzione di *quality brokering*. Una seconda funzione, chiamata *quality improvement*, permette, al fine di riconciliare copie diverse degli stessi dati ricevuti in risposta alla richiesta, di selezionare la copia con qualità più elevata e di proporla alle organizzazioni, le quali possono rifiutarla o aggiornare la loro copia a quella più qualitativa.

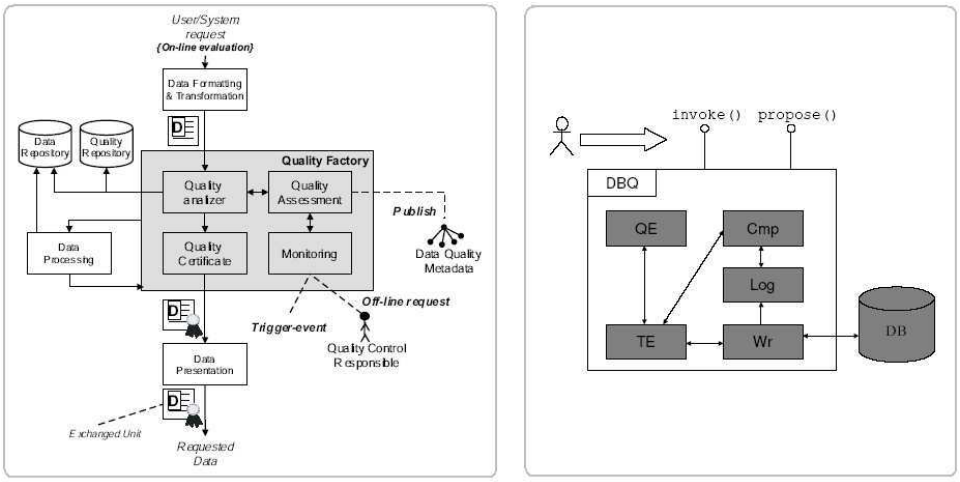


FIGURA 2. Architettura (a) del Quality Factory e (b) del Data Quality Broker

Il Data Quality Broker è sostanzialmente un sistema peer-to-peer di data integration (LENZERINI 2002) il quale permette di porre query quality-aware formulate rispetto uno schema globale e di selezionare le risposte che soddisfano tali requisiti. Dal punto di vista architetturale, il broker è implementato come un servizio peer-to-peer, il quale è posizionato in ogni organizzazione del sistema, ed è costituito da diversi componenti quali: il *Query Engine*, responsabile dell'esecuzione e dell'ottimizzazione delle query, il *Wrapper*, il quale traduce la query dal linguaggio utilizzato dal broker a quello utilizzato dalla specifica sorgente, il *Transport Engine*, che fornisce delle facility per il trasferimento delle query e dei loro risultati tra wrapper e query engine, *Schema Mapper*, che memorizza i mapping (statici) tra classi globali e locali e il *Comparator*, il cui compito è quello di comparare i diversi risultati ottenuti al fine di selezionare la copia di qualità maggiore. L'architettura generale del data quality broker è mostrata in figura 2b.

- QUALITY NOTIFICATION SERVICE. Si tratta di un engine di sottoscrizione/pubblicazione utilizzato come un bus di messaggi riguardo la qualità dei dati tra servizi e/o organizzazioni. Più specificatamente, consente di notificare agli utenti che si sono sottoscritti al servizio il cambiamento della qualità dei dati di loro interesse. Ad esempio, un'organizzazione può voler sapere se i dati che essa usa degradano sotto un certo valore soglia o quando dati di qualità elevati sono disponibili. Come per il Data Quality Broker, anche il Quality Notification Service è implementato come un sistema peer-to-peer, dove ogni organizzazione mantiene un'istanza del servizio.
- RATING SERVICE. Associa dei valori di fiducia ad ogni sorgente dati nel sistema. Questi valori sono utilizzati dal Data Quality Broker per determinare l'affidabilità della valutazione della qualità effettuata dalle organizzazioni. Gli indici di fiducia sono aggiornati automaticamente dalle statistiche dal Quality Knowledge Repository e tenendo anche in considerazione la disponibilità corrente della sorgente. In particolare, il modello di trust di DaQuinCIS prevede

che ogni organizzazione Org_i che ha interagito con una seconda organizzazione Org_j ricevendo da quest'ultima dei dati, associ un valore di fiducia alla coppia <Org_j, D>, dove D rappresenta l'insieme dei dati ricevuti, è invii al servizio di rating tale valore. Al contrario degli altri componenti dell'architettura, il servizio di Rating è centralizzato e non distribuito sui nodi della rete.

- ❖ QUALITY IMPROVER. Tale componente rappresenta un insieme di tool che consentono alle organizzazione di migliorare la qualità dei propri dati. Ad esempio, contiene il *Record Matcher* e il *Record Improver* che periodicamente vengono eseguiti per riconciliare i dati all'interno del sistema. A fronte di un insieme di dati ottenuti in risposta ad una query, il Record Matcher permette di riconoscere istanze diverse dello stesso oggetto e il Record Improver sceglie l'istanza di qualità più elevata. Il primo tool è basato su un algoritmo derivato da Sorted Neighborhood Method e sulla scelta di opportune matching key.

Query Processing e Object Fusion. Il processo di query answering è una delle principali funzionalità offerte dal Data Quality Broker. L'idea sul quale si fonda l'approccio di DaQuinCIS è quella di fare esportare sia i dati che si intendono scambiare ma anche i metadati che caratterizzano la loro qualità. Per riuscire in questo intento, è stato sviluppato uno specifico modello dei dati, D^2Q, che verrà illustrato successivamente. Sulla base di questa caratterizzazione della qualità dei dati esportati, le query vengono processate in modo che la risposta con qualità migliore sia ritornata come risultato.

Le query sullo schema globale sono processate secondo l'approcio *global-as-view* (*GAV*) attraverso il processo di unfolding, cioè sostituendo ogni atomo della query originale con la corrispondente vista sulle sorgenti locali. Quando si definiscono i mapping tra concetti dello schema globale e quelli dello schema locale, mentre l'estensione dei concetti globali può essere recuperata da sorgenti multiple, i mapping sono definiti per recuperare l'unione delle estensioni delle sorgenti locali. Tale definizione di mapping dipende dall'assunzione che lo stesso concetto può avere estensioni diverse a livello di sorgente locale. Quindi, quando i dati vengono recuperati, possono essere confrontati e può essere selezionata, o costruita, la copia di qualità migliore.

In particolare, il query processing in DaQuinCIS consiste dei seguenti step:

(1) QUERY UNFOLDING. Una query globale Q viene scomposta in base ai mapping statici che definiscono ogni concetto dello schema globale in termini delle sorgenti locali; questi mapping sono definiti per poter recuparare tutte le copie dello stesso dato che sono disponibili, le quali vengono esportate in accordo al modello D^2Q. Quindi, la query Q è decomposta nelle query Q_1, ..., Q_k poste sulle sorgenti locali. Queste query vengono successivamente poste alle rispettive sorgenti; le sorgenti eseguono le query e ritornano un insieme di risultati R_1,. . . ,R_k (vedi figura 3).

(2) EXTENSIONAL CHECKING. In questo step, viene applicato un algoritmo di record matching sull'insieme $R_1 \cup R_2 \cup \ldots \cup R_k$. Il risultato dell'esecuzione di tale algoritmo è la costruzione di un insieme di cluster composti dai record che si riferiscono agli stessi oggetti del mondo reale C_1,. . . , C_z (figura 3, centro).

(3) RESULT BUILDING. Il risultato è costruito in base alla *best quality default semantics*. Per ogni cluster, viene selezionato, o costruito, il valore di qualità migliore. Ogni record nel cluster è composto da coppie alle quali un *quality value* `q` è associato ad ogni *field value* `f`. Per ogni cluster, viene selezionato, se esiste, il record che ha i valori di qualità migliore per ogni campo.

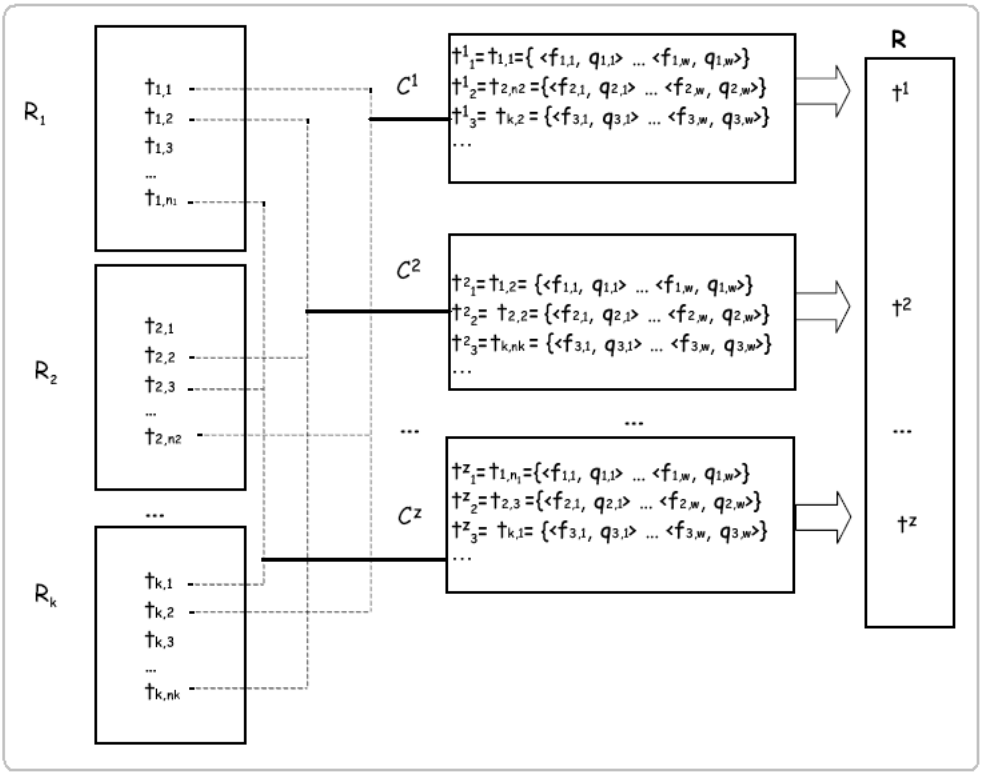


FIGURA 3. Costruzione dei risultati in DaQuinCIS

Se tale record non esiste, viene costruito componendo i campi dai record del cluster che hanno la qualità migliore. Quando i record rappresentativi di ogni cluster sono selezionati, il risultato *R* è costruito come l'unione di tutti i rappresentativi dei cluster (figura 3, destra). Ogni valore di qualità q è un vettore di di valori, uno per ogni dimensione. Ad esempio, q può contenere i valori per accuranzy, currency, completeness e consistency.

3. Il modello D²Q

Tutte le organizzazioni del sistema DaQuinCIS esportano i loro dati applicativi e di qualità in modo conforme al modello *Data and Data Quality* (*D²Q*) (SCANNAPIECO *et al.* 2004). In particolare, tale modello definisce i tipi di dato e le dimensioni di qualità consentiti; inoltre definisce sia il modello con cui esportare i dati, quello con cui esportare i dati di qualità e per realizzare le associazioni tra elementi nei due modelli.

Dimensioni. Le dimensioni utilizzate nel sistema DaQuinCIS sono 23 le quali vengono raggruppate in 4 categorie (CAPPIELLO *et al.* 2002):

- SUBJECT CATEGORY: include tutte le dimensioni la cui valutazione dipenda dall'utente.

- ❖ OBJECT CATEGORY: include tutte le dimensioni intrinseche e tutte quelle dimensioni misurabili semplicemente considerando i dati.
- ❖ ARCHITECTURAL CATEGORY: riguarda le dimensioni relative all'architettura che supporta un CIS.
- ❖ PROCESS CATEGORY: include le dimensioni contestuali e tutte le dimensioni specifiche dei processi cooperativi.

Le dimensioni specifiche di ogni categoria sono riportate in tabella 1. Per motivi di semplicità, verranno proposte, nel resto del capitolo, solo 4 dimensioni considerate necessarie per i CIS (SCANNAPIECO *et al.* 2004; MECELLA *et al.* 2002), ovvero *accuracy*, *completeness*, *currency* e *internal consistency*. Tali dimensioni corrispondono anche a quelle identificate, nel capitolo precedente, come le uniche adottate in ogni proposta analizzata.

Accuracy. Per *accuracy* si intende la distanza tra i valori *v* e *v'*, dove *v'* è il valore considerato corretto. Si consideri il seguente esempio: `Student` è uno schema element (ovvero una classe di UML o un'entità del modello E/R, etc.) con un attributo `Name` e `p` è un'istanza di `Student`. Se `p.Name` ha un valore *v* = NDREA, mentre *v'* = ANDREA, allora il valore corrispondente di accuracy sarà basso in quanto NDREA non è un nome ammissibile nella lingua italiana.

Tale dimensione è relativamente semplice da calcolare per valori alfanumerici, in quanto è possibile confrontare tali valori con dizionari di riferimento attraverso la edit distance; per valori numerici tale compito è più difficile, in quanto è necessario definire nozioni di edit distance più complesse.

Completeness. Per *completeness* si intende il grado in cui i valori di uno schema element sono presenti nelle istanze dell'elemento stesso; essa rappresenta dunque *l'estensione per la quale i dati non sono mancanti e sono di sufficiente respiro e profondità per le operazioni da gestire* (PIPINO, LEE & WANG 2002).

In base alle definizione si possono distinguere: (i) la completeness di un valore di un attributo, dipendente dal fatto che l'attributo sia o meno presente; (ii) la completeness di un'istanza di uno schema element, dipendente dal numero di valori di attributi presenti.

Altri tipi di completeness sono stati definiti in (PIPINO, LEE & WANG 2002), come schema completeness, column completeness e population completeness, i quali però non sono stati considerati in D2Q, in quanto l'obbiettivo non è quello di avere informazioni per una valutazione generale della qualità dei dati memorizzati da una organizzazione, bensì di poter valutare la qualità di ogni singolo dato.

Nella valutazione della completeness è necessario saper gestire i valori null: per attributi mandatori, un valore null è associato ad un basso livello di completeness; al contrario, per attributi opzionali o inapplicabili la completeness non è influenzata dalla presenza di valori nulli.

Ad esempio, si consideri l'attributo `E-mail` dell'elemento `Student`: un valore null può avere diverse semantiche. Se lo studente in questione non ha un indirizzo e-mail, cioè l'attributo è inapplicabile, la completeness non ne risulta influenzata. Se, al contrario, lo studente possiede un indirizzo e-mail che però non è strato memorizzato, allora la completeness sarà bassa.

Categoria *Subject*	
Interpretability	Misura quanto i dati sono presenti nel linguaggio appropriato e quanto le definizioni sono chiare.
Easy of understanding	Misura quanto i dati sono chiari e semplici da comprendere.
Concise representation	Misura quanto i dati sono rappresentati in maniera compatta, senza compromissione della completezza e dell'accuratezza.
Accessibility	Misura la disponibilità e la facilità di reperimento dei dati.
Categoria *Object*	
Believability	Misura quanto i dati sono considerati credibili o reali o veritieri.
Accuracy	Misura la correttezza dei dati.
Objectivity	Misura quanto i dati sono imparziali.
Reputation	Fiducia valutata per i dati, in termini della loro sorgente o del loro contenuto.
Representational Consistency	Misura quanto i dati sono presenti nello stesso formato e sono compatibili con i dati precedenti.
Internal Consistency	Proprietà per la quale i valori di una entità non si contraddicono fra loro.
Data Completeness	Misura sia la completezza del valore di un attributo, sia la completezza dell'istanza di una entità.
Categoria *Architectural*	
Availability	La capacità dell'architettura di fornire i dati richiesti.
Responsiveness	La capacità dell'architettura di fornire tempestivamente i dati richiesti.
Source Availability	La capacità di una sorgente informativa di fornire specifici dati in risposta ad una rischiesta.
Source Responsiveness	La capacità della sorgente di fornire i dati richiesti tempestivamente.
Categoria *Process*	
Relevancy	Quanto i dati sono utili per una determinata azione.
Timeliness	Quanto i dati sono recenti.
Appropriate Amount of Data	Misura se la quantità dei dati disponibili è appropriata.
Process Completeness	Misura quanto la portata dei dati è utile per l'azione da compiere.
Value-added	Misura se l'utilizzo dei dati fornisce un vantaggio.
Access Security	Misura quanto l'accesso ai dati può essere vincolato e limitato.
History	Descrivere la storia delle manipolazioni applicate ai dati.
Cost	Valuta quanto "costano" gli errori dovuti a scarsa qualità.

TABELLA 2. Categorie e relative dimensioni della qualità definite dal modello D^{2}Q

Currency. Per *currency* si intende la distanza tra l'instante in cui un valore cambia nel mondo reale e l'istante in cui il corrispondente valore memorizzato nel sistema informativo

viene modificato.

Tale dimensione si riferisce solo a quei valori che possono variare nel tempo, come, ad esempio, Address; DateOfBirth è un attributo che può essere invece considerato invariante. Per misurare questa dimensione è possibile associare ad ogni valore un *updating time-stamp* oppure un *transaction time* analogamente a quanto avviene nei database temporali.

Si nota inoltre che tale dimensione non è presente in Tabella 1: essa è infatti una sottodimensione di *timeliness*, come *volatility* e *currency level*.

Internal Consistency. La consistenza implica che due o più valori non sia in conflitto con ogni altro valore. *Internal consistency* significa che tutti i valori da comparare per valutare la consistenza sono contenuti in una specifica istanza di uno schema element. Più specificatamente, per internal consistency si intende il grado con cui i valori degli attributi di una istanza di uno schema element soddisfano l'insieme di semantic rules definite sullo schema element. Per semantic rule si intende un qualsiasi constraint che vincola i valori degli attributi di uno schema element.

Se si considera, ad esempio, uno schema element `Citizen` con attributi `DateOfBirth` e `DateOfDeath` allora una possibile semantic rule è quella che implica il fatto che `DateOfBirth` precede sempre `DataOfDeath`. Se una data istanza di `Citizen` non dovesse rispettare tale regola, il corrispondente valore di internal consistency sarebbe basso.

Tipi di dato. Per quanto riguarda i tipi di dato, il modello distingue tra *base data type* e quality type. Un *base data type* è un identificatore il quale denota il dominio dei valori trattati dal modello. D^2Q prevede come tipi di dato di base quelli tipici dei linguaggi di programmazione e di SQL, ovvero: Integer, Real Boolean String Date, Time, Interval, Currency, Any. I quality type sono dei tipi di dato speciali i quali caratterizzano il dominio dei valori che uno specifico valore di qualità può assumere. Per questo motivo, ad ogni dimensione considerata, dim_i, viene assegnato uno specifico data type, τ_{dim_i}. Sono quindi stati definiti i seguenti tipi di dato:

- $\tau_{accuracy}$, relativo alla dimensione accuracy. Per quanto concerne il range di valori supportati, si possono definire tre valori diversi, "low", "medium" e "high" i quali indicano rispettivamente bassa, media e alta accuratezza.
- $\tau_{completeness}$, relativo alla dimensione completeness. Anche per tale dimensione si possono prevedere due valori "false" e "true" per indicare l'incompletezza e la completezza di un attributo.
- $\tau_{consistency}$, relativo alla dimensione internal consistency. Tale dimensione supporta due possibili valori "false" e "true", relativi al risultato della valutazione di regole semantiche.
- $\tau_{currency}$, relativo alla dimensione currency. Per tale dimensione, si assume che essa venga espressa da un qualsiasi numero reale nell'intervallo 0 - 100.

Si nota infine che ogni dominio dei quality type prevede un simbolo speciale, $\perp$, per indicare che nessun valore di qualità è stato indicato per quella dimensione.

§

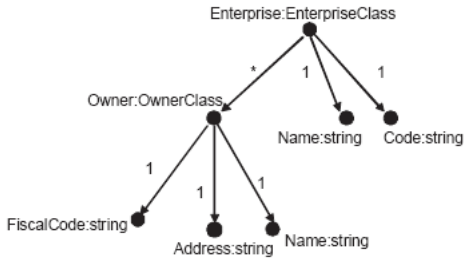


FIGURA 4. Esempio di D^2Q data schema

Il Data Model. Il modello D^2Q è un modello semi-strutturato, il quale è costituito da un *data model* e da un *quality model*. In generale, il modello è basato sul modello dei dati sottostante a XML-QL. Infatti, ogni modello conforme a D^2Q viene tradotto in un documento XML con un insieme di data items e una Document Type Definition (DTD), o un XML Schema, composta da data e quality class. In particolare, un documento XML $D^2Q - compliant$ contiene sia dati applicativi, nella forma di D^2Q *data graph*, e i relativi valori di qualità, nella forma di quattro D^2Q *quality graph*, uno per ogni dimensione considerata. I nodi del D^2Q data graph sono collegati ai rispettivi nodi del D^2Q quality graph attraverso link. Nel data model, presentato di seguito, viene fornita una definizione formale di data class e D^2Q data schema. Il quality model è definito invece nel prossimo paragrafo.

DEFINITION 3.1. (*Data Class*). Una *data class* $\delta(\pi_1, \ldots, \pi_n)$ consiste di:

- un nome δ;
- un insieme di proprietà $\pi_i =< name_i : type_i >$, $i = 1 \ldots n, n \geq 1$, dove $name_i$ è il nome della proprietà e $type_i$ può essere:
  - un tipo di dato di base;
  - oppure una data class;
  - oppure un tipo `set-of < X >`, dove `< X >` può essere un tipo di base o una classe.

DEFINITION 3.2. (D^2Q *Data Schema*). Un D^2Q *Data Schema* S_D è un grafo con le seguenti caratteristiche:

- un insieme di nodi N_D, rappresentati da data class;
- un insieme di foglie T_D, le quali sono tutte proprietà con tipo di base;
- un insieme di archi $\varepsilon_D \subseteq N_D \times (N_D \cup T_D)$;
- un insieme di label di nodi e foglie $L_D = L_{N_D} \cup L_{T_D}$, dove L_{N_D} è l'insieme dei label dei nodi costituiti da $\langle$`data class name: data class`$\rangle$; L_{N_D} è invece l'insieme dei label delle foglie del grafo, costituiti da $\langle$`property name: basic type`$\rangle$.
- Un insieme di label di archi L_{ε_D}, definito come segue. Sia $\langle n_1, n_2 \rangle \in \varepsilon_D$. Sia n_1 una data class e $l_{1,2}$ il label sull'arco $\langle n_1, n_2 \rangle$. Questo label rappresenta il numero di occorrenze di n_2, ovvero il label specifica la cardinalità di n_2 nella relazione con n_1.

Un D^2Q Data Schema deve inoltre rispettare le due seguenti proprietà:

- ❖ *Proprietà di chiusura*: Se S_D contiene una data class δ, la quale possiede come proprietà un'altra classe δ', allora anche δ' appartiene allo schema S_D.
- ❖ *Aciclicità*: In uno schema D2Q non esistono cicli.

In figura 3 è riportato un esempio di D^2Q data schema; lo schema descrive una generica impresa, `Enterprise`, con nome e codice (`Name` e `Code`) e con zero o più `Owner`, ognuno rappresentato da codice fiscale (`FiscalCode`), indirizzo (`Address`) e nome (`Name`).

Il Quality Model. Viene ora descritto il modello utilizzato per descrivere i dati di qualità nel sistema DaQuinCIS. In analogia con il data model, il $D^2Qqualitymodel$ definisce i concetti di *quality class* e *quality schema*. In particolare, una quality class viene associata ad ogni quality class e ad ogni proprietà. Nel seguito viene utilizzata la seguente notazione: se $\delta(name_\delta, \pi_1 \dots \pi_i \dots \pi_n)$, allora λ_δ è la relativa quality class associata, mentre $\lambda_{\delta::\pi_i}$ è la quality class associata alla proprietà (di tipo base) π_i.

DEFINITION 3.3. ($\lambda_{\delta::\pi_i}$ - Quality class associata al tipo base π_i). Sia $\delta(name_\delta, \pi_1, \dots, \pi_i, \dots, \pi_n)$una data class e π_i una propriotà, di tipo base, di δ. La quality class $\lambda_{\delta::\pi_i}$ è definita da:

- ❖ un nome $name_{\lambda_{\delta::\pi_i}}$;
- ❖ quattro proprietà $\kappa_{accuracy}$, $\kappa_{completeness}$, $\kappa_{consistency}$, $\kappa_{currency}$ tali che $(\kappa_{accuracy} : \tau_{accuracy}) \cup (\kappa_{completeness} : \tau_{completeness}) \cup (\kappa_{consistency} : \tau_{consistency}) \cup (\kappa_{currency} : \tau_{currency})$.

DEFINITION 3.4. (λ_δ - Quality class associata ad una data class). Sia $\delta(name_\delta, \pi_1 \dots \pi_i \dots \pi_n)$ una data class. Una quality class $\lambda_\delta(name_{\lambda_\delta}, \kappa_{accuracy}, \kappa_{completeness}, \kappa_{consistency}, \kappa_{currency}, \pi_1^\lambda, \dots, \pi_n^\lambda)$ consiste di:

- ❖ un nome $name_{\lambda_\delta}$;
- ❖ un insieme di proprietà $(\kappa_{accuracy}, \kappa_{completeness}, \kappa_{consistency}, \kappa_{currency}, \pi_1^\lambda, \dots, \pi_n^\lambda)$ tali che:
  - $\kappa_{accuracy}, \kappa_{completeness}, \kappa_{consistency}, \kappa_{currency}$ sono tali che $(\kappa_{accuracy} : \tau_{accuracy}) \cup (\kappa_{completeness} : \tau_{completeness}) \cup (\kappa_{consistency} : \tau_{consistency}) \cup (\kappa_{currency} : \tau_{currency})$;
  - $(\pi_1^\lambda, \dots, \pi_n^\lambda)$ è un insieme di proprietà $\pi_i^\lambda = \langle name_i^\lambda : type_i^\lambda \rangle$, che corrispondono biunivocamente a $(\pi_1, \dots, \pi_n)$ (proprietà di δ) ed è definito come:
    - se π_i è una proprietà di tipo base, allora $type_i^\lambda$ è $\lambda_{\delta::\pi_i}$, dove $\lambda_{\delta::\pi_i}$ è la quality class associata a $\delta :: \pi_i$;
    - se π_i è un insieme di tipi $\langle X \rangle$, dove X è un tipo base, allora $type_i^\lambda$ è un insieme di $\langle \lambda_{\delta::\pi_i} \rangle$, dove $\lambda_{\delta::\pi_i}$ è la quality class associata a $\delta :: \pi_i$;
    - se π_i è una data class δ', allora $type_i^\lambda$ è $\lambda_{\delta'}$, dove $\lambda_{\delta'}$ è la quality class associata a δ';
    - se π_i è un insieme $\langle X \rangle$, dove X è una data class δ', allora $type_i^\lambda$ è un insieme di $\langle \lambda_{\delta'} \rangle$, dove $\lambda_{\delta'}$ è la quality class associata a δ'.

DEFINITION 3.5. (D^2Q *Quality Schema*). Un D^2Q *Quality Schema* S_Q è un grafo diretto con le seguenti caratteristiche:

- ❖ un insieme di nodi N_Q, ognuno dei quali rappresenta una quality class;
- ❖ un insieme di foglie T_Q, le quali rappresentano proprietà di tipo quality type;

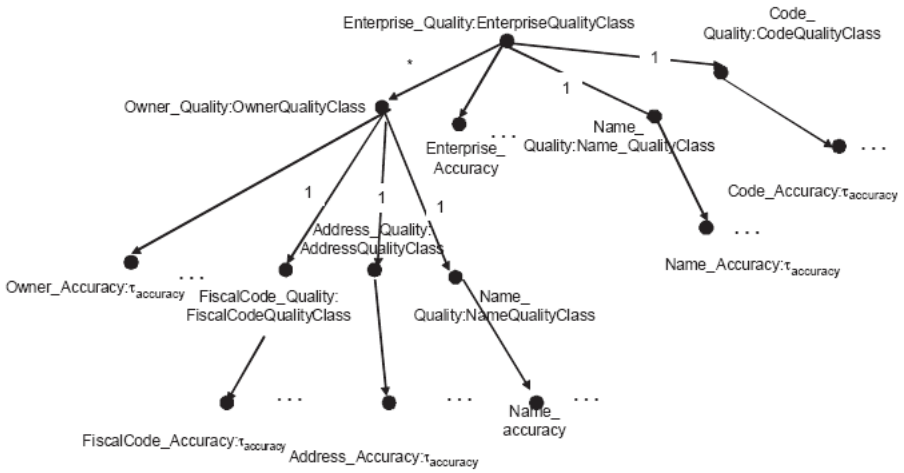


FIGURA 5. Esempio di un D^2Q quality schema

- un insieme di archi $\varepsilon_Q \subseteq N_Q \times (N_Q \cup T_Q)$;
- un insieme di etichette per nodi e label $L_Q = L_{N_Q} \cup T_Q$ dove L_{N_Q} è un insieme di etichette per nodi e ogni etichetta consiste di ⟨`quality class name`:`quality class`⟩; L_{T_Q} è l'insieme delle etichette delle foglie, le quali consistono di ⟨`quality property name`:`quality type`⟩;
- un insieme di etichette per gli archi L_{ε_Q} definiti come segue. Sia $\langle n_1, n_2 \rangle \in \varepsilon_Q$. Sia n_1 una quality class e sia $l_{1,2}$ una etichetta sull'arco $\langle n_1; n_2 \rangle$; tale etichetta è definita rispetto a n_2 come segue:
  - se n_2 è una foglia, allora $l_{1,2}$ è la stringa `NULL`, in quanto non abbiamo bisogno di specificare la cardinalità
  - se n_2 è una quality class, allora $l_{1,2}$ è il numero di occorenze di n_2 nella relazione con n_1.

In figura 5 è riportato l'esempio del D^2Q quality schema relativo al data schema di figura 4. Come si vede, lo schema è composto da due quality class, `Enterprise_Quality` e `Owner_Quality`, associate rispettivamente alle classi `Enterprise` e `Owner` del data schema. Nella figura vengono riportati anche i nodi relativi alle dimensioni di accuracy; si nota che anche i nodi relativi a completeness, currency e consistency sono presenti nel grafo, anche se non sono riportati. Ad ogni modo, è possibile che per certe dimensioni non siano riportati i rispettivi valori di qualità: il simbolo $\perp$ è infatti stato aggiunto al dominio dei valori possibili per poter rappresentare questa eventualità.

Il modello D^2Q: data + quality. Di seguito viene definita la relazione tra un D^2Q *data schema* e il relativo D^2Q *quality schema*; in particolare, si definirà la nozione di *quality association* fra i nodi dei due grafi.

DEFINITION 3.6. (*Quality association*). Sia $A = N_D \cup T_D$ l'insieme di tutti i nodi di un D^2Q data schema S_D; chiamiamo $B = N_Q$ l'insieme di tutti i nodi non foglia di un D^2Q quality schema S_Q. Una *quality association* è una funzione: $qualityAss : A \rightarrow B$ tale che:

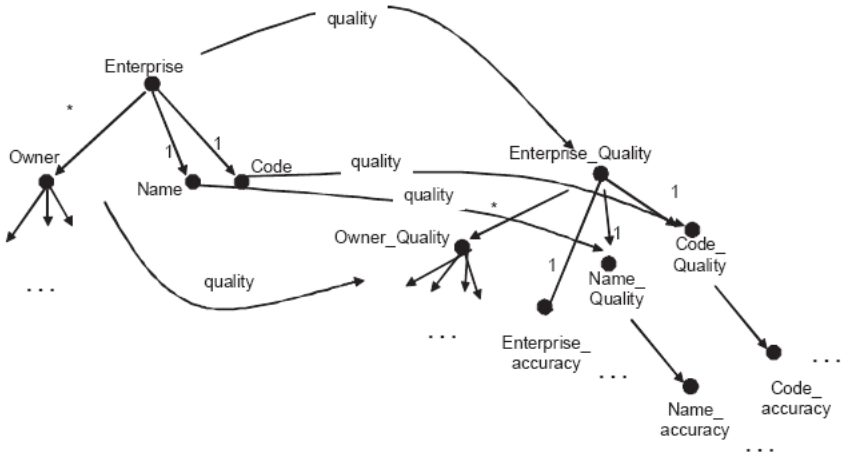


FIGURA 6. Esempio di D^2Q Schema

- ($\forall x \in A \; \exists$ *un'unica* $y \in B$ *tale che* $y = qualityAss(x)) \wedge (\forall y \in B \; \exists$ *un'unica* $x \in A$ *tale che* $y = qualityAss(x)) \Longrightarrow$ `qualityAss` è una funzione biunivoca.

DEFINITION 3.7. (*D^2Q Schema*). Dato un D^2Q data schema S_D e il relativo quality schema S_Q, un $D^2QSchemaS$ è un grafo con le seguenti caratteristiche:

- un insieme di nodi $N = N_D \cup N_Q$, derivati dall'unione dei nodi di S_D e di S_Q;
- un insieme di foglie $T = T_D \cup T_Q$;
- un insieme di archi $\varepsilon = \varepsilon_D \cup \varepsilon_Q \cup \varepsilon_{DQ}$, dove $\varepsilon_{DQ} \subseteq N_D \cup T_D \times N_Q$. In particolare, vengono introdotti degli archi speciali utilizzati per connettere i nodi di S_D con i nodi di S_Q, dove una `qualityAss` è definita. Tali archi sono aggiunti all'unione degli archi da S_D e quelli da S_Q;
- un insieme di etichette sui nodi $L_N = L_D \cup L_Q$;
- un insieme di etichette sugli archi $L_\varepsilon = L_{\varepsilon_D} \cup L_{\varepsilon_Q} \cup L_{\varepsilon_{DQ}}$, dove $L_{\varepsilon_{DQ}} = quality$ è un insieme di etichette `quality`.

Un esempio di *D^2Q Schema*, relativo all'esempi precedenti, è riportato in figura 6.

Istanze di uno schema D^2Q. Le istanze degli schemi dei dati e della qualità si possono facilemente creare in accordo alla struttura *class-based* di un modello D^2Q. In particolare:

- le istanze di una data class sono *data object*;
- le istanze di una quality class sono *quality object*;
- i valori delle quality association corrispondono a *quality link*.

Un'istanza di un D^2Q data schema è un grafo con nodi corrispondenti a data object e foglie corrispondenti ai valori delle proprietà. I nodi sono etichettati con i nomi dei data object e le foglie con una coppia $\langle$`basic type property name`, `basic type property value`$\rangle$. Gli archi non sono etichettati.

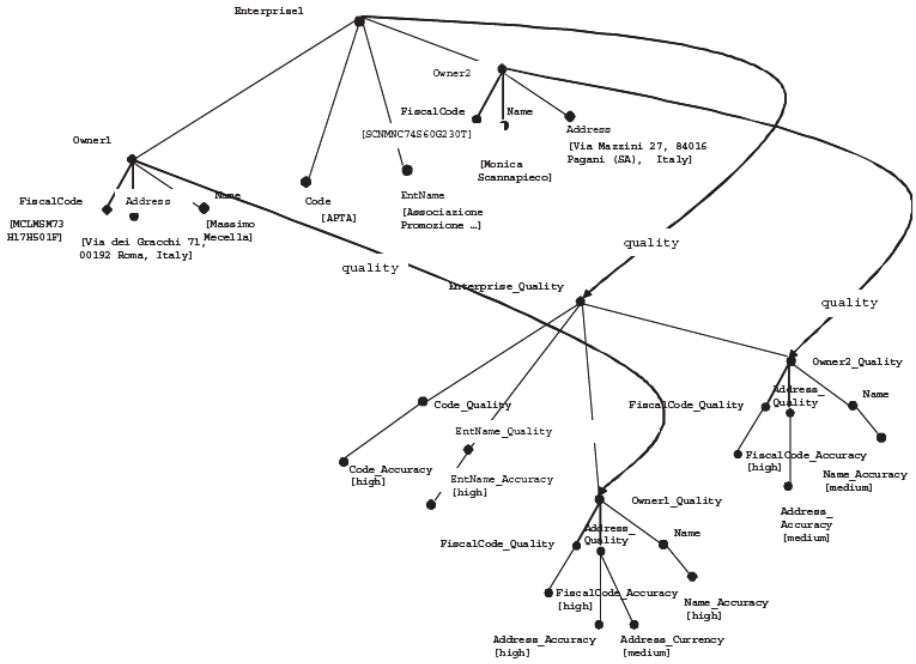


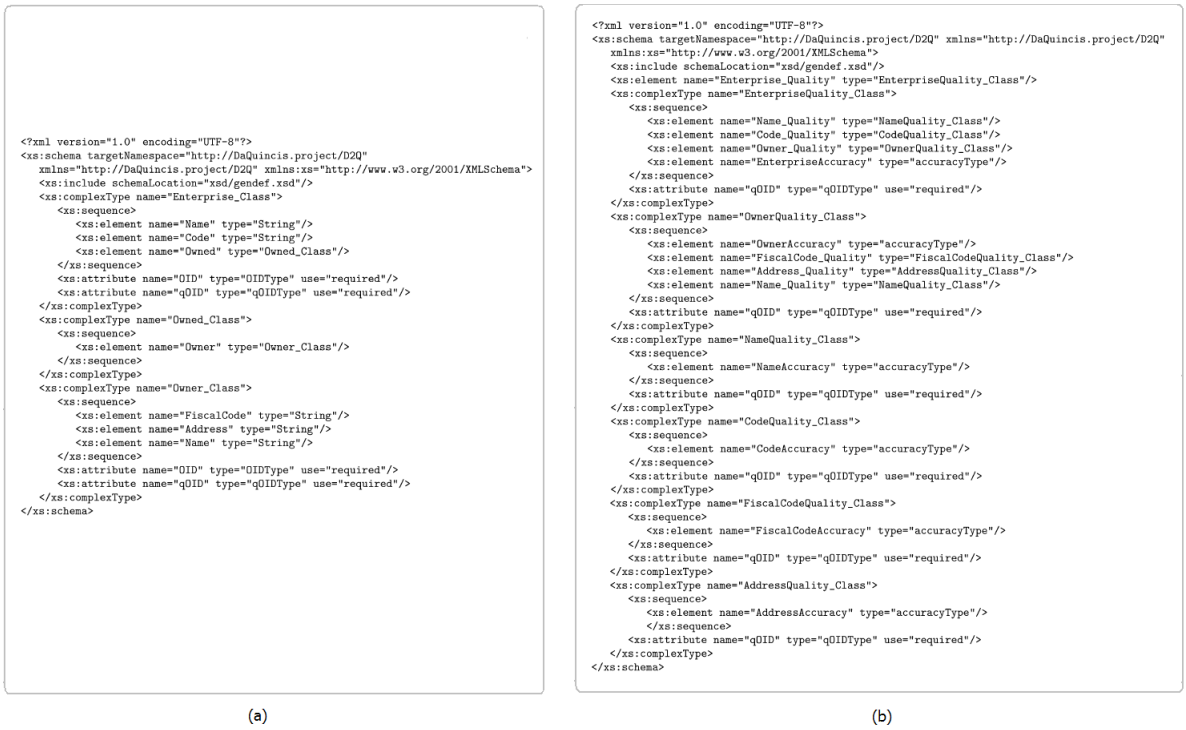
FIGURA 7. Esempio di istanza di uno schema D^2Q

In modo analogo, un'istanza di un D^2Q quality schema è un grafo i cui nodi e foglie corrispondono ai valori delle quality property. Le foglie sono etichettate con una coppia $\langle$`quality type property name, quality type property value`$\rangle$. Infine, un'istanza di uno schema D^2Q, oltre ad essere composta dalle istanza dei data e quality schema, include anche le istanze delle quality association, i quality link, i quali connettono i data object ai rispettivi quality object.

Un esempio di istanza dello schema D^2Q di figura 5 è mostrato in figura 6. L'oggetto Enterprise1, istanza della classe Enterprise, ha due proprietari, Owner1 e Owner2, istanze della classe Owner; in figura sono riportati anche i valori di qualità associati.

Implementazione di un modello D^2Q. Nel sistema DaQuinCIS, gli schemi di un modello D^2Q vengono tradotti in XML Schema, che corrispondono agli schemi locali da essere interrogati in modo da poter recuperare i dati e la loro qualità. Inoltre, verrà descritto come le istanze degli schemi D^2Q vengono tradotte in documenti XML.

Da uno schema D^2Q *a uno schema XML*. Uno schema D^2Q viene tradotti in due schemi XML, uno per i dati e uno per la qualità. Partendo da uno schema D^2Q vi sono molti modi per tradurlo in uno schema XML. Il metodo scelto prevede di convertire la struttura a grafo degli schemi D^2Q nella struttura ad albero del modello dei dati sottostante a XQuery, che è il linguaggio di interrogazione utilizzato in DaQuinCIS. In particolare, questa conversione prevede di replicare gli oggetti dove necessario. Ad esempio, nel modello dei dati di D^2Q una data class può essere proprietà di diverse classi; in questi casi, la data class è replicata e, a livello di oggetto, le stesse istanze possono essere identificate introducendo in modo esplicito gli *Object Identifier* (*OID*). I quality link sono implementati attaverso i *Quality Object Identifier* (*QOID*), presenti sia sui dati che sui rispettivi quality object.



```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://DaQuincis.project/D2Q"
   xmlns="http://DaQuincis.project/D2Q" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:include schemaLocation="xsd/gendef.xsd"/>
   <xs:complexType name="Enterprise_Class">
      <xs:sequence>
         <xs:element name="Name" type="String"/>
         <xs:element name="Code" type="String"/>
         <xs:element name="Owned" type="Owned_Class"/>
      </xs:sequence>
      <xs:attribute name="OID" type="OIDType" use="required"/>
      <xs:attribute name="qOID" type="qOIDType" use="required"/>
   </xs:complexType>
   <xs:complexType name="Owned_Class">
      <xs:sequence>
         <xs:element name="Owner" type="Owner_Class"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="Owner_Class">
      <xs:sequence>
         <xs:element name="FiscalCode" type="String"/>
         <xs:element name="Address" type="String"/>
         <xs:element name="Name" type="String"/>
      </xs:sequence>
      <xs:attribute name="OID" type="OIDType" use="required"/>
      <xs:attribute name="qOID" type="qOIDType" use="required"/>
   </xs:complexType>
</xs:schema>
```

(a)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://DaQuincis.project/D2Q" xmlns="http://DaQuincis.project/D2Q"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:include schemaLocation="xsd/gendef.xsd"/>
   <xs:element name="Enterprise_Quality" type="EnterpriseQuality_Class"/>
   <xs:complexType name="EnterpriseQuality_Class">
      <xs:sequence>
         <xs:element name="Name_Quality" type="NameQuality_Class"/>
         <xs:element name="Code_Quality" type="CodeQuality_Class"/>
         <xs:element name="Owner_Quality" type="OwnerQuality_Class"/>
         <xs:element name="EnterpriseAccuracy" type="accuracyType"/>
      </xs:sequence>
      <xs:attribute name="qOID" type="qOIDType" use="required"/>
   </xs:complexType>
   <xs:complexType name="OwnerQuality_Class">
      <xs:sequence>
         <xs:element name="OwnerAccuracy" type="accuracyType"/>
         <xs:element name="FiscalCode_Quality" type="FiscalCodeQuality_Class"/>
         <xs:element name="Address_Quality" type="AddressQuality_Class"/>
         <xs:element name="Name_Quality" type="NameQuality_Class"/>
      </xs:sequence>
      <xs:attribute name="qOID" type="qOIDType" use="required"/>
   </xs:complexType>
   <xs:complexType name="NameQuality_Class">
      <xs:sequence>
         <xs:element name="NameAccuracy" type="accuracyType"/>
      </xs:sequence>
      <xs:attribute name="qOID" type="qOIDType" use="required"/>
   </xs:complexType>
   <xs:complexType name="CodeQuality_Class">
      <xs:sequence>
         <xs:element name="CodeAccuracy" type="accuracyType"/>
      </xs:sequence>
      <xs:attribute name="qOID" type="qOIDType" use="required"/>
   </xs:complexType>
   <xs:complexType name="FiscalCodeQuality_Class">
      <xs:sequence>
         <xs:element name="FiscalCodeAccuracy" type="accuracyType"/>
      </xs:sequence>
      <xs:attribute name="qOID" type="qOIDType" use="required"/>
   </xs:complexType>
   <xs:complexType name="AddressQuality_Class">
      <xs:sequence>
         <xs:element name="AddressAccuracy" type="accuracyType"/>
         </xs:sequence>
      <xs:attribute name="qOID" type="qOIDType" use="required"/>
   </xs:complexType>
</xs:schema>
```

(b)

FIGURA 8. Esempio di traduzione di uno schema D^2Q: (a) data schema e (b) quality schema

Le scelte principali per rappresentare gli schemi D^2Q come schemi XML sono:

- ❖ rappresentare le data class, le quality class e le loro proprietà come elementi XML;
- ❖ limitare l'utlizzo degli attributi di XML al solo scopo di rappresentare informazioni di controllo, come OID e QOID;
- ❖ definizione di tipi di XML Schema che corrispondono ai tipi base e di qualità del modello D^2Q;
- ❖ definizione dell'elemento radice (root) per ogni data e quality schema.

Un esempio relativo allo schema D^2Q di figura 6 è riportato in fugura 8a per il data schema e 8b per il quality schema.

Dalle istanze degli schemi D^2Q ai file XML. In corrispondenza con il caso precedente, due documenti XML corrispondono alle istanze di uno schema D^2Q: un documento è istanza di un D^2Q data schema e l'altro è istanza del D^2Q quality schema. Tali documenti si ottengono dai rispettivi modelli D^2Q seguendo le seguenti regole:

- ❖ vengono definiti un elemento radice che contiene tutti i data object e uno che contiene tutti i quality object;

```
<?xml version="1.0" encoding="UTF-8"?>
<Enterprise xmlns:n="http://DaQuincis.project/D2Q"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://DaQuincis.project/D2Q
   I:\...\XMLSchemaData.xsd" OID="OID1" qOID="QOID1">
   <Name qOID="QOID11">Associazione Promozione Tecnologie Avanzate</Name>
   <Code qOID="QOID12">APTA</Code>
   <Owned>
      <Owner qOID="QOID13">
         <FiscalCode qOID="QOID131"> MCLMSM73H17H501F </FiscalCode>
         <Address qOID="QOID132">Via dei Gracchi 71 00192 Roma Italy</Address>
         <Name qOID="QOID133">Massimo Mecella</Name>
      </Owner>
      <Owner qOID="QOID14">
         <FiscalCode qOID="QOID141">SCNMNC74S60G230T</FiscalCode>
         <Address qOID="QOID142">Via Mazzini 27 84016 Pagani  Italy</Address>
         <Name qOID="QOID143">Monica Scannapieco</Name>
      </Owner>
   </Owned>
</Enterprise>
```

FIGURA 9. Esempio di istanza relativa allo schema di figura Xa

- ❖ un oggetto *d*, istanza di una data class, è mappato in un elemento XML con un attributo OID che identifica l'oggetto e un attributo qOID che dentifica il quality object ad esso associato;
- ❖ una proprietà di una data class è tradotta in un elemento XML;
- ❖ analogamente, un oggetto d_q, istanza di una quality class, è mappato in un elemento XML con un attributo qOID che lo identifica;
- ❖ una proprietà di una quality class è anch'essa mappata in un elemento XML.

In figura 9 è mostrato un esempio di documento XML rappresentante una possibile istanza del data schema di figura 8a.

4. Tecniche di Query Processing Quality-Driven

In questa sezione verranno analizzati alcuni sistemi esistenti che permettono di effettuare un query processing basato sulla qualità, ovvero che ritornano una risposta ad una global query considerando esplicitamente la qualità fornita dalle sorgenti locali.

Il sistema QP-alg. Viene presentato ora l'approccio descritto in (NAUMANN *et al.* 1999), chiamato nel seguito QP-alg. L'obbiettivo di questo lavoro è quello di incorporare gli aspetti relativi alla qualità delle informazioni nel query planning. Il framework descritto riguarda un sistema di query processing multidatabase basato su uno schema globale dove i mapping tra sorgenti locali e schema globale sono specificati in termini di *query correspondence assertion* (*QCA*). La caratteristica principale di questo sistema è la stretta integrazione tra il classico processo di query planning con valutazioni e considerazione riguardo la qualità delle informazioni: lo scopo è quello di misurare la qualità delle query e, in base a questo, riuscire a rispondere alle query globali utilizzando solo quelle query che sono eseguibili da qualche sorgente.

Le QCA sono equazioni tra query poste rispetto le sorgenti locali e query poste sullo schema globale, e la loro semantica è quella di inner-join. Esse assumo la seguente forma generale:

$$MQ \leftarrow S_i.v_j \leftarrow WQ$$

dove MQ è la query congiuntiva posta sul mediatore (query globale), $S_i.v_j$ denota una vista v_j sulla sorgente s_i e WQ è la query posta su un wrapper (query locale). Si nota che la vista v_j deve essere garantita in entrambe le direzioni, ovvero le variabili nella vista devono comparire in entrambe le query. Il mapping può essere classificato come global-local-as-view (GLAV), dal momento che la query globale è definita in termini di query poste sulle sorgenti.

Sono definite tre classi diverse di dimensioni per il data quality, chiamate *information quality criteria* (IQ criteria):

- *Source-specific criteria*, i quali definiscono la qualità dell'intera sorgente. Dimensioni contenute in questa categoria sono la *reputation* della sorgente, misurata in base alle preferenze personali, e *timeliness*, valutata in base alla frequenza di aggiornamento della sorgente.
- *QCA-specific criteria*, i quali determinano la qualità di una specifica query computabile da una sorgente. Il prezzo pagato per una query, price, e il tempo di risposta, *response time*, di una query rispetto una sorgente sono esempi di dimensioni QCA-specific.
- *Attribute-specific criteria*, i quali valutano la qualità di una sorgente rispetto alla risposta fornita ad una specifica query. Tra le dimensioni di questa categoria è inclusa la *completeness*.

Alcuni metriche per gli IQ criteria sono predeterminate, altre sono calcolate dinamicamente e il risultato è un insieme di vettori di IQ criteria utilizzati per fare un rank delle sorgenti e dei plan di esecuzione delle query. Si nota che nei DBMS ogni plan relativo ad una query produce sempre lo stesso risultato e, pertanto il piano da eseguire è selezionato rispetto ad un modello dei costi; con l'algoritmo QP-alg i piani vengono selezionati in base alla correttezza semantica, pertanto piani diversi della stessa query possono produrre risultati diversi (in quanto possono coinvolgere sorgenti diverse); pertanto i query plan sono selezionati in base alla correttezza semantica, ovvero si seleziona il piano di esecuzione che fornisce la risposta più completa. In particolare, l'algoritmo di integrazione delle informazioni quality-driven QP-alg è composto da 3 fasi, illustrate di seguito e mostrate in figura 10, ovvero selezione delle sorgenti, creazione dei piani e selezione del piano migliore.

Selezione delle Sorgenti. L'obbiettivo della prima fase è quello di cercare di ridurre il numero di piani che potenzialmente possono essere generati per la valutazione di una query. Per tale motivo, i criteri source-specific sono utilizzati per eliminare fin dall'inizio le sorgenti le cui informazioni non sono nel complesso qualitative come nelle altre. Lo scopo è quello di trovare un certo numero di sorgenti migliori indipendentemente da qualsiasi peso introdotto dagli utenti e di ridurre la complessità computazionale delle fasi successive dell'algoritmo. Tale fase viene eseguita dal mediatore una sola volta, ovvero dopo lo start-up del sistema; essa viene rieseguita solo nei casi un cui una sorgente cambi drasticamente i criteri source-specific o quando una sorgente è aggiunta al sistema. Per poter classificare le sorgenti in base ai vettori di criteri IQ, viene utilizzato un metodo di decision making multiattributo, ovvero la Data Envelopment Analysis (CHARNES *et al.* 1978).

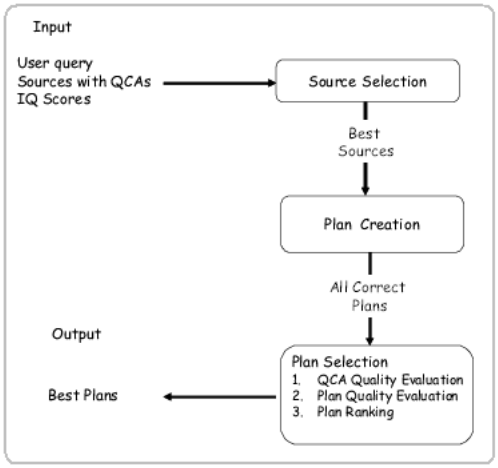


FIGURA 10. Fasi dell'algoritmo QP-alg

Creazione dei piani di esecuzione. L'obbiettivo di questa seconda fase è quello di trovare tutte le combinazioni di QCA che permettono di ottenere un risultato semanticamente corretto ad una query. Dal momento che ogni QCA definisce una vista sullo schema globale, tale fase ricerca le combinazioni di tali viste che producono tuple di dati corrette. In particolare, per ogni relazione di una query posta da un utente, *UQ*, si determina l'insieme delle QCA le quali contengono la relazione nella loro query globale *MQ*. Inoltre, viene verificato se ogni QCA esporta tutti i necessari attributi, ovvero quelli richiesti nella UQ. L'insieme di QCA ricavato viene memorizzato in un bucket. Successivamente, si enumerano i prodotti cartesiani di tutti i bucket e, infine, si verifica se le combinazioni sono soddisfacibili, se sono semanticamente contenute in *UQ* e se possono essere minimizzate. Il risultato finale di questi passi è costituito da tutti i piani che permettono di ottenere risultati semanticamente validi per la query *UQ*.

Selezione del piano migliore. L'intento, in questa fase, è quello di classificare i piani creati al passo precedente in base ai valori di qualità e di restringere l'esecuzione ad una qualche percentuale di piani o, alternativamente, ai soli piani necessari per raggiungere certi vincoli di costo o di qualità. Questa terza fase è suddivisa in altri 3 step. Inizialmente, viene valutata la qualità di ogni QCA (step 1 in plan selection, figura 10). In particolare, per ogni QCA vengono calcolati i criteri QCA-specific e user-query specific. Successivamente, la qualità di un piano viene calcolata (step 2 in plan selection, figura 10) affidandosi ad una procedura simile a quella dei modelli di costo nei DBMS. Per ogni piano viene costruito un albero, *tree*, dove le foglie sono QCA e i nodi interni sono gli operatori di join. Il vettore dei criteri IQ è calcolato ricorsivamente per un nodo, partendo dai suoi nodi figli.Per ogni criterio di qualità, allo scopo di combinare i diversi vettori di criteri, si definisce un insieme di funzioni "merge". Ogni criterio viene valutato in base ad una specifica funzione di merge. Ad esempio, la funzione di merge per il criterio price è definita come la somma sia del figlio destro che del figlio sinistro di un certo nodo, che significa che entrambe le query devono essere valutate. In figura 11 è riportato come è computato il price di un piano.

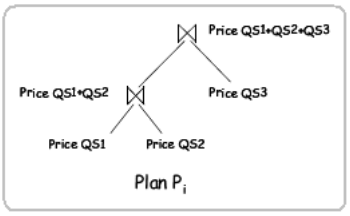


FIGURA 11. Computazione del criterio price per il piano P_i

Il plan ranking è eseguito per mezzo del metodo Simple Additive Weighting (SAW) (step 3 in plan selection, figura 10). In particolare, il punteggio finale di qualità per un piano è computato come la somma pesata dei criteri normalizzati, dove i pesi rappresentano l'importanza di ogni criterio per l'utente. In fine, in base al ranking effettuato, il sistema è in grado di scegliere il piano o i piani migliori.

Fusionplex. Si tratta di un sistema di integrazione dei dati (MOTRO ET AL. 2004; MOTRO & ANOKHIN 2005) che si basa su *(i)* uno schema globale relazionale D, *(ii)* un insieme di sorgenti locali relazionali (D_i, d_i), dove d_i è l'istanza dello schema locale D_i e *(iii)* su un insieme di schema mapping (D, D_i). La definizione dei mapping segue l'approccio GLAV. In FusionPlex, si assume la *schema consistency assumption*, che significa che non vi sono errori nella modellazione delle sorgenti locali, ma solo modellazioni diverse. A livello di istanze, invece, si assume valida la *instance inconsistency assumption*, cioè la stessa istanza del mondo reale può essere rappresentata in diversi modi nelle varie sorgenti locali. Per poter gestire queste inconsistenze a livello di istanza, Fusionplex introduce un insieme di metadati, chiamati *features*, riguardo le sorgenti integrate.

Alcuni esempi di feature sono time stamp, availability e accuracy. La definizione del framework di integrazione dei dati è quindi estesa includendo la feature nella definizione di schema mapping. In particolare, i mapping sono triple consistenti di una vista dello schema globale D, di una vista dello schema locale D_i e le feature associate alla vista locale. Fusionplex include una estensione dell'algebra relazionale che prende in considerazione le associazioni di un insieme di caratteristiche $F = \{F_1 \ldots F_n\}$ con le relazioni della sorgente. Ad esempio, il prodotto cartesiano esteso concatena i valori delle relazioni partecipanti, ma fonde il valore delle loro feature. Il metodo di fusion dipende dalla particolare feature. Quindi, ad esempio, il valore di availability della nuova tupla è il prodotto dei valori dell'availability delle tuple di input; il time stamp è il minimo dei time stamp in input. In questo scenario, il query processing è svolto in diversi step:

(1) Data una query Q, viene definito l'insieme delle *contributing views*. Inzialmente, l'insieme degli attributi della query e ogni contributing view sono intersecati. Se l'intersezione è vuota, allora la contributing view non è rilevante. Successivamente, viene eseguito un join tra i predicati di selezione della query e le contributing view. Se il risultato è vero, allora la contributing view è considerata rilevante per la query.
(2) Quando le contributing view sono state identificate, vengono derivati i *query fragments*, ovvero le unità dell'informazione considerate adatte per popolare la risposta alla query. Un query fragment si ottiene rimuovendo dalla contributing view di tutte le tuple e attributi che non sono richiesti nella query, e dall'aggiunta di valori null per gli attributi della query mancanti nella contributing view.

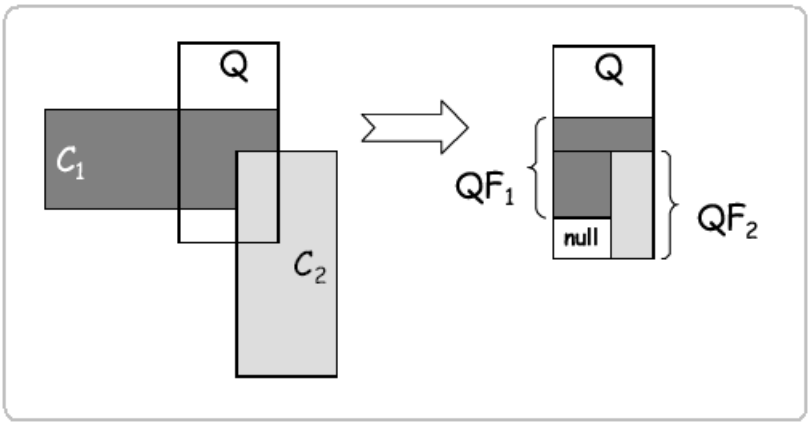


FIGURA 12. Esempio di costruzione del query fragment

Ad esempio, in figura 12 sono mostrate due contributing view, C_1 e C_2, e i corrispondenti query fragment, QF_1 e QF_2.

(3) Un query fragment, che può essere vuoto, viene costruito da ogni contributing view rilevante. L'unione di tutti i query fragment non vuoti viene detta *poly-instance query*. Intuitivamente, una polyinstance include tutte le informazioni derivate dalla sorgente dati in risposta ad una query.

Per fornire un'unica risposta alla query Q, i conflitti a livello di istanza presenti nella poly-instance devono essere risolti. La strategia per la risoluzione dei conflitti è descritta nella prossima sezione.

Confronto fra le tecniche di query processing quality-driven. Le tecniche presentate, ovvero DaQuinCIS, QP-alg e Fusionplex sono comparate rispetto le seguenti caratteristiche:

- *Metadati di qualità*. Mostrano che ogni tecnica è basata sull'utilizzo di un insieme di metadati che forniscono un supporto alle attività di query processing.
- *Granularità del modello di qualità*. Indica a quali elementi è possibile associare i metadati di qualità. QP-alg associa i metadati non solo alle sorgenti ma anche alle QCA e alle query poste dagli utenti. DaQuinCIS, grazie alla natura semi-strutturata del suo modello dei dati, permette di avere metadati associati a elementi a diversi livelli di granularità. Infine, Fusionplex consente le associazioni alle sole sorgenti.
- *Tipi di mapping*. Fusionplex e QP-alg presentano un approccio GLAV, mentre DaQuinCIS utilizza un approccio GAV.
- *Supporto alla quality algebra*. I valori di qualità associati ai dati delle sorgenti locali necessitano di essere processati per mezzo di specifici operatori algebrici. Vi sono alcune indicazioni in questa direzione, ma è tutt'ora un problema di ricerca aperto. Tra i sistemi analizzati, QP-alg propone delle funzioni di fusione mentre Fusionplex propone un'estensione dei classici operatori relazionali. DaQuinCIS, al contrario degli altri due sistemi, non fa nessuna proposta a riguardo.

Un riassunto del confronto fra i tre sistemi è riportato in tabella 3.

Tecnica	Metadati di Qualità	Granularità	Tipi di Mapping	Supporta all'algebra di qualità
QP-alg	SI	Orgenti, Query, QCA.	GLAV	Funzioni di merge
DaQuinCIS query processing	SI	Ogni elemento di un modello di dati semi strutturato	GAV	NO
Fusionplex query processing	SI	Sorgente	GLAV	Estensione degli operatori algebrici classi del modello relazionale.

TABELLA 3. Confronto fra le tecniche di query-processing quality-driven

Dal confronto si nota come DaQuinCIS definisca un modello dei dati più flessibile rispetto agli altri, in quanto i metadati di qualità possono essere presenti a diversi livelli di granularità: questo permette di avere più metadati di qualità associati ai valori dei singoli attributi. Rispetto agli approcci di QP-alg e Fusionplex, il vantaggio consiste nella possibilità di avere risposte più accurate alle query e alla possibilità di sviluppare efficaci funzioni di risoluzione per la risolizione di eventuali conflitti fra istanze. Inoltre, nell'approccio definito da QP-alg, si corre il rischio di eliminare fino dal primo step dell'algoritmo di query processing, sorgenti che avrebbero potuto dare risposte significative alla query posta. Ciononostante, avere metadati che esprimano un giudizio complessivo sulla qualità di una sorgente dati, o sull'istanza di una relazione, o un suo attributo, di un database è sufficiente per creare meccanismi di query processing o conflict resolution efficaci ed, in particolare, efficienti. Infatti, il modello adottato da DaQuinCIS, costringe a dover scambiare ed elaborare una grossa quantità di metadati.

5. Risoluzione Quality-Driven di Conflitti a Livello di Istanza

La risoluzione dei conflitti a livello di istanza rappresenta uno dei problemi principali da risolvere nei sistemi di integrazione delle informazioni ed, in particolare, nei PDMS.

Tali conflitti possono essere distinti in due diverse categorie: *attribute* (o *entity*) *conflict* e *key* (o *tuple*)*conflict*. Si considerino in particolare due tabelle relazionali, $S_1(A_1,\ldots,A_k, A_{k+1},\ldots, A_n)$ e $S_2(B_1,\ldots,B_k,B_{k+1},\ldots, B_m)$, dove $A_1 = B_1 \ldots A_k = B_k$. Si consideri inoltre che le tuple t_1 di S_1 e t_2 di S_2 rappresentino lo stesso oggetto del mondo reale e che $A_i = B_i$. Allora, si ha che:

- un *attribute conflict* sorge se e solo se
$$t_1.A_i \neq t_2.B_i$$
- Si supponga inoltre che A_i e B_i siano, rispettivamente, le chiavi primarie di S_1 e S_2. Allora, un *key conflict* nasce se e solo se, per tutti i $j = 1,\ldots,K$ con $i \neq j$, si verifica che:

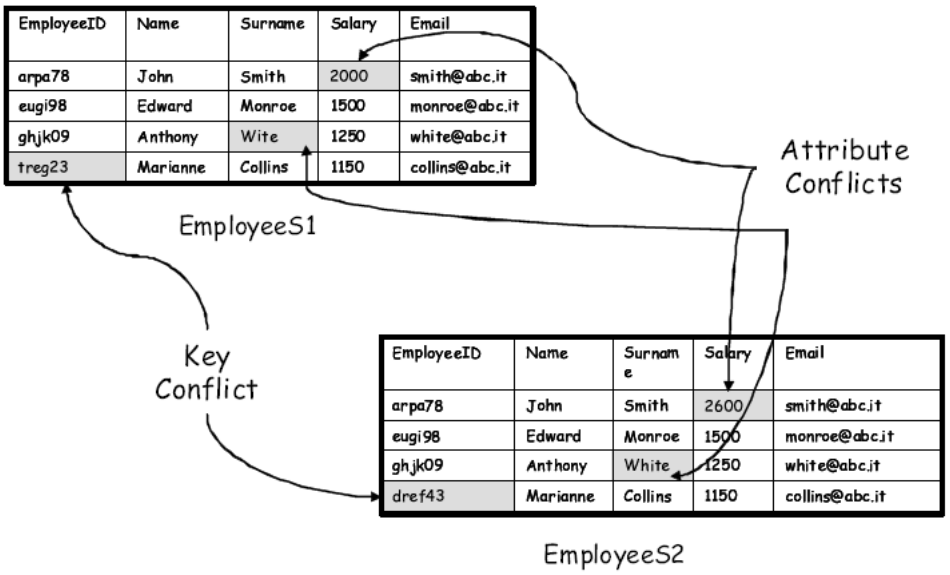


FIGURA 13. Esempio di conflitto a livello di istanza

$$\mathtt{t_1.A}_i \neq \mathtt{t_2.B}_i \text{ e } \mathtt{t_1.A}_j \neq \mathtt{t_2.B}_j$$

In figura 13 è riportato un esempio di conflitti a livello d'istanza. La figura mostra due relazioni `EmployeeS1` e `EmployeeS2`, che rappresentano informazioni riguardo gli impiegati di una azienda. Tali relazioni presentano due conflitti sugli attributi, riguardanti il `Salary` dell'impiegato `arpa78` e il `Surname` di `ghjk09`. Un esempio di conflitto sulla chiave primaria è mostrato per l'impiegata `Marianne Collins`.

Spesso, tali inconsistenze sono causate dalla bassa qualità dei dati; in particolare esse possono essere generate da errori nel processo di collezione e di generazione dei dati o perchè essi non sono aggiornati. Nel seguito verranno presentate e confrontate alcune delle proposte presenti in letteratura che si prefiggono di gestire le inconsisitenze a livello di istanza basandosi sulla qualità dei dati.

Aurora. Si tratta di un sistema di integrazione dei dati basato su mediatore (YAN & OZSU 1999). L'approccio propone un modello di query *conflict-tolerant*, che presenta le seguenti caratteristiche:

- Due operatori, per la risoluzione di conflitti di attributi, chiamati resolve *attribute-level conflict* (RAC) e per la risoluzione dei conflitti a livello di tuple, chiamati *resolve tuple-level conflict* (RTC). Tali operatori prendono come parametri una funzione di risoluzione. Si consideri ad esempio la relazione globale `Employee` di figura 14, la quale rappresenta l'istanza globale risultato dell'integrazione delle relazioni `EmployeeS1` e `EmployeeS2` mostrate in figura 13.

TupleID	EmployeeID	Name	Surname	Salary	Email
t_1	arpa78	John	Smith	2000	smith@abc.it
t_2	eugi98	Edward	Monroe	1500	monroe@abc.it
t_3	ghjk09	Anthony	Wite	1250	white@abc.it
t_4	treg23	Marianne	Collins	1150	collins@abc.it
t_5	arpa78	John	Smith	2600	smith@abc.it
t_6	eugi98	Edward	Monroe	1500	monroe@abc.it
t_7	ghjk09	Anthony	White	1250	white@abc.it
t_8	dref43	Marianne	Collins	1150	collins@abc.it

FIGURA 14. Relazione globale `Employee`

TupleID	EmployeeID	Name	Surname	Salary	Email
t_1	arpa78	John	Smith	2000	smith@abc.it
t_2	eugi98	Edward	Monroe	1500	monroe@abc.it
t_3	ghjk09	Anthony	White	1250	white@abc.it
t_4	treg23	Marianne	Collins	1150	collins@abc.it

RAC(Employee,Salary(MIN), Surname(Longest), EmployeeID(Any))

FIGURA 15. Risoluzione dei conflitti sugli attributi

Un esempio del funzionamento dell'operatore RAC è ripostato in figura 15, dove le specifiche funzioni di risoluzione sono `MIN` per `Salary`, `LONGEST` per `Surname` e `ANY` per `EmployeeID`. Un esempio di utilizzo di RTC è invece mostrato in figura 16, dove la funzione di risoluzione applicata è `ANY`.

- Tre strategie per la risoluzione dei conflitti: HighConfidence, RandomEvidence e PossibleAtAll. Queste strategie consentono all'utente di definire il grado di conflitti permessi, e sono utilizzati in congiunzione con gli operatori descritti in precedenza quando una query viene formulata. La prima strategia consente di specificare che nessun conflitto su di uno specifico attributo è consentito; la seconda specifica che nel caso in cui siano presenti conflitti, una funzione eseguite a runtime deve selezionare i valori di ritorno; infine, la terza strategia ritorna tutti i valori che rispondono correttamente alla query, indipendentemente dai conflitti.

Un esempio di query *conflict-tolerant* è la seguente:

```
SELECT EmployeeID, Name (ANY), Salary[MIN]
FROM Employee
WHERE Salary > 1800
WITH HighConfidence
```

Tale query ritorna gli impiegati con `Salary` maggiore di 1800 euro. Se nella risposta ci sono conflitti, la query seleziona gli impiegati il cui salario è maggiore di 1800 in tutte le sorgenti. Quindi, basandosi sulla figura 14, vengono ritornate solo le tuple t_1 e t_5. Successivamente, applicando la funzione di risoluzione `MIN` su `Salary`, il risultato si riduce alla sola tupla t_1.

TupleID	EmployeeID	Name	Surname	Salary	Email
t_1	arpa78	John	Smith	2600	smith@abc.it
t_2	eugi98	Edward	Monroe	1500	monroe@abc.it
t_3	ghjk09	Anthony	Wite	1250	white@abc.it
t_4	dref43	Marianne	Collins	1150	collins@abc.it

RTC(Employee, ANY)

FIGURA 16. Risoluzione dei conflitti sulle tuple

Fusionplex. L'approccio adotatto nel sistema Fusionplex per la risoluzione dei conflitti è molto simile a quello di DaQuinCIS, in quanto entrambi si basano sui metadati associati ai dati delle sorgenti informative. I metadati usati in Fusionplex, detti *feature*, sono: *time stamp*, la quale indica il tempo in cui il sistema è stato validato; *cost*, che può indicare sia il costo di trasmissione nella rete o l'importo da pagare per ottenere le informazioni o entrambi; *accuracy*, valutata secondo un approccio probabilistico; *availability*, probabilità che l'informazione sia disponibile; *clearance*, ovvero la chiarezza delle informazioni presentate.

Inoltre, entrambi gli approcci collezionano le risposte significative ad una query in cluster contenenti diverse copie dello stesso oggetto, sui quali sono applicati delle strategie di risoluzione.

La differenza fra i due sistemi consiste nel processo di costruzione del risultato finale. In Fusionplex, come descritto nella sezione precedente, la fase di collezione dei risultati ottenuti dalle sorgenti locali termina con la costruzione di una polyinstance, sulla quale è applicata una strategia di risoluzione. Tale risoluzione è eseguita in due fasi distinte: nella prima fase si introducono le preferenze dell'utente in base ad una utility function, mentre nella seconda viene eseguita la fusione.

Per quanto riguarda la prima fase, gli utenti possono specificare l'importanza che vogliono assegnare ad ogni feature. Sulla base di tali valori, una apposita funzione calcola la somma pesata dei valori delle feature della sorgente, la quale fornisce una soglia di utility in base alla quale è possibile fare un ranking delle sorgenti.

Nella seconda fase, la risoluzione delle inconsitenze può essere eseguita sia sulla base delle feature (*feature-based resolution*) che sulla base dei dati (*content-based resolution*).

Una politica di risoluzione consiste di una sequenza di:

- *elimination functions*, che possono essere feature-based, come `MAX(timestamp)` e `MIN(cost)`, o content-based, come `MAX(Salary)`.
- *fusion functions*, che sono sempre content-based. `ANY` e `AVERAGE` ne sono degli esempi.

Si nota che la politica di risoluzione è completamente definita dall'utente. Inoltre, Fusionplex ammette tre livelli di tolleranza. Il primo (*no resolution*), consente ad una risposta con conflitti di essere ritornata all'utente; il secondo (*pruning of polytuple*) rimuove le tuple che che non soddisfano la selezione delle feature o che non superano la soglia di utility.

EmployeeID	Salary
arpa78	2000

EmployeeID	Salary
eugi98	1500

EmployeeID	Salary
ghjk09	1250
treg23	1150

FIGURA 17. Risultato della query context-aware

Infine, il terzo livello di tolleranza (*selective attribute resolution*) forza la risoluzione solo su certi attributi.

FraSQL-Based Conflict Resolution. Tale approccio (SCHALLEHN *et al.* 2002) propone una estensione al linguaggio di interrogazione per multidatabase FraSQL, il quale fornisce operazioni per trasformare e integrare dati eterogenei. L'idea principale è di utilizzare il raggruppamento per l'eliminazione dei duplicati e l'aggregazione per la risoluzione dei conflitti. FraSQL fornisce, per la risoluzione dei conflitti, sia un'aggregazione che un raggruppamento *user-defined*. La prima è particolarmente vantaggiosa per la risolu-zione dei conflitti, in quanto consente la selezione di un valore rappresentativo da un gruppo di valori corrispondenti allo stesso oggetto del mondo reale. Il raggruppamento user-defined può essere di due tipi: *(i)* context free e *(ii)* context aware. Il primo è l'approccio tipico; un esempio di query che utilizza il raggruppamento user-defined e context free è la seguente:

```
SELECT avg (Temperature), rc
FROM Weather
GROUP BY regionCode (Longitude, Latitude) AS rc
```

dove `regionCode` è una funzione esterna che calcola la regione indicando la sua posizione geografica.

Il raggruppamento context-aware è stato introdotto con lo scopo di superare le limitazioni di quello classico. Un esempio è rappresentato dalla query sottostante:

```
SELECT EmployeeID, Salary
FROM EmployeeS1
GROUP maximumDifference(Salary, diff-150)
BY CONTEXT
```

Tale query considera la relazione `EmployeeS1` di figura 13 e raggruppa le tuple come mostrato in figura 17, generando tre insiemi corrispondenti alle tuple per le quali il `Salary` differisce di massimo 150.

Confronto fra le tecniche proposte. Si considerano come parametri di confronto le strategie di tolleranza adottate dalle varie tecniche e il modello di query. Per quanto riguarda le strategie di tolleranza, Aurora e Fusionplex propongono un grado di flessibilità che può essere selezionato al manifestarsi del conflitto. Entrambi propongono tre livelli di tolleranza. In particolare, la politica no resolution di Fusionplex corrisponde alla politica PossibleAtAll di Aurora, in quanto in entrambi gli approcci la risposta con conflitti è ritornata all'utente; la politica pruning of polytuple di Fusionplex, la quale rimuove le

tuple che non soddisfano le feature selezionate o la soglia di utility, è più specifica della politica RandomEvidence di Aurora; la politica selective attribute di Fusionplex, la quale permette di lasciare non risolti certi conflitti, è uno specifico caso di politica no resolution, con granularità maggiore.

Per quanto riguarda i modelli di query, le tecniche presentate propongono tutte soluzioni differenti: Fusionplex si basa su un modello relazionale esteso, DaQuinCIS su un modello semi-strutturato (XML), mentre Aurora e FraQL presentano soluzioni ad-hoc.

6. Implementazione Relazionale del Modello D^{2}Q

In questa sezione e nella successiva, viene proposta una possibile implementazione del modello D^{2}Q per il sistema MOMIS, in quanto entrambi i componenti rappresentano la base per quella che sarà il componente di data management in NeP4B. Dal momento che la rappresentazione dei dati in MOMIS è relazionale, anche le informazioni relative al data quality (schemi e istanze) dovranno essere modellate secondo tale formalismo.

In particolare, ciò che si è voluto realizzare è una struttura dati adeguata per memorizzare in un DBMS la qualità delle sorgenti informative. Per far questo, si è voluto mantenere una certa conformità con il modello D^{2}Q, considerando il fatto che il database della qualità non è integrato con quello dei dati ma separato e che deve essere popolato gradualmente, in quanto i dati sulla qualità non sono pre-esistenti. Si è cercata inoltre una implementazione che necessiti del minor numero possibili di modifiche allo schema della sorgente informativa.

Come si vedrà nel seguito, la problematica principale affrontata è stata l'individuazione di un metodo efficiente di identificazione delle tuple e di associazione dei dati con i relativi valori di qualità.

Sempre nell'ambito di database relazionali, WANG e KON (1995) propongono un'estensione al modello relazione, nella quale una chiave surrogata, detta *quality key*, viene aggiunta ad ogni attributo e ad ogni cella. Tale approccio non prevede di avere quality object che si riferiscono ad una tupla intera (ma solo ad attributi), anche se ciò sarebbe risolvibile prevedendo un'ulteriore attributo per ogni relazioni che mi identifichi univocamente la tupla ed associare ad esso un quality object. Inoltre, si nota che una quality key viene aggiunta ad ogni cella, modificando quindi l'intera base di dati e incrementando notevolmente le dimensioni della stessa.

Come detto, l'approccio seguito mira a modificare il meno possibili la corgente orignale; partanto dovrà essere studiato un meccanismo di associazione diverso da quello proposto da Wang e Kon.

Naming Convention. Viene ora illustrata la convenzione scelta per assegnare i nomi alle tabelle e ai campi delle tabelle del database della qualità.

L'importanza della naming convention è data dal fatto di poter rendere leggibile e facilmente utilizzabile l'applicazione DB alle applicazioni che dovranno sfruttarla. Si nota comunque che tali regole non sono assolutamente vincolanti e che possono quindi anche non essere seguite.

(1) Il nome del database della qualità è composto dal nome del database dei dati più la stringa "_Quality". Ad esempio, "HotelDB_Quality" indica il nome del database della qualità di "HotelDB".

(2) Analogamente al caso precedente, il nome delle relazioni nella sorgente di qualità sono costituiti dal nome della relativa relazione concatenata con la stringa "_Quality". Ad esempio, la qualità della relazione "Hotel" di "HotelDB" sarà memorizzata nella relazione "Hotel_Quality" di "HotelDB_Quality".
(3) Il nome delle relazioni di qualità relative ad attributi è composto dal nome della relazione, dal nome dell'attributo e dalla stringa "_Quality". Ad esempio, la qualità degli attributi di "Hotel", come "Name" o "Address", sono memorizzate nella relazioni "Hotel_Name_Quality" e "Hotel_Address_Quality".
(4) Ogni relazione del quality schema prevede una serie di proprietà che rappresentano le dimensioni della qualità. Seppure il modello D^2Q non prevede nessun vincolo sul nome che tali proprietà possono avere, nel seguito si associa ad ogni attributo di qualità il nome della dimensione stessa, ovvero accuracy, completeness, consistency e currency. Ad esempio, se voglio introdurre nella relazione "Hotel_Name_Quality" un attributo indicante l'accuratezza, in nome di questo attributo sarà "accuracy"; se "Address_Quality" specifica sia accuratezza che completezza, allora i campi di questa relazione saranno "accuracy" e "completeness".
(5) L'attributo che rappresenta la chiave primaria delle relazioni di qualità è indicato con il termine `id`.

Mapping tra il modello D2Q e il modello relazionale. Il modello D^2Q definisce due schemi, il Data e il Quality Schema. Entrambi gli schemi, i quali sono semi-strutturati, devono essere mappati in uno schema relazionale.

Mapping Data Schema - Relational Schema.

- Traduzione di data class. Una classe dello schema dei dati è tradotta in una relazione, dove il nome della classe diventa il nome della relazione.
- Traduzione delle proprietà. Le proprietà vengono tradotte in attributi delle relazioni. In particolare, le proprietà che come tipo dato hanno una classe sono mappate come una chiave esterna che referenzia la specifica relazione. Il nome della proprietà diventa il nome dell'attributo.
- Le cardinalità dello schema relazionale rispettano quelle dello schema D^2Q.

Mapping Quality Schema - Relational Schema.

- Traduzione di quality class. D^2Q distingue due tipi di quality class: quelle associate ad un tipo base (un attributo) e quelle associate ad una data class (una relazione). Le prime vengono tradotte con relazioni composte da quattro attributi, che corrispondono alle quattro dimensioni della qualità. Per quanto riguarda il secondo tipo di quality class, una traduzione fedele al modello D^2Q prevede che vengano tradotte in relazioni composte dai 4 attributi relativi alle dimensioni della qualità (come nel caso precedente) e altri attributi i quali contengono un riferimento (chiave esterna) ad altre relazioni (sia relazioni relative ad attributi o relative ad altre relazioni). In realtà questo tipo di traduzione dovrà essere opportunamente valutato. Secondo la naming convention adottata, il nome di tali relazioni è rappresentato dalla concatenazione del nome dell'attributo a cui sono associate con la stringa "Quality", mentre il nome degli attributi di tale relazione è il nome stesso della dimensione di qualità (accurancy, currency, completeness e consistency).

- ❖ Si nota che il modello D^{2}Q prevede di definire nuovi tipi di dato relativi alle dimensioni della qualità. In modo analogo, nella effettiva implementazione con un DBMS è possibile estendere i tipi di dato del sistema. In questo caso, si assume che ogni dimensione possa assumere 3 possibili valori (stringhe):
  - low: è un basso indice di qualità.
  - medium: indice medio di qualità.
  - high: alto valore di qualità.

Mapping D^2Q Schema - Relational Schema. Tale schema rappresenta l'unione dei due precedenti. In più vengono definite le quality association, le quali collegano le tuple e i record della data source con quelle della quality source. La traduzione di queste associazioni non è banale per due motivi principali:

- ❖ data e quality source sono due database separati. Pertanto non è possibile indicare chiavi esterne.
- ❖ le associazioni non riguardano sole le tuple ma anche i valori dei singoli attributi. Pertanto, non è possibile implementare un semplice meccanismo che recuperi le tuple in base ad un riferimento indicato come attributo della relazione dei dati. Questo meccanismo verrà analizzato nella prossima sezione

7. Scelta delle Chiavi e della Struttura dei Dati

In generale, le alternative di suddividono in quelle che prevedono la modifica del database dei dati e quelle che invece non lo alterano. Nel seguito vedremo i vantaggi e gli svantaggi delle varie soluzioni identificate.

Prima soluzione: QID sulle relazioni dati. Tale soluzione prevede di aggiungere alle tabelle del database della sorgente un campo `qid`, il quale fa riferimento alla chiave primaria della relazione nel database della qualità relativa alla qualità della relazione dati stessa. Le relazioni di qualità associate ad una relazione dello schema dei dati prevedono, in analogia con il modello D^2Q, i seguenti attributi:

- ❖ `id`, il quale rappresenta la chiave primaria.
- ❖ Un attributo per ogni dimensione di qualità previsto dallo schema. Secondo la naming convention, il nome di tali attributi è identico al nome delle dimensioni, ovvero accuracy, completeness, ecc.
- ❖ Un attributo, con semantica di chiave esterna, che fa riferimento alle tuple delle relazioni di qualità relative agli attributi della relazione dei dati.

Le relazioni di qualità associate agli attributi delle relazioni nello schema dei dati sono invece composte dal solo `id` e dalle dimensioni di qualità.

Si nota che questa struttura suddivide il database della qualità in relazioni posizionate su due livelli. Le *relazioni di primo livello* sono relative alla qualità delle tuple della relazione dati, mentre le *relazioni di secondo livello* esprimono i valori di qualità per gli attributi delle tuple. Ogni relazione di primo livello contiene i riferimenti (chiavi esterne) alle relazioni di secondo livello. Pertanto, l'attributo `qid` previsto per le relazioni dati contiene il valore della chiave primaria della relazione di primo livello relativa a quel data object.

Un esempio della struttura dati realizzata è mostrata in figura 18.

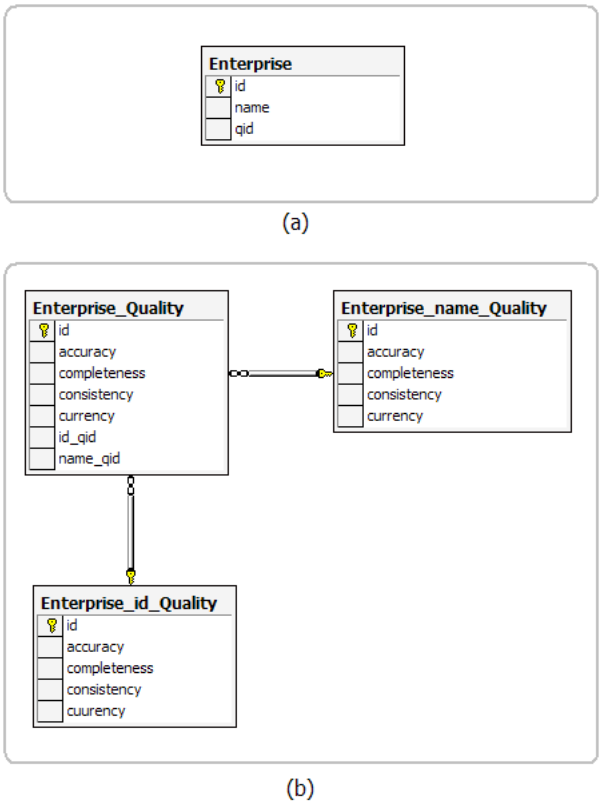


FIGURA 18. Struttura dati relativa alla prima soluzione: (a) data schema, (b) quality schema

Tale figura mostra che l'attributo `qid` di `Enterprise` è una chiave esterna che fa riferimento all'attributo `id` di `Enterprise_Quality`, la quale rappresenta la relazione di primo livello. La relazione `Enterprise_Quaity` prevede degli attributi `id_qid` e `name_qid`, anch'essi chiavi esterne che si rifersiscono agli attributi `id` delle relazioni di secondo livello `Enterprise_id_Quality` e `Enterpise_name_Quality`.

Cerchiamo ora di valutare i pregi e i difetti di tale soluzione. Per prima cosa è necessario stabilire la scelta delle chiavi primarie per le relazioni nel database della qualità. Una prima soluzione è rappresentata dall'utilizzo di una chiave surrogata; un esempio di chiave surrogata è rappresentato da un numero intero progressivo univoco all'interno dell'istanza di una relazione.

L'efficienza della struttura dati proposta è valutata in base alle seguenti analisi:

(1) Costo per la modifica del database dei dati.
(2) Costo per le operazioni di aggiunta di nuovi valori di qualità.
(3) Costo per le operazioni di rimozione dei valori di qualità.
(4) Costo per le operazioni di aggiornamento dei valori di qualità.

1. Analisi del costo di modifica del database dei dati. Come descritto in precedenza, tale soluzione prevede di aggiungere un attributo `qid` ad ogni relazione del database. Ciò significa eseguire la seguente operazione per ogni tabella del database:

```
ALTER TABLE nome_tabella ADD qid INTEGER
```

dove `nome_tabella` deve essere sostituito con il nome della tabella da modificare.

Questa operazione accede ad ogni tupla del database ed inserisce nel nuovo campo aggiunto il valori `NULL`. Per database di grandi dimensioni, tali operazioni di scrittura possono essere onorose.

Tale soluzione ha inoltre il difetto di occupare memoria per valori `NULL`. Infatti si assume che al momento della creazione del database della qualità, esso non sia istanziato; pertanto, nel database della sorgente viene occupata memoria senza nessuno scopo immediato.

2. Aggiunta di nuovi valori di qualità. Questa operazione modifica sia il database dei dati che della qualità. Si considerano due casi diversi: *(i)* primo istanziamento delle relazioni di qualità relativamente ad una specifica tupla e *(ii)* istanziamento già effettuato.

Nel caso *(i)*, è necessario istanziare la relazione di qualità (di primo e/o di secondo livello) e successivamente scrivere il valore della chiave della tuple di primo livello nel campo `qid` della tupla dati. Si nota che con questa soluzione non è possibile avere tuple di secondo livello senza la relativa tupla di primo livello; ciò è dovuto al fatto che il campo qid della relazione dati contiene sempre e solo la chiave della tupla di primo livello. Pertanto, riferendosi all'esempio in figura 1, per recuperare l'accuracy del nome di un certo `Enterprise`, dovrò per forza accedere alla relazione di primo livello `Enterprise_Quality` per leggere il valore dell'attributo `name_qid`; in base a tale valore sarò in grado di recuperare correttamente il valore di accuratezza cercato. Quindi, questra struttura implica che ogni volta che devo inserire un valore di qualità relativo ad un attributo (secondo livello), devo creare anche, se non esiste già, la tupla di primo livello corrispondente, perchè questo è fondamentale per il meccanismo di recupero dei dati di qualità. L'analisi degli accessi necessari per tale operazione è riportata in tabella 4; si nota che l'accesso alla relazione di secondo livello non deve essere considerato se l'aggiunta del valore di qualità e relativo ad una relazione di primo livello.

CONCETTO	ACCESSI	TIPO
Relazione dati	2	L
Relazione dati	1	S
Relazione secondo livello	1	S
Relazione di primo livello	1	S

TABELLA 4. Accessi necessari per l'inserimento di valori di qualità relativamente al caso *(i)*

Per quanto riguarda il caso *(ii)*, è necessario accedere prima la relazione dati per recuperare il qid. Con questo valore si è in grado di accedere in modo corretto al database della qualità. L'analisi degli accessi per questa operazione sono riportate in tabella 5 per relazioni di primo livello e in tabella 6 per quelle di secondo livello.

CONCETTO	ACCESSI	TIPO
Relazione dati	1	L
Relazione di primo livello	1	S

TABELLA 5. Accessi necessari per relazioni di I livello relativamente al caso *(ii)*

CONCETTO	ACCESSI	TIPO
Relazione dati	1	L
Relazione di primo livello	1	L
Relazione di secono livello	1	S

TABELLA 6. Accessi necessari per le relazioni di II livello relativamente al caso *(ii)*

3. Eliminazione dei valori di qualità. La rimozione di valori di singoli attributi (accuracy, completeness, ecc.) sono equivalenti al (ii) dell'aggiunta di nuovi valori, quindi gli accessi sono riportati nelle tabelle 5 e 6.

Potrebbe essere necessario rimuovere l'intera tupla in quanto richiesto direttamente dall'utente o perchè tutti i valori della tupla sono NULL. In generale, la rimozione di una tupla provoca la rimozione di tutti campi che referenziano a tale tuple. Ad esempio, se intendo rimuovere una tupla della relazione di secondo livello `Enterprise_name_Quality`, allorà dovrò modificare il campo `name_qid` nella relativa tupla di `Enterprise_Quality` impostandolo come NULL. Se intendo rimuovere un tupla di una relazione di primo livello, dovrò impostare a NULL il campo `qid` della relazione dati e dovrò eliminare tutte le tuple che referenzia nelle relazioni di II livello. Ad esempio, se voglio eliminare una tupla di `Enterprise_Quality`, dovrò eliminare anche le tuple di `Enterprise_name_Quality` e `Enterpise_id_Quality` referenziate attraverso i campi `name_qid` e `id_qid`.

4. Modifica dei valori di qualità. Per poter accedere al database della qualità devo prima recuperare il valore di qid relativamente alla tupla della relazione su cui intendo effettuare la modifica. Serve pertanto un accesso in lettura al database dei dati. Un altro accesso in lettura è necessario per la relazione di primo livello. Ho due possibilità: se il valore da modificare è un campo della relazione di primo livello, allora necessito di un accesso in scrittura su tale tupla; nel caso in cui debba modificare una relazione di secondo livello, allora devo recuperare prima la sua chiave dalla relazione di primo livello, accedo poi in lettura alla relazione di secondo livello e infine effettuo la modifica. I relativi accessi sono riportati nelle tabelle 7 e 8.

CONCETTO	ACCESSI	TIPO
Relazione dati	1	L
Relazione I livello	1	L
Relazione I livello	1	S

TABELLA 7. Accessi per la modifica di relazioni di primo livello

CONCETTO	ACCESSI	TIPO
Relazione dati	1	L
Relazione I livello	1	L
Relazione II livello	1	L
Relazione II livello	1	S

TABELLA 8. Accessi per la modifica di relazioni di secondo livello

5. Considerazioni finali. Tale soluzione, che deriva dall'applicazione diretta del modello D^2Q al caso relazionale, presenta diversi svantaggi. In primo luogo, tale struttura implica due costi diverso di accesso ai dati di qualità. Infatti, detta `d(X, qid)` la relazione del data schema, `q1(accuracy,..., currency, q2_qid)` la relazione di primo livello del quality schema e `q2(accuracy,..., currency)` la relazione di secondo livello del quality schema ho che:

- l'accesso alle relazioni di primo livello implica il seguente equijoin:

$$\texttt{d} \bowtie_{\texttt{d.qid=q1.id}} \texttt{q1}$$

- l'accesso alle relazioni di secondo livello implica, oltre al join precedente, anche il seguente equijoin:

$$\texttt{q1} \bowtie_{\texttt{q1.q2\_qid=q2.id}} \texttt{q2}$$

Ad esempio, considerando lo schema di figura 1, l'accesso alle tuple di `Enterprise_Quality` avvieni tramite il seguente join:

$$\texttt{Enterprise} \bowtie_{\texttt{allele.qid=allele\_Quality.id}} \texttt{Enterprise\_Quality}$$

Quindi, si paga un costo maggiore nell'accedere alle relazioni di secondo livello. Inoltre, dall'analisi degli accessi mostrati precedentemente, si ottiene che certe operazioni possono essere decisamente onoreso, come il caso dell'inserimento di nuovi valori di qualità

Si nota pertanto che la presenza delle relazioni di primo livello è necessaria per il funzionamento del sistema. Questo potrebbe portare ad un problema di occupazione della memoria. Infatti, aggiungere una tupla ad una relazione di secondo livello implica la creazione di tupla per la relazione di primo livello con valori dei inizializzati a NULL. Il rischio è pertanto quello di avere un database composto da molti record, molti dei quali sono composti da campi NULL.

Infine, un ultimo rproblema ruguarda il fatto che tale struttura dati costringe il sistema a modificare le relazioni del data schema aggiungendo un attributo `qid`.

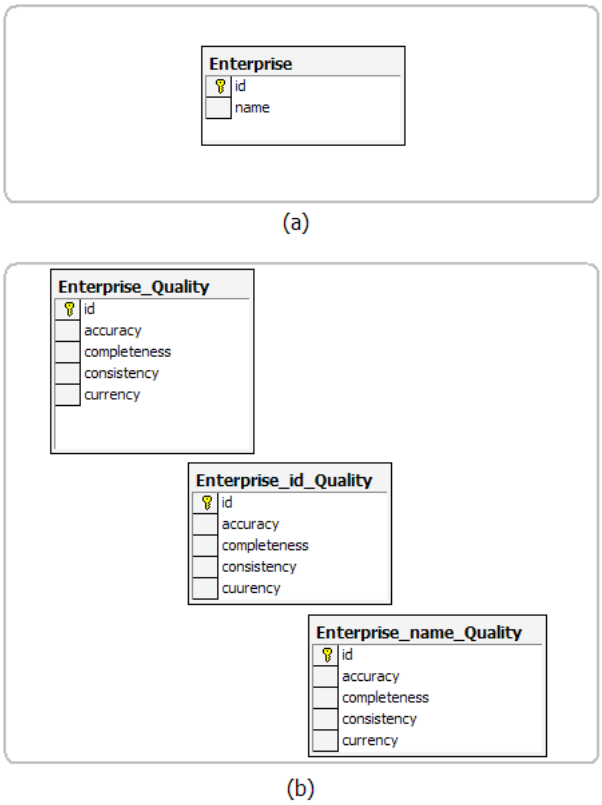


FIGURA 19. Struttura dati relativa alla seconda soluzione: (a) data schema e (b) quality schema

Soluzione ottimizzata. I problemi relativi alla proposta precedente possono essere risolti facilmente utilizzando la chiave primaria della relazioni dello schema dei dati per referenziare sia le relazioni di primo che di secondo livello nel database della qualità. In particolare, tutte le relazioni relative alla qualità (sia di primo che di secondo livello) prevedono:

- una chiave primaria, `id`, che è la chiave primaria della rispettiva tupla nel database dei dati.
- un attributo di tipo *quality type* per ogni dimensione definita dal quality schema.

Pertanto, rispetto al caso precedente:

- non è necessario modificare il database dei dati, in quanto non è prevista l'aggiunta di nessun attributo `qid` alle relazioni;
- è necessario un solo join per accedere alle relazioni di secondo livello, in quanto anche esse sono identificate dalla chiave primaria della relazione sui dati.
- le dimensioni del database sui dati sono più contenute in quanto sono eliminati i vincoli di integrità referenziale tra relazioni di primo e secondo livello.

Relativamente all'esempio di riferimento, il nuovo schema è rappresentato in figura 18. Si nota che nella relazione `Enterprise_Quality` non sono più necessari gli attributi `name_qid` e `id_qid`, in quanto non è previsto nessun vincolo referenziale tra tale relazione con quelle di secondo livello.

Conseguentemente anche il numero di accessi è inferiore rispetto il caso precedente. Di seguito sono descritte le operazioni di inserimento, rimozione e aggiornamento nella nuova struttura dati.

1. Modifiche al database dei dati. Con tale soluzione nessuna modifica è prevista al database dei dati, in quanto la struttura per creare le quality association si basa sulle chiavi primarie delle relazioni di tale database. Esistono tuttavia delle situazione nelle quali è possibile precedere la creazione di chiavi surrogate per l'istanza del database; in particolare, questa possib ilità si verifica quando la chiave primaria è composta e non semplice. In tali casi, aggiungo alla relazioni un attributo che contiene una chiave surrogata e uso questa chiave per creare le quality association.

2. Aggiunta di nuovi valori di qualità. Questa operazione consiste nel recuperare l'identificatore della tupla dati e successivamente di inserire il valore di qualità richiesto nella giusta relazione di qualità. Gli accessi al database sono mostrati in tabella 9.

CONCETTO	ACCESSI	TIPO
Relazione dati	1	L
Relazione qualità	1	S

TABELLA 9. Aggiunta di valori di qualità nella seconda soluzione

Si nota pertanto che tale operazione richiede meno accessi rispetto il caso precedente. Inoltre, il procedimento seguito per l'inserimento dei valori è sempre lo stesso, e pertanto non si pagano costi diversi.

3. Eliminazione dei valori di qualità. Tale operazione è analoga alla precedente in quanto consiste nel: recupero della chiave primaria dal database dei dati e accesso al database della qualità per eliminare il valore. Pertanto, gli accessi necessari sono gli stessi del processo di inserimento, per cui sono riportati in tabella 9.

4. Aggiornamento di valori di qualità. Anche questo caso è analogo ai precedenti, in quanto sono previsti gli stessi accessi.

5. Considerazioni finali. Da quanto illustrato si nota che l'eliminazione del vincolo di integrità referenziale tra le relazioni di primo e secondo livello permette di ottenere più omogeneità negli accessi ai valori di qualità. Inoltre, il numero stessi di accessi necessari sono sensibilmente inferiori nel caso attuale, garantendo un costo inferiore per le operazioni sui database. In questo caso infatti è sufficiente un singolo equijoin tra relazione dati e relazioni di qualità.

CAPITOLO 8

Conclusioni

Nella presente tesi sono stati affrontati alcuni dei problemi principali riguardanti la gestione e l'integrazione di sorgenti informative e di risorse eterogenee in contesti altamente distribuiti quale quello peer-to-peer.

In particolare si è visto che MDA e MOF propongono una soluzione all'eterogeneità tecnica dei sistemi, definendo una meta-architettura a 4 livelli che permette di progettare modelli PIM di sistemi dai quali possono essere ricavati diverse rappresentazioni PSM e quindi diverse implementazioni tecnologiche. Tuttavia, questa architettura da sola non consente di superare le eterogeneità a livello di schema, a causa della mancanza di semantica. Questa mancanza ha portato l'OMG ha rilasciare una RFP per un Ontology Definition Metamodel (ODM), con lo scopo di riutilizzare ontologie già definite grazie agli strumenti del Semantic Web.

Digital Business Ecosystem definisce infatti un framework MDA-based in base al quale sono stati definiti tutti i linguaggi utilizzati per catturare e rappresentare la conoscenza condivisa dai peer del sistema. Tra di essi è stato definito, in risposta alla RFP di OMG, un Ontology Definition Metamodel e un mapping fra esso e OWL-DL, che consente di riutilizzare in DBE le ontologie definite in contesti diversi. Ad ogni modo, in DBE i concetti del Semantic Web non vengono recepiti in modo completo: ODM è infatti utilizzato esclusivamente per fornire tipi di dati per gli elementi definiti nei modelli BML, SSL e SDL. In generale, lapproccio semantico di DBE differisce in modo consistente da quello di NeP4B, in quanto non vengono definite relazioni semantiche tra diversi peer per mezzo di mapping; lo stesso ODM non supporta la definizione di relazioni intensionali in quanto non prevede costrutti adeguati per definire tale tipo di relazioni. Inoltre, lo stesso linguaggio non prevede un mapping con le Logiche Descrittive, limitando quindi la possibilità di sviluppare meccanismi di inferenza avanzati.

Altre limitazioni di DBE sono state analizzate nei capitoli 4 e 5 relativamente alla modellazione di servizi. Come si è visto, BML non permette di definire la molteplicità degli attributi dei suoi elementi, limitando di molto la potenzialità espressiva del linguaggio e costringendo a ricorrere ad alternative che però compromettono la semantica del modello rappresentato.

Per quanto riguarda il data quality si è visto come esso possa essere utilizzato nei sistemi di integrazione delle informazioni per creare sistemi di query processing quality-driven e per definire funzioni di risoluzione dei conflitti a livello d'istanza, anche se tuttora non vi è un accordo comune sui modelli di dati migliori e sulle dimensioni da adottare (anche su alcune, come accuracy e completeness, vengono adottate con definizioni simili in molti lavori). *DaQuinCIS*, sistema progettato per i *Cooperative Information Systems* (CIS), definisce un modello flessibile ma che costringe ad avere molti metadati associati alle informazioni. In un contesto relazionale, gestire molti metadati significa dover mantenere e gestire grosse

basi di dati, con possibili problemi di efficienza. Altri approcci, come QP-alg, Fusionplex e Aurora permettono invece di avere metadati con livelli di granularità diversi relativamente alle informazioni alle quali sono associate. In particolare, soluzioni che permettano di esprimere giudizi sulla qualità a livello di sorgente, di query, o di schema element assumono una notevole importanza in ambito peer-to-peer e, pertanto, dovranno essere studiati anche nellambito di NeP4B.

Infine, un aspetto molto importante è stato la possibilità che questa attività progettuale mi ha dato di partecipare a due meeting del progetto NeP4B, esperienza che mi ha permesso di entrare in contatto con i meccanismi del mondo del lavoro. Molto utile è stato inoltre l'utilizzo di Mailing List e Forum al fine di interagire con personale competente, soprattutto per quanto riguarda il progetto DBE.

Possibili sviluppi futuri associati alla tesi riguardano:

- ❖ Approfondimento delle strategie di traduzione tra modelli PIM e PSM nella tecnologia MDA e della loro applicazione ad una architettura wrapper-mediatore, come quella di MOMIS.
- ❖ Versificare se nella release finale di DBE vengono risolti i problemi e ele lacune della versione attuale.
- ❖ Implementazione e sviluppo dello scenario descritto nel capitolo 5.
- ❖ Creazione di strutture dati che raccolgano attributi aggregati i quali forniscano un valore cumulativo sulla qualità di un sorgente informativa.

Bibliografia

ARENAS, M., KANTERE, V., KEMENTSIETSIDIS, A., KIRINGA, I., MILLER, R. J. & MYLOPOULOUS, J. (2003). The Hyperion Project: From Data Integration to Data Coordination. *SIGMOD Record, 32(3)*.

BALLOU, D.P. & PAZER, H.L. (1985). Modeing data and process quality in multi-input information systems. *Management Science, vol. 31, no. 2*.

BENETTI, I., GUERRA, F., VINCINI, M. & BERGAMASCHI, S. (2004). SOAP-enabled web services for knowledge management. *International Journal of Web Engineering and Technology, Vol. 1, No. 2*.

BENEVENTANO, D., BERGAMASCHI, S., GUERRA, F. & VINCINI, M. (2005). Querying a Super-Peer in a Schema-Based Super-Peer Network. In *Proceedings of the 3rd VLDB International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P 2005), Trondheim, Norway*.

BERGAMASCHI, S., BENEVENTANO, D., CASTANO, S. & VINCINI, M. (1998). Integrazione di informazione: il linguaggio ODL_{I^3} e la logica descrittiva OLCD. *Technical Report T3-R03, Università degli Studi di Modena e Reggio Emilia*.

BERGAMASCHI, S., BENEVENTANO, D., VINCINI, M. (1999). Progettazione di Basi di Dati Relazionali: Lezioni ed Esercizi. *Pitagora Editrice*.

BERGAMASCHI, S., BOUQUET, P., CIACCIA, P. & MERIALDO, P. (2005). Speaking Words of WISDOM: Web Intelligent Search based on DOMain ontologies. In *Proceedings of the 2nd SWAP Italian Semantic Web Workshop*.

BERGAMASCHI, S., CASTANO, S., BENEVENTANO, D., VINCINI, M. (2001). Semantic Integration of Hetergogeneous Information Sources. *Data & Knowledge Engineering, 36(1)*.

BERGAMASCHI, S., GUERRA, F. & VINCINI, M. (2003). A Peer-to-Peer Information System for the Semantic Web. In *Proceedings of the International Workshop on Agents and Peer-to-Peer Computing (AP2PC03), Melbourne, Australia*.

BERNERS-LEE, T., HENDLER, J. & LASSILA, O. (2001). The Semantic Web. *Scientific American*.

BERNESTEIN, P. A., GIUNCHIGLIA, F., KEMENSTSIETSIDIS, A., MYLOPOULOS, J., SERAFINI L. & ZAIHRAYEU, I. (2002). Data Management in Peer-to-Peer Computing: A Vision. In *Proceedings of the 5th International Workshop on the Web and Databases (WebDB 2002), Madison, Wisconsin, USA.*

BÉZEVIN, J. (2004). In Search Of Basic Principle For Model Driven Engineering. *UPGRADE Vol. V, No. 2.*

BOLDSOFT, INTERNATIONAL BUSINESS MACHINES CORPORATION (IBM), IONA & ADAPTIVE LTD. (2003). OCL 2.0 OMG Final Adopted Specification. *OMG Adopted Specification, http://www.omg.org/docs/ptc/03-10-14.*

BOUZEGHOUB, M. (2003). MDA: Some Lessons Learned from Data Base Technology and Design. In *OMG Meetings, Paris.*

BOVEE, M., STRIVASTRAVA, R. P. & MAK, B. R. (2001). A conceptual Framework and Belief-Function Approach to Assessing Overall Information Quality. In *Proceedins of the 6th International Conference on Information Quality, Boston, MA.*

BRICKLEY, D., GUHA, R.V. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendaation, http://www.w3.org/TR/rdf-schema/.*

BRISCOE, G. & DE WILDE, P. (2005). D6.4 Intelligence, Learning and Neural Networks in Distributed Agent Systems. *DBE Project Deliverable, http://www.digital-ecosystem.org/.*

CAPPIELLO, C., FRANCALANCI, C., MISSIER, P., PERNICI, B., PLEBANI, P., SCANNAPIECO, M. & VIRGILLITO, A. (2002). DL2: Presentation of Metadata and of the Quality Certificate. *DaQuinCIS Project Deliverable, http://www.dis.uniroma1.it/~dq/.*

CHARNES, A., COOPER, W. W. & RHODES, E. (1978). Measuring the Efficiency of Decision Making Units. *European Journal of Operational Research, 2.*

CHEN, P. P. (1976). The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transaction on Database Systems, Vol. 1, No. 1, pp. 9-37.*

CHRISTODOULAKS, S., GIOLDASIS, N., KAZASIS, F.G., MARAGOUDAKIS, Y., CORALLO, A. & DA TOMASI, M. (2005). D14.1 DBE Knowledge Representation models. *DBE Project Deliverable, http://www.digital-ecosystem.org/.*

CRESPO, A. & GARCIA-MOLINA, H. (2003). Semantic Overlay Networks. *Technical Report, Stanford University.*

DASWANI, N., GARCIA-MOLINA, H., AND YANG, B. (2003). Open Problems in Data-Sharing Peer-to-Peer Systems. In *Proocedings of the 9th International Conference on Database Theory, Siena, Italy.*

DE GIACOMO, G., LEMBO, D., LENZERINI, M. & ROSATI, R. (2004). Tackling Inconsistencies in Data Integration through Source Preferences. In *Proceedings of the International Workshop on Information Quality in Information Systems (IQIS 2004)*, pp 27-34.

DE TOMASI, M., CORALLO, A. & CISTERNINO, V. (2005A). D15.1 Business Modelling Language 1.0. *DBE Project Deliverable, http://www.digital-ecosys-tem.org/.*

DE TOMASI, M., CORALLO, A. & CISTERNINO, V. (2005B). D15.3 BML Framework 2nd Release. *DBE Project Deliverable, http://www.digital-ecosys-tem.org/.*

FELLEGI, I.P. & SUNTER, A.B. (1969). A Theory for Record Linkage. *Journal of the America Statistical Association, vol. 64.*

FERRONATO, P. & NEAGU, A. (2005). D21.2 Architecture Scope Document. *DBE Project Deliverable, http://www.digital-ecosystem.org/.*

FRANKEL, D., HAYES, P., KENDALL, E. & MCGUINNESS, D. (2004). The Model Driven Semantic Web. In *Proocedings of the First International Workshops on Model-Driven Semantic Web (MDSW2004), Monterey, CA.*

GRUBER, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Aquisition, Vol. 5, No. 2.*

HAASE, P. & BROCKMANS, S. (2006). A Metamodel and UML Profile for Networked Ontologies - The Complete Reference. *Technical Report, Università di Karlsruhe.*

HALEVY, A., IVES, Z., MADHAVAN, J., MORK, P., SUCIU, D. & TATARINOV, I. (2004). The Piazza Peer Data Management System. *IEEE Transaction on Knowledge and Data Engineering, Vol. 16, No. 7.*

HALEVY, A. Y., IVES, Z. G., MORK, P., TATARINOV, I. (2003). Piazza: Data Management Infrastructure for Semantic Web Applications. In *Proceedings of the 12th International World Wid Web Comference (WWW2003), Budapest, Hungary*.

HERNANDEZ, M.A. & STOLFO, S.J. (1998). Real-word Data is Dirty: Cleansing and The Merge/Purge Problem. *Journal of Data Mining Knowledge Discovery, 1(2).*

JARKE, M., LENZERINI, M., VASSILIOU, Y. & VASSILIADIS, P. (1999). Fundamentals of Data Warehouses. *Spinger Verlag.*

KAHN, B. K., STRONG, D. M., WANG, R. Y. (2002). Information Quality Benchmarks: Product and Service Performance. *Communication of the ACM, vol. 45, no. 4, pp. 184-192.*

KAZASIS, F. G., KOTOPOULOS, G., KOTOPOULOS, Y., CHRISTODOULAKIS, S., GIOLDAS, N. & PAPPAS, N. (2005). D17.1 Recommender. *DBE Project Deliverable, http://www.digital-ecosystem.org/*.

KOTOPOULOS, G., KOTOPOULOS, Y., PAPPAS, N. & KAZASIS, F. (2006). D14.4 Second Release of the Recommender. *DBE Project Deliverable, http://www.digital-ecosystem.org/*.

LENZERINI, M. (2002). Data Integration: a Theoretical Perspective. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems (PODS 2002), Madison, Wisconsin, USA*.

LENZERINI, M. (2004). Quality-Aware Data Integration in Peer-to-Peer Systems. In *Proocedings of the First International Workshops on Information Quality in Information Systems (IQIS' 2004), Paris, France*.

LOSER, A., NAUMANN, F., SIBERSKY, W. , NEJDL, W. & THADEN, U. (2003). Semantic Overlay Clusters Within Super-Peer Networks. In *Aberer K., Kalogeraki, V., Koubarakis, M., editords, DBISP2P, volume 2944 of Lecture Notes in Computer Science, pages 33-47, Springer*.

MANOLA, F. & MILLER, E. (2004). Resource Description Framework (RDF) Model and Syntax Specification. *W3C Recommendation, http://www.w3.org/TR/rdf-primer/*.

MCGUINNESS, D.L. & VAN HARMELEN, F. (2004). OWL Web Ontology Language Overview. *W3C Recommendation, http://www.w3.org/TR/owl-features/*.

MCKITTERICK, D., NICKEL, L. & DOWLING, J. (2005). D17.2 Manual Composer. *DBE Project Deliverable, http://www.digital-ecosystem.org/*.

MECELLA, M., SCANNAPIECO, M., VIRGILLITO, A., BALDONI, R., CATARCI, T. & BATINI, C. (2002). Managing Data Quality in Cooperative Information Systems. In *Proceedings of the 10th International Conference on Cooperative Information Systems (CoopIS'02), Irvine, CA*.

MICROSOFT, IBM, SIEBEL SYSTEMS, BEA & SAP (2003). Business Process Execution Language for Web Service, version 1.1. *http://www-128.ibm.-com/developerworks/library/specification/ws-bpel/*.

MONTANARI, M., FERRONATO, P., NEAGU, A. & TRENTIN, V. (2005A). D16.1 Service Description Models and Language Definition - Part 1 SDL Definition. *DBE Project Deliverable, http://www.digital-ecosystem.org/*.

MONTANARI, M., FERRONATO, P., NEAGU, A. & TRENTIN, V. (2005B). D16.1 Service Description Models and Language Definition - Part 2 Service Manifest Conceptual and Software Models. *DBE Project Deliverable, http://www.digital-ecosystem.org/*.

MOTRO, A., BERLIN, J. & ANOKHIN P. (2004). Multiplex, Fusionplex and Autoplex - Three Generations of Information Integration. *ACM SIGMOD Record, 33(4)*.

MOTRO, A. & ANOKHIN, P. (2005). Fusionplex: Resolution of Data Inconsistencies in the Data Integration of Heterogeneous Information Sources. *Information Fusion*.

NACHIRA, F. (2002). Towards a Network of Digital Business Ecosystems Fostering the Local Development. *http://www.digital-ecosystems.org/doc/discus-sionpaper.pdf*

NAUMANN, F. (2002). Quality-Driven Query Answering for Integrated Information Systems. *Lecture Notes in Computer Science, 2261*.

NAUMANN, F., LESER, U. & FREYTAG, J.C. (1999). Quality-Driven Integration of Heterogeneous Information Systems. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99), Edinburgh, Scotland, UK*.

NEILING, M., JURK, S., LENZ, H-J. & NAUMANN, F. (2003). Object Identification Quality. *Workshop on Data Quality in Cooperative Information Systems (DQCIS2003), Siena, Italia*.

NEJDL, W., WOLF, B., QU, C., DECKER, S., SINTEK, M., NAEVE, A., NILSSON, M., PALMER, M. & RISCH, T. (2003). Editella: A P2P Networking Infrastructure Based on RDF. In *Proceedings of the 12th International Conference on World Wide Web (WWW2003)*.

OMG (2000). XML Metadata Interchange Specification. *Object Management Group Specification, http://www.omg.org/cgi-bin/doc?formal/2000-06-01/*.

OMG (2002). OMG/RFP/QVT MOF 2.0 Query/Views/Transformations RFP. *Object Management Group Document, http://www.omg.org/docs/ad/02-04-10/*.

OMG (2003A). MDA Guide Version 1.0.1. *Object Management Group Document, http://www.omg.org/cgi-bin/doc?omg/03-06-01/*.

OMG (2003B). UML 2.0 OCL Specification. *Object Management Group Specification http://www.omg.org/docs/ptc/03-10-14/*.

OMG (2003C). Ontology Definition Metamodel Request For Proposal. *OMG Document, http://www.omg.org/docs/ad/2003-03-40/*.

OMG (2005A). Meta Object Facility (MOF) Specification Version 1.4. *Object Management Group Specififcation, http://www.omg.org/cgi-bin/doc?for-mal/05-05-05/*.

OMG (2005B). XML Metadata Interchange Specification Version 2.0.1. *Object Management Group Specification, http://www.omg.org/cgi-bin/doc?for-mal/2005-05-06/*.

OMG (2006A). Meta Object Facility (MOF) Core Specification Version 2.0. *Object Management Group Specification, http://www.omg.org/cgi-bin/doc?-formal/2006-01-01/.*

OMG (2006B). Unified Modeling Language: Infrastructure Specification Version 2.0. *Object Management Group Specification http://www.omg.org/cgi-bin/doc?formal/05-07-05*.

ORR, K. (1998). Data Quality and Sysrems Theory. *Communication of the ACM, vol. 4, no. 2*.

PAN, J. & HORROCKS, I. (2001). Metamodeling Architecture of Web Ontology Languages. In *Proceedings of the First Semantic Web Working Symposium (SWWS'01), Stanford, CA.*

PIPINO, L. L., LEE, Y. W. & WANG, R. Y. (2002). Data Quality Assessment. *Communication of the ACM, Vol. 45, No 4*.

RAHM, E. & BERNSTEIN, P.A. (2001). A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal, 10(4)*.

REDMAN, T. C. (1996). Data Quality for the Information Age. *Artech House*.

SCANNAPIECO, M., VIRGILLITO, A., MARCHETTI, M., MACELLA, M. & BALDONI, R. (2004). The DaQuinCIS Architecture: a Platform for Exchanging and Improving Data Quality in Cooperative Information Systems. *Information Systems, 29(7)*.

SCHALLEHN, E., SATTLER, K. U. & SAAKE, G. (2002). Extensible and Similarity-Based Grouping for Data Integration. In *Proceedings of the 18th Internation Conference on Data Engineering, San Jose, California, USA.*

SHANKARANARAYAN, G., WANG, R.Y. & ZIAD, M. (2000). Modeling the Manufacture of an Information Product with IP-MAP. In *Proceedings of the 6th International Conference on Information Quality, Boston, MA.*

THEOTOKIS, S. A. & SPINELLIS, D. (2004). A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Survey, Vol. 36, No. 4, pp 335-371*.

WAND, Y. & WANG, R. Y. (1996). Anchoring Data Quality Dimensions in Ontological Foundations. *Communication of the ACM, vol. 39, no. 11*.

WANG, R.Y., & STRONG, D.M. (1996). Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems, 12(4), 5–34*.

WANG, R.Y. & KON, H. B. (1995). Toward Quality Data: An Attribute-based Approach. *Decision Support Systems, 13, pp 349-372*.

WANG, R.Y., KON, H.B. & MADNICK, S.E. (1993). Data Quality Requirements: Analysis and Modeling. In *Proceedings of the 9th Internationl Conference on Data Engineering (ICDE'93), Vienna, Austria*.

YAN, L. L. & OZSU, T. (1999). Conflict Tolerant Queries in AURORA. In *Proceedings of the 4th Internation Conference on Cooperative Information Systems (CoopIS'99), Edinburgh, Scozia.*