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

Estrazione di relazioni lessicali con WordNet nel sistema MOMIS

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Giovanni Malvezzi

Correlatore
Ing. Alberto Corni

Controrelatore
Chiar.mo Prof. Michele Colajanni

Anno Accademico 1999 - 2000

Parole chiave:

Linguistica computazionale
Integrazione semantica
Relazioni lessicali
Thesaurus
WordNet

RINGRAZIAMENTI

Un sentito ringraziamento va alla Professoressa Sonia Bergamaschi per l'aiuto che mi ha fornito durante la realizzazione della presente tesi e per la costante disponibilità dimostrata.
Vorrei inoltre ringraziare tutti i componenti del team MOMIS, in particolare l'Ing. Alberto Corni, per i consigli ed i chiarimenti di ordine pratico ed implementativo, e l'Ing. Domenico Beneventano per la preziosa collaborazione.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

La presente tesi si colloca nel proficuo filone di ricerca che mira a condurre un'analisi su come sia possibile ottenere maggiore integrazione tra sorgenti dati eterogenee.

Per sorgente dati si intende una qualunque fonte di conoscenza anche semistrutturata, sviluppata in maniera autononoma senza nessuna predisposizione per la fusione con altre sorgenti dati. Ad esempio database relazionali, database ad oggetti, documenti semistrutturati, quali documenti XML e file dati.

L'integrazione è una esigenza molto sentita a partire dai sistemi informativi delle piccole aziende cresciuti rapidamente e confusamente intorno a più fornitori, fino alle multinazionali protagoniste di grandi fusioni nell'ambito del mercato globale.

Anche i cittadini privati, navigando su Internet, possono ottenere benefici da una maggiore integrazione delle informazioni: *shop bot*, agenti automatici per la rilevazione del prezzo più basso o dei servizi migliori; *motori di catalogazione*, creano una vista virtuale su contenuti affini distribuiti; *portali per l'e-commerce*, reali *mall* in grado di offrire servizi diversificati, ma omogenei [1].

Più in particolare questa tesi si inserisce nel progetto **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) [2, 3, 4, 5, 6, 7, 8, 9, 10] una ricerca condotta dal Dipartimento di Scienze dell'Ingegneria di Modena e dal Dipartimento di Scienze dell'Informazione di Milano, ricerca svolta nell'ambito del progetto INTERDATA - MURST 40% 97-98.

MOMIS si pone l'obiettivo di integrare sorgenti dati eterogenee distribuite ed eventualmente semistrutturate, e proporne all'utilizzatore una vista globale, aggregata, virtuale che permetta all'utente di formulare interrogazioni in modo trasparente.

MOMIS è formato da due macro-componenti:

Global Schema Builder: è il modulo di integrazione degli schemi locali, che, partendo dalle descrizioni delle sorgenti, genera un unico schema globale da presentare all'utente. L'interfaccia grafica di questo modulo, cioè il tool di ausilio al progettista, è SI-Designer.

Query Manager: è il modulo di gestione delle interrogazioni. In particolare, genera le query da inviare ai wrapper partendo dalla singola query formulata dall'utente sullo schema globale

La presente tesi si pone nella prima fase (di ausilio al progettista) ed, in breve, l'obiettivo è quello di individuare relazioni implicite tra classi (o entità o tabelle) di sorgenti diverse in modo da identificare/accomunare classi simili tra sorgenti diverse. Le relazioni individuate dal componente sviluppato nella tesi sono le relazioni lessicali tra i nomi delle classi e degli attributi. In particolare, si lavora sul significato delle parole usate per descrivere il contenuto delle classi e degli attributi: un tipo di conoscenza implicita nel nome assegnato dal progettista. Su questi nomi, un sistema di gestione di basi di dati esegue solo semplici controlli sintattici, ma per il resto li tratta come identificatori. E' compito del progettista attribuire ai campi un nome significativo. Per cui c'è un'incertezza di interpretazione insita nell'ambiguità del linguaggio; Bates in [Bate86] scrive *"the probability of two persons using the same term in describing the same thing is less than 20%"*. Questa conoscenza è comunque un'opportunità che deve essere sfruttata per estrarre relazioni, e non si può pensare di farlo in modo manuale, cioè chiedere ad un operatore di analizzare gli schemi sorgenti, confrontare ogni termine ed inserire tutte le relazioni in un thesaurus, perché è un compito che, col crescere del numero degli schemi, diventa di complessità non trattabile per un essere umano.

Per trovare le relazioni lessicali non vengono in aiuto i comuni strumenti del campo delle basi di dati, ma bisogna affidarsi a strumenti di linguistica computazionale.

In questa tesi è stato utilizzato WordNet [11, 12, 13, 14, 15], un database lessicale che è considerato la più importante risorsa disponibile per i ricercatori nei campi della linguistica computazionale, dell'analisi testuale, e di altre aree associate.

La tesi segue un percorso dal generale al particolare ed è così articolata:

Capitolo 1. *Architettura generale di MOMIS*: introduzione generale a MOMIS (per approfondimenti consultare la bibliografia), ed inquadramento della seguente tesi all'interno del progetto generale.

Capitolo 2. *Il database lessicale WordNet*: analisi delle funzionalità messe a disposizione da questo strumento.

Capitolo 3. *Estrazione lessicale di relazioni*: parte centrale della tesi, che illustra il problema e la soluzione proposta.

Capitolo 4. *Tool grafico semiautomatico: progetto e realizzazione*: il componente grafico integrato dentro a SI-Designer per l'interazione con l'utente.

Capitolo 5. *L'organizzazione del software*: viene illustrata la struttura del software realizzato durante lo svolgimento della presente tesi.

La tesi è correlata dalle seguenti appendici:

- *Statistiche su WordNet*: grafici e diagrammi non inclusi nel capitolo 2 per non appesantirlo.
- *Glossario WordNet*: utile per districarsi nella terminologia di WordNet mutuata dalla linguistica.
- *Esempi di riferimento*: Codice degli esempi di riferimento utilizzati in questo lavoro in linguaggio ODL_{I^3} .
- *Tecnologia JavaBeans*: breve introduzione ad una tecnologia utilizzata in questa tesi: i JavaBeans.

Il lavoro svolto nella presente tesi ha ottenuto un riconoscimento nazionale ed internazionale.

I risultati della tesi sono contenuti in un articolo presentato all'Ottavo Convegno Nazionale su Sistemi Evoluti per Basi di Dati - {SEBD2000}, L'Aquila, 26-28 giugno 2000, autori D. Beneventano, S. Bergamaschi, A. Corni, R. Guidetti e G. Malvezzi dal titolo "SI-Designer: un tool di ausilio all'integrazione intelligente di sorgenti di informazione".

Il componente realizzato durante la tesi è parte importante del sistema MOMIS che verrà presentato alla prossima conferenza internazionale Very Large DataBase {VLDB2000}, Cairo (Egitto), 10-14 settembre 2000, autori D. Beneventano, S. Bergamaschi, S. Castano, A. Corni, R. Guidetti, G. Malvezzi, M. Melchiori e M. Vincini dal titolo "Information Integration: the MOMIS Project Demonstration".

Capitolo 1

Architettura generale di MOMIS

Il progetto MOMIS ha come obiettivo l'integrazione intelligente delle informazioni in sorgenti di dati sia strutturate che semistrutturate. SI-Designer (*Source Integrator Designer*) è un tool di supporto al progettista per l'integrazione semi-automatica di schemi di sorgenti eterogenee (relazionali, ad oggetti e semistrutturate).

Realizzato nell'ambito del progetto MOMIS, SI-Designer esegue l'integrazione seguendo un *approccio semantico*, che fa uso di tecniche intelligenti, basate sulla *Description Logics* **OLCD** [16, 17] (*Object Language with Complements allowing Descriptive cycles*), di tecniche di clustering e di un linguaggio object-oriented per rappresentare le informazioni estratte ed integrate, $\mathrm{ODL_{I^3}}$ [18], derivato dallo standard ODMG.

Partendo dalle descrizioni delle sorgenti in $\mathrm{ODL_{I^3}}$ (gli *schemi locali*) SI-Designer assiste il progettista nella creazione di una vista integrata di tutte le sorgenti (*schema globale*) anch'essa espressa in linguaggio $\mathrm{ODL_{I^3}}$.

Lo schema globale viene ottenuto in fasi successive, creando un *Common Thesaurus* di relazioni intra ed inter-schema.

Le sorgenti da integrare vengono descritte attraverso il linguaggio $\mathrm{ODL_{I^3}}$ e, utilizzando le tecniche di inferenza di **OLCD** , vengono estratte relazioni intensionali intra-schema che sono inserite nel *Common Thesaurus*.

Dopo questa fase iniziale il *Common Thesaurus* viene arricchito aggiungendo relazioni inter-schema ottenute:

- utilizzando il sistema lessicale WordNet, che identifica le affinità tra concetti inter-schema sulla base di lessico/significato delle loro denominazioni;
- utilizzando il sistema ARTEMIS [18] che calcola le affinità strutturali di concetti inter-schema.

Partendo dal *Common Thesaurus* ottenuto ed usando ancora le tecniche di

inferenza di **OLCD** e le tecniche di clustering di ARTEMIS, si definisce uno schema globale che contiene la visione d'insieme delle sorgenti integrate.

1.1 L'architettura di MOMIS

MOMIS è stato progettato per fornire un accesso integrato ad informazioni eterogenee memorizzate sia in database di tipo tradizionale (e.g. relazionali, object-oriented) o file system, sia in sorgenti di tipo semistrutturato. Seguendo l'architettura di riferimento I^3 [19], in MOMIS si possono distinguere quattro componenti principali per la fase di integrazione delle sorgenti (Figura 1.1):

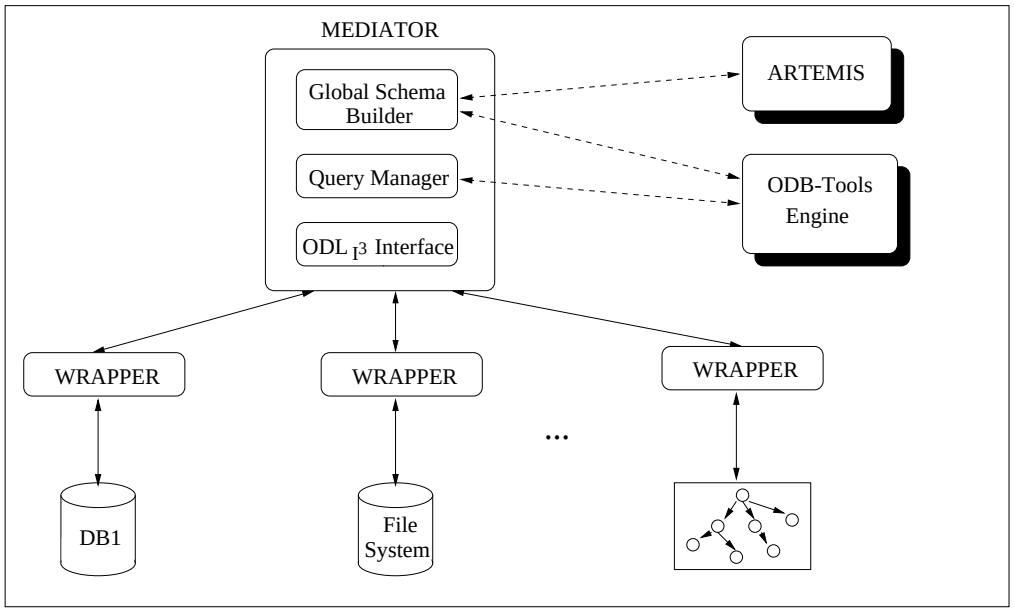


Figura 1.1: Architettura di MOMIS

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

   - in fase di integrazione, forniscono la descrizione delle informazioni in essa contenute. Questa descrizione viene fornita attraverso il linguaggio ODL$_{I^3}$;
   - in fase di query processing, traducono la query ricevuta dal mediatore (espressa quindi nel linguaggio comune di interrogazione OQL$_{I^3}$

, definito a partire dal linguaggio OQL) in una interrogazione comprensibile dalla sorgente stessa. Devono inoltre esportare i dati ricevuti in risposta all'interrogazione, presentandoli al mediatore attraverso il modello comune di dati utilizzato dal sistema.

2. *Mediatore*: è il cuore del sistema, ed è composto da due moduli distinti.
   - *Global Schema Builder (GSB)*: è il modulo di integrazione degli schemi locali, che, partendo dalle descrizioni delle sorgenti espresse in ODL_{I^3} , genera un unico schema globale da presentare all'utente. L'interfaccia grafica di GSB, cioè il tool di ausilio al progettista, è SI-Designer.
   - *Query Manager* (QM): è il modulo di gestione delle interrogazioni. In particolare, genera le query in linguaggio OQL_{I^3} da inviare ai wrapper partendo dalla singola query formulata dall'utente sullo schema globale. Servendosi di tecniche delle *Description Logics* di ODB-Tools il QM genera automaticamente la traduzione della query sottomessa nelle corrispondenti sub-query delle singole sorgenti.
3. *ODB-Tools Engine*, un tool basato sulle *Description Logics* OCDL [16, 17] che compie la validazione di schemi e l'ottimizzazione di query [20, 21, 22].
4. *ARTEMIS-Tool Environment*, un tool basato sulle tecniche di clustering *affinity-based* che compie l'analisi ed il clustering delle classi $\mathrm{ODL}_{\mathrm{I}^3}$ [18].

1.1.1 L'architettura di SI-Designer

L'integrazione delle sorgenti realizzata dal *Global Schema Builder* si basa sull'individuazione di una antologia, comune alle diverse sorgenti, sotto forma di thesaurus: un insieme di relazioni terminologiche chiamato *Common Thesaurus*. Come mostrato in Figura 1.2, *GSB* è composto da due moduli:

- SIM (*Source Integrator Module*): estrae relazioni intensionali intra-schema sulla base della struttura delle classi $\mathrm{ODL}_{\mathrm{I}^3}$ e delle sorgenti relazionali, utilizzando ODB-Tools. Oltre a ciò, il modulo si occupa della "validazione semantica" delle relazioni e ne inferisce di nuove tramite ODB-Tools.
- SLIM (*Schemata Lessical Integrator Module*): estrae relazioni intensionali inter-schema tra nomi di attributi e classi $\mathrm{ODL}_{\mathrm{I}^3}$ utilizzando la conoscenza espressa in WordNet.

Il tool SI-Designer (vedi Figura 1.2 e Figura 1.3) mette a disposizione del progettista una interfaccia per interagire con i moduli SIM, SLIMed ARTEMIS,

mostrando, via via, le relazioni estratte ed assistendo il progettista nella creazione del *Common Thesaurus*, cioè di un'ontologia comune alle diverse sorgenti.

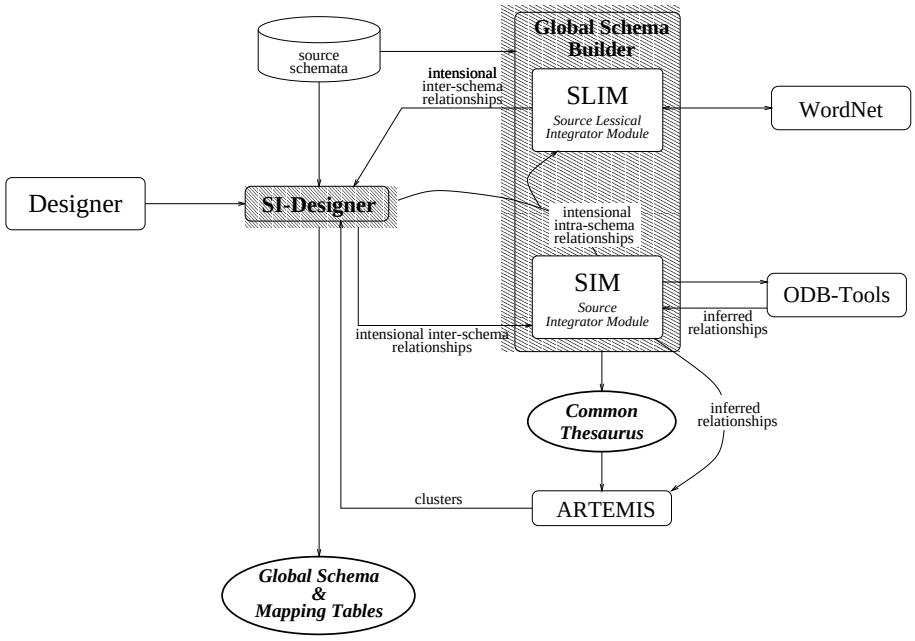


Figura 1.2: Architettura di Global Schema Builder.

Costruito il *Common Thesaurus*, SI-Designer utilizza nuovamente il modulo ARTEMIS per individuare, tramite un algoritmo di clustering, gli insiemi disgiunti di classi affini: i *cluster*. Ad ogni cluster corrisponderà una *classe globale* (una vista su tutte le classi affini appartenenti al cluster) caratterizzata da un insieme di attributi globali e da una *mapping-table*. SI-Designer costruisce quindi un insieme di attributi globali per ogni classe globale e la relativa *mapping table*: questo viene realizzato attraverso un processo semi-automatico in cui il progettista deve revisionare l'insieme di attributi globali e la mapping-table ed assegnare un nome ad ogni classe globale, in modo da pervenire ad uno schema globale chiaro. In sostanza, il procedimento di integrazione può essere suddiviso in due fasi: (**1**) Generazione del *Common Thesaurus*, (**2**) Generazione delle classi globali.

1.1.2 Esempio di riferimento

In Appendice C.1 è riportato per esteso l'esempio che verrà utilizzato in questo capitolo per illustrare il funzionamento di SI-Designer. Sono presenti 3 sorgenti eterogenee: una sorgente relazionale (`University`), una sorgente semistrut-

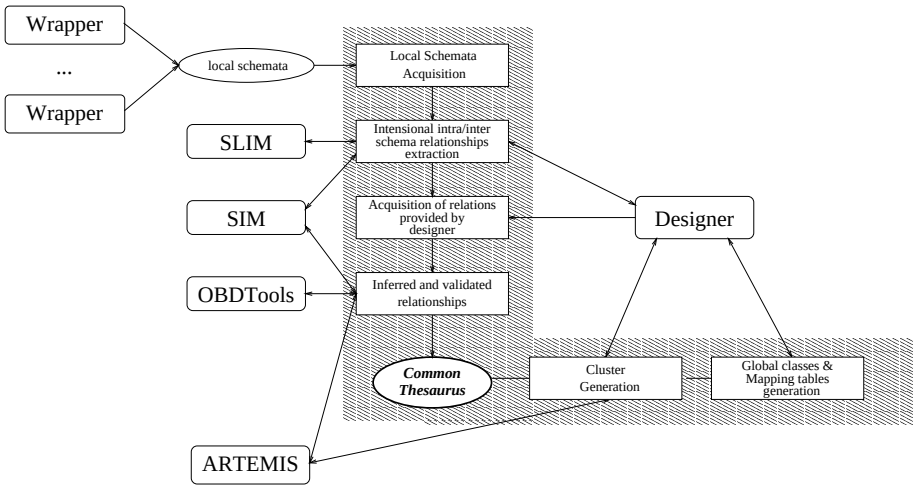


Figura 1.3: Architettura di SI-Designer.

turata (`Computer_Science`) (originariamente un file XML) e una sorgente costituita da un semplice file (`Tax_Position`).

La sorgente `University` contiene le informazioni sugli studenti e lo staff di un'università ed è composta da cinque relazioni: `Research_Staff`, `School_Member`, `Department`, `Section` e `Room`. Ogni professore in `Research_Staff` ha associati un dipartimento (`dept_code`) e una sezione (`section_code`). Ogni sezione (in `Section`) è associata ad un'aula (`room_code`) mentre ogni studente (in `School_Member`) è caratterizzato da un `name`, una `faculty` e l'anno di immatricolazione `year`.

La sorgente semistrutturata `Computer_Science` contiene informazioni sulla facoltà di informatica della medesima università descritta dalla sorgente relazionale. Ci sono sei classi: `CS_Person`, `Professor`, `Student`, `Division`, `Location` e `Course`. Le informazioni rappresentate sono simili a quelle della sorgente relazionale: anche qui sono presenti professori e studenti, e un professore è associato ad una `Division`, che è una logica specializzazione di `Department`. Di uno studente possiamo conoscere che corsi frequenta, l'anno di immatricolazione e il suo stato (`rank`), cioè se è in corso, fuori corso, laureando oppure dottorando.

L'ultima sorgente rappresenta un file, `Tax_Position`, in cui è rappresentata la posizione fiscale di ciascuno studente. In particolare sono presenti i seguenti campi: `name`, `student_code`, `faculty_name` e `tax_fee`.

Nel seguito del capitolo, le relazioni tra classi o attributi verranno indicate specificando solamente il nome delle classi, o degli attributi, e il tipo di relazione. In realtà SI-Designer, per evitare ambiguità tra nomi uguali in sorgenti diverse,

considera una rappresentazione dei nomi in *dot notation*: per esempio, la relazione tra nomi di attributi `dept_code BT belongs_to` viene rappresentata internamente ad SI-Designer come:
`University.Department.dept_code BT Computer_Science.Division.belongs_to`.

1.2 Generazione del *Common Thesaurus*

Le relazioni terminologiche del *Common Thesaurus* esprimono la conoscenza inter-schema e intra-schema riguardo gli schemi delle sorgenti in esame; sono di tipo intensionale e possono essere espresse per classi ed attributi. Le relazioni sono di tre tipi:

- SYN (SYNonym-of, *relazioni di sinonimia*): definita tra 2 termini che possono essere scambiati nelle sorgenti senza modificare il concetto del mondo reale rappresentato. Esempio: `faculty SYN faculty_name`.
- BT (Broader-Term, *relazioni di specializzazione)*: definita tra 2 termini t_i e t_j tali che t_i ha un significato più generale di t_j; NT (Narrower-Term) è la relazione opposta di BT. Esempio: `CS_Person BT Professor`, che è equivalente a `Professor NT CS_Person`.
- RT (Related-Term, *relazioni di aggregazione*): definita tra 2 termini t_i e t_j tra i quali esiste un legame generico. Esempio: `Research_Staff RT Department`.

La costruzione del *Common Thesaurus* è un processo incrementale durante il quale vengono aggiunte relazioni secondo il seguente ordine:

1. *relazioni semantiche intra-schema*
2. *relazioni lessicali*
3. *relazioni aggiunte dal progettista*
4. *relazioni intensionali inferite*

1.2.1 Relazioni semantiche intra-schema

Il modulo SIM (*Source Integrator Module*) estrae le relazioni semantiche analizzando la struttura dei sorgenti:

- Sorgenti ad oggetti:
  - dalla gerarchia di generalizzazione estrae relazioni di tipo BT, NT.

  - dalla gerarchia di aggregazione estrae relazioni di tipo RT.

- Sorgenti relazionali:

  - dalle definizioni di chiavi esterne estrae relazioni RT, BT.

Per esempio, dalla sorgente relazionale `University` è estratta la relazione `Research_Staff RT Department` derivante dal fatto che l'attributo `dept_code` della relazione `Research_Staff` è foreign key per la relazione `Department`, mentre dalla sorgente semistrutturata `Computer_Science`, per sua natura, non vi sono informazioni che permettano di estrarre relazioni automaticamente.

Se invece la sorgente `Computer_Science` fosse ad oggetti ed avessimo una gerarchia di ereditarietà tra Professor *IS_A* CS_Person e Student *IS_A* CS_Person, allora verrebbe estratto `Professor NT CS_Person` e `Student NT CS_Person`.

Queste relazioni sono estratte dal modulo SIM analizzando direttamente le classi $\mathrm{ODL_{I^3}}$ delle sorgenti relazionali e sfruttando ODB-Tools per estrarre le relazioni dagli altri tipi di sorgente.

1.2.2 Relazioni lessicali

L'estrazione delle relazioni lessicali è l'argomento di questa tesi e questa fase verrà trattata per esteso nel capitolo 3, mentre il componente grafico incaricato di interagire con il progettista, SLIM (Schemata Lessical Integrator Module), sarà mostrato nel capitolo 4. Per adesso si propone solo una breve introduzione per dare una visione generale.

Le relazioni lessicali vengono estratte sulla base delle relazioni lessicali tra i nomi delle classi e degli attributi, derivanti dal significato delle parole usate: un tipo di conoscenza non esplicitata tramite costrutti di un linguaggio di definizione dei dati, ma impliciti nel nome assegnato dal progettista. È compito del progettista attribuire nomi descrittivi o che possano essere interpretati correttamente; per cui c'è un'incertezza di interpretazione insita nell'ambiguità del linguaggio.

Le relazioni trovate sono sia intra-schema che inter-schema. Nel caso di sorgenti ad oggetti e, meno, nel caso di sorgenti relazionali, se gli schemi sono stati strutturati bene, non dovrebbero venir trovate relazioni intra-schema che SIM non abbia già inserito nel Thesaurus alla fase precedente.

Comunque le relazioni di maggior valore che vengono trovate da questa fase sono quelle inter-schema che non possono venir trovate da SIM.

Per ora, non avendo ancora presentato WordNet (Capitolo 2), è sufficiente sapere che è possibile estrarre da WordNet le seguenti relazioni di tipo lessicale[1]:

Sinonimia, **Iperonimia**[2], **Iponimia**[3], **Olonimia**[4], **Meronimia**[5], **Correlazione**[6]

Siccome iponimia e meronimia sono le relazioni inverse di iperonimia e olonimia, rispettivamente, l'insieme delle relazioni di interesse è il seguente: $\mathcal{W} = \{\mathbf{S}_{inonimia}, \mathbf{I}_{pernimia}, \mathbf{O}_{lonimia}, \mathbf{C}_{orrelazione}\}$.

Le relazioni derivanti da WordNet vengono proposte come relazioni semantiche da inserire nel *Common Thesaurus* in base alla seguente corrispondenza:

Sinonimia: corrisponde ad una relazione SYN.

Iperonimia: corrisponde ad una relazione BT.

Olonimia: corrisponde ad una relazione RT.

Correlazione: corrisponde ad una relazione RT.

Il *Common Thesaurus* deve contenere solo relazioni semantiche, quindi questa fase si limita a proporre le relazioni al progettista che, se le valida, le promuove a relazioni semantiche.

Nella presente tesi è stato sviluppato ed implementato un algoritmo che, acquisiti in input i nomi relativi agli schemi da integrare a cui sia stato assegnato il significato utilizzato nel contesto, restituisce come output le relazioni semantiche individuate. Gli aspetti implementativi dell'algoritmo sono riportati nella Sezione 3.3. Nel seguito si faranno alcune considerazioni sull'utilizzo del tool realizzato.

Partendo dagli schemi da integrare, il progettista deve fissare un'associazione tra ogni nome (di classe o di attributo) ed il significato utilizzato nel contesto. Cioè dato un nome si devono scegliere i significati ad esso associati.
Tale scelta è eseguita in due fasi:

[1] Questa affermazione non è del tutto vera: si rimanda alla sezione 3.1

[2] Relazione di generalizzazione

[3] Relazione di specializzazione

[4] Relazione di aggregazione, lato *parte*

[5] Relazione di aggregazione, lato *tutto*

[6] La correlazione è la relazione che lega due termini che appartengono a due insiemi di sinonimi che condividono uno stesso iperonimo, cioè lo stesso padre.

1. **Scelta della forma base.** In tale scelta il progettista è assistito dal sistema che gli propone la forma base (word form) usando il processore morfologico presente in WordNet. Per forma base si intende la parola tolti suffissi dovuti alla declinazione o coniugazione. Ad esempio, in figura 1.4, selezionando l'attributo `address`, si ottiene dal processore morfologico la forma base di `address`.

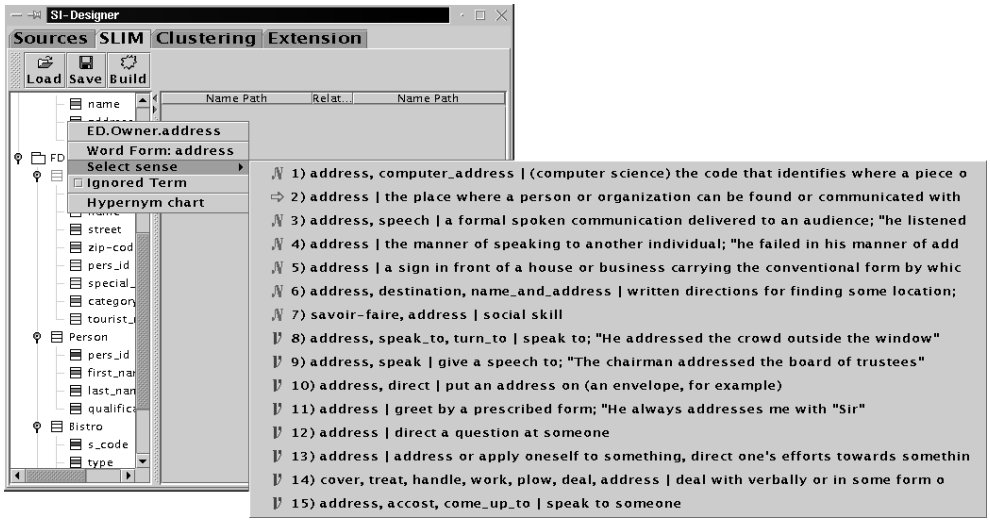


Figura 1.4: Significati di `address`

   Se tale forma non è trovata, oppure se c'è ambiguità[7], oppure non è soddisfacente, il progettista può immetterla direttamente.

2. **Scelta del significato.** Il progettista può decidere di far corrispondere ad un nome zero, uno o più significati. Ad esempio in figura 1.4 per la forma base `address` ottengo tutti i 15 significati che WordNet le attribuisce tra cui scegliere quello calzante con il contesto.

Alla fine della procedura di scelta dei significati, il progettista preme il pulsante "build" e vengono calcolate le relazioni come si vede in figura 1.5.

Da queste il progettista può scartare quelle sbagliate o fuorvianti; le altre vengono accettate ed immesse nel *Common Thesaurus* come relazioni semantiche intensionali.

[7]Per esempio di `axes` vengono trovate 3 forme base: `ax` (1 senso), `axis` (5 sensi), `axe` (2 sensi).

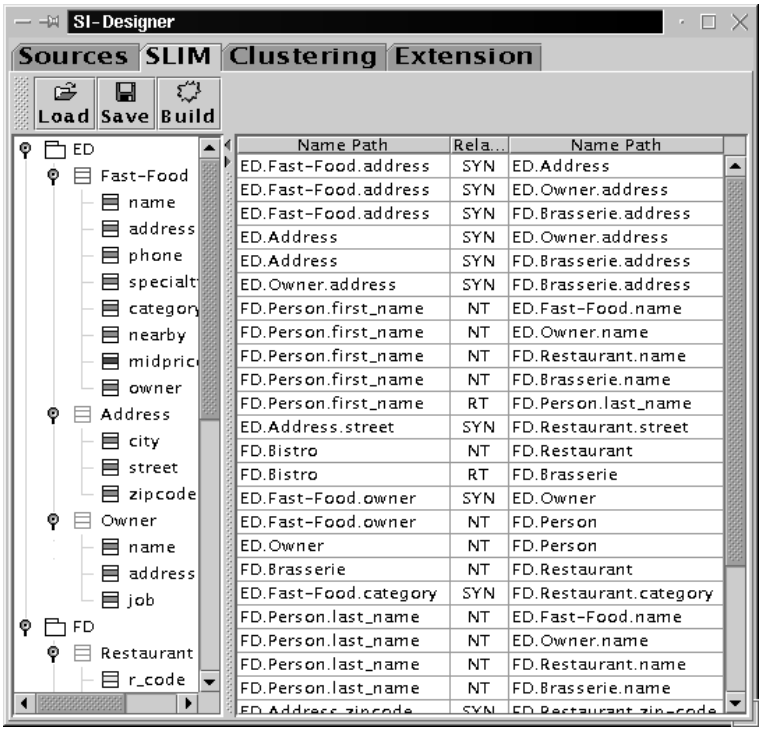


Figura 1.5: Relazioni intensionali ottenute dopo SLIM

1.2.3 Relazioni intensionali inter-schema aggiunte dal progettista

Il progettista può aggiungere al *Common Thesaurus* ottenuto nelle fasi precedenti delle nuove relazioni intensionali inter-schema (non individuate tramite WordNet perchè non "affini" in termini linguistici). Per aiutare il progettista in tale compito, vengono calcolate le *affinità* strutturali tra classi, tramite il modulo ARTEMIS. L'affinità tra ogni coppia di classi è espressa attraverso un coefficiente, chiamato *Global Affinity (GA)*, calcolato come combinazione lineare di due fattori: il primo valuta l'affinità strutturale delle classi, e il secondo dei nomi delle classi e degli attributi [10]. Tali coefficienti vengono calcolati riorganizzando il *Common Thesaurus* in una struttura simile alle Associative Networks [23], dove i nodi (ciascuno dei quali rappresenta genericamente un termine, sia esso il nome di una classe o il nome di un attributo) sono uniti attraverso le relazioni che sono presenti nel *Common Thesaurus*, assegnando un peso a ciascuno dei tre tipi di relazione intensionale.

Ad esempio, analizzando i seguenti coefficienti *Global Affinity* calcolati da ARTEMIS:

GA(`University.Research_Staff`, `Computer_Science.Professor`) = 0.4
GA(`University.Section`, `Computer_Science.Course`) = 0.66

il progettista può decidere di inserire o meno una relazione (SYN o BT o RT) fra le classi; ad esempio può aggiungere le seguenti relazioni:

⟨`University.Research_Staff` BT `Computer_Science.Professor`⟩
⟨`University.Section` SYN `Computer_Science.Course`⟩

Il progettista può inoltre aggiungere esplicitamente altre relazioni, quali ad esempio:

⟨`University.Research_Staff.dept_code` BT `Computer_Science.Professor.belongs_to`⟩
⟨`University.Department.dept_area` SYN `Computer_Science.Division.Sector`⟩

Il *Common Thesaurus* ottenuto fino a questo punto viene visualizzato da SI-Designer al progettista come in figura 1.5.

1.2.4 Validazione delle relazioni tra attributi valore

Le relazioni tra attributi valore inserite nei passi precedenti nel *Common Thesaurus* devono essere esaminate per verificare la compatibilità dei loro domini; tale fase è detta *validazione* e viene svolta dal modulo SIM attraverso ODB-Tools. Considerata una coppia di attributi posti in relazione, $a_i = \langle na_i, da_i \rangle$ e $a_j = \langle na_j, da_j \rangle$, la validazione viene attuata secondo i seguenti criteri:

- $\langle na_i$ SYN $na_j\rangle$: è *valida* se i domini da_i e da_j sono equivalenti o se uno dei due è più specializzato dell'altro;

- $\langle na_i$ BT $na_j\rangle$: è *valida* se da_i contiene o è equivalente a da_j.

Come esempio si mostrano alcune delle relazioni tra attributi che hanno subito la validazione ('[1]' significa relazione validata mentre '[0]' indica il contrario):

⟨`Tax_Position.Un_Student.name` BT `Computer_Science.CS_Person.first_name`⟩ [1]
⟨`Tax_Position.Un_Student.name` BT `Computer_Science.CS_Person.last_name`⟩ [1]
⟨`University.Research_Staff.dept_code` BT `Computer_Science.Professor.belongs_to`⟩ [0]
⟨`University.Department.dept_name` SYN `Computer_Science.Office.description`⟩ [1]
⟨`University.Section.section_name` SYN `Computer_Science.Course.course_name`⟩ [1]
⟨`University.School_Member.faculty` SYN `Tax_Position.Un_Student.faculty_name`⟩ [1]

Nell'esempio si nota come la relazione `dept_code` BT `belongs_to` non abbia superato la validazione poichè il dominio dell'attributo `dept_code` è un numero o una stringa (tipici domini di codici) mentre il dominio dell'attributo `belongs_to` è un attributo complesso.

1.2.5 Relazioni intensionali inferite

In quest'ultima fase si vogliono inferire nuove relazioni semantiche, partendo da quelle già introdotte nel *Common Thesaurus* ed utilizzando le tecniche di inferenza di **OLCD** . Siccome le relazioni semantiche di generalizzazione (BT) ed equivalenza (SYN) stabilite nel *Common Thesaurus* tra nomi di classi possono essere in conflitto con le descrizioni strutturali delle classi correlate, si produce uno *schema virtuale*, sempre espresso in ODL_{I^3} , che contiene una descrizione degli schemi sorgenti "ristrutturata" sulla base delle relazioni semantiche stabilite nel *Common Thesaurus*:

- per le relazioni $\langle C_1$ `SYN` $C_2\rangle$ è necessario "uniformare" le descrizione delle classi C_1 e C_2, in modo che la struttura sia la stessa;

- per le relazioni $\langle C_1$ `BT` $C_2\rangle$ è necessario "uniformare" la descrizione della classe C_1 a quella della classe C_2, in modo che la struttura di C_1 sia una specializzazione di quella di C_2. In sostanza, questo significa aggiungere gli attributi della classe C_2 alla struttura della classe C_1;

- per le relazioni $\langle C_1$ `RT` $C_2\rangle$ è necessario aggiungere alla descrizione della classe C_1 un nuovo attributo di aggregazione che si riferisce alla classe C_2.

Lo schema virtuale costituisce solo uno strumento tecnico per l'inferenza di nuove relazioni e rimane completamente trasparente all'utente.

Al termine di questa fase, SI-Designer visualizza le relazioni che hanno subito la validazione, evidenziando quelle che l'hanno superata, e le nuove relazioni inferite (vedere fig. 1.6).

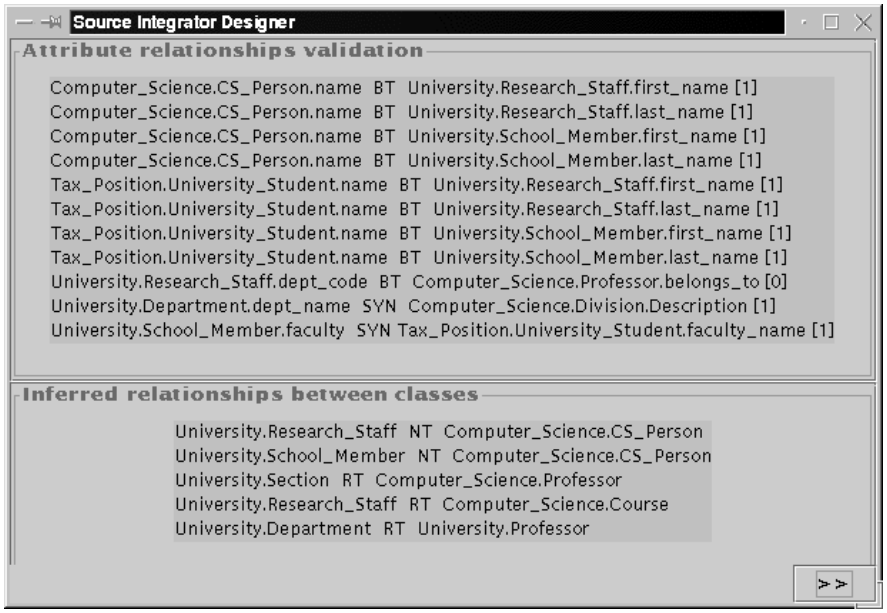


Figura 1.6: Relazioni inferite e validate

1.3 Generazione delle classi globali

Una volta costruito il *Common Thesaurus*, SI-Designer può generare le classi globali. Tale operazione viene attuata dal tool in questo modo:

1. Calcolo delle affinità,
2. Generazione dei cluster,
3. Generazione degli attributi globali e delle *mapping-table*.

Nella prima fase, SI-Designer funge da interfaccia tra il modulo ARTEMIS e il progettista che può interagire più volte con ARTEMIS, fino a quando non è soddisfatto dell'insieme di cluster ottenuto. Nella seconda fase invece, il tool costruisce, per ogni cluster, una classe globale a cui è associato un insieme di attributi globali

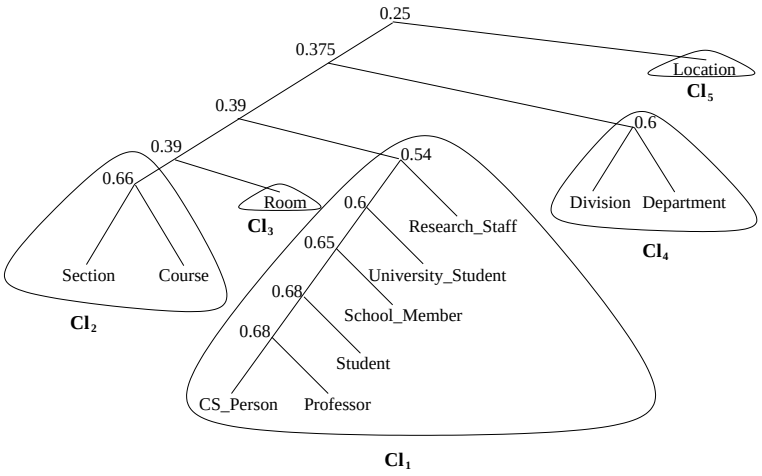


Figura 1.7: Albero delle affinità: cluster con $\sigma = 0.5$.

e la relativa mapping-table, mettendo a disposizione del progettista una interfaccia per revisionare gli insiemi di attributi globali e le mapping-table proposti dal tool. Con la medesima interfaccia, il progettista può infine assegnare un nome ad ogni classe globale.

1.3.1 Generazione dei cluster

Per la generalizzazione dei cluster, ARTEMIS utilizza tecniche di clustering attraverso le quali le classi sono automaticamente classificate in una struttura ad albero dove:

- le foglie rappresentano tutte le classi locali: foglie contigue sono classi caratterizzate da alta affinità, mentre foglie tra loro molto lontane rappresenteranno invece classi a bassa affinità;
- ogni nodo rappresenta un livello di clusterizzazione ed ha associato il coefficiente di affinità tra i due sottoalberi (cluster) che unisce.

Con SI-Designer, il progettista può immettere ad ogni iterazione un valore di soglia in base al quale ARTEMIS costruisce i cluster: ogni cluster sarà costituito da tutte le classi appartenenti ad un sottoalbero che al nodo radice ha un coefficiente maggiore del valore di soglia.

La Fig. 1.7 mostra l'albero delle classi locali e i cluster individuati da ARTEMIS sulla base dell'esempio presentato, ponendo un coefficiente di soglia σ=0.5.

1.3.2 Generazione degli attributi globali e delle mapping-table

Per ogni cluster, SI-Designer crea un insieme di attributi globali e, per ognuno di essi, determina la corrispondenza con gli *attributi locali* (quelli delle classi appartenenti al cluster cui corrisponde la classe globale). In alcuni casi, la corrispondenza è unica, mentre in altri ci sono diversi tipi di corrispondenza che il tool individua ma di cui non può risolvere l'ambiguità: in questo caso il tool chiede al progettista di scegliere quella giusta.

Il tool costruisce l'insieme degli attributi globali da associare ad un cluster in due fasi:

1. Unione degli attributi di tutte le classi affini appartenenti al cluster
2. Fusione degli attributi "simili"

L'unione degli attributi delle classi affini consiste in una mera raccolta di tutti gli attributi di ogni classe del cluster in un insieme. Tale insieme di attributi è un possibile insieme di attributi globali, in generale ridondanti.

Nella seconda fase (fusione degli attributi "simili"), SI-Designer tenta di eliminare queste ridondanze considerando le relazioni terminologiche del *Common Thesaurus*. Il procedimento di fusione è sempre automatico per gli attributi legati da relazioni validate, mentre lo è solo in certi casi se gli attributi sono legati da relazioni non validate. In particolare SI-Designer opera in questo modo:

- **Attributi in relazioni validate.** Per questi attributi la fusione è sempre automatica:
  - Agli attributi legati da relazioni `SYN` SI-Designer farà corrispondere un unico attributo globale: il dominio è lo stesso; come gli attributi locali ed il nome può essere scelto dal progettista tra le proposte di SI-Designer oppure inserito esplicitamente. Per esempio: agli attributi `description` e `dept_name`, legati dalla relazione `description SYN dept_name`, viene associato l'attributo globale `description`.
  - Gli attributi legati da relazioni `BT` vengono trattati da SI-Designer sostituendoli con un attributo globale che ha lo stesso nome e lo stesso dominio dell'attributo generalizzazione. Per esempio: gli attributi `name`, `first_name` e `last_name` sono legati dalle relazioni di specializzazione `name BT first_name` e `name BT last_name` per cui i tre attributi saranno rappresentati dall'attributo globale `name`.
- **Attributi in relazioni non validate.** A questa categoria appartengono gli attributi delle relazioni del *Common Thesaurus* che non hanno superato la

validazione: SI-Designer è in grado di individuare un attributo globale in modo automatico solo per un numero limitato di casi, lasciando al progettista il compito di aggiungere altri attributi globali per completare l'integrazione. In particolare, l'individuazione automatica di un attributo globale, in presenza di relazioni non validate, è possibile se gli attributi nelle relazioni soddisfano i seguenti requisiti:

1. sono legati da relazioni `SYN` o `BT`;
2. le classi in relazione appartengono ad uno stesso cluster;
3. rappresentano gerarchie di aggregazione (sono attributi complessi o foreign key);

Come esempio, si consideri la relazione non validata `dept_code BT belongs_to`. Gli attributi `dept_code` e `belongs_to` soddisfano tutte le condizioni elencate prima, infatti:

1. i due attributi sono legati da una relazione `BT`;
2. le classi mappate dai due attributi, `Department` e `Division`, sono nello stesso cluster (fig. 1.7);
3. `dept_code` è foreign key per `Research_Staff` e `belongs_to` è un attributo complesso di tipo `Division`;

Dunque è possibile aggiungere un attributo globale `works` che corrisponda ai due attributi locali `dept_code` e `belongs_to` (vedere mapping-table di fig. 1.8).

L'insieme di attributi globali così individuato può essere ulteriormente ampliato dal progettista per rappresentare pienamente tutte le informazioni delle sorgenti locali: questo accade soprattutto quando alcune informazioni sono contenute nelle sorgenti sotto forma di metadato.

Contemporaneamente alla creazione degli attributi globali, SI-Designer costruisce una *mapping-table* (vedere Fig. 1.8). Essa é una tabella $MT[CL][AG]$ dove CL é l'insieme delle classi locali che appartengono al cluster cui la mapping-table si riferisce e AG é l'insieme degli attributi globali creato da SI-Designer. Indicando con C il nome di una classe locale, con A il nome di un attributo globale e con AL il nome di un attributo locale, ogni elemento $MT[C][A]$ della tabella può assumere i seguenti valori:

- AL, con $AL \in C$.
  Questo valore viene inserito quando:

  - l'attributo globale A deve rappresentare l'informazione contenuta nel solo attributo locale AL. Per esempio, l'attributo globale `name` corrisponde all'omonimo attributo `name` della classe locale `CS_Person`;
  - ci sono relazioni di specializzazione che legano tra loro attributi appartenenti a classi diverse. Per esempio, l'attributo globale `faculty` della classe globale `University_Person` corrisponde all'omonimo attributo `faculty` della classe locale `School_Member` e all'attributo `faculty_name` della classe locale `University_Person`;

- AL_1 ***and*** AL_2 ***and ... and*** AL_n, con $AL_i \in C, i = 1, ..., n$.
  Usato quando il valore dell'attributo A é il concatenamento dei valori di piú attributi appartenenti alla medesima classe locale C. Per esempio, l'attributo globale `name` della classe globale `University_Person` corrisponde al concatenamento degli attributi `first_name` e `last_name` della classe locale `School_Member`;

- ***case of*** AL $\quad cost_1$**:** AL_1 $\quad cost_2$**:** AL_2 $\quad \ldots \quad cost_n$**:** AL_n
  dove $AL, AL_i \in C, i = 1, ..., n$ e $cost_i, i = 1, ..., n$ sono delle costanti.
  Questa situazione avviene quando l'attributo globale A può assumere il valore di uno tra un insieme di attributi locali $\{AL_i\}$ appartenenti alla medesima classe e la scelta avviene attraverso un terzo attributo locale AL, appartenente sempre alla stessa classe locale, che funge da selettore. Per esempio, l'attributo globale `email` deve assumere il valore di `home_email` o `phd_email` a seconda del valore assunto dall'attributo locale `rank` di `Student`: se `rank` = `'course'` allora `email` assume il valore di `home_email` altrimenti se `rank` = `'phd'` allora `email` assume il valore di `phd_email`;

- $costante$.
  Si contempla il caso in cui l'attributo globale A non corrisponde ad alcun attributo della classe locale C. Il valore assunto da A viene attribuito dal progettista in base al significato dato all'attributo globale. Per esempio, l'attributo globale `rank` della classe globale `University_Person` assume il valore costante `'Professor'` se occorre accedere alla classe locale `Research_Staff` e il valore costante `'Student'` se occorre accedere alla classe locale `University_Student`.

- $null$.
  Questo é il caso in cui l'attributo globale A, durante un accesso alla classe locale C, non assume alcun valore. Per esempio, l'attributo globale A

University_Person	**name**	**rank**	**works**	**faculty**	**email**	. . .
Research_Staff	first_name *and* last_name	'Professor'	dept_code	null	email	. . .
School_Member	first_name *and* last_name	'Student'	null	faculty	null	. . .
CS_Person	name	null	null	'Computer_Science'	null	. . .
Professor	name	rank	belongs_to	'Computer_Science'	null	. . .
Student	name	rank	null	'Computer_Science'	*tag* rank 'course':home_mail 'phd':phd_mail	. . .
University_Student	name	'Student'	null	faculty_name	null	. . .

Workplace	**name**	**area**	**employee_nr**	**budget**	. . .
Department	dept_name	dept_area	null	budget	. . .
Division	description	sector	employee_nr	fund	. . .

Figura 1.8: Mapping-table di `University_Person` e `Workplace`.

`faculty` non assume alcun valore nella classe locale `Research_Staff`: ciò viene indicato nella mapping-table con la costante $null$.

SI-Designer crea una *mapping-table* per ogni classe globale e mette a disposizione del progettista una interfaccia che permette sia di avere una visione completa di tutte le classi globali (nomi ed attributi), incluse le relative mapping-tables, che l'inserimento dei nomi delle classi globali e la modifica delle *mapping table*.

Il tool conclude l'integrazione con la creazione della descrizione $\mathrm{ODL_{I^3}}$ dello schema globale [10].

Capitolo 2

Il Database lessicale WordNet

WordNet [11, 12] è sviluppato dal Cognitive science Laboratory sotto la direzione del professore George A. Miller presso l'università di Princeton. Il sito web ufficiale è all'indirizzo http://www.cogsci.princeton.edu/∼wn/ presso cui si può scaricare il pacchetto completo senza che sia richiesto di corrispondere un compenso. WordNet è un copyright (1997) dell'Università di Princeton. La licenza d'uso di WordNet ne prevede un utilizzo gratuito anche per fini al di fuori della ricerca, cioè commerciali, con l'unico obbligo di citare gli autori e l'indirizzo del sito web.

WordNet, attualmente (19 luglio 2000) giunto alla versione 1.6, datata 27 febbraio 1998, conta 129625 lemmi organizzati in 99759 insiemi di sinonimi (*synset*), ed è considerato la più importante risorsa disponibile per i ricercatori nei campi della linguistica computazionale, dell'analisi testuale, e di altre aree associate.

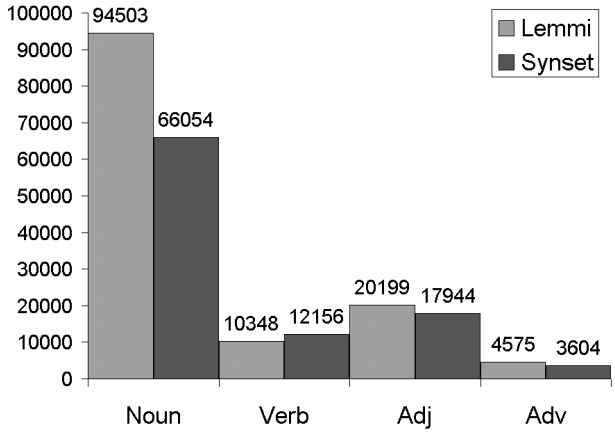


Figura 2.1: Conteggio di lemmi e *synset* per categoria sintattica

WordNet è sistema di gestione di un dizionario lessicale basato sulle attuali teorie psicolinguistiche della memoria lessicale umana. Ogni categoria sintattica:

nomi, verbi, aggettivi e avverbi, è organizzata in insiemi di sinonimi che rappresentano un inerente concetto lessicale. Gli insiemi di sinonimi sono collegati da diversi tipi di relazioni.

I dizionari cartacei sono organizzati con un ordinamento alfabetico perché è l'unico modo che permette ad un lettore di trovare le parole cercate sfogliandolo una sola volta. Questo approccio però è lontano dall'essere perfetto, infatti vengono accostate parole con significati diversi e i termini correlati sono sparpagliati in modo casuale. Per ogni lemma vengono presentati tutti i significati assieme, anche se possono non aver nulla in comune, per esempio `number` ha i significati di "cifra" o "quantità", ma anche di "performance teatrale" o di fascicolo di una "rivista".

Anche un dizionario dei sinonimi e contrari è un concetto riduttivo: fissa l'attenzione sui termini intesi come stringhe di caratteri ed è dunque ridontante: se due parole sono sinonimi i loro significati appaiono due volte, sotto l'ordine alfabetico di entrambe le parole.

WordNet non si pone in competizione con i dizionari tradizionali per le informazioni reperibili, per esempio non mostra la sillabazione, la pronuncia, le forme derivate, l'etimologia, le definizioni ed esempi di usi alternativi, note sugli usi speciali, immagini o grafici descrittivi, né per completezza lessicale: il numero di termini lo pone al livello di un dizionario da *college*.
Quello che è innovativo è:

- la comprensione della differenza tra lemmi e significati (sezione 2.1);
- le relazioni, che permettono navigabilità alle informazioni (sezione 2.2);
- la strutturazione interna delle categorie sintattiche (sezione 2.3).

2.1 La Matrice Lessicale

Il punto di partenza della semantica lessicale è il riconoscimento che esiste un'associazione convenzionale fra la forma delle parole (il modo in cui, cioè, vengono pronunciate e scritte) e il concetto/significato che esse esprimono; tale associazione è di tipo molti-a-molti (vedi Figura 2.2), dando luogo alle proprietà di:

Sinonimia: proprietà di un concetto/significato di avere due o più parole in grado di esprimerlo. Un gruppo di sinonimi prende il nome di *synset*.

Polisemia: proprietà di una stessa parola di avere due o più significati.

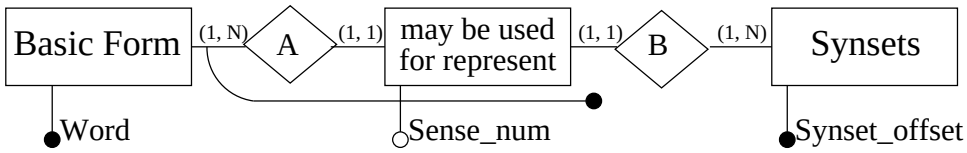


Figura 2.2: Relazione tra lemmi e significati

La corrispondenza tra la forma delle parole e il significato che esprimono viene sintetizzata nella cosiddetta *Matrice Lessicale*, nella quale le righe riportano il significato delle parole (quindi una riga rappresenta un *synset*) e le colonne rappresentano la forma delle parole (forma/lemma base) (vedi Figura 2.3). Una $\text{entry}_{i,j}$ non nulla nella matrice lessicale significa che una word form j puo' essere usata per esprimere un concetto i.

Word Meanings	Word Forms				
	F_1	F_2	F_3		F_n
M_1	$E_{1,1}$	$E_{1,2}$			
M_2		$E_{2,2}$			
M_3			$E_{3,3}$		
. . .			. . . . .		
M_m					$E_{m,n}$

Figura 2.3: La Matrice Lessicale

Ogni elemento della matrice è una definizione (*entry*), $e = (f, m)$, dove f è la *forma base* e m (*meaning*) è il contatore del significato; ad esempio (`test`, `1`) si riferisce all'azione di fare un tentativo; mentre (`test`, `4`) si riferisce a fare un questionario o esercizi per valutare capacità o conoscenze (vedi Figura 2.4).

Il termine "parola" è inerentemente ambiguo, perché può essere interpretata come il suono ed il modo di essere scritta o come il concetto associato. D'ora in poi si utilizzerà al suo posto:

Lemma: per indicare la forma scritta o il suono,

Significato: per indicare il concetto associato.

Word Meanings	Word Forms ...	exam	examination	run	test	trial	tryout	...
...								
a set of questions or exercises evaluating ...		exam#1	examination#2		test#4			
the act of testing something				run#2	test#6	trial#2		
trying something to find out about it					test#1	trial#5	tryout#1	
...								

Figura 2.4: Esempio di una vista della matrice lessicale

Una seconda ambiguità è data dal fatto che un lemma è rappresentato con una parola, ed un *synset* con un insieme di parole: per cui sembra più una relazione uno-a-molti e invece è relazione molti-a-molti; il fatto poi che un insieme di parole possa essere formato da una sola parola complica ulteriormente le cose.

Il problema è dovuto alla difficoltà di rappresentare un significato che è una *psychological feature* (una caratteristica di una mente di un organismo vivente) tramite parole che sono delle *abstraction* (concetti generali creati riconoscendo aspetti comuni da casi specifici).
Dunque i modi per rappresentare un significato usati in WordNet sono:

- Glossa: un commento/spiegazione del significato. È molto aleatoria oltre che scomoda per la sua lunghezza.

- Elenco dei sinonimi: insieme di tutti i lemmi che singolarmente possono essere utilizzati per rappresentare il significato. Ad un madrelingua inglese è sufficiente la lista dei sinonimi per capire il contesto in esame ed escludere gli altri; per esempio tra `‘‘doctor, doc, physician, MD, Dr., medico’’` e `‘‘Doctor, Doctor of the Church’’` la differenza è evidente. Mentre è impossibile fare distinzione tra *synset* che contengono uno solo o pochi sinonimi, per esempio: `‘‘doctor’’` con la glossa `give medical treatment to`, oppure `‘‘doctor, Dr.’’` con la glossa `a person who holds Ph.D. degree from an academic institution`.

- Chiave surrogata: un identificatore univoco, ma scorrelato dai contenuti del record. In WordNet viene usato il `synset_offset`: la posizione all'interno dei file dati a cui si trova il *synset*. È un valore non umanamente leggibile ed a solo uso interno.

- SenseKey: da notare che anche una entry, cioè un lemma più il numero del significato, è chiave per un *synset*, cioè identifica univocamente l'unico *synset* a cui appartiene. Con una visione relazionale lemma+numero_senso sono chiave per la relazione che collega le due entità lemmi e significati (vedi Figura 2.2), dunque una tupla della relazione che è dal lato *molti* individua una tupla della entità che è dal lato *uno*.

La sola lista dei sinonimi non è sufficiente: infatti, in media, un *synset* contiene 1,75 termini (vedi Figura 2.5).

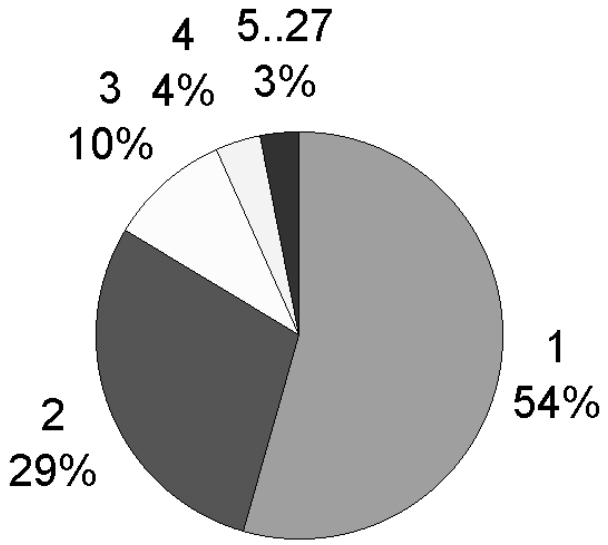


Figura 2.5: Percentuale dei *synset* per numero di parole contenute

Il 54% dei *synset* possono essere rappresentati con un solo termine che non ha sinonimi per quel significato. In Appendice A.2 sono riportate statistiche più approfondite.

Riassumendo viene usato un `synset_offset` (chiave surrogata) per motivi di performance; un SenseKey quando è importante l'immutabilità rispetto alle versioni di WordNet, infatti la posizione di un *synset* all'interno dei file dati in generale cambia con il succedersi delle versioni; e la lista dei sinonimi più la glossa per essere compreso da un lettore umano.

2.2 Relazioni tra lemmi e significati

In WordNet sono presenti due famiglie di relazioni a seconda del tipo degli operatori, cioé ci sono relazioni che esprimono un legame tra singoli lemmi e altre tra significati (Vedi Figura 2.6; schematizzazione con notazione ER: le relazioni sono rappresentate come autoanelli).
In particolare nella terminologia di WordNet vengono indicate come:

Relazioni semantiche: stabiliscono un nesso tra significati, per esempio relazioni di specializzazione/generalizzazione o di parte/tutto.

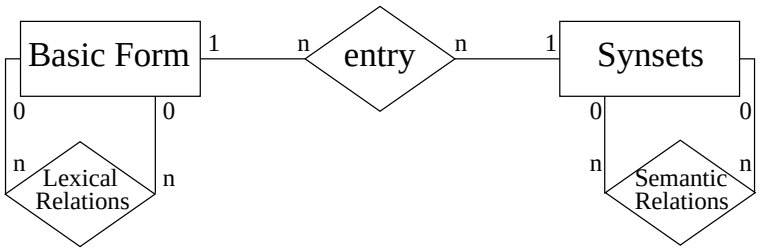


Figura 2.6: Autoanelli: relazioni lessicali e semantiche

Relazioni lessicali: stabiliscono un nesso tra singoli lemmi, per esempio un contrario non è detto che sia valido per tutti i termini di un *synset*, ma solo per uno in particolare.

In Tabella 2.1 sono mostrati i 17 tipi di relazioni base di WordNet. I valori sono normalizzati sul numero di *synset* e disaggregati per categoria sintattica; le relazioni sono conteggiate una per ogni operatore: per esempio una relazione iperonimo/iponimo è conteggiata sia sotto HYPERPTR che HYPOPTR. Si nota che:

- La relazione di sinonimia non è presente: sarà spiegato al paragrafo 2.2.1
- Esistono relazioni caratteristiche di un'unica categoria sintattica: esempio la relazione di *entailment* (un verbo X implica (entail) Y se X non può essere fatto a meno che Y sia, o sia stato, fatto)
- Per ogni categoria sintattica ci sono relazioni decisamente più frequenti ed alle altre rimangono le briciole: vedi Appendice A.3
- In media i *synset* partecipano a poche relazioni: vedi Tabella 2.2

Tipo	Nomi	Verbi	Aggettivi	avverbi
ANTPTR	0.0280	0.0850	0.2293	0.2013
HYPERPTR	1.0134	0.9512	-	-
HYPOPTR	1.0134	0.9512	-	-
ENTAILPTR	-	0.0352	-	-
SIMPTR	-	-	1.2224	-
ISMEMBERPTR	0.1794	-	-	-
ISSTUFFPTR	0.0107	-	-	-
ISPARTPTR	0.1042	-	-	-
HASMEMBERPTR	0.1794	-	-	-
HASSTUFFPTR	0.0107	-	-	-
HASPARTPTR	0.1042	-	-	-
MERONYM	-	-	-	-
HOLONYM	-	-	-	-
CAUSETO	-	0.0184	-	-
PPLPTR	-	-	0.0050	-
SEEALSO	-	0.0520	0.1513	-
PERTPTR	-	-	0.2244	0.8786
ATTRIBUTE	0.0098	-	0.0362	-
VERBGROUP	-	0.0431	-	-

Tabella 2.1: Relazioni per *synset*: conteggio dei puntatori fratto numero dei *synset*

Relazioni	Nomi	Verbi	Aggettivi	avverbi	Totale
Conteggio	175201	25908	33482	3861	238452
Normalizzato	2.65	2.13	1.87	1.08	2.34

Tabella 2.2: Media di relazioni per *synset*

2.2.1 Sinonimia

La sinonimia sarebbe una relazione lessicale, ma non è espressa formalmente come le altre relazioni, infatti è alla base della struttura di WordNet: i termini vengono raggruppati per insiemi di sinonimi (*synset*), dunque esiste una relazione di sinonimia implicita tra tutte le coppie di elementi appartenenti ad ogni *synset*.

Secondo una definizione (comunemente attibuita a Leibniz):

> Due espressioni sono sinonime se la sostituzione di una per l'altra non cambia il vero valore della sentenza nella quale è fatta la sostituzione.

Da questa definizione si nota che i veri sinonimi sono molto rari o inesistenti. Una definizione meno forte fa cadere l'ipotesi sul contesto, cioè impone la sinonimia a prescindere dal contesto. Quindi:

> Due espressioni sono sinonime in un contesto linguistico C se la sostituzione di una per l'altra, nel contesto C, non cambia il vero valore della sentenza.

Esempio: `plank` e `board` sono sinonimi solo in un contesto di falegnameria. In altri contesti, `plank` può avere il significato di "piattaforma politica", mentre `board` quelli di "tavola", "circuito stampato", "commissione di supervisione", ...

Questa definizione di sostituibilità implica che non ci possano essere sinonimi tra categorie sintattiche diverse. Questo è uno dei motivi per cui i *synset* in WordNet sono rigidamente separati per categoria sintattica come si vedrà alla sezione 2.3.

2.2.2 Antinomia

La antinomia è una relazione lessicale tra singoli lemmi ed associa due termini l'uno il contrario dell'altro. In una logica booleana l'antinomo di x è $\neg x$, in linguistica non è sempre vero. Per esempio `rich` e `poor` sono antinomi, ma dire che una persona non ricca ($\neg rich$) non implica che sia povera: molte persone non si considerano nè ricche nè povere.

La antinomia è una relazione lessicale e non una relazione semantica perché per esempio i *synset* `rise, ascend` e `fall, descend` possono essere concettualmente opposti, ma non sono antinomi; [`rise`/`fall`] sono antinomi e così anche [`ascend`/`descend`].

Num	Nomi		Verbi		Aggettivi		Avverbi		Totale	
0	64342	98%	11137	92%	14007	78%	2938	83%	92424	93%
1	1530	2%	949	8%	3716	21%	567	16%	6762	7%
2	142	0%	41	0%	184	1%	59	1%	426	0%
3	9	0%			8	0%	9	0%	26	0%
4	2	0%					2	0%	4	0%

Tabella 2.3: Distribuzione dei *synset* per numero di Antinomi

2.2.3 Iponimia

> Un concetto rappresentato dal *synset* $\{x, x^i, \dots\}$ è un iponimo di un concetto rappresentato dal *synset* $\{y, y^i, \dots\}$ se un madrelingua inglese valida una sentenza costruita come: x è un (*is a, kind of*) y.

Nel caso dei verbi la relazione di iponimia viene chiamata troponimia:

> Un verbo che esprime una maniera specifica di operare di un altro verbo: X è un troponimo di Y se fare X è fare Y in una qualche maniera.

La relazione inversa è la relazione di *iperonimia*: la relazione che lega un sovraordinato, un concetto generale, ad un concetto più specifico che ne eredita le caratteristiche generali differenziandosi almeno per qualcosa.

Iperonimia/iponimia sono relazioni semantiche, e sono l'equivalente della gerarchia di generalizzazione/specializzazione dei modelli relazionali o dell'ereditarietà dei modelli ad oggetti.

Dato un *synset* si può calcolare la chiusura transitiva degli iperonimi, cioè trovare tutti i concetti più generali del concetto in esame.
Come esempio in Figura 2.7 è riportato il diagramma degli iperonimi del *synset* che contiene il termine hoodoo#1 con glossa "un praticante di voodoo".
L'esempio non è stato scelto a caso e permette di mostrare diversi aspetti:

- la ereditarietà è multipla.
- le radici sono più di una e confluiscono in un unico grafo. In particolare i beginner per i nomi sono 9, ma formano un unico diagramma di 66025 *synset*. Il *synset* di hoodoo#1 è l'unico *synset* il cui diagramma ha tre beginner.
- *synset* possono avere iperonimi a livelli diversi di profondità: hoodoo#1 ha profondità 7,8 o 12 a seconda del percorso che si sceglie.

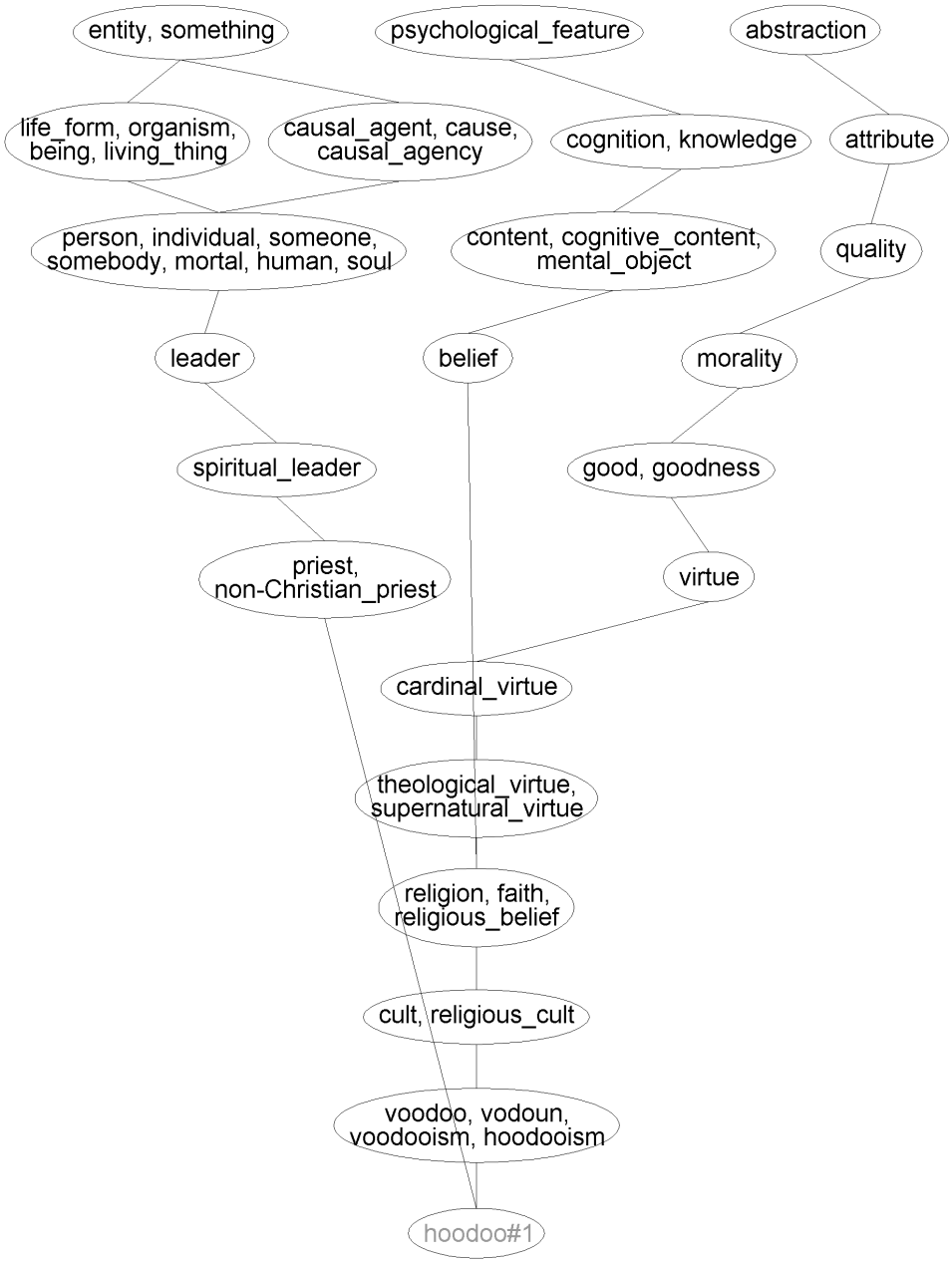


Figura 2.7: Diagramma degli iperonimi di hoodoo#1

La iperonimia/iponimia è definita solo per i nomi ed i verbi, ed è la relazione di gran lunga più frequente, infatti ci sono in media 1.013404 e 0.951265 iponimi e iperonimi per ogni *synset* rispettivamente dei nomi e dei verbi (vedi Tabella 2.1). Gli iponimi e gli iperonimi sono in numero uguale essendo una relazione opposta dell'altra, però è importante la distribuzione, che è molto diversa.
La Tabella 2.4 mostra il numero dei *synset* divisi per numero di relazioni di iperonimia a cui appartengono. Si vede che la quasi totalità dei *synset* ha un solo sovraordinato, cioè l'eraditarietà multipla è usata solo nell'1% dei *synset*, ed il massimo sono quattro diretti iperonimi.

Numero	Nomi		Verbi	
0	9	0%	617	5%
1	65144	99%	11484	95%
2	852	1%	26	0%
3	18	0%		
4	2	0%		

Tabella 2.4: Distribuzione dei *synset* per numero di iperonimi

Per gli iponimi invece:

- il 78% dei *synset* sono foglie (75% per i verbi), cioè non hanno iponimi.
- la Figura 2.8 (Figura 2.8 per i verbi) mostra la distribuzione dei *synset* per numero di iponimi tra 1 e 26: calano molto rapidamente.
- i *synset* con più di 26 iponimi sono pochi: 96 (16 per i verbi) ed il massimo è raggiunto dal *synset* che contiene il termine bird_genus con 397 iponimi, tutti i tipi di volatili, appiattiti su di un solo livello. Il caso di bird_genus mostra come la cura di WordNet vari molto da contesto a contesto.

È interessante confrontare il livello di profondità tra nomi (vedi Figura 2.10) e verbi (vedi Figura 2.11). Per profondità intendo il percorso più lungo per raggiungere un beginner ed è indice del lavoro che è stato fatto per raffinare differenze tra i *synset*.

Il punto di picco della profondità è 6 per i nomi, mentre solo 3 per i verbi.

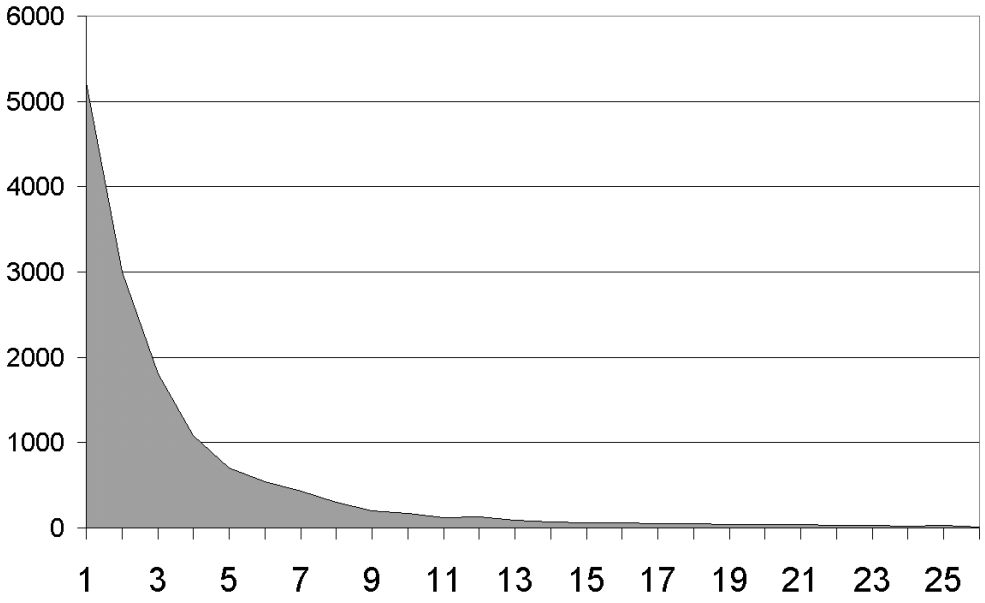


Figura 2.8: Distribuzione dei *synset* per numero di iponimi nei nomi nell'intervallo 1..26

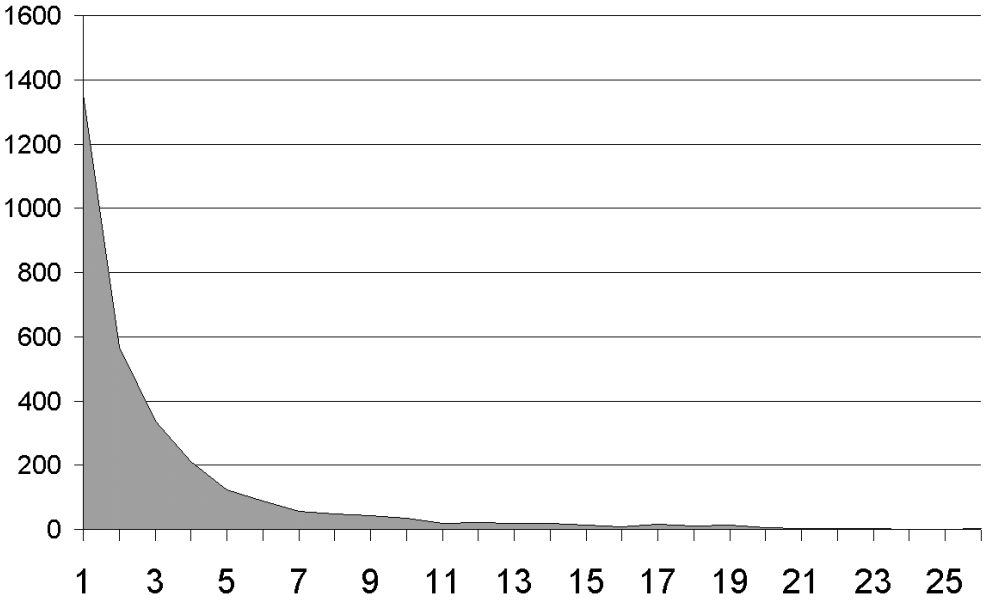


Figura 2.9: Distribuzione dei *synset* per numero di iponimi nei verbi nell'intervallo 1..26

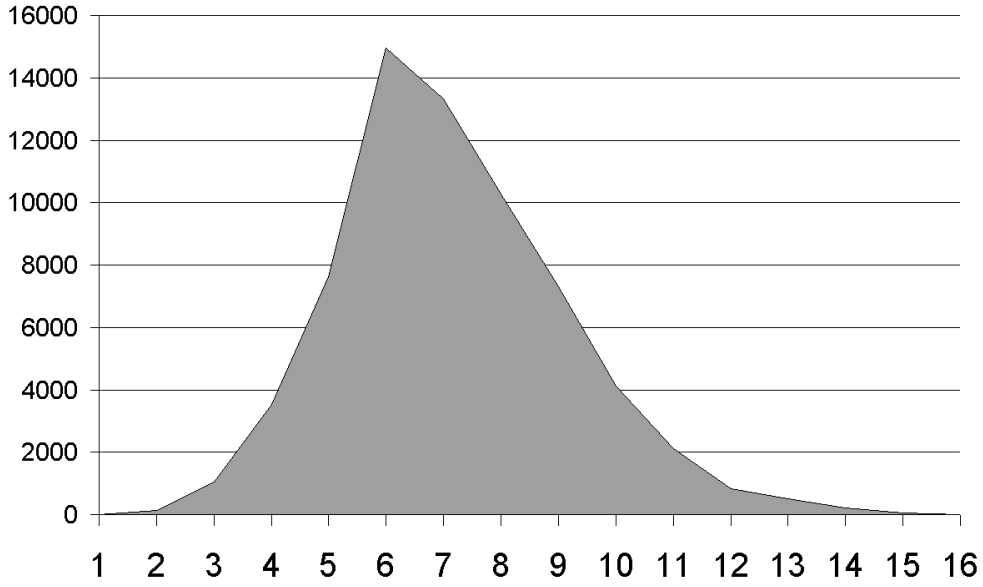


Figura 2.10: Distanza dai beginner nei nomi

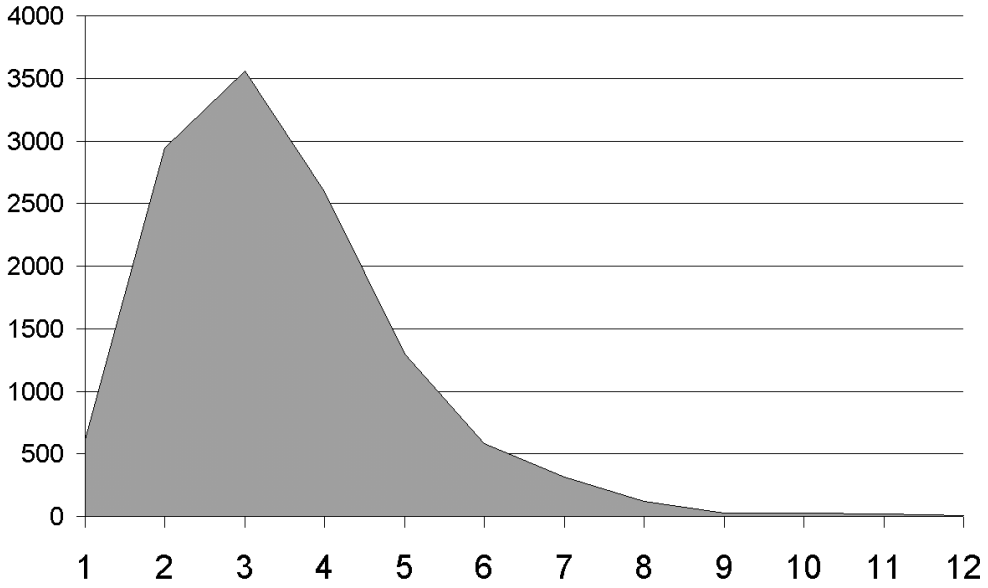


Figura 2.11: Distanza dai beginner nei verbi

2.2.4 Meronimia

> Un concetto rappresentato dal *synset* $\{x, x^i, \dots\}$ è un meronimo di un concetto rappresentato dal *synset* $\{y, y^i, \dots\}$ se un madrelingua inglese valida una sentenza costruita come: y ha un (*has an*) x (come parte/sostanza/membro), oppure x è una parte di y.

Sono definiti tre tipi di aggregazione:

membro esempio school (nel senso di istituzione educativa), Figura 2.12

parte esempio school (nel senso del luogo dove i giovani ricevono l'educazione), Figura 2.12

sostanza esempio quartz, Figura 2.13

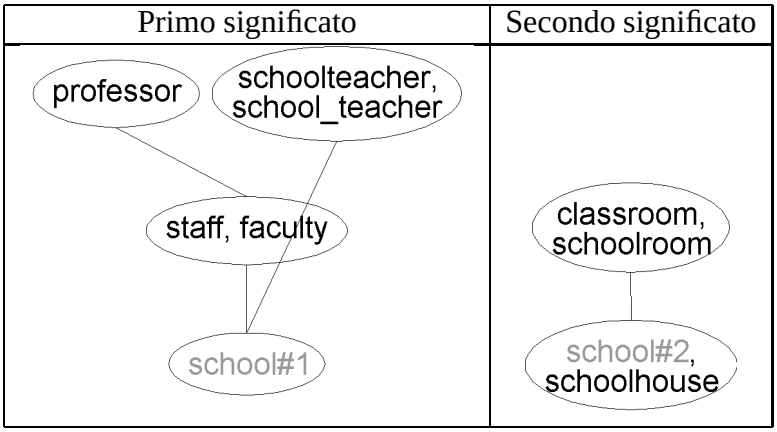


Figura 2.12: Meronimi di school

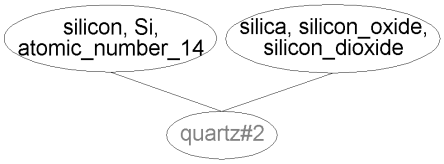


Figura 2.13: Meronimi di quartz

Da notare che per avere tutti i meronimi di un *synset* bisogna aggiungere anche i meronimi degli iperonimi. Per esempio school (Figura 2.12) ha solo "classroom, schoolroom" come diretto meronimo, perchè eredita anche quelli di building (Figura 2.14) e di structure (Figura 2.15).

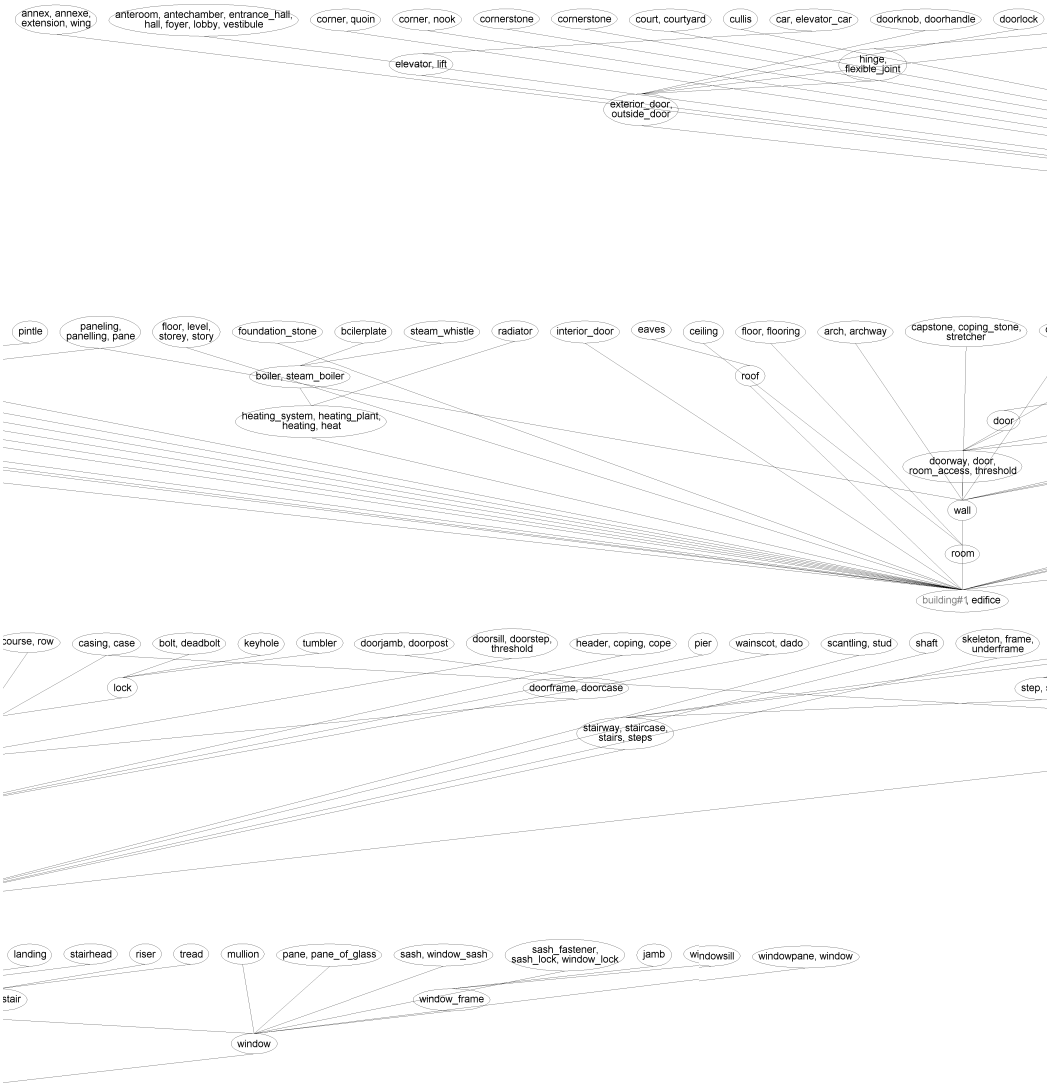


Figura 2.14: Meronimi diretti di building (il diagramma è spezzato verticalmente in quattro)

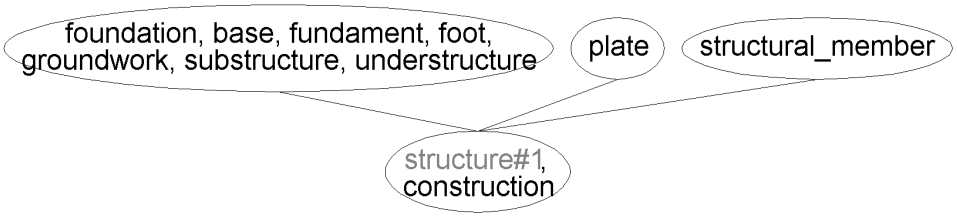


Figura 2.15: Meronimi diretti di Structure

2.2.5 Implicazione, (*entailment*)

Un verbo X implica (entail) Y se X non può essere fatto a meno che Y sia, o sia stato, fatto.

	0	1	2	3
conteggio	11719	391	15	2
percentuale	97%	3%	0%	0%

Tabella 2.5: Distribuzione dei *synset* per numero di *entailment*

La relazione di *entailment* nei verbi è molto simile alla relazione di meronimia dei nomi. Per esempio (Figura 2.16) l'azione di paracadutarsi è composta da lanciarsi, planare (*glide*) ed andare giù: sono la sequenza temporale delle azioni svolte durante l'attività con il paracadute. Così anche comprare un bene si compone prima di una fase decisionale e poi di un pagamento.

2.2.6 Relazione causale, (*cause to*)

La relazione causale è simile alla relazione di *entailment* però senza l'inclusione temporale.

	0	1	2
conteggio	11905	220	2
percentuale	98%	2%	0%

Tabella 2.6: Distribuzione dei *synset* per numero di *cause to*

Dagli esempi (Figura 2.17):

- "costringere", "forzare" implica un "azione" o un "movimento" (dopo).

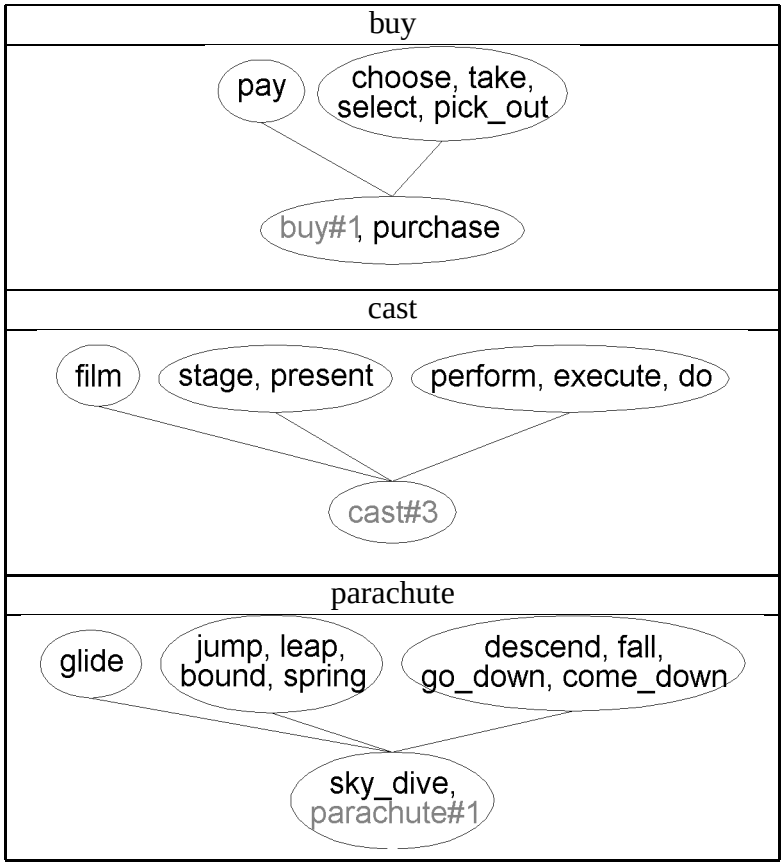


Figura 2.16: Esempi di *entailment*

- "compiacere", "far piacere" a implica "piacere" o "essere affezionato".
- "avere un impatto emotivo o cognitivo su" implica "provare" o "imparare".

2.2.7 Vedi anche, *see also*

È una relazione lessicale che lega singoli lemmi di diversi *synset*. Esempio: il *synset* che contiene i lemmi "put, set, place, pose, position, lay" (con glossa: put into a certain place: "Put your things here"; "Set the tray down"; "Set the dogs on the scent of the missing children"; also with abstract objects and locations: "Place emphasis on a certain point") ha i seguenti puntatori lessicali di tipo *see also*:

put $\longrightarrow$ put off#4; put down#4; put off#3; put out#3; put to sleep#3; put out#8; put under#1; put away#4; put through#2; put to sleep#2; put up#6; put through#1; put out#5

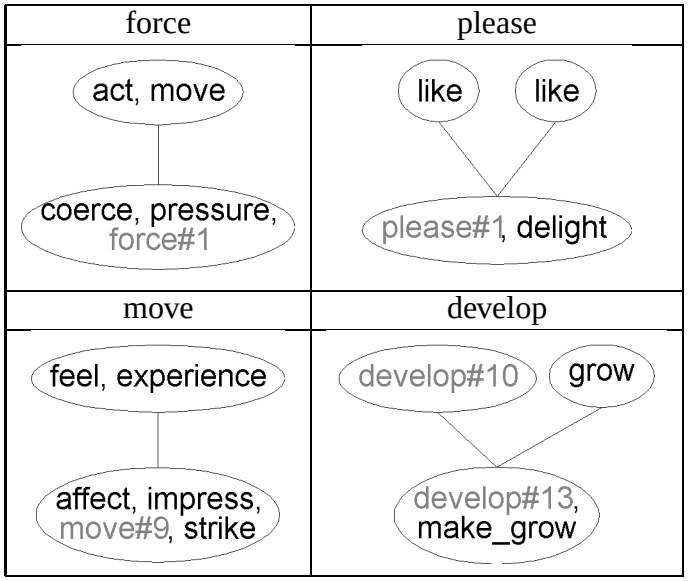


Figura 2.17: Esempi di *cause to*

set $\longrightarrow$ put back#1; put back#2; put off#1; put down#3; put away#3; put together#1; put on#7; put up#1; put down#2; put across#1

place $\longrightarrow$ *null*

pose $\longrightarrow$ *null*

position $\longrightarrow$ *null*

lay $\longrightarrow$ put over#2

numero	0	1	2	3	4	5	6	7	8	9	10
verbi	11822	177	69	21	15	4	7	5	0	1	1
aggettivi	16575	619	351	209	80	49	26	5	1		

numero	11	12	13	14	15	16	17	18	19	20	21
verbi	0	1	1	1	0	0	1	0	0	0	1

Tabella 2.7: Distribuzione dei *synset* per numero di *see also*

2.2.8 Raggruppamenti di verbi, *verb group*

Esempi:

1. Il *synset* “mistake, confuse, confound” con glossa (mistake one thing for another; “you are confusing me with the other candidate”; “I mistook her for the secretary“) è raggruppato con i seguenti *synset*:

   - “confuse, blur, obscure”
   - “confuse”
   - “jumble, confuse, mix up”

2. Il *synset* “see, check, insure, see to it, ensure, control, ascertain, assure” con glossa (be careful or certain to do something; make certain of something; “He verified that the valves were closed”; “See that the curtains are closed”; “control the quality of the product”):

   - “check, check of, mark, mark of, tick of”
   - “check”
   - “control, verify”

	0	1	2	3
conteggio	11669	396	59	3
percentuale	97%	3%	0%	0%

Tabella 2.8: Distribuzione dei *synset* per numero di *verb group*

2.2.9 Similarità (relazione di capo-satellite negli aggettivi)

Le relazioni di similarità sono alla base dell’organizzazione degli aggettivi (vedi Tabella 2.10). La maggior parte degli aggettivi sono raggruppati intorno a coppie o triplette di antinomi. Le relazioni di antinomia legano solo due (o tre) *synset* chiamati *synset* principali (*head synset*); a questi sono collegati *synset* satellite che ne condividono indirettamente le relazioni di antinomia.

Il tipo di relazione che collega i *synset* principali con i *synset* satellite si chiama relazione di similarità.

Come esempio riporto il *synset* con più relazioni di similarità: “chromatic, colored” con glossa (being or having or characterized by hue). Il termine `chromatic` ha una relazione di antinomia con `achromatic`, ed i *synset* satelliti, dunque in relazione di similarità, sono in Tabella 2.9.

→ amber, brownish-yellow, yellow-brown – (a medium to dark brownish yellow color)	→ amethyst – (of a moderate purple color)
→ aureate, gilded, gilt, gold, golden – (having the deep slightly brownish color of gold; ”long aureate (or golden) hair”; ”a gold carpet”)	→ azure, cerulean, sky-blue, bright blue – (of a deep somewhat purplish blue color similar to that of a clear October sky; ”October’s bright blue weather”)
→ blue, bluish, blueish, light-blue, dark-blue – (having a color similar to that of a clear unclouded sky; ”October’s bright blue weather”- Helen Hunt Jackson; ”a blue flame”; ”blue haze of tobacco smoke”)	→ blushful, rosy – (blush colored; ”blushful mists”)
→ bottle-green – (of a dark to moderate grayish green color)	→ bronze, bronzy – (of the color of bronze)
→ brown, brownish, dark-brown – (of a color similar to that of wood or earth)	→ buff – (of the color of buff leather)
→ canary, canary-yellow – (having the color of a canary; a light to moderate yellow)	→ caramel, caramel brown – (having the color of caramel; a moderate yellow-brown)
→ carnation – (having the color of a carnation)	→ chartreuse – (having the yellowish green color of Chartreuse liqueur)
→ dun – (of a dull grayish brown to brownish gray color; ”the dun and dreary prairie”)	→ earthlike – (earth colored; having a color of soil or earth; ”a range of earthlike colors”)
→ fuscous – (having a dusky brownish gray color)	→ green, greenish, light-green, dark-green – (similar to the color of fresh grass)
→ jade, jade-green – (similar to the color of jade; especially varying from bluish green to yellowish green)	→ khaki – (of a yellowish brown color similar to olive drab)
→ lavender, lilac – (of a pale purple color)	→ mauve – (of a pale to moderate grayish violet color)
→ moss green, mosstone – (of a moderate somewhat dull yellow-green color)	→ mousy, mouse-colored, mouselike – (having a drab pale brown color resembling a mouse; ”a mousy grownish-gray color”; ”mouse-colored hair”; ”a mouselike rodent”)
→ ocher, ochre – (of a moderate orange-yellow color)	→ olive-brown – (of a brown color with a greenish tinge)
→ olive-drab – (of an olive-brown color similar to khaki)	→ olive – (of a yellow-green color similar to that of an unripe olive)
→ orange, orangish – (similar to the color of a ripe orange)	→ peacock-blue – (bright greenish blue)
→ pink, pinkish – (similar to the natural color of pinks)	→ purple, violet, purplish – (of a color midway between red and blue)
→ red, reddish, ruddy, blood-red, carmine, cerise, cherry, cherry-red, crimson, ruby, ruby-red, scarlet – (having any of numerous bright or strong colors reminiscent of the color of blood or cherries or tomatoes or rubies)	→ rose, roseate, rosaceous – (having a dusty purplish pink color; ”the roseate glow of dawn”)
→ rose-red – (of a deep slightly bluish red color)	→ rust, rusty – (of the color of rust)
→ sapphire – (having the color of a blue sapphire; ”sapphire eyes”)	→ snuff, snuff-brown, mummy-brown, chukker-brown – (snuff colored; grayish to yellowish brown)
→ sorrel, brownish-orange – (of a light brownish color)	→ stone – (of any of various dull tannish-gray colors)
→ straw – (of a pale yellow color like straw; straw colored)	→ tan – (of a light yellowish-brown color)
→ tangerine – (of a strong reddish orange color)	→ tawny – (of a light brown to brownish orange color)
→ ultramarine – (of a brilliant pure blue to purplish blue color)	→ umber – (of the color of any of various natural brown earth pigments)
→ vermilion, vermillion, cinibar, Chinese-red – (of a vivid red to reddish-orange color)	→ yellow, yellowish – (similar to the color of an egg yolk)
→ yellow-green – (midway between yellow and green)	→ avocado – (of the dull yellowish green of the meat of an avocado)
→ beige – (of a light grayish-brown color)	→ blae – ((Scot) bluish-black or gray-blue)
→ chestnut – (used of hair; of a golden brown to reddish brown color; ”a chestnut horse”; ”chestnut hair”)	→ coppery, copper colored – (having the color of copper)
→ coral – (of a strong pink to yellowish-pink color)	→ creamy – (of the color of cream; ”creamy translucent pebbles”)
→ cress green, cresson, watercress – (of a moderate yellow-green color that is greener and deeper than moss green and yellower and darker than pea green)	→ hazel – (of a light brown or yellowish brown color)
→ honey – (having the color of honey)	→ magenta – (deep purplish red)
→ maroon – (dark brownish to purplish red)	→ mosslike, mossy – (resembling moss)
→ pea-green – (of a moderate slightly yellowish-green color)	→ powder blue – (moderate to pale blue or purplish blue)
→ russet – (brown with a reddish tinge)	→ sage, sage-green – (of the gray-green color of sage leaves)
→ sea-green – (of the color of the sea; bluish green)	

Tabella 2.9: Aggettivi satelliti di “chromatic, colored”

0	1	2	3	4	5	6	7	8	9
4312	11574	585	369	314	186	130	94	78	58
10	11	12	13	14	15	16	17	18	19
43	35	15	17	21	15	8	10	10	7
20	21	22	23	24	25	26	27	28	29
5	7	5	1	1	2	2	3	1	0
30	31	32	33	34	35	36	37	38	39
0	0	1	0	0	1	0	0	0	0
40	41	42	43	44	45	46	47	48	49
0	0	0	0	1	0	2	0	0	1
50	51	52	53	54	55	56	57	58	59
0	0	0	0	0	0	0	0	0	0
60	61	62	63	64	65	66	67		
0	0	0	0	0	0	0	1		

Tabella 2.10: Distribuzione dei *synset* (degli aggettivi) per numero di similarità

2.2.10 Relazione di pertinenza, *pertainym*

Gli aggettivi che rimangono fuori dall'organizzazione per cluster contrapposti, sono gli aggettivi relazionali o "pertinenti".

Un aggettivo "pertinente" è usualmente definito da una frase come "di o pertinente a" e non ha antinomi. Può puntare ad un nome o ad un altro *pertainym* con una relazione chiamata appunto di pertinenza.

Aggettivi	0	1	2	3	4	5	6	7
conteggio	14817	2349	622	87	35	3	2	
percentuale	84%	13%	3%	0%	0%	0%	0%	
Avverbi	0	1	2	3	4	5	6	7
conteggio	1166	1884	376	109	31	2	5	2
percentuale	33%	52%	11%	3%	1%	0%	0%	0%

Tabella 2.11: Distribuzione dei *synset* per numero di *pertainym*

Esempio un *synset* di aggettivi che deriva da due *synset* di nomi:
"amoebic, amebic, amoeban, ameban, amoebous, amebous" – (pertaining to or resembling amoebae; "amoebic dysentry")
Pertains to noun amoeba (Sense 1)
$\longrightarrow$ ameba, amoeba – (naked freshwater or marine or parasitic protozoa that form

temporary pseudopods for feeding and locomotion)
⟶ rhizopod, rhizopodan – (protozoa characterized by a pseudopod)

Esempio un *synset* di avverbi che deriva da un *synset* di aggettivi:
"craftily, cunningly, foxily, knavishly, slyly, trickily, artfully" – (in an artful manner; "he craftily arranged to be there when the decision was announced"; "had ever circumstances conspired so cunningly?")
Derived from adj crafty (Sense 1)
⟶ crafty, cunning, dodgy, foxy, guileful, knavish, slick, sly, tricksy, tricky, wily – (marked by skill in deception; "cunning men often pass for wise"; "deep political machinations"; "a foxy scheme"; "a slick evasive answer"; "sly as a fox"; "tricky Dick"; "a wily old attorney")

2.2.11 Relazione participiale, *participle*

Aggettivi participiali cioè derivati da un verbo. Come si vede dalla Tabella 2.12 sono in numero non rilevante.

Esempio, "burned, burnt" – (having undergone oxidation: "burned powder")
Participle of verb burn (Sense 3)
→ burn, combust – (undergo combustion; "Maple wood burns well")
→ change state, turn – (undergo a transformation or a change of position; "We turned from Socialism to Capitalism")

Aggettivi	0	1	2
conteggio	17833	74	8
percentuale	100%	0%	0%

Tabella 2.12: Distribuzione dei *synset* per numero di *Participle*

2.2.12 Attributo, *attribute*

La relazione che lega un aggettivo ad un nome a cui esprime un valore. Per esempio `light` e `heavy` esprimono un valore al nome `weight`.
Esempio thin:
Sense 1
thin (vs. thick) – (of relatively small extent from one surface to the opposite or in

Aggettivi	0	1	2
conteggio	17284	612	19
percentuale	97%	3%	0%

Tabella 2.13: Distribuzione dei *synset* per numero di *attribute*

cross section; “thin wire”; “a thin chiffon blouse”; “a thin book”; “a thin layer of paint”)
→ thickness – (the dimension through an object as opposed to its length or width)

Sense 2
thin (vs. fat), lean – (lacking excess flesh; “you can’t be too rich or too thin”; “Yon Cassius has a lean and hungry look”-Shakespeare)
→ body weight – (the weight of a person’s body)

Sense 6
thin (vs. thick) – (relatively thin in consistency or low in density; not viscous; “air is thin at high altitudes”; “a thin soup”; “skimmed milk is much thinner than whole milk”; “thin oil”)
→ thickness – (resistance to flow)
→ consistency, consistence, body – (the property of holding together and retaining its shape; “when the dough has enough consistency it is ready to bake”)

2.2.13 Coordinati

Non è un tipo di relazione base, ma derivato. Due *synset* sono coordinati se condividono uno stesso iperonimo, cioè se sono la specializzazione dello stesso concetto.

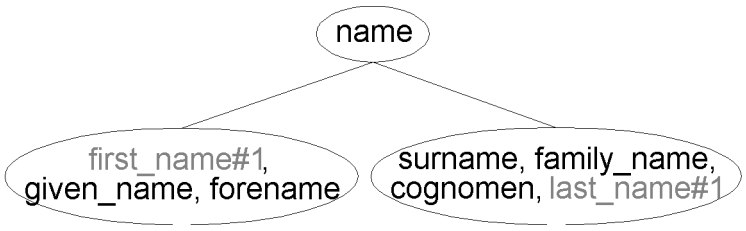


Figura 2.18: Esempio di *synset* coordinati

2.2.14 Citazione delle glosse

Non è una relazione semantica né lessicale, ma indica semplicemente che un termine è citato nella glossa di un altro. Non è dunque simmetrica, ed è totalmente aleatoria, visto che si possono usare parole diverse per descrivere uno stesso concetto.

Infatti viene utilizzato un indice a parte che contiene una lista di lemmi con associato per ognuno una lista di *synset* che citano il termine in esame nella proprio glossa.

Per esempio la parola `golf` è citata in `golfclub` , `golfer`, `golfers`, `sudden death`, ...

2.3 Organizzazione per categorie sintattiche

Le parole in WordNet sono divise per categoria sintattica. In particolare, per ora, le categorie sono: nomi, verbi, aggettivi ed avverbi. Come futura espansione verà aggiunta una quinta categoria per gli aggettivi di tipo satellite.

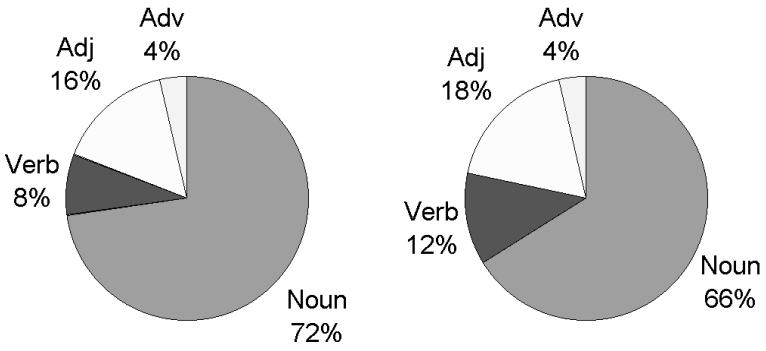


Figura 2.19: Percentuale di lemmi (sinistra) e *synset* (destra) per categoria sintattica

Questa divisione genera un po' di ridondanza, infatti la maggioranza dei termini ha significati in diverse categorie sintattiche, per esempio "back" è presente in misura di 9/10/3/6 synset in nomi/verbi/aggettivi/avverbi; e per cercare i significati di una parola bisogna entrare nell'indice di ogni categoria per vedere se è presente.

Il vantaggio è che sono rese manifeste a livello di struttura le differenze di organizzazione semantica. Infatti il metodo di categorizzazione delle categorie sintattiche è:

nomi gerarchia di argomenti

verbi una varietà di relazioni di implicazione

aggetivi e avverbi iperspazio n-dimensionale

2.3.1 Nomi

I nomi sono organizzati in una gerarchia di specializzazione (iperonimi/iponimi).

Nelle moderne teorie psicolinguistiche per definire un nome di solito si usa il termine sovraordinato più dei fattori distintivi; in questo modo è organizzato WordNet: un iponimo eredita dai termini sovraordinati tutte le loro caratteristiche, ma se ne differenzia almeno per una cosa.

2.3.2 Verbi

Anche i verbi sono strutturati con una gerarchia di specializzazione (troponimia, modo particolare di compiere un'azione), ma a differenza dei nomi i beginner sono 617 invece che 7, ed il livello di profondità è molto più piatto.

Per i verbi sono molto importanti, invece, le relazioni di implicazione (*entailment* e *cause to*).

2.3.3 Aggettivi ed Avverbi

Gli aggettivi sono strutturati in cluster che contengono un *synset* principale e dei *synset* satellite. Ogni cluster è organizzato intorno a coppie di antinomi (occasionalmente triplette). Le coppie di antinomi sono indicate nei *synset* principali dei cluster e sono dette relazioni di antinomia dirette. I *synset* principali sono collegati ai *synset* satellite da una relazione di similarità ed i *synset* satelliti mediano tramite il *synset* principale le relazioni di antinomia che diventano derivate. Un modo per pensare l'organizzazione a cluster è visualizzare una ruota, con il *synset* principale al centro ed i satelliti sui raggi. Due o più ruote sono connesse logicamente con la relazione di antinomia che può essere pensata come un asse tra le due ruote.

Gli aggettivi pertinenti sono aggettivi relazionali che non seguono la struttura a cluster: non hanno antinomi, di solito i *synset* contengono un solo lemma ed hanno una relazione con un nome. Gli aggettivi partecipiali hanno una relazione lessicale con il verbo dal quale derivano.

Gli avverbi spesso derivano da un aggettivo, e di tanto in tanto hanno un antinomo. Di solito contengono solo la relazione con l'aggettivo da cui derivano.

Capitolo 3

Estrazione lessicale di relazioni

In questo capitolo si ritorna a parlare di MOMIS, e viene presa in esame la fase di *Estrazione lessicale di relazioni* anticipata per sommi capi alla sezione 1.2.2.

3.1 Mapping della relazioni da WordNet a MOMIS

La terminologia usata in WordNet ha diverse differenze rispetto a quella utilizzata in MOMIS. Infatti WordNet viene da un ambito linguistico, mentre MOMIS da un ambito più strettamente informatico.
Per cui si propone un quadro riassuntivo:

Relazione Lessicale

MOMIS	WordNet
Relazione estratta utilizzando il database lessicale WordNet	Relazione tra lemmi, in contrapposizione ad una relazione semantica (vedi Sezione 2.2)

Relazione Semantica

MOMIS	WordNet
Le relazione che vengono incluse nel *Common Thesaurus* sono tutte semantiche, quelle estratte con l'ausilio di SLIM vengono promosse a semantiche dalla validazione del progettista	Relazioni tra significati (vedi Sezione 2.2)

Riassumendo, le fasi che portano a trasformare una relazione da lessicale a semantica sono, dati due nomi (di classi o attributi) degli schemi sorgenti:

- si assegna ad ogni nome un significato (o più di uno)

- si trovano i *synset* che corrispondono a questi significati
- si cerca in WordNet se ci sia una relazione semantica tra questi *synset*, per esempio il primo specializza il secondo
- si traduce questa relazione semantica per WordNet in una relazione lessicale per MOMIS facendo il percorso inverso
- dai *synset* si trova il significato che rappresentano
- dal significato si trovano i nomi (di classi o attributi) a cui è stato assegnato questo significato
- quindi si è trovata una relazione lessicale tra i due nomi; nell'esempio una relazione **NT** tra il primo ed il secondo nome
- il tool presenta la relazione al progettista
- se il progettista l'accetta, viene aggiunta al *Common Thesaurus* come relazione semantica, quindi con lo stesso valore di ranking di quelle estratte da SIM.

Delle relazioni che si possono ricavare da WordNet (vedi Sezione 2.2) sono state prese in esame le relazioni di seguito riportate:

Sinonimia si trasforma in una relazione SYN.

Iperonimia si trasforma in una relazione BT (Broader-Term), relazione di generalizzazione.

Iponimia si trasforma in una relazione NT (Narrower-Term), relazione di specializzazione.

Olonimia è assunta come una relazione di tipo RT (Related-Term); cioè l'aggregazione è considerata un legame più debole rispetto alla specializzazione.

Meronimia è assunta come una relazione di tipo RT (Related-Term); La meronimia (contiene) è la relazione opposta alla olonimia (è contenuto).

Correlazione è assimilata ad una relazione di tipo RT (Related-Term); la correlazione è la relazione che lega 2 *synset* che condividono uno stesso iperonimo, cioè lo stesso padre.

Sono state selezionate solo queste perché sono la maggior parte delle relazioni in WordNet, come si evince dalla Appendice A.3, dedicata alle statistiche su WordNet. Negli esempi di riferimento proposti (Appendice C) non sono presenti altri tipi di relazioni; questo però non significa che come passo futuro non si possa allargare anche ad altri tipi di relazioni. In questa tesi, partendo da nulla, si è cercato di fare un prodotto verticale, dalla libreria a basso livello al front-end per l'utente, ed in questa fase si è mirato all'efficienza invece che al massimo dell'efficacia.

Un altro spunto di ricerca è l'arbitrarietà della scelta di considerare come coordinati i termini che condividono uno stesso iperonimo salendo la gerarchia di un solo nodo e non più di uno. Navigando in WordNet si nota che il salto di un livello gerarchico (tra padre e figlio) non sempre corrisponde ad un uguale livello di affinamento, per esempio tra `abstraction` e `relation` c'è un "salto" maggiore che tra `department` e `division`. E non è solo in funzione del livello di profondità (`abstraction` è un *beginner*), ma sembra anche funzione dei recensori che hanno redatto i singoli file lessicografici all'origine di WordNet (vedi Appendice A.4); per esempio la parte sulle religioni è molto definita (vedi Figura 2.7), mentre tutti gli uccelli sono "piatti" sotto `bird_genus` (397 *synset*).

3.2 La libreria per interagire con WordNet

Il pacchetto WordNet[1] oltre ai file di dati e di indice fornisce un programma per compiere semplici interrogazioni ed una libreria con i sorgenti in *C*.

SI-Designer, invece, per scelta architetturale, utilizza il linguaggio *Java*. Una via percorribile sarebbe stata quella di richiamare le funzioni C tramite metodi nativi, cioè che incapsulano codice *platform dependent*. Questo approccio è stato seguito, per esempio, dal progetto Teseo [24, 25]. Il problema della libreria fornita a corredo di WordNet è che presenta solo funzioni ad alto livello, cioè troppo flessibili per offrire un controllo preciso.

La funzione principale, `findtheinfo`, ritorna la ricerca in un buffer come output formattato. Esempio, iperonimi di `address#2`:

```
{06263815} <noun.location> address#2 -- (the place where a person or ...
 => {06324198} <noun.location> geographic point#1, geographical point ...
  => {06351684} <noun.location> point#2 -- (the precise location of ...
   => {00014887} <noun.Tops> location#1 -- (a point or extent in space)
    => {00009457} <noun.Tops> object#1, physical object#1 -- (a physical ..
     => {00001740} <noun.Tops> entity#1, something#1 -- (anything  ...
```

[1] Reperibile all'indirizzo http://www.cogsci.princeton.edu/~wn/

Dunque i difetti sono:

- Necessità di codificare un parser per estrarre dal testo le informazioni cercate; Teseo ne offre un esempio.
- Il buffer dell'output è limitato e può andare in overrun.
- Non si riescono ad avere tutte le informazioni contenute in WordNet.
- Il codice è dipendente dalla piattaforma per il quale sono compilate le librerie.

Si è deciso di scrivere una libreria [2] interamente in *pure Java*. Questa libreria è molto versatile: ci permette di navigare tra i *synset*, oppure offre funzioni di alto livello che, a differenza di quelle di WordNet, riportano i dati in strutture dati native di Java.

In breve WordNet è organizzato in definizioni e (*entry*): una coppia $e = (f, m)$ dove f è la forma base e m (*meaning*) è il contatore del significato, esempio (`address`, `2`) si riferisce all'indirizzo presso il quale si può trovare una persona o una organizzazione; mentre (`address`, `1`) si riferisce all'indirizzo di un computer in ambito informatico.

Alcuni esempi:

`Synset trovaSynset(String word, int senseNum)` dato una definizione e (forma base, contatore del significato) trova dai file di indice il *synset* y in cui è presente.

$$f_{trovaSynset} : e \longrightarrow y \qquad y : e \subset y$$

`Vector trovaRicorsivo(Synset inpynset, String tipo)` a partire da un *synset* y trova tutti i *synset*[2] che si possono raggiungere con un puntatore[3] di tipo fissato, per esempio seguendo la gerarchia di specializzazione o aggregazione.

$$f_{trovaRicorsivo} : (y, r) \longrightarrow \{y_1, \ldots, y_n\} \qquad y_i : \langle y_i \, r \, y \rangle \quad \forall i = 1 \ldots n$$

Le prestazioni in definitiva sono buone: l'inefficienza per interpretare il bytecode Java viene controbilanciata da algoritmi con minore overhead.

[2] Un insieme $\{y_1, \ldots, y_n\}$

[3] Che implica una relazione di tipo $r \in \{\text{SYN, BT, NT, RT}\}$

3.3 L'algoritmo per trovare le relazioni tra termini

Per trovare le relazioni tra coppie di termini si è usato il seguente algoritmo:

1. Input.
   In ingresso si prende il vettore di termini (nomi di attributo e di interfaccia) e si suppone già definito il loro significato.

2. Creazione della struttura dati di lavoro.
   Viene creato un Hash di [synset, vettore dei termini sinonimi]. Per come è costruito Wordnet un *synset* può essere usato come chiave di un Hash senza ulteriori manipolazioni. Come dato associato alla chiave c'è un vettore contenente i termini trovati negli schemi che appartengono al *synset* chiave. In particolare:

   Per ogni termine
   viene trovato il *synset*
   si cerca nel Hash se e' gia' presente:
   se non c'e' si crea l'elemento del Hash
   se no si aggiunge al vettore corrispondente alla chiave

3. Sezione principale: trova le relazioni tra gli elementi.

   Per ogni elemento del Hash
   si scrivono le relazioni di sinonimia tra gli elementi associati alla chiave attuale
   per ogni elemento del vettore
   si trovano gli iperonimi
   per ogni iperonimo (che e' un *synset*) si cerca nel Hash se e' presente
   se c'e' si scrivono le relazioni di NT con tutti i termini del vettore
   si trovano gli iponimi e i termini correlati
   per ognuno di questi si cerca nel Hash se e' presente
   se c'e' si scrivono le relazioni di RT con tutti i termini del vettore

4. Eliminazione relazioni duplicate o inconsistenti.

   - Elimina le relazioni tra termini della stessa tabella.
   - Elimina relazioni duplicate, cioè con invertito il primo ed il secondo termine.

5. Output.
   Il risultato è un vettore di coppie di termini con la relazione che li lega.

3.4 L'ambiguità dei significati

Ricordando l'ipotesi che è stata posta, questo algoritmo funziona a condizione che si assegni il significato al lemma.

Per esempio: `address`, in WordNet, ha 15 diversi significati: nell'esempio di riferimento address è un attributo di `office` di tipo `location`, dunque in questo contesto `address` è usato con il significato numero `2` (indirizzo del tipo via, numero, città, ...). `Address#2` è contenuto in un solo *synset*, quindi ci permette di stabilire in modo univoco il suo *synset*, da confrontare con i *synset* degli altri identificatori per calcolare le relazioni.

Nello schema sorgente, però, non è indicato il numero del significato: se cerchiamo le relazioni con gli altri termini a partire da tutti i suoi 15 significati, troviamo che `address` è una specializzazione di `name`, il che è certamente vero, infatti il verbo `address` nel senso {`greet by a prescribed form; "He always addresses me with "Sir"`} ha come iperonimo il verbo `name` nel senso {`name call, assign a specified name to; "They named their son David"; "The new school was named after the famous Civil Rights leader"; "Call me Boris"`}. Oppure, nello stesso senso, condivide un iperonimo, cioè è termine correlato, con `title` nel senso {`entitle title, give a title to`}. Tutto vero, però non in questo contesto in cui `address` è un indirizzo postale, `name` è un nome usato per identificare una persona o una cosa, e `title` è la qualificazione di un professore.

In particolare, analizzando l'esempio di riferimento e cercando le relazioni tra tutti i significati di ogni lemma, alcuni risultati "*sbagliati*", cioè non calzanti con il contesto in esame, sono:

address IS_A name	number IS_A name	department COORD city
address IS_A relation	number IS_A relation	faculty COORD rank
course IS_A location	room IS_A relation	last_name COORD title
description IS_A relation	title IS_A name	number COORD rank
first_name IS_A relation	title IS_A relation	room COORD street
last_name IS_A relation	address COORD description	title COORD address
length IS_A section	address COORD name	title COORD first_name
name IS_A relation	department COORD county	title COORD section

Si noti che, per quanto alcune di queste relazione possano sembrare curiose, sono vere, in contesti particolari.

Se invece ci poniamo nel caso ottimo, cioè supponiamo che, in qualche modo, sia annotato nel codice sorgente il numero del significato, le relazioni estratte a partire dal solo significato "*giusto*" del lemma:

name SYN name	first_name IS_A name	title IS_A name
rank SYN rank	last_name IS_A name	un_student IS_A cs_perso
year SYN year	professor IS_A cs_person	un_student IS_A student
address IS_A location	professor IS_PART faculty	first_name COORD last_na
city IS_A location	school_member IS_A cs_person	
country IS_A location	student IS_A cs_person	

Queste relazioni sono ottime nell'ipotesi che i significati forniti siano corretti.

In Tabella 3.1 e nelle Figure 3.2 e 3.3 sono riportate per esteso le statistiche relative ai tre esempi di riferimento. La voce *lemma* si riferisce all'utilizzo di tutti i significati del lemma, mentre la voce *tag* solo di quelli ritenuti giusti.

Quindi il problema si sposta nel risolvere l'ambiguità sul significato, per poter fornire a WordNet, per ogni identificatore, una coppia (forma base, numero del significato) appropriata al contesto. Per trovare la forma base a partire da un nome di identificatore declinato o coniugato WordNet fornisce un processore morfologico. Per esempio di `axes` vengono trovate 3 forme base: `ax` (1 senso), `axis` (5 sensi), `axe` (2 sensi).

Il processore morfologico però non può interpretare le abbreviazioni (`dept` per `department`) o gli attributi formati da più parole (`seats_number`) che non siano locuzioni (`go back` sinonimo di `return`), che vengono trattare come una singola forma base.

3.5 Componente grafico semiautomatico

Vista la complessità del problema, la soluzione che si propone è semiautomatica. In breve un tool grafico che per ogni identificatore chiede al progettista preposto all'integrazione degli schemi di scegliere il significato corretto nel contesto.

Per esempio in Figura 3.1 sono mostrati i significati che verrebbero proposti per `address`.

Il progettista può impostare un filtro in base alla categoria sintattica (nomi, verbi, aggettivi, avverbi) e farsi visualizzare solo i nomi: i primi 7 significati, e da questi selezionare il secondo.

In più, se il processore morfologico fallisce, il progettista deve immettere la forma base. Compito semplice nel caso di una abbreviazione, più complesso se sono coinvolte più parole.

Per esempio `dept_name` della tabella `Department`, tra `department` e `name` si sceglie `name` perché `dept` serve solo a rafforzare ciò che è ovvio trattandosi della tabella `department`; mentre in `faculty_name` della tabella `un_student` si sceglie `faculty` perché `name` è un qualificatore di tipo per

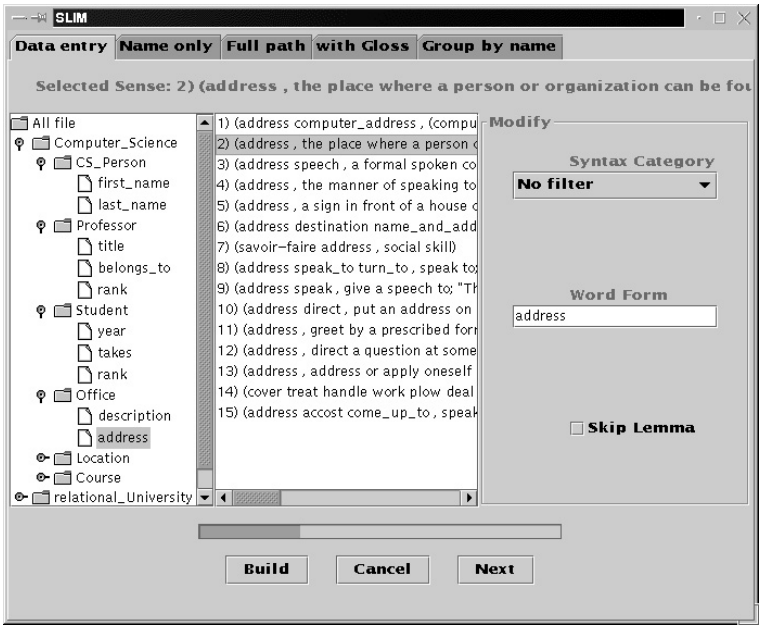


Figura 3.1: Significati di `address`

`faculty`, cioè specifica che vi è memorizzato come nome e non, per esempio, come codice.

Il progettista ha comunque l'opzione di ignorare un termine se troppo ambiguo o fuorviante. Perché:

- è un concetto troppo complesso per essere espresso da una sola parola: `seats_number`;
- appartiene ai *tops*, cioè ai concetti più generici, e dunque si troverebbe ad essere in relazione con tutto: `relation`;
- è una chiave surrogata, non aggiunge conoscenza: `dept_code` della tabella `Department`;
- è usata come *foreign key*, dunque questa relazione è già stata espressa durante l'estrazione di relazioni dalla struttura degli schemi: `dept_code` della tabella `research_staff`

Questo approccio semiautomatico riduce la complessità per il progettista: infatti, viene scomposto un solo problema "*difficile*", trovare le relazione tra tutti i termini, in tanti problemi "*facili*", scegliere da una lista il senso di ogni termine, in numero lineare rispetto alla quantità degli identificatori. In pratica questo

è problema 80/20 cioè l'80% dei termini si risolve nel 20% del tempo, solo il tempo di leggere le definizioni, mentre il restante 20% occupa l'80% del tempo, perché bisogna scegliere tra significati molto simili.

Per velocizzare l'80% ci si serve di una cache dei termini già decisi. Infatti l'ambito, che ci permette di eliminare le ambiguità, è diviso in 3 strati:

Basi di dati: se appare l'attributo name in qualunque tabella, è sempre nel senso di nome usato per identificare una cosa o persona, e non, per esempio, il verbo "dare il nome".

Schema sorgente: in un contesto universitario `course` è sempre usato nel senso di una serie di lezioni, e continua a mantenere questo significato in tutto il sorgente.

classe: sono i termini che vengono resi chiari solo dal nome della tabella o dagli altri campi; in questo caso la cache non ci può aiutare.

Comunque i termini trovati in cache non vengono decisi automaticamente, ma vengono solo evidenziati per lasciare l'ultima parola al progettista.

Il 20% di termini difficili, sono ambigui perché presentano significati molto simili. Ad esempio nella seguente frase:

> John likes to eat rabbit ... He shot it in the hills nearby.

`Rabbit` si riferisce al cibo coniglio o all'animale coniglio ?

WordNet permette di raggruppare i *synset* per similarità. In particolare usa le relazioni di

cuginanza, due termini appartenenti a gerarchie di specializzazione diverse come animal/food

sorellanza, condividono uno stesso iperonimo,

gemellanza, tre termini uguali tra i *synset*.

Nell'esempio sopra verrebbero raggruppati i significati di `rabbit` come animale, cibo e pelliccia, mentre separati gli altri come ad esempio "uomo codardo".

Il progettista può scegliere uno qualunque dei *synset* raggruppati perché l'algoritmo li tratta come identici. Si perde una sfumatura, ma si rischia meno che lo stesso termine venga scelto con significati diversi.

Computer_Science/University/Tax_Position

	syn	ipon	olon	coord	corelex	gloss
Lemma	5	21	2	15	1	5
Tag	16	20	2	1	0	0
	syn	ipon	olon	coord	corelex	gloss
Lemma	5	28	2	20	1	5
Tag	23	27	2	1	0	0

Cardiology_Department/Intensive_Care

	syn	ipon	olon	coord	corelex	gloss
Lemma	15	12	0	11	5	7
Tag	15	2	0	1	0	0
	syn	ipon	olon	coord	corelex	gloss
Lemma	34	62	0	41	16	21
Tag	34	12	0	4	0	0

Eating_Source/Food_Guide

	syn	ipon	olon	coord	corelex	gloss
Lemma	5	11	1	4	4	12
Tag	10	6	0	2	0	0
	syn	ipon	olon	coord	corelex	gloss
Lemma	16	39	4	23	5	25
Tag	21	16	0	2	0	0

Tabella 3.1: Relazioni estratte per esempio di riferimento

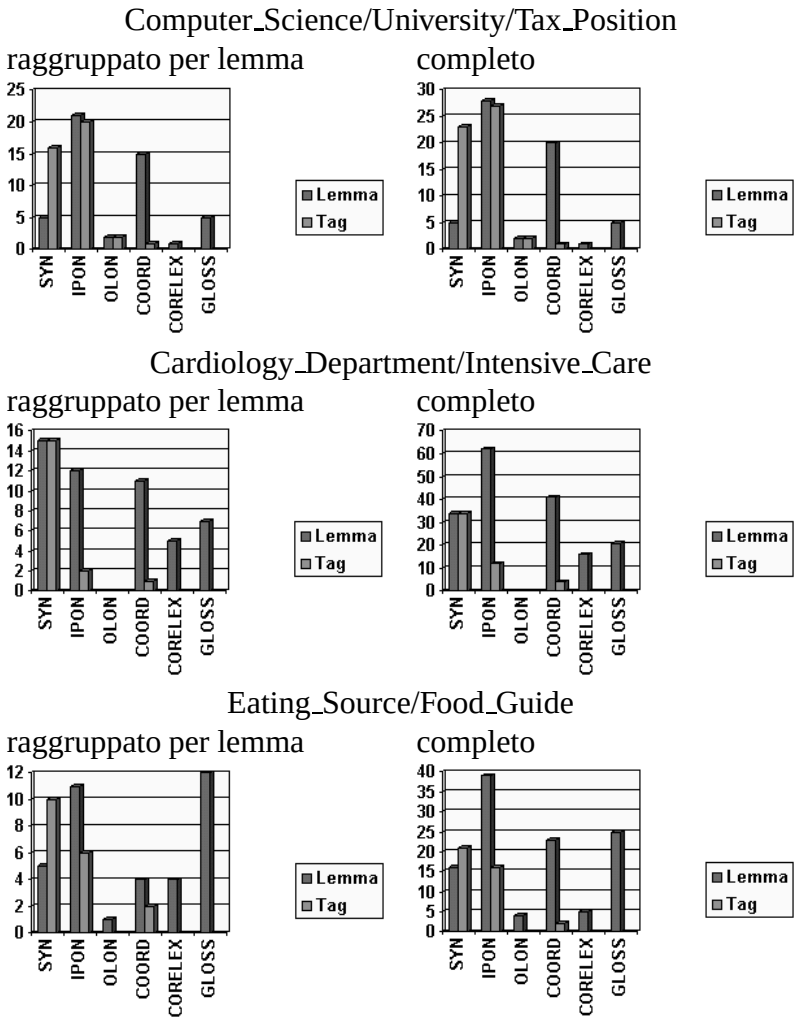


Figura 3.2: Relazioni estratte per esempio di riferimento (disagregato)

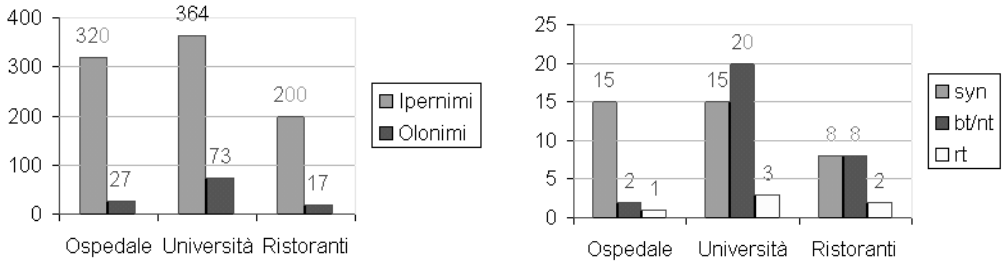


Figura 3.3: Relazioni estratte per esempio di riferimento (confronto)

Capitolo 4

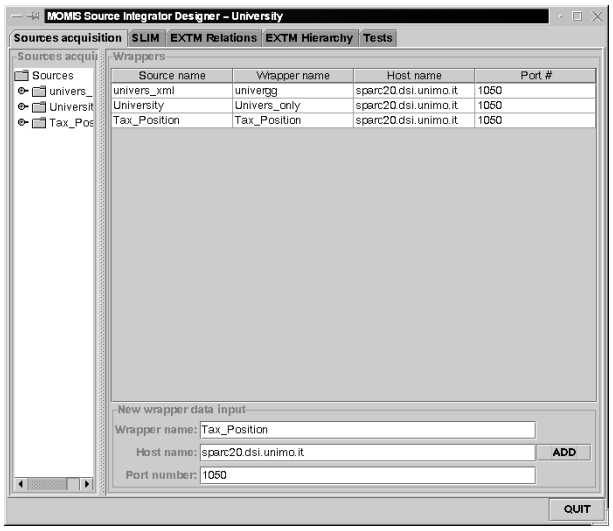
Componente grafico semiautomatico: progetto e realizzazione

Nella sezione 3.5 sono state elencate le motivazioni che hanno portato alla realizzazione di un componente grafico semiautomatico per la rimozione delle ambiguità. In questo capitolo viene preso in esame solo il punto di vista implementativo.

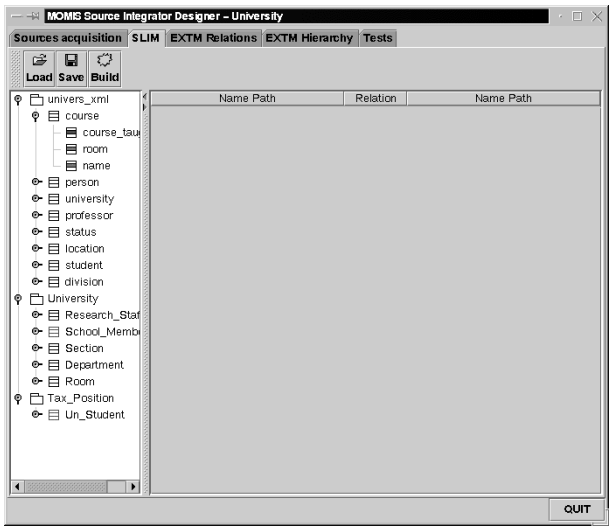
Il modulo è stato chiamato SLIM (**S**chemata **L**exical **I**ntegration **M**odule) ed, in breve, offre al progettista una lista di lemmi, gli identificatori usati per le classi e gli attributi dei sorgenti da integrare, e chiede al progettista di selezionare il significato calzante con il contesto in esame dall'elenco di quelli presenti in WordNet per la parola in esame.

4.1 Esempio d'uso

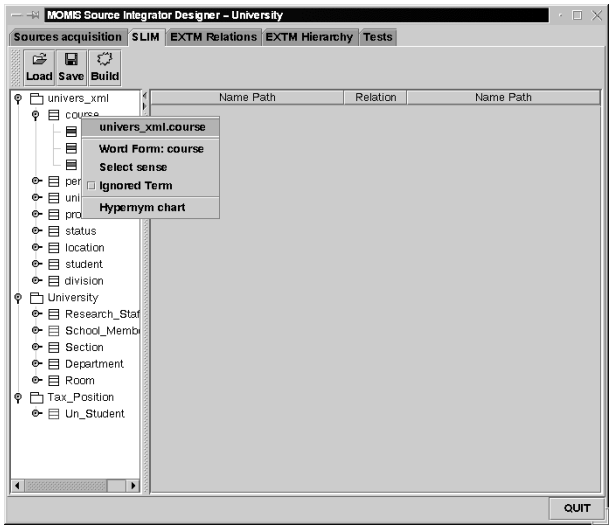
Un breve esempio introduttivo per spiegare meglio le funzionalità.



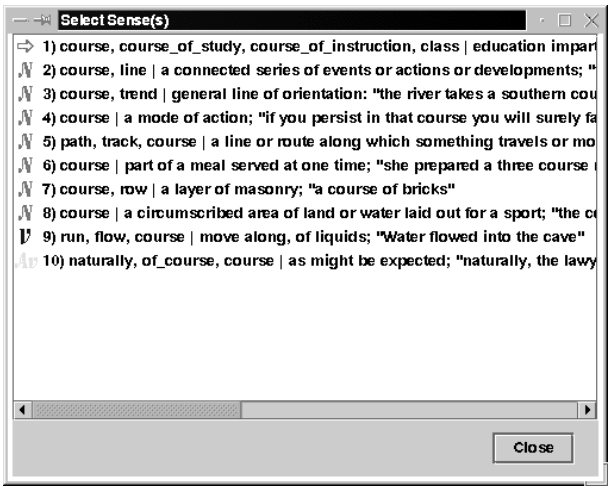
Il componente prima di SLIM. Il modulo per l'acquisizione delle sorgenti ha concluso il suo compito ed ora si può passare a SLIM premendo l'etichetta in alto.



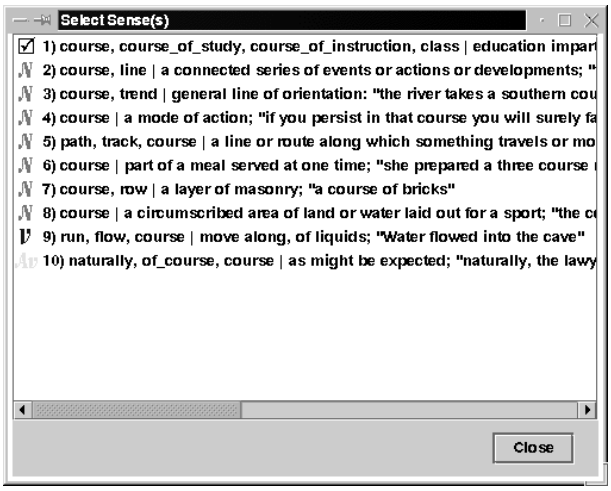
Come si presenta SLIM inizialmente. Il progettista ha aperto una sorgente (radice dell'albero) ed al suo interno alcune classi (rami) per visualizzarne gli attributi (foglie).



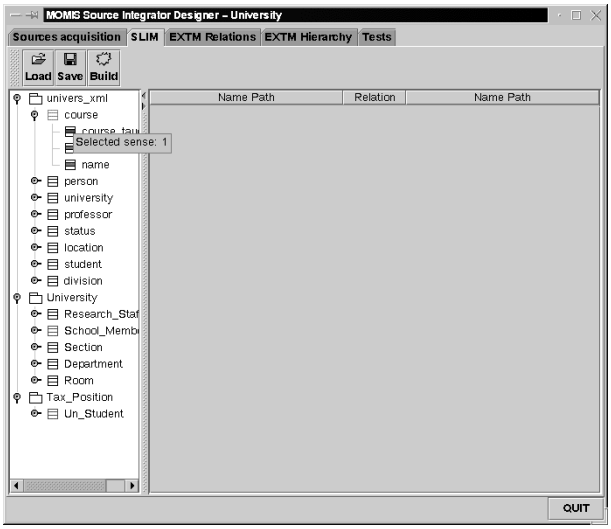
Il progettista ha visualizzato il menù contestuale dell'attributo `course` da cui può scegliere le azioni disponibili per l'elemento in esame. La procedura per far apparire il menù è *system dependant*: tasto destro per windows o KDE, mela più tasto per MacIntosh.



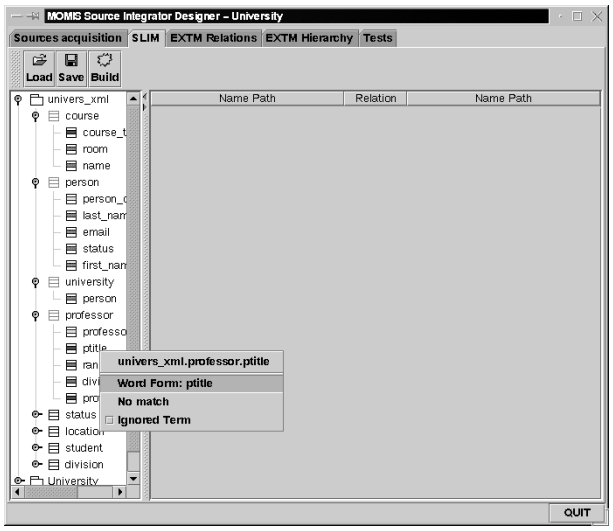
Scegliendo la voce "Select Sense" appare una finestra di dialogo che elenca i significati di `course`. Un'icona all'inizio della riga indica la categoria sintattica.



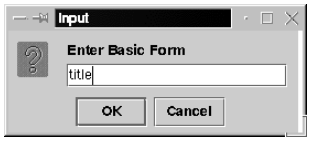
Al posto dell'icona della categoria sintattica può apparire una freccia: è il significato che il progettista ha selezionato in un'occasione precedente per quel termine. Il progettista, in questo caso, conferma la scelta precedente ed appare un segno di spunta.



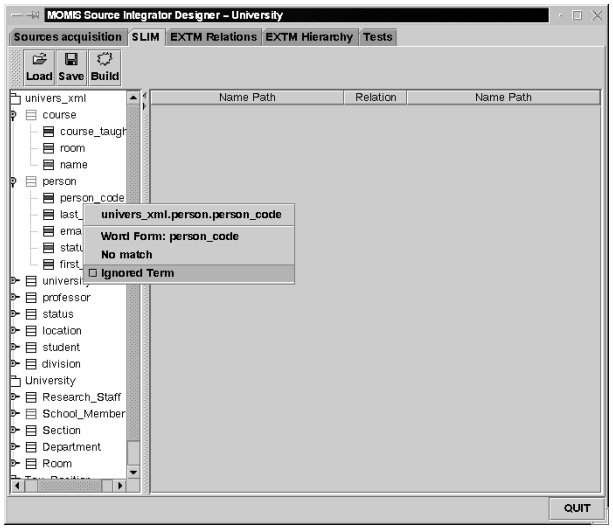
Il progettista chiude la dialog, ed ora l'elemento `course` appare in verde. Il tooltip conferma la scelta avvenuta.
Il progettista svolge questa procedura per ogni termine.



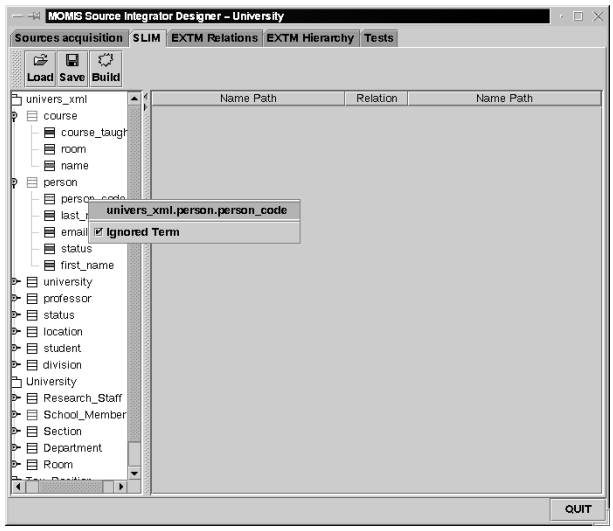
Un caso particolare che può presentarsi nel corso di questa procedura, è che il motore morfologico di WordNet non sia in grado di trovare la forma base.



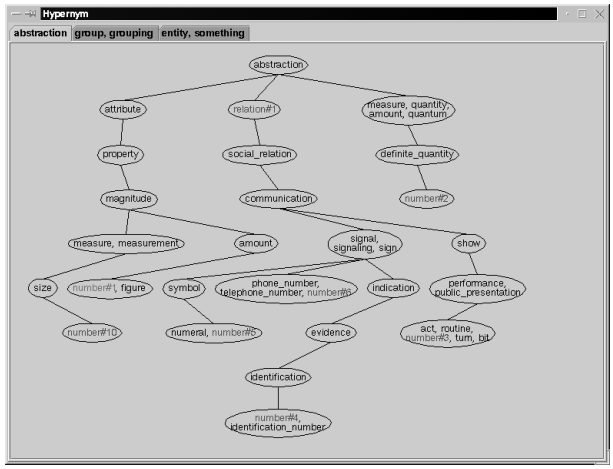
Selezionando dal menù “Word Form” si può immettere un nuovo valore.



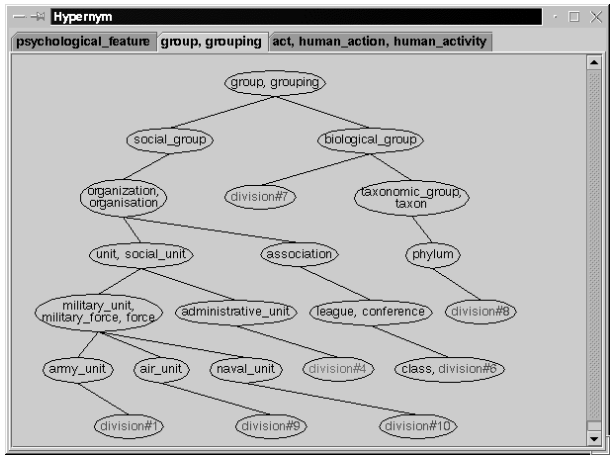
Inoltre può rendersi necessario ignorare un termine. In questo caso perché è una chiave surrogata che non aggiunge conoscenza. Dal pop-up c’è una casella di spunta “Ignored Sense”.



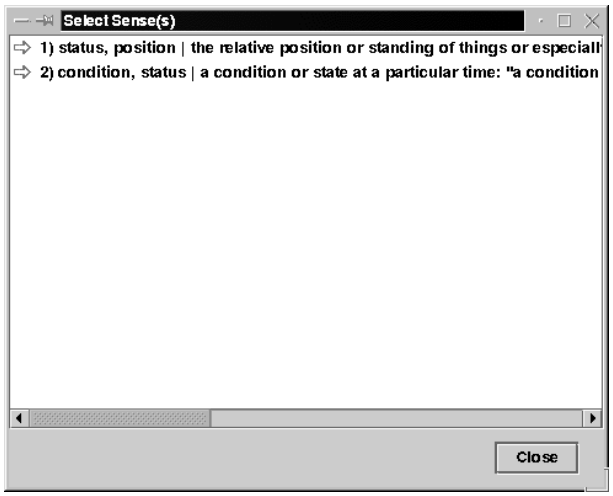
Il menù si adatta al nuovo stato e, dunque, l'unico comando disponibile è quello per includere di nuovo il termine.



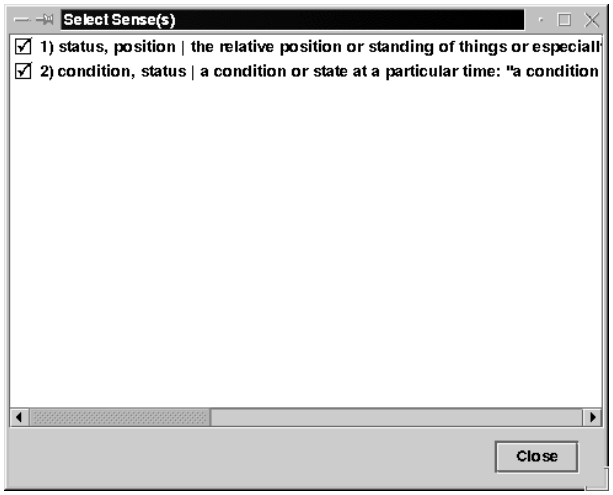
Il progettista, tramite il comando "Hypernym Chart", che è presente solo se la forma base è consistente, può visualizzare il diagramma degli iperonimi della parola selezionata. In Figura la parola `number`: le cui occorrenze sono riportate in rosso.



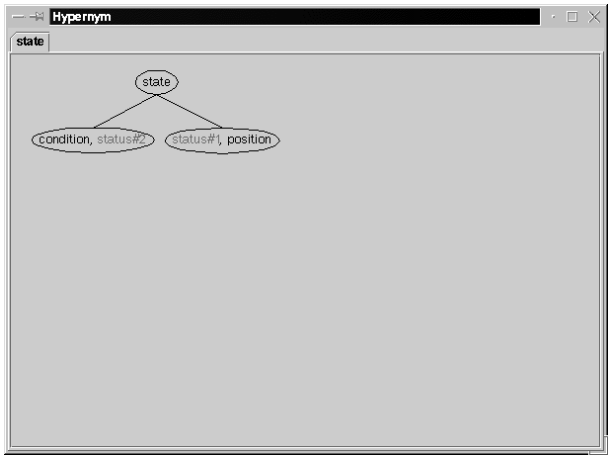
I diagrammi aiutano nelle decisioni più difficili: nel caso in esame si vede chiaramente che il significato corretto è il *synset* che contiene `division#4` in quanto specializzazione di `administrative_unit`.



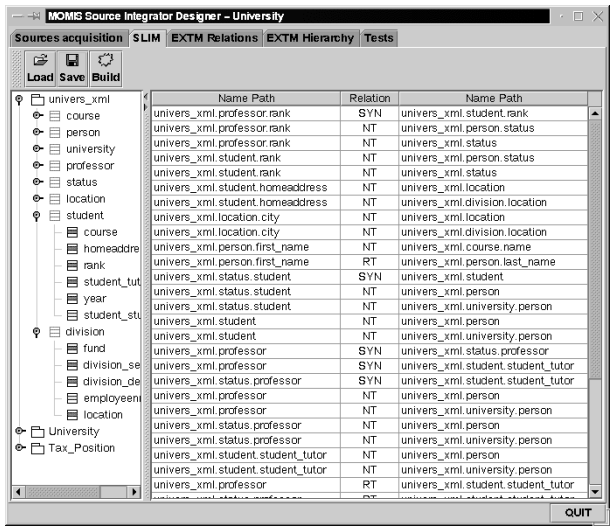
Il progettista può selezionare più significati: premendo su un significato viene messo o tolto il segno di spunta a seconda dello stato precedente. Quando il progettista è soddisfatto preme il pulsante “close”.



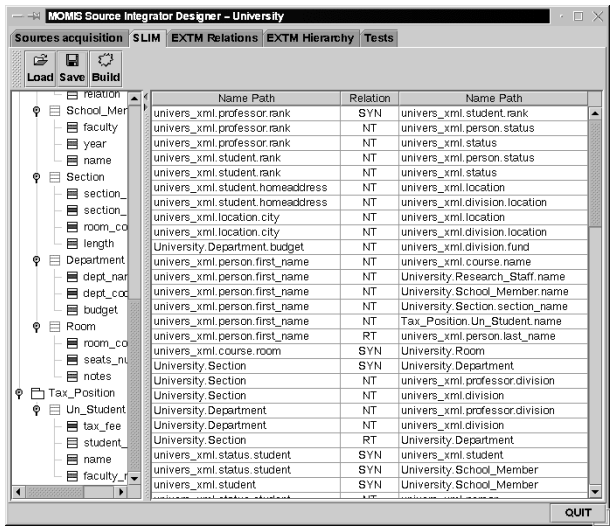
Nel caso di `status` basta selezionare uno solo dei due significati che automaticamente appaiono come selezionati entrambi. Infatti il tool riconosce una certa similarità tra i *synset*.



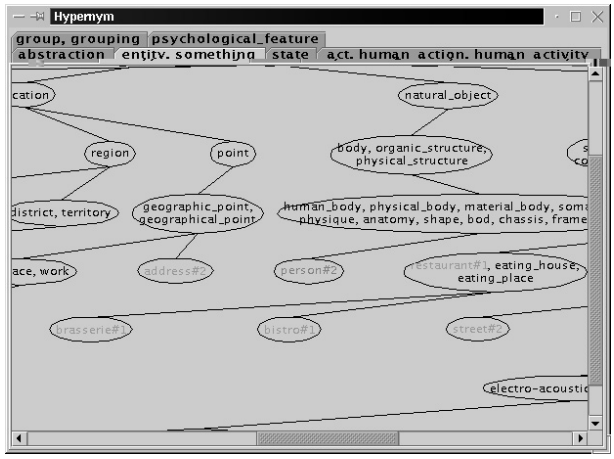
Dal diagramma degli iperonimi si vede che i due *synset* sono *sisters*, cioè condividono uno stesso diretto iperonimo.



Il progettista può premere il pulsante "Build" per calcolare le relazioni. In figura il risultato parziale, dopo aver terminato il data entry della prima sorgente: sono solo relazioni intra-schema.



A sinistra è mostrato il risultato finale. Solo parte delle relazioni sono visualizzate. Da notare che il *dot path* viene sempre visualizzato indipendentemente dalla dimensione della cella, in quanto il *font* utilizzato è adattivo dello spazio disponibile.



Dal menù contestuale dell'intero albero è possibile accedere al diagramma degli iperonimi somma di tutti gli alberi degli iperonimi dei termini.

4.2 Specifiche funzionali

Oltre alle specifiche funzionali già descritte, le altre specifiche non visibili sono:

- Integrazione in SI-Designer:
  - estendere SIDPhase: una classe condivisa che ha la funzione di interfaccia comune per i componenti che occupano i pannelli di SI-Designer,
  - estrarre i nomi attributo/interfaccia dalla gerarchia Java rappresentativa degli schemi ODL_{I^3} ,
  - immettere le relazioni lessicali nel *Common Thesaurus* costruendo i puntatori agli attributi/interfacce.
- Memorizzare le scelte effettuate dal progettista per riproporgliele nel caso si presenti lo stesso nome (*cache* dei significati).
- Mediare le chiamate all'oggetto CORBA che incapsula la libreria (Sezione 3.2) per interrogare WordNet tramite uno strato software (*proxy*) che minimizzi il numero di chiamate CORBA memorizzando quelle già effettuate.
- Gestire la selezione multipla dei significati ed espandere i significati selezionati a tutti i parenti (*sister*, *twin* e *cousin* vedi Sezione 3.5) dopo ogni selezione.
- Revisione/integrazione da parte del progettista: il menù contestuale della tabella delle relazioni permette di aggiungere o togliere le relazioni estratte.
- Tenere traccia delle operazioni svolte in modo che se il progettista ritorna all'immissione dati e riesegue il calcolo delle relazioni non debba, di nuovo, aggiungere e togliere le relazioni.

Dal punti di vista tecnologico, invece, le specifiche sono:

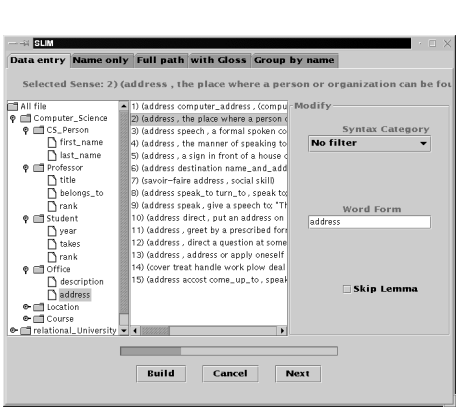
- Utilizzo del linguaggio Java, per essere integrato in SI-Designer.
- Interfaccia grafica con classi *swing* per uniformità grafica
- Tecnologia CORBA per interrogare la libreria di WordNet.

4.3 Primo prototipo

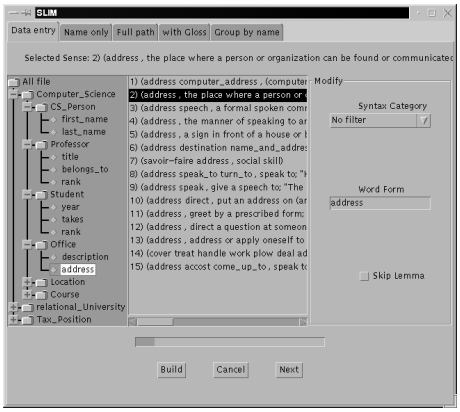
Inizialmente è stato costruito un prototipo. Per ogni progetto non banale e non ben definito può essere utile sviluppare un'applicazione di prova con lo scopo di affinare le specifiche, capire le vere esigenze e scoprire i punti algoritmicamente più difficili.

SI-Designer, poi, non era completo durante le prime fasi di sviluppo, per cui il primo SLIM doveva funzionare anche stand-alone, per poter essere testato, mentre queste funzionalità sono superflue e dannose per la versione definitiva.

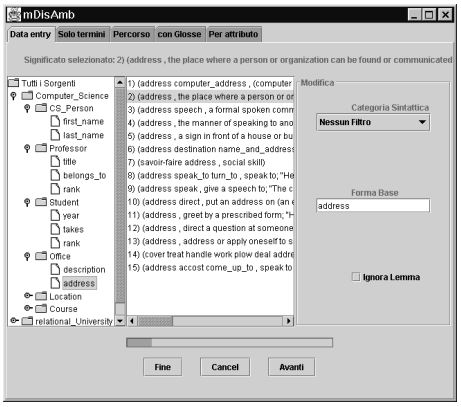
In Figura 4.1 si può vedere il suo aspetto grafico. La libreria *swing* permette di scegliere tra due *look and feel* (di base): multipiattaforma o nativo.



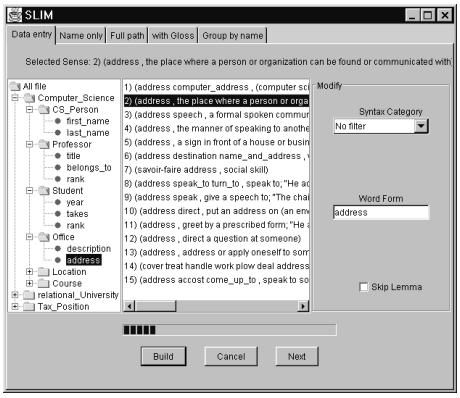
KDE multipiattaforma



KDE nativo



Windows multipiattaforma

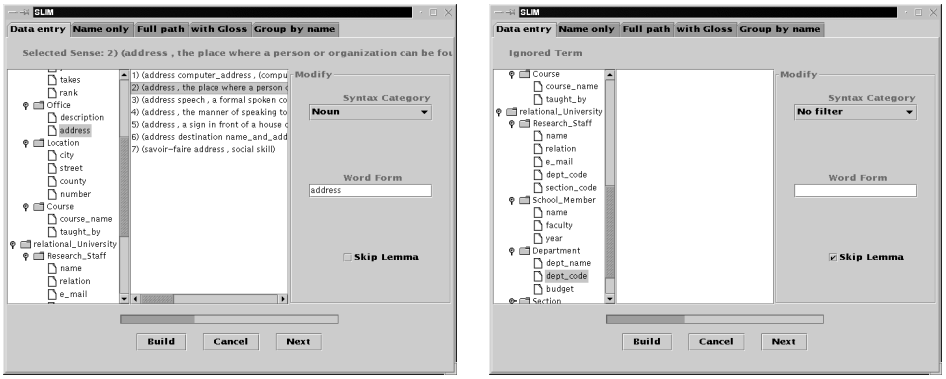


Windows nativo

Figura 4.1: Prototipo di SLIM

A sinistra è presente una lista ad albero sui sorgenti da cui scegliere i termini. Al centro la lista dei significati del termine selezionato. A destra un pannello

con i comandi per imporre un filtro per categoria sintattica (Figura 4.2.a); per cambiare la forma base; per ignorare il termine (Figura 4.2.b). In alto è riportato il significato selezionato ed in basso sono situati i pulsanti per uscire, creare le relazioni e passare al termine successivo.



a) filtro per categoria b) ignorare un termine

Figura 4.2: Funzioni del prototipo di SLIM

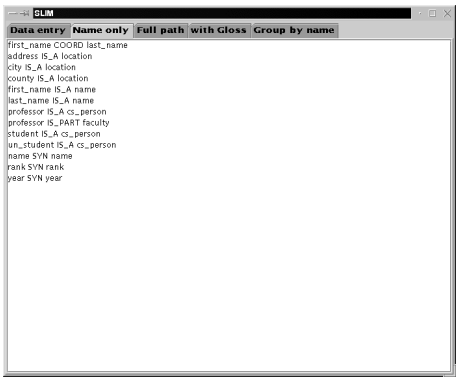
Tramite le linguette in alto si visualizzano i pannelli dei risultati (Figura 4.2):

1. visualizzando solo il termine e non tutto la *dot path* che crea confusione, e raggruppando le relazioni per termini uguali.

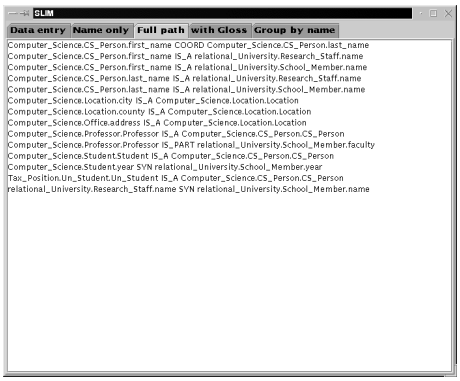
2. completa di *dot path*.

3. con in più anche le glosse dei termini delle relazione per controllarne la veridicità.

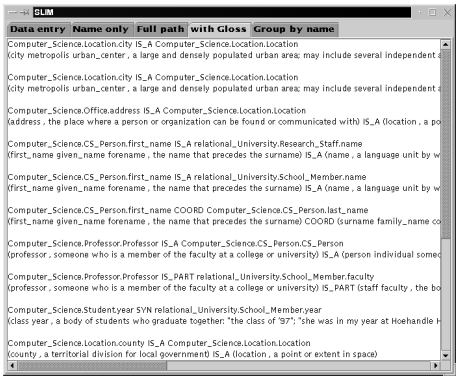
4. per ogni *dot path* vengono elencate le relazioni in cui partecipa, in modo da controllare più facilmente se i risultati ottenuti sono quelli previsti.



1) solo i nome



2) con *dot path*

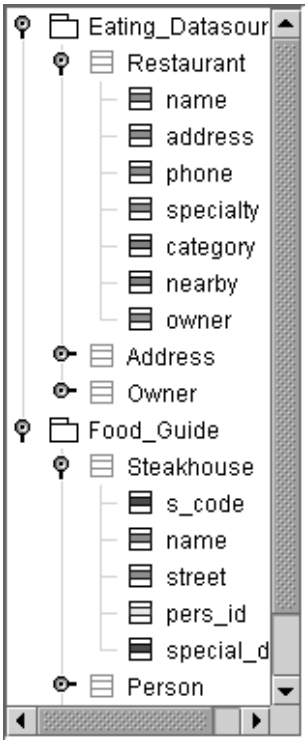


3) con glosse

4) elenco per termine

Figura 4.3: Output del prototipo di SLIM

4.4 Versione finale



Lo spazio occupato da SLIM è stato ridotto a quello di un `JTree`, ed è un oggetto che incapsula i path ed i loro significati, permettendone la modifica da parte dell'utente attraverso il `JTree`, ma anche da parte di altri componenti tramite metodi setter, e può essere usato come *repository* per i termini disambiguati da prelevare con i metodi getter e registrandosi listener delle sue modifiche. I bug che c'erano prima non ci sono più e sono state aggiunte diverse funzionalità: selezione multipla, significati "parenti", cache, proxy, diagramma degli iperonimi.

4.4.1 Meno spazio occupato

In questo modo per essere integrato non c'è bisogno di un `JTabbePane` o di aprire nuovi `Frame`. Prima la pagina di data entry di SLIM (Figura 4.1) occupava l'intero schermo quindi non era possibile per l'utente vedere contemporaneamente SLIM e altri componenti grafici che vanno aggiornati in risposta alle modifiche si SLIM, o che ne sono causa.

Visione d'insieme: SlimTree integrato

In Figura 4.4 si può osservare SlimTree integrato dentro a SI-Designer. Le linguette in alto sono gestite da SI-Designer e permettono al progettista di passare da una fase alla successiva. Più in basso c'è la barra dei pulsanti di SLIM: raccoglie i pulsanti per salvare e ripristinare in locale i dati privati di SLIM, per calcolare le relazioni. La parte centrale è divisa tra SlimTree (a sinistra) e la tabella che contiene le relazioni estratte; da questa il progettista può cancellarne o aggiungerne.

Il flusso dei dati va da sinistra a destra: premendo il pulsante "Build" vengono estratti i termini disambiguati da SlimTree, calcolate le relazioni ed inserite nella tabella a destra.

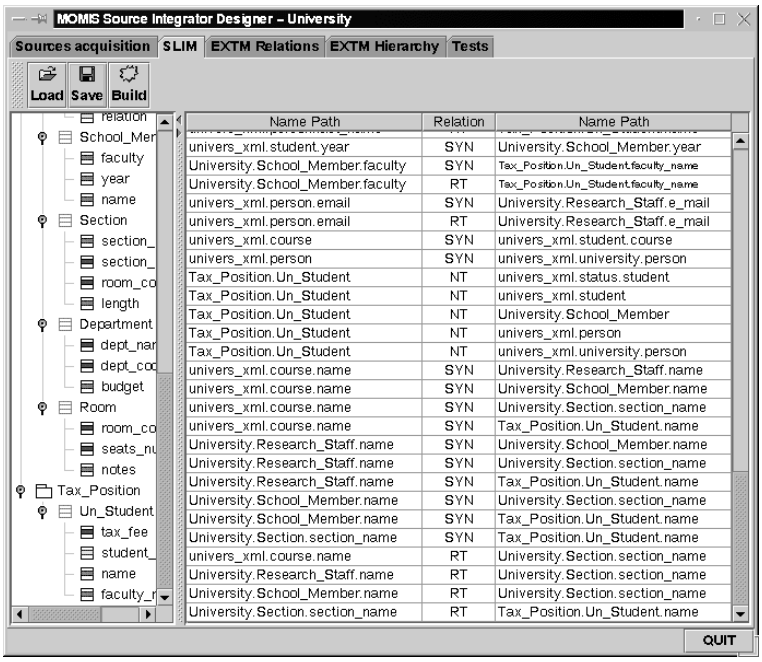


Figura 4.4: Visione d'insieme: SlimTree integrato

4.4.2 Significato delle icone dei rami e delle foglie di SlimTree

Icone		Colori	
	Sorgente		Sensi non selezionati
	Interfaccia		Forma base non trovata
	Attributo		Termine ignorato
			Sensi selezionati

4.4.3 Input tramite menù contestuale generato dinamicamente

Tutte le funzioni di input sono accessibili tramite un menù contestuale generato dinamicamente, in modo da occupare spazio solo quando necessario (vedi Figura 4.5).

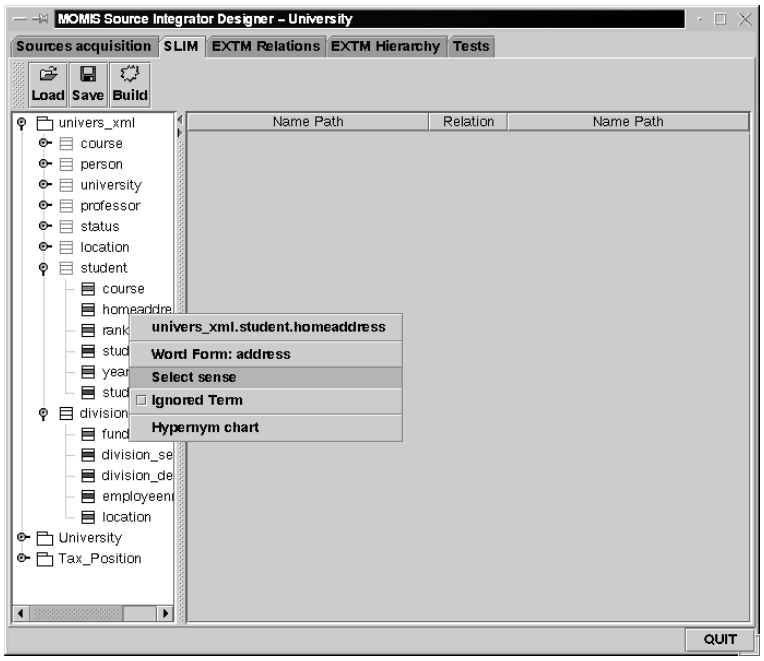
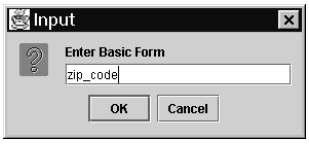


Figura 4.5: Menù contestuale

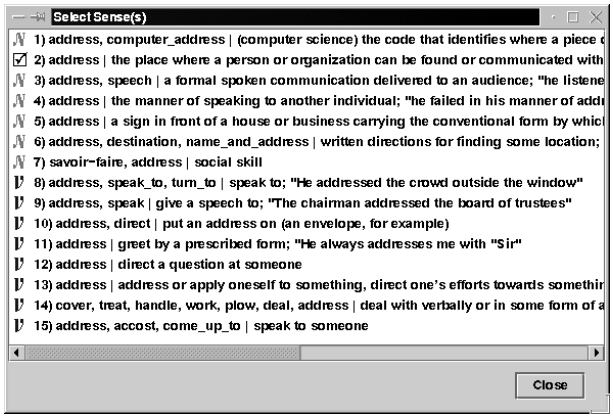
Forma Base

Per cambiare la forma base si preme Word Form: address e viene aperta una InputDialog.



Selezionare un significato

Selezionare da il menù Select sense ed appare la dialog con i significati. Cliccare sui significati per far apparire il segno di spunta. Dopo la selezione il menù Selected sense: 2 conferma la selezione.



Categoria sintattica

Un'icona indica la categoria sintattica senza bisogno di una ComboBox per fare la selezione.

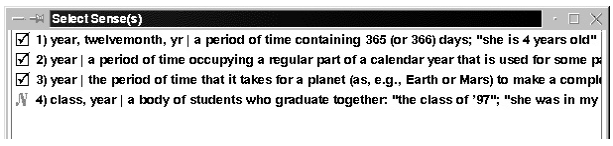
N	Nome
V	Verbo
Aj	Aggettivo
Av	Avverbio

Selezionare più di un significato

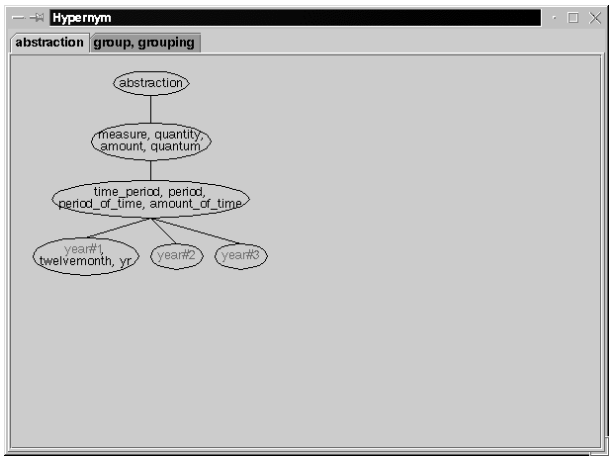
Selected sense: 2 Nella dialog si possono selezionare più significati. Se si seleziona un significato già scelto viene tolto dai selezionati: il funzionamento è del tipo set/reset.

Espansione dei significati ai significati affini

Se si seleziona un significato vengono in automatico selezionati tutti i significati "parenti".



Come si vede dal diagramma degli iperonimi, il significato 1, 2 e 3 sono *sisters*.



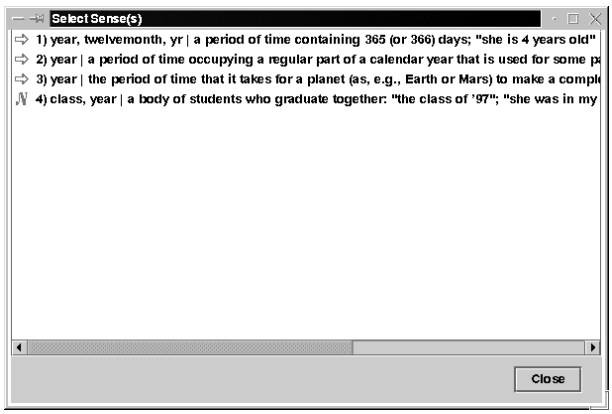
Ignorare un senso

Tramite un checkbox 

si può ignorare un termine perché fuorviante o troppo generale.

“Cache” delle scelte effettuate precedentemente

Vengono memorizzate le scelte fatte per riproporle successivamente. Un’icona del colore della categoria sintattica mostra i significati in cache.
La cache propone solo: sta al progettista accettare o meno i suggerimenti selezionando i significati nel modo usuale.



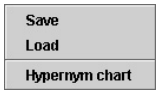
4.4.4 Diagrammi degli iperonimi

Non c’è stato bisogno di implementare un algoritmo incrementale per visualizzare gli iperonimi. L’albero viene ricalcolato al volo a richiesta, infatti, grazie alla *proxy* il tempo impiegato è di 0..2 secondi per far apparire il frame con i diagrammi contro quasi un minuto che il programma mGrafico impiega per generare lo stesso risultato come una pagina html. Oltre alla proxy, c’è anche il vantaggio che i diagrammi vengono disegnati solo per la zona che viene effettivamente visualizzata.

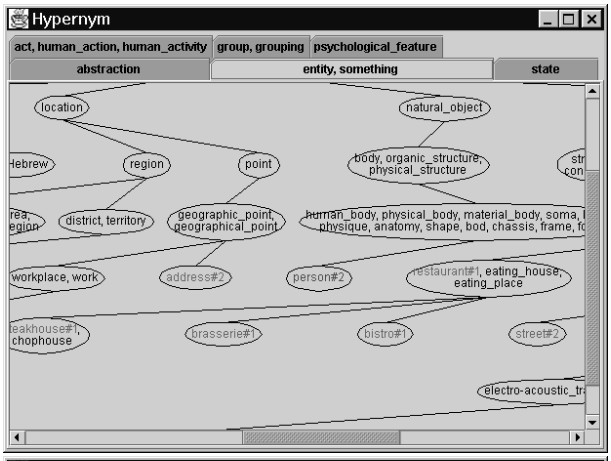
Come strategia di ridisegno si utilizza il ricalcolo dinamico della zona invalidata, invece che creare una immagine in memoria e visualizzarne solo la parte in finestra.

La seconda strategia permette uno scroll molto più fluido, però con il ricalcolo si risparmia memoria; infatti si possono aprire contemporaneamente 168 diagrammi di iperonimi globali (dell'esempio di riferimento Ristoranti con 64MB di heap) prima di andare in `OutOfMemoryError`.

Lo scroll non è fluido e si percepisce un ritardo negli spostamenti, ma è totalmente accettabile.

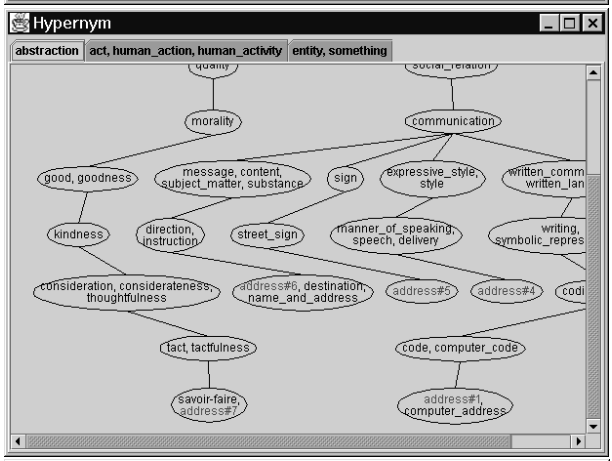


Accedendo dal menù contestuale di SlimTree si ottiene il diagramma degli iperonimi a partire da tutti i significati presenti in SlimTree.

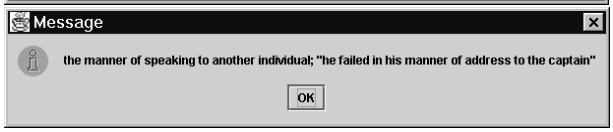


Mentre dal menù 

della foglia, solo quelli del termine.



Clickando su un *synset* appare la glossa



4.4.5 Proxy per velocizzare le chiamate a CORBA_WordNet

SlimTree si pone come *proxy* di CORBA_WordNet: istanzia e dispone il server, nasconde la serializzazione presentando i risultati in modo nativo e non come stringhe. Mantiene una copia delle ultime interrogazioni in modo da accelerarne

un successivo accesso.
Le funzioni sono le stesse di CORBA_WordNet:

```
/**
 * trovaSensi (=search for meanings) method finds the meanings related
 * to a particular word. Morphogical expansion is performed to retrieve
 * basic word form(s).
 * @param word A word or collocation (e.g.: set_up); ignorecase,
 * decline or conjugate
 * @return Meanings Vector of wn2slim.Meaning:
 * <ul><li>String formaBase;</li>
 * <li>Synset synset;</li>
 * <li>String glossa;</li>
 * </ul>
*/
public java.util.Vector trovaSensi(java.lang.String word)

/**
 * trovaParenti (= search for correlated senses) calculates the sense's
 * expansion to his near senses.
 * @param formaBase normalized form
 * @param synset as pos + synset_offset: 2 integer not a serialized class
 * @return Synsets a Vector of Synset
 */
public java.util.Vector trovaParenti(java.lang.String formaBase
                                  , int synset_offset, int pos)

/**
 * creaThes: Builds lexical relation to be inserted into thesaurus,
 * if ok for designer,<br>
 * @param namePathTaged String represents serialized form of a Vector
 * of wn2slim.NodeTag:
 * <ul><li>NamePath namePath;</li>
 * <li>String formaBase;</li>
 * <li>Synset synset;</li>
 * </ul>
 * @return Thesaurus String represents serialized form of a Vector
 * of wn2slim.ThesaurusEntry:
 * <ul><li>NamePath primo, secondo;</li>
 * <li>String relazione;</li>
 * <li>String gloss1, gloss2;</li>
 * </ul>
*/
public java.util.Vector creaThes(java.util.Vector vNamePathTaged)
```

```
/**
 * trovaHashObject (=search for hypernym tree), only in 'noun' syntax
 * category<br>
 * For any meaning of any basic form of the word returns all his
 * hypernyms<br>
 * @param word A word or collocation (es: set_up); ignorecase,
 * decline or conjugate
 * @return hypernyms String represents serialized form of a Vector
 * of Vector of wn2slim.HashObject:
 * <ul><li>public String[] term; //array of the synset's words</li>
 * <li>public String synset;</li>
 * <li>public int occurence; //always = 1</li>
 * <li>public String gloss; //with also semanticfield</li>
 * <li>public String semanticfield; //if no semanticfield
 * then semanticfield=null</li>
 * <li>public Vector hyper; //if no hyper then hyper.isEmpty()=true</li>
 * <li>public Vector hypon; //if no hypon then hypon.isEmpty()=true</li>
 * </ul>One Vector for any (word form/meaning) retrieved, if param word
 * is not found return a empty Vector.
*/
public java.util.Vector trovaHashObject(java.lang.String word)
```

SlimTree offre in più funzioni simili di più alto livello (vedere Sezione 4.4.6).

4.4.6 Funzione getter

```
/**
 * getRelations: build lexical relation to be inserted into thesaurus,
 * if ok for designer,<br>
 * The input are only the disambigued sense in SlimTree
 * @return Thesaurus Vector of wn2slim.ThesaurusEntry:
 * <ul><li>NamePath primo, secondo;</li>
 * <li>String relazione;</li>
 * <li>String gloss1, gloss2;</li>
 * </ul>
 */
public java.util.Vector getRelations()

/**
 * NodeTag of disambigued senses in SlimTree
 * @return namePathTaged String Vector of wn2slim.NodeTag:
 * <ul><li>NamePath namePath;</li>
```

```
 * <li>String formaBase;</li>
 * <li>Synset synset;</li>
 * </ul>
 */
public java.util.Vector getNodeTags()

/**
 * @return NamePaths The dot notation of all the node of SlimTree
 */
public java.util.Vector getNamePaths()

/**
 * Returns only the namepath whose morphy has not found the basic form
 * @return NamePaths The dot notation of the node not found
 */
public java.util.Vector getNotFound()
```

4.4.7 Funzioni setter

```
/**
 * Set (reset) a sense (tag) for a qualified path (NamePath)<br>
 * if sense is selected is reset<br>
 * Important! Call a <istanceSlimTree>.getJTree().repaint() to UI update
 * @param NodeTag tag(basic form,Synset)+NamePath
 * @return success True if NamePath present in SlimTree
 */
public boolean setNodeTag(NodeTag nt)

/**
 * Set (reset) a sense (tag) for a qualified path (NamePath)<br>
 * if sense is selected is reset<br>
 * Important! Call a <istanceSlimTree>.getJTree().repaint() to UI update
 * @param NamePath to be set/reset
 * @param senseNum sense number
 * @return success True if NamePath present in SlimTree
 */
public boolean setNodeTag(NamePath np, int senseNum)
```

4.4.8 ActionEvent source

```
public void addActionListener(java.awt.event.ActionListener l)
public void removeActionListener(java.awt.event.ActionListener l)
```

Capitolo 5

L’organizzazione del software

In questo capitolo viene illustrata la struttura del software realizzato durante lo svolgimento della presente tesi.

In breve i macro componenti sviluppati:

- Libreria per interrogare WordNet: come descritto alla Sezione 3.2, è stato necessario codificare delle funzioni (*parser*) per estrarre le informazioni dai file dati o indice di WordNet, che sono in formato ASCII; e funzioni di più alto livello come calcolare la chiusura transitiva di una relazione.
- Server CORBA: la libreria è stata incapsulata in un oggetto CORBA che ne permette un utilizzo distribuito.
- Dati CORBA: sono le classi condivise tra SLIM e la libreria per scambiarsi i dati tramite servizi CORBA.
- SLIM: *front-end* grafico per l’interazione con il progettista.
- Diagrammi: JavaBean per visualizzare i diagrammi degli iperonimi; richiamabile dalla linea di comando: genera una pagina *html*.

I sorgenti sono disponibili sul server `sparc20.dsi.unimo.it` presso il dipartimento di Scienze dell’Ingegneria, nella directory comune dedicata allo sviluppo del prototipo del progetto MOMIS:
`/export/home/progetti.comuni/Momis/prototype`

Più in particolare i componenti realizzati in questa tesi sono disposti:

componente	directory
Libreria	`modules/WordNet/mWordNet`
Dati CORBA	`shared/wn2slim`
Server CORBA	`modules/WordNet`
SLIM	`modules/SIDesigner/SLIM`
Diagrammi	`modules/WordNet/mGrafico`

La documentazione in formato `html` generata tramite l'utility `JavaDoc` è disponibile presso l'indirizzo:
`http://sparc20.dsi.unimo.it/Momis/prototipo/docsOnline/sources/`

componente	l. tot	l. comm.	l. codice	% comm.	% costo
Libreria	2421	98	1966	5%	93.24%
Dati CORBA	455	43	351	12%	6.72%
Server CORBA	532	199	264	75%	0%
SLIM	2589	333	2037	16%	0%
Diagrammi	1232	84	1071	8%	0%
Totale	7229	757	5689	13%	100%

Legenda:

componente	package o macro funzionalità
l. comm.	conteggio del numero delle linee che contengono unicamente commenti. Per ottenere queste importanti statistiche è stato realizzato appositamente un piccolo programma.
l. codice	conteggio del numero delle linee che contengono codice.
l. tot	numero di linee totali calcolate come con `wc -l`. È la somma delle linee con commenti, di quelle con codice e di quelle vuote.
% comm.	Percentuale commenti/codice.
% costo	Tempo impiegato per eseguire il calcolo delle relazione in modalità *batch* normalizzato sull'unità.

5.1 Libreria per interrogare WordNet

In Figura 5.1 è riportata la gerarchia delle classi. La radice (`mObject`) fornisce servizi per la gestione del file di *log* e degli errori. La interfaccia (`mConst`) permette di condividere tutte le constanti. Le altre classi sono divise (da sinistra verso destra) in classi che offrono funzioni ad alto livello (algoritmo), classi che si occupano di formattare l'output, classi che incapsulano i 6 tipi di file, classi che si occupano del parsing degli stessi, e classi che contengono le informazioni estratte.

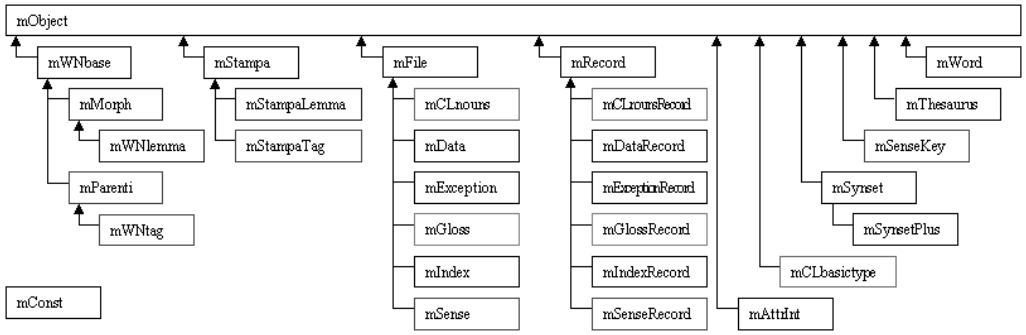


Figura 5.1: Mappa delle classi

Oppure dividendo per macro moduli funzionali:

Nome	l. tot	l. comm.	l. codice	% comm.	% costo
Kernel	454	13	383	3%	8.20%
Core	363	24	267	9%	62.33%
Strutture Dati	66	15	40	37%	0.49%
Sense	180	1	150	1%	1.18%
Gloss	77	1	62	1%	0%
CoreLex	242	7	202	3%	0%
Morph	114	5	92	5%	0%
Parenti	161	18	129	14%	1.03%
Algoritmo	392	14	328	4%	11.51%
Formattazione Output	372	17	313	5%	8.50%
Totale	2421	98	1966	5%	93.24%

5.1.1 Kernel

Interfacce, classi astratte o comunque al top della gerarchia di specializzazione.

Nome	linee	% commenti	% costo	Descrizione
mConst	128	2%	0%	Permette di condividere tutte le costanti.
mFile	212	2%	5.98%	Incapsula un `RandomAccessFile` e offre servizi di alto livello mantenendo privato il file descriptor.
mObject	114	7%	2.22%	Classe base del package, offre la gestione degli errori e del logfile.
mRecord	10	20%	0%	Classe base per i record.
Totale	454	3%	8.20%	

Per quanto riguarda i risultati del *profiler*, appare curioso che il conteggio del tempo trascorso dentro alla classe `mFile` sia solo del 6%. Questa classe è infatti l'unica che utilizza le funzioni di input dal disco, le classi che la specializzano non hanno accesso ai `file descriptor` e passano solo attraverso le sue funzioni per questi compiti; infatti al massimo incidono per lo 0,44% del totale (`mData`). Il tempo è speso, come è prevedibile, dentro alle funzioni `BinSearch` (2%) e `readLine` (3%).

Il 2% di `mObject` è giustificato dal fatto che fornisce funzioni di giustificazione dell'output alle classi che lo estendono.

5.1.2 Core

Classi obbligatorie; sono necessarie per interrogare i file di dati, quelli che contengono i synset, e i file di indice che da una forma base trovano i puntatori ai synset in cui è compreso un suo significato.

Nome	linee	% commenti	% costo	Descrizione
mData	35	0%	0.44%	Incapsula un file di dati, quelli che contengono i synset.
mDataRecord	193	12%	60.45%	Parser del record dei file dati.
mIndex	43	0%	0.35%	Incapsula un file di indice, cioè che permette, dato un lemma, di trovare i synset corrispondenti.
mIndexRecord	92	10%	1.09%	Parser del record dei file indice.
Totale	363	9%	62.33%	

`mDataRecord` utilizza il 60.45% del tempo totale per calcolare le relazioni in modalità *batch* dell'esempio di riferimento dell'Università (profili simili vengono trovati per gli altri due esempi di riferimento).

Non è un dato previsto, infatti la classe `mDataRecord` non chiama nessuna funzione di input/output, ma contiene solo il *parser* per i *synset*.

Evidentemente ci deve essere qualcosa che non va: seguendo lo stack delle chiamate si scopre che il tempo è quasi completamente utilizzato dal *parser* ed all'interno di questo dalla classe di libreria `StringTokenizer`; una classe che restituisce uno per volta gli elementi sintattici riconosciuti in una stringa. Per finire si scopre che quasi il 60% del tempo totale è utilizzato dal metodo `charAt` della classe `String`. Le stringhe in Java sono oggetti, a differenza di altri linguaggi di programmazione in cui sono vettori di byte, quindi per accedere ad un carattere bisogna passare attraverso un metodo d'interfaccia, appunto `charAt(i)` che restituisce l'i-esimo carattere della stringa; questo è reso necessario dal fatto che non è pubblica l'organizzazione interna dei dati, e poi per il fatto che i caratteri sono codificati in *Unicode* a 16 bit (vedi http://www.unicode.org).

Le prestazioni non sono un elemento critico per un prototipo, e nel caso particolare sono comunque del tutto accettabili; è certo però che lavorare per ottimizzare la classe `mDataRecord` può dare ottimi risultati.

Bisognerebbe fare dei tentativi: controllare il codice, che è pubblico (http://java.sun.com), della classe `StringTokenizer` per vedere se si può fare qualcosa: può essere che la Sun abbia pensato questa classe per usi meno *pesanti* di quelli assegnati da questo lavoro. Oppure trasformare le stringhe in `byteArray`. È certo che il modo per ottenere un sicuro successo è di chiamare un metodo nativo scritto in linguaggio C, però non è percorribile, perché si perderebbe il beneficio di utilizzare un linguaggio multipiattaforma.

5.1.3 Strutture Dati

Record delle strutture dati usate solo internamente dalla libreria; i record condivisi con SLIM sono nel `package wn2slim`, vedi Sezione 5.2.

Nome	linee	% commenti	% costo	Descrizione
mpSynsetPlus	43	48%	0.49%	Puntatore ad un synset con il tipo di puntatore (iponimo, iperonimo, ...). È nella forma: pointer_symbol synset_offset pos source/target. source/target si riferiscono alle parole all'interno dei *synset* se la relazione è lessicale.
mWord	23	20%	0%	Parola: lemma + lex_id. Meglio usare un mpSynset per prestazioni, o un mSenseKey per compatibilità futura.
Totale	66	37%	0.49%	

5.1.4 Sense

Chiave di tipo sense_key. Un tipo di chiave immutabile con la versione di WordNet. Infatti bisogna pagare l'accesso nell'indice incapsulato dalla classe `mSense` per trovare il `synset_offset` corrispondente.

Nome	linee	% commenti	% costo	Descrizione
mSense	30	0%	0.64%	Incapsula il file `sense` che mappa i sense_key sui synset_offset.
mSenseKey	100	1%	0.34%	Chiave per indicare un synset immutabile con le versioni di WordNet. Contiene `mSenseKey.lexSense`: Codice del significato.
mSenseRecord	50	0%	0.20%	Parser del record del file `sense`.
Totale	180	1%	1.18%	

5.1.5 Gloss

Indice delle citazioni nelle glosse: lista di lemmi con associato una lista di *synset* nei quali il lemma è citato nelle glosse.

Nome	linee	% commenti	% costo	Descrizione
mGloss	29	0%	0%	Incapsula `gloss`, il file contiene i riferimenti alle parole citate nelle glosse.
mGlossRecord	48	2%	0%	Parser del record dei file delle glosse.
Totale	77	1%	0%	

5.1.6 CoreLex

Vedi: http://www.dfki.de/ paulb/corelex.html
Un *related work* di WordNet. Una via tentata, ma poi abbandonata.

Nome	linee	% commenti	% costo	Descrizione
mCLbasictype	181	4%	0%	Incapsula un Hashtable che mappa i tipi CoreLex sulla loro descrizione. Vengono letti sia `corelex_nouns.basictypes.synset` che `corelex_nouns.classes`.
mCLnouns	23	4%	0%	Incapsula `corelex_nouns`.
mCLnounsRecord	38	0%	0%	Parser del record del file `corelex_nouns`
Totale	242	3%	0%	

5.1.7 Morph

Processore morfologico.
WordNet propone un algoritmo per trovare una forma base, il file delle eccezioni si riferisce ai casi in cui l'algoritmo sbaglia.

Nome	linee	% commenti	% costo	Descrizione
mException	43	0%	0%	Incapsula il file di eccezioni usato da morphy.
mExceptionRecord	53	13%	0%	Parser del record dei file `.exc`.
mMorph	18	0%	0%	Preprocessore morfologico.
Totale	114	5%	0%	

Il processore morfologico non è stato al momento terminato.

5.1.8 Parenti

Permette allargare un concetto anche ai significati affini.

Nome	linee	% commenti	% costo	Descrizione
mParenti	161	14%	1.03%	Calcola i sensi parenti: cugini, sorelle, gemelli.
Totale	161	14%	1.03%	

5.1.9 Algoritmo

Classi che contengono gli algoritmi per calcolare le relazioni lessicali.

Nome	linee	% commenti	% costo	Descrizione
mWNbase	237	6%	10.82%	Apre i file e li nasconde, offrendo funzioni di alto livello.
mWNlemma	119	3%	0%	Funzioni per confrontare tutti i significati dei lemmi.
mWNtag	36	0%	0.69%	Funzioni per confrontare gli attributi.
Totale	392	4%	11.51%	

5.1.10 Formattazione Output

Classi `main` per la modalità *batch*.
Uso:

```
java mWordNet\mStampaLemma [1|2|3]
```

Stampa le relazioni a partire da ogni significato del lemma.

```
java mWordNet\mStampaTag [1|2|3]
```

Stampa le relazioni a partire solo dal significato deciso precedentemente. `1|2|3` numero dell'esempio di riferimento da elaborare.

```
java mWordNet\mStampaTag/$CreaTag
```

Crea in modo interattivo i *tag* (forma base+lexsense).

Nome	linee	% commenti	% costo	Descrizione
mStampa	85	0%	3.76%	Funzioni per formattare l'output.
mStampaLemma	100	10%	0%	`main`, parsing della linea di comando e usa un `mWNlemma`.
mStampaTag	187	4%	4.74%	`main`, parsing della linea di comando e usa un `mWNtag`. Contiene `mStampaTag.CreaTag`: programma interattivo che crea un tag a partire da un lemma e dal suo significato.
Totale	372	5%	8.50%	

5.2 Dati CORBA

Classi dati condivise da SLIM e dal server `CORBA_WordNet`, per passarsi informazioni tramite i servizi CORBA. I dati da elaborare ed i risultati sono trasformati tramite serializzazione in una stringa e poi codificati a base64.

Nome	linee	% commenti	% costo	Descrizione
HashObject	193	20%	0%	Chiusura transitiva degli iperonimi, serve per calcolare l'albero degli iperonimi.
Meaning	17	27%	0%	*Holder* per informazioni da trasferire tramite CORBA: forma base, *synset* e glossa
NamePath	98	9%	4.60%	Incapsula una *dot path*: `sourceName.interfaceName.attributeName`.
NodeTag	18	36%	0%	NamePath+forma base+*synset*.
Synset	58	0%	1.38%	Puntatore ad un *synset*.
ThesaurusEntry	71	0%	0.74%	Un elemento del Thesaurus: contiene due NamePath, le loro glosse e la relazione trovata.
Totale	455	12%	6.72%	

5.3 Server CORBA

Il server istanzia `mWNtag` e `mWNlemma` all'occorrenza.

Nome	linee	% commenti	Descrizione
CORBA_WordNetImplBase	278	50%	Server che offre i servizi.
WordNetFactoryImplBase	254	117%	Crea un'istanza di `CORBA_WordNetImplBase` per ogni client.
Totale	532	75%	

L'interfaccia di `CORBA_WordNetImplBase` è già stata riportata nella Sezione 4.4.5.

5.4 SLIM

Si veda la documentazione completa al Capitolo 4.

Nome	linee	% commenti	Descrizione
SLIM	271	50%	Pannello contenitore.
SlimGrafico	747	9%	Genera e visualizza il diagramma degli iperonimi.
SlimNode	320	7%	Elemento dell'albero di `SlimTree`.
SlimTree	698	28%	Albero per il data entry, e repository dei dati.
ThesaurusAddDialog	308	5%	Finestra di dialogo per l'inserimento di nuove relazioni.
ThesaurusTable	245	11%	Tabella che contiene le relazioni estratte e permette l'inserimento/cancellazione.
Totale	2589	16%	

5.5 Diagrammi

Nome	linee	% commenti	Descrizione
mGrafico	1232	8%	Genera dalla linea di comando i diagrammi degli iperonimi in formato `html`.
Totale	1232	8%	

Come si nota dai parametri che si possono impostare, si può generare la chiusura transitiva di qualunque tipo di relazione semantica. Il programma è stato

pensato per gli iperonimi, quindi con altri tipi di relazioni i risultati possono non essere ottimali. Per esempio a calcolare gli iponimi di uno qualunque degli esempi di riferimento si ottiene un grafico da circa 10.000 *synset*, pari a un'immagine da 759 megapixel cioè una pagina `html` da 37 m^2 a 72 dpi (circa 500 pagine in stampa a corpo 4), ed occorre un'ora e venti per ottenere il risultato.

In più c'è il problema degli autoanelli o dei cicli. Cioè il caso in cui un *synset* sia in relazione con sè stesso, per esempio una parte di una disciplina scientifica è una disciplina scientifica, o un'università è formata da università. Questi esempi sono discutibili, infatti, utilizzando il programma d'interrogazione fornito con WordNet, questo segnala `''WordNet internal error: cyclical path detected''`.

Per testare il javaBean in modo accurato gli è stato fatto calcolare l'albero degli iperonimi a partire da tutti i *synset* in WordNet: per i nomi non ci sono percorsi ciclici, nei verbi ce ne sono 11, e 4 nei meronimi dei nomi. Come recovery da questo errore viene adottata una politica molto semplice: viene ignorato il *synset*, per cui schemi che comprendano *synset* fonte di percorsi ciclici possono venire graficamente non accurati.

```
Uso: mGrafico [opzioni] (formabase[#numsenso])+|ospedale|universita|ristora
Esempio: java -jar mGrafico.jar -d=tmp patient doctor#1 nurse

-f=nome Nome del file .htm e radice per le .gif [Default="Ipernimi"]
-d=dir  Directory in cui crea i file
-p=ptr  Tipo ricerca: ipenimi|olonimi|meronimi|iponimi|CHARPTR
-na     Se viene indicato il sensenumber ignora gli altri significati
-ni     Non ignorare i synset che non hanno puntatori del tipo richiesto
-t=tipo Tipo del font: SansSerif|Serif|Dialog|DialogInput|Monospaced
-s=stl  Stile del font: PLAIN|BOLD|ITALIC
-c=n    Dimesione del font
-mx=n   Pixel lasciati bianchi ai margini destro e sinistro della pagina
-my=n   Pixel lasciati bianchi ai margini sopra e sotto della pagina
-bx=n   Distanza dei bordi destro e sinistro dell'ellisse dal testo
-by=n   Distanza dei bordi sopra e sotto dell'ellisse dal testo
-dx=n   Distanza orizzontale minima in pixel tra 2 ellissi
-dy=n   Distanza verticale minima in pixel tra 2 ellissi
-px=n   Dimensione orizzontale della pagina:
        esempio: A4 landscape -> (12"-1" per i bordi) * 300dpi = 3300pixel
-py=n   Minimo spazio occupato verticamente dalla pagina
-72g    Preimposta per monitor, caratteri grandi (default)
-72p    Preimposta per monitor, caratteri piccoli
-300    Preimposta per stampante inkjet (300dpi, A4 landscape)
-600    Preimposta per stampante laser (600dpi, A4 landscape)
```

La maggior parte dei paramentri servono solo per ottenere una formattazione più adatta all’uso che se ne prevede: visualizzare a video o stampare.

La dimensione orizzontale della pagina può non essere rispettata, infatti la macchina virtuale Java ha uno *heap* limitato (impostabile tramite il parametro `-Xmx`, 64 MB il default), quindi in caso di `OutOfMemoryError`, come recovery, il programma dimezza la dimensione della banda da allocare. Comunque ai nomi di file `gif` viene aggiunto un pedice progressivo per permettere la ricostruzione delle pagine.

Note conclusive

Questa tesi è stata sviluppata nell'ambito del progetto MOMIS ed ha avuto come obiettivo il progetto e l'implementazione del componente che realizza l'estrazione di relazioni lessicali tra sorgenti dati eterogenee per facilitare l'integrazione di sorgenti di informazione.

Partendo da una precedente tesi [5] in cui era stata ipotizzata l'interazione tra SI-Designer e WordNet, il lavoro di questa tesi è cominciato con uno studio di fattibilità che ha espresso un risultato positivo sull'uso di WordNet, a condizione di creare una libreria a basso livello per interrogare WordNet. Questa libreria si è resa necessaria per ottimizzare i tempi di risposta e per integrare più facilmente le funzioni di WordNet all'interno di un ambiente scritto in un linguaggio multipiattaforma (Java).

È stato poi realizzato l'algoritmo per estrarre le relazioni lessicali, da cui è emerso che i risultati calcolati a partire da tutti i significati in WordNet di ogni termine non davano buoni risultati; per cui è stata scartata l'idea di creare un tool totalmente automatico, e si è resa necessaria l'interazione con il progettista all'interno di SI-Designer; per questo compito è stato creato SLIM.

In seguito gli sforzi si sono diretti verso una maggior integrazione di SLIM all'interno di SI-Designer, la cui implementazione si è sviluppata in parallelo con il lavoro della presente tesi. Dunque la libreria è stata trasformata in un oggetto CORBA *stand alone* e SLIM in un pannello di SI-Designer. Nel frattempo SLIM è stato riscritto, ottimizzandolo per le esigente che via via erano emerse, e già pensato per utilizzi futuri.

Infine è stato realizzato un JavaBean per visualizzare il diagramma degli iperonimi: era emersa, provando in prima persona ad utilizzare il modulo d'integrazione, la necessità di una visione più naturale, dunque visiva, della gerarchia di specializzazione dei significati.

Per esprimere i risultati ottenuti dalla presente tesi mi affido alla lusinghiera conclusione dell'articolo accettato e presentato all'Ottavo Convegno Nazionale su Sistemi Evoluti per Basi di Dati - {SEBD2000}, L'Aquila, 26-28 giugno 2000, autori D. Beneventano, S. Bergamaschi, A. Corni, R. Guidetti e G. Malvezzi dal titolo "SI-Designer: un tool di ausilio all'integrazione intelligente di sorgenti di informazione"[26]:

> *Un'interazione diretta con WordNet senza il supporto di SLIM è improponibile come strumento di effettivo ausilio al progettista nella fase di integrazione di schemi a causa del numero eccessivo delle relazioni proposte da WordNet per ogni termine e dal fatto che solo una minima parte di queste relazioni è consistente con le sorgenti oggetto dell'analisi.*
>
> *L'interazione con il progettista realizzata attraverso SLIM è allo stato attuale di buon livello e costituisce quindi un valido strumento per l'attività di integrazione di sorgenti.*

Ricordo inoltre che il lavoro della tesi è parte dell'articolo accettato alla Conferenza Internazionale sulle Basi di Dati, Very Large DataBase {VLDB2000}, Cairo (Egitto), 10-14 settembre 2000, autori D. Beneventano, S. Bergamaschi, S. Castano, A. Corni, R. Guidetti, G. Malvezzi, M. Melchiori e M. Vincini dal titolo "Information Integration: the MOMIS Project Demonstration".

Appendice A

Statistiche su WordNet

In questa appendice sono riportati per intero i dati utilizzati per la costruzione di grafici su WordNet nell'arco della tesi. Questi dati sono raggruppati in appendice per non appesantire la trattazione e distrarre dal filo del discorso, ma per correttezza e per riscontrabilità devono essere presentati.

L'origine dati è la versione di WordNet1.6, data 27 febbraio 1998.

Per il conteggio delle statistiche sono stati creati brevi programmi in Java per l'occasione.

A.1 Conteggio e percentuale di forme base e *synset* per categoria sintattica

Categoria	Forma Base		*synset*	
Nomi	94503	72%	66025	66%
Verbi	10348	8%	12127	12%
Aggettivi	20199	16%	17915	18%
Avverbi	4575	4%	3575	4%
Totale	129625	100%	99642	100%

Tabella A.1: Conteggio e percentuale di forme base e *synset* per categoria sintattica

A.2 Distribuzione del numero di parole contenute nei *synset*

Categoria	Parole	Synset	Parole/Synset
Nomi	116364	66025	1,7624233
Verbi	22073	12127	1,8201534
Aggettivi	29892	17915	1,8689367
Avverbi	5679	3575	1,5885315
Totale	174008	99642	1,7463318

Tabella A.2: Numero di lemmi per *synset*

Cnt	N	V	Aj	Av	Tot	Cnt	N	V	Aj	Av	Tot
1	33926	7032	10962	2250	54170	15	3	2	0	0	5
2	21214	2782	4335	851	29182	16	0	2	1	0	3
3	6640	1181	1451	302	9574	17	0	0	0	0	0
4	2551	539	611	96	3797	18	1	2	0	0	3
5	973	270	259	49	1551	19	0	0	0	0	0
6	406	139	141	10	696	20	0	0	1	0	1
7	170	78	62	10	320	21	0	0	0	0	0
8	70	45	43	2	160	22	0	0	0	0	0
9	28	22	24	4	78	23	0	2	0	0	2
10	15	11	9	1	36	24	0	0	1	0	1
11	9	9	6	0	24	25	0	0	0	0	0
12	9	6	5	0	20	26	0	0	0	0	0
13	3	5	4	0	12	27	2	0	0	0	2
14	5	0	0	0	5						

Tabella A.3: Distribuzione del numero di parole contenute nei *synset* per categoria sintattica

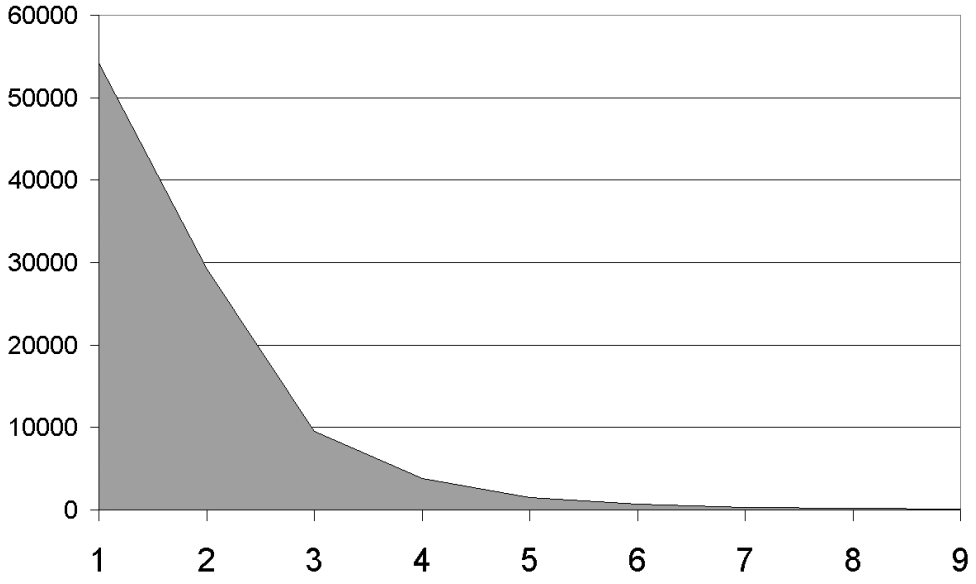


Figura A.1: Distribuzione del numero di parole contenute nei *synset*, totale

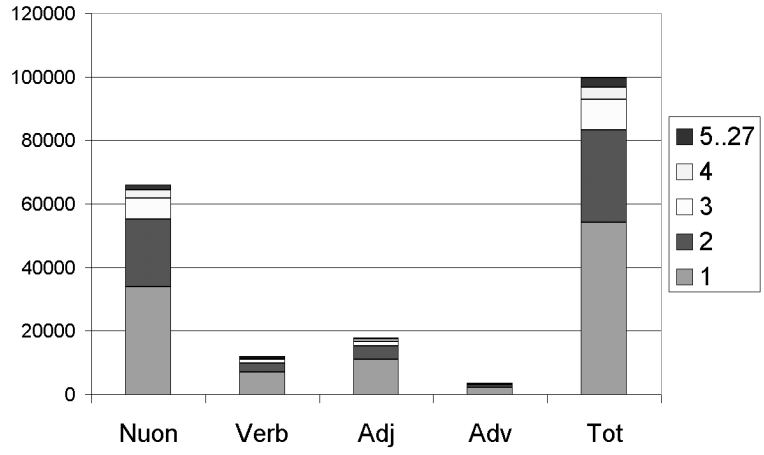


Figura A.2: Distribuzione del numero di parole contenute nei *synset* per categoria sintattica

A.3 Conteggio relazioni

Tipo	Nomi	Verbi	Aggettivi	avverbi
ANTPTR	1849	1031	4108	720
HYPERPTR	66910	11536	-	-
HYPOPTR	66910	11536	-	-
ENTAILPTR	-	427	-	-
SIMPTR	-	-	21901	-
ISMEMBERPTR	11849	-	-	-
ISSTUFFPTR	709	-	-	-
ISPARTPTR	6883	-	-	-
HASMEMBERPTR	11849	-	-	-
HASSTUFFPTR	709	-	-	-
HASPARTPTR	6883	-	-	-
MERONYM	-	-	-	-
HOLONYM	-	-	-	-
CAUSETO	-	224	-	-
PPLPTR	-	-	90	-
SEEALSO	-	631	2712	-
PERTPTR	-	-	4021	3141
ATTRIBUTE	650	-	650	-
VERBGROUP	-	523	-	-

Tabella A.4: Conteggio delle relazioni

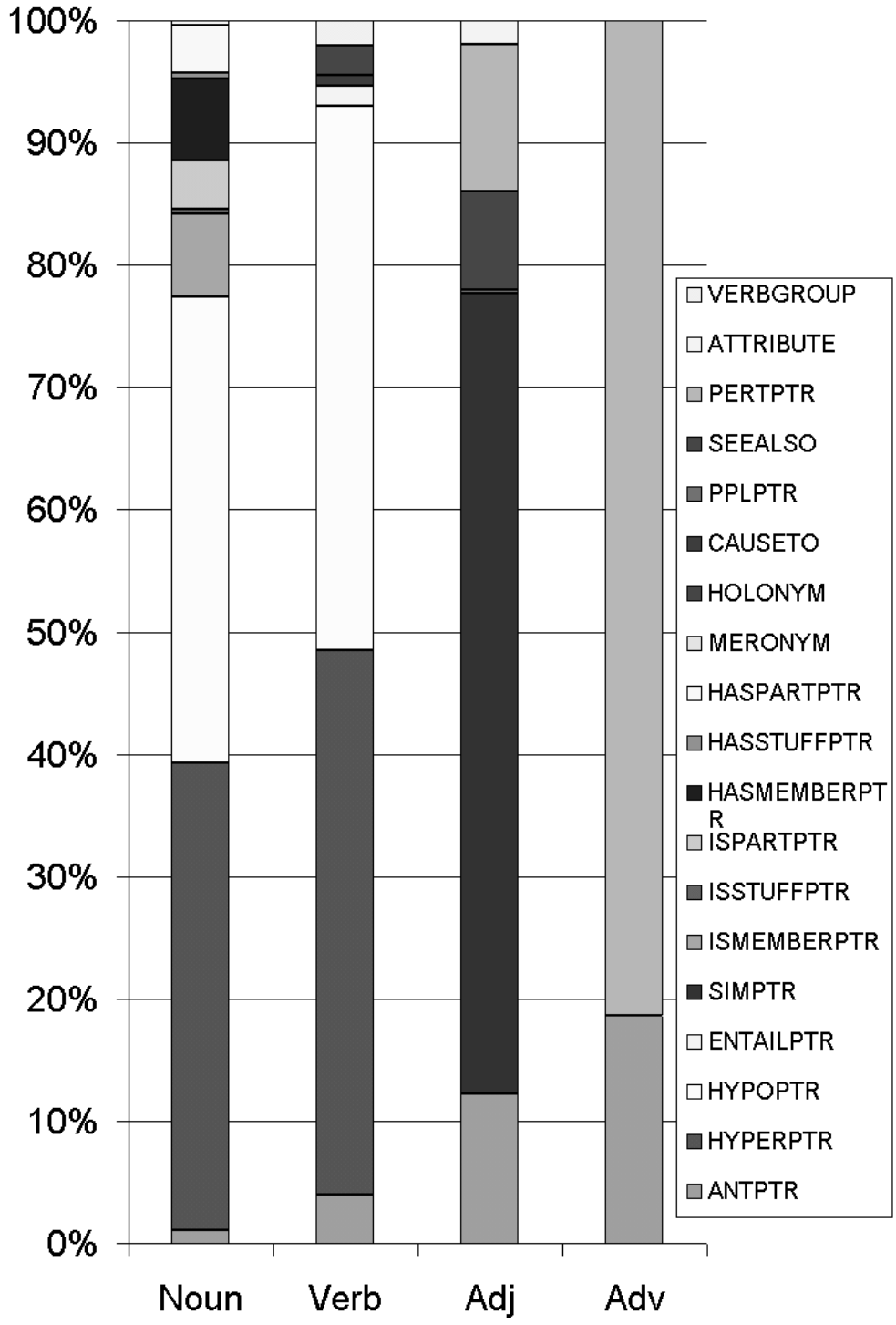


Figura A.3: Percentuale dei tipi delle relazioni

A.4 Conteggio dei *synset* per file lessicografico

Num	Nome	Contenuto	Tot.
0	adj.all	all adjective clusters	14734
1	adj.pert	relational adjectives (pertainyms)	3099
2	adv.all	all adverbs	3575
3	noun.Tops	unique beginners for nouns	35
4	noun.act	nouns denoting acts or actions	5372
5	noun.animal	nouns denoting animals	7294
6	noun.artifact	nouns denoting man-made objects	9810
7	noun.attribute	nouns denoting attributes of people and objects	2633
8	noun.body	nouns denoting body parts	1592
9	noun.cognition	nouns denoting cognitive processes and contents	2260
10	noun.communication	nouns denoting communicative processes and contents	4547
11	noun.event	nouns denoting natural events	850
12	noun.feeling	nouns denoting feelings and emotions	393
13	noun.food	nouns denoting foods and drinks	2377
14	noun.group	nouns denoting groupings of people or objects	1831
15	noun.location	nouns denoting spatial position	2123
16	noun.motive	nouns denoting goals	40
17	noun.object	nouns denoting natural objects (not man-made)	1050
18	noun.person	nouns denoting people	6409
19	noun.phenomenon	nouns denoting natural phenomena	523
20	noun.plant	nouns denoting plants	7872

Figura A.4: Conteggio dei *synset* per file lessicografico (1 di 2)

Num	Nome	Contenuto	Tot.
21	noun.possession	nouns denoting possession and transfer of possession	907
22	noun.process	nouns denoting natural processes	521
23	noun.quantity	nouns denoting quantities and units of measure	1104
24	noun.relation	nouns denoting relations between people or things or ideas	369
25	noun.shape	nouns denoting two and three dimensional shapes	299
26	noun.state	nouns denoting stable states of affairs	2549
27	noun.substance	nouns denoting substances	2391
28	noun.time	nouns denoting time and temporal relations	874
29	verb.body	verbs of grooming, dressing and bodily care	495
30	verb.change	verbs of size, temperature change, intensifying, etc.	2006
31	verb.cognition	verbs of thinking, judging, analyzing, doubting	635
32	verb.communication	verbs of telling, asking, ordering, singing	1388
33	verb.competition	verbs of fighting, athletic activities	411
34	verb.consumption	verbs of eating and drinking	229
35	verb.contact	verbs of touching, hitting, tying, digging	1953
36	verb.creation	verbs of sewing, baking, painting, performing	606
37	verb.emotion	verbs of feeling	303
38	verb.motion	verbs of walking, flying, swimming	1247
39	verb.perception	verbs of seeing, hearing, feeling	410
40	verb.possession	verbs of buying, selling, owning	688
41	verb.social	verbs of political and social activities and events	1007
42	verb.stative	verbs of being, having, spatial relations	671
43	verb.weather	verbs of raining, snowing, thawing, thundering	78
44	adj.ppl	participial adjectives	82

Figura A.5: Conteggio dei *synset* per file lessicografico (2 di 2)

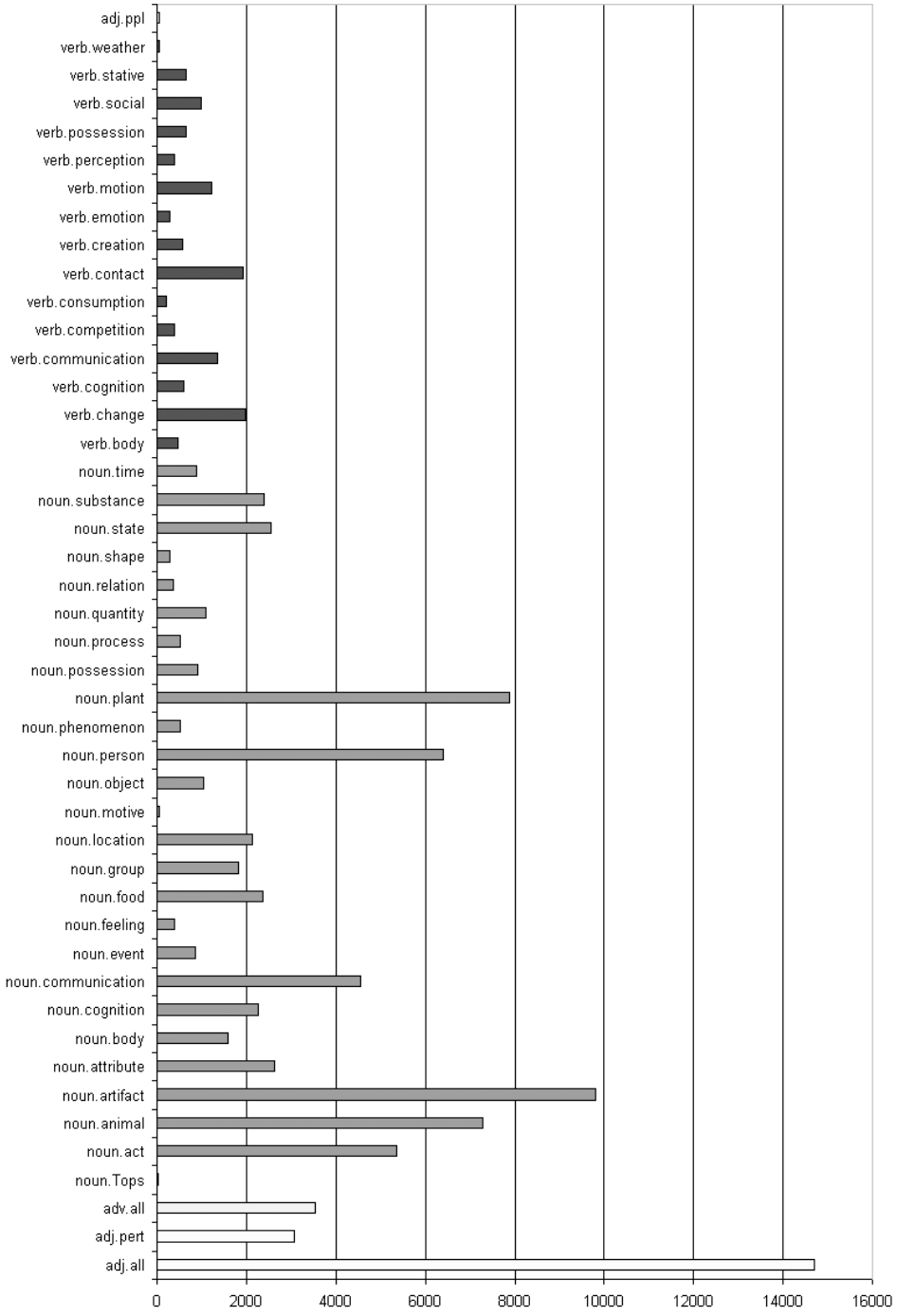


Figura A.6: Conteggio dei *synset* per file lessicografico

Appendice B

Glossario WordNet

In questa appendice sono riportati i termini usati da WordNet con la spiegazione. Durante la trattazione della tesi si è sempre cercato di utilizzare la traduzione in italiano del termine invece che l'originale inglese; nel glossario appaiono in tutte e due le lingue, per chiarezza.

adjective cluster Un gruppo di *synset* di aggettivi organizzati intorno a coppie o triplette di antinomi. Un cluster di aggettivi contiene due o più *synset* principali (*head synset*) che rappresentano i concetti opposti. Ogni *synset* principale ha uno o più *synset* satellite.

attribute Un nome per il quale degli aggettivi esprimono un valore. Esempio il nome `weight` è un *attribute* per il quale `light` e `heavy` esprimono un valore.

base form La forma base di un termine è la parola privata di declinazioni o coniugazioni.

collocation (locuzioni) Una locuzione in WordNet è una stringa di due o più parole connesse da spazi o trattino. Esempio: `man-eating shark`, `blue-collar`, `depend on`, `line of products`. Nei file del database gli spazi sono sostituiti con il carattere di sottolineatura ('_').

coordinate Termini coordinati sono parole che condividono uno stesso iperonimo.

cross-cluster pointer Un puntatore semantico da un cluster di aggettivi ad un altro.

cousin Significati i cui iponimi generano una specifica relazione tra di loro.

direct antonyms Una coppia di parole tra le quali vi è una relazione di antinomia (contrari). Nei cluster degli aggettivi le relazioni di antinomia sono solo tra *synset* origine.

entailment Un verbo X implica (*entail*) Y se X non può essere svolto a meno che Y sia, o sia stato, eseguito.

exception list Trasformazione morfologica per le parole che non sono regolari e non possono essere processate in maniera algoritmica.

group Significati simili per via di relazioni di tipo *cousin*, *sister* o *twin*.

gloss Definizione e/o sentenza d'esempio per un *synset*.

head synset *synset* in un cluster di aggettivi che contiene almeno una parola come diretto antinomo.

holonym Il "tutto" in una relazione di aggregazione. Y è un olonomo di X se X è una parte di Y.

hypernym Il termine sovraordinato in una gerarchia di specializzazione. Y è un iperonimo di X se X è un (*is_a*) Y.

hyponym Il termine sottoordinato in una gerarchia di specializzazione. X è un iponimo di Y se Y è un (*is_a*) X.

indirect antonym Un aggettivo in un *synset* satellite che non ha antinomi diretti, ma mediati attraverso il *synset* principale.

lemma rappresentazione tramite forma scritta o suoni di una parola.

lexical pointer Una relazione tra due singoli lemmi.

monosemous Un lemma che è contenuto in un solo *synset* in una categoria sintattica.

meronym La "parte" in una relazione di aggregazione. X è un olonomo di Y se Y è una parte di X.

part of speech Categoria sintattica: nomi, verbi, aggettivi ed avverbi.

participial adjective Un aggettivo derivato da un verbo.

pertainym Un aggettivo relazionale. Un aggettivo "pertinente" è usualmente definito da una frase come "di o pertinente a" e non ha antinomi. Può puntare ad un nome o ad un altro *pertainym*.

polysemous Un lemma contenuto in più *synset* in una categoria sintattica.

polysemy count Conteggio dei significati di un lemma in una categoria sintattica in WordNet.

postnominal Un aggettivo che si trova unicamente dopo il nome che modifica.

predicative Un aggettivo che può essere usato solo in una posizione predicativa. Se X è un aggettivo predicativo, può essere usato in una frase come "esso è X".

prenominal Un aggettivo che si trova unicamente prima del nome che modifica.

satellite synset Un *synset* in un cluster di aggettivi che rappresenta un concetto che è simile in significato al concetto del suo *synset* principale.

semantic pointer Un puntatore semantico che indica una relazione tra *synset*.

sense Un significato di una parola in WordNet. Ogni senso di una parola è in un *synset* diverso.

sense key Informazione necessaria per trovare un senso in WordNet. Un *sense key* è formato da lemma, informazioni lessicografiche e, per gli aggettivi, informazioni a riguardo del *synset* principale. È chiave univoca ed immutabile con le diverse versioni di WordNet.

sister Un coppia di *synset* che siano iponimi dello stesso diretto sovraordinato.

synset Insieme di sinonimi; un insieme di parole che possono essere scambiate in certi contesti.

troponym Un verbo che esprime una maniera specifica per compiere un'azione di un altro verbo. X è un troponimo di Y se fare X è fare Y in una certa maniera.

twin *synset* che hanno almeno tre parole in comune.

unique beginner Un *synset* dei nomi che non ha iperonimi.

Appendice C

Esempi di riferimento in ODL$_{I^3}$

Quella che segue è la descrizione in linguaggio ODL$_{I^3}$ degli esempi a cui si è fatto riferimento in questa tesi.

C.1 Università

C.1.1 `Computer_Science`

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

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

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

}
 union status1
{
 attribute professor professor;

};

interface student
(source semistructured Computer_Science
extent student
key (student_studentid)
candidate_key cK0 (student.tutor))
{
 attribute string student_studentid;
 attribute string student_tutor;
 attribute string year;
 attribute set<course> course;
 attribute string homeaddress;
 attribute string rank;
};

interface professor
(source semistructured
Computer_Science
extent professor
key (professor_prof_code))
```

```
{
 attribute string
        professor_prof_code;
 attribute string
        professor_office_phone;
 attribute string ptitle;
 attribute division division;
 attribute string rank;
};

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

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

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

C.1.2 University

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

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

interface Department
 ( source relational University
   extent Department
   key (dept_code) )
{       attribute string dept_name;
        attribute integer dept_code;
        attribute integer budget;
};

interface Section
( source relational University
   extent Section
```

```
   key (section_code)
   foreign_key ( room_code )
references Room )
{       attribute string
            section_name;
        attribute integer
            section_code;
        attribute integer length;
        attribute integer room_code;
};

interface Room
 ( source relational University
   extent Room
   key (room_code) )
{       attribute integer
            room_code;
        attribute integer
            seats_number;
        attribute string notes;
};
```

C.1.3 Tax_Position

```
interface Un_Student
 ( source file Tax_Position
   extent University_Student
   key (student_code) )
{       attribute string name;
        attribute integer
            student_code;
        attribute string
            faculty_name;
        attribute integer tax_fee;
};
```

C.2 Ospedale

C.2.1 Cardiology_Department

```
interface Patient
 ( source semistructured
      Cardiology_Department )
{ attribute string         name;
  attribute string         address;
  attribute set<Exam>      exam*;
  attribute integer        room;
  attribute integer        bed;
  attribute string         therapy*;
  attribute set<Physician> physician*;
};

interface Nurse
 ( source semistructured
      Cardiology_Department )
{ attribute string       name;
  attribute string       address;
  attribute integer      level;
  attribute set<Patient> patient;
};

interface Physician
 ( source semistructured
      Cardiology_Department )
{  attribute string  name;
   attribute string  address;
   attribute integer phone;
   attribute string specialization;
};

interface Exam
( source semistructured
     Cardiology_Department )
{  attribute integer date;
   attribute string  type;
   attribute string  outcome;
};
```

C.2.2 Intensive_Care

```
interface Patient
( source relational Intensive_Care
  extent Patients
  key  code
  foreign_key(test)
```

```
            references Test(number)
  foreign_key(doctor_id)
            references Doctor (id))
{ attribute string  code;
  attribute string  first_name;
  attribute string  last_name;
  attribute string  address;
  attribute string  test;
  attribute string doctor_id;
};

interface Test
( source relational Intensive_Care
  extent Tests
  key number )
{ attribute integer number;
  attribute string  type;
  attribute integer date;
  attribute string  laboratory;
  attribute string  result;
};

interface Doctor
( source relational Intensive_Care
  extent Medical_Staffers
  key id )
{ attribute string  id;
  attribute string  first_name;
  attribute string  last_name;
  attribute integer phone;
  attribute string  address;
  attribute string  availability;
  attribute string  position;
};

interface Dis_Patient
  ( source relational Intensive_Care
    extent Dis_Patients
    key code
    foreign_key(code)
    references Patient )
{ attribute string  code;
  attribute integer date;
  attribute string  note;
};
```

C.3 Ristoranti

C.3.1 Eating_Datasource

```
interface Restaurant
( source semistructured Eating_Datasource )
{ attribute string name;
    attribute Address
address;
attribute integer phone*;
attribute
set<string>specialty;
attribute string category;
attribute Restaurant
nearby*;
attribute Owner owner*;
};

interface Address
( source semistructured Eating_Datasource )
{ attribute string city;
attribute string street;
attribute string zipcode;
};

interface Owner
( source semistructured Eating_Datasource )
{ attribute string name;
attribute string address;
attribute string job;
};
```

C.3.2 Food_Guide

```
interface Steakhouse
( source relational Food_Guide
  key s_code
  foreign_key(pers_id) references Person )
{ attribute string s_code;
```

```
attribute string name;
attribute string street;
attribute integer pers_id;
attribute string
special_dish;
};

interface Person
( source relational Food_Guide
  key pers_id )
{ attribute integer pers_id;
attribute string first_name;
attribute string last_name;
attribute string
qualification;
};

interface Bistro
( source relational Food_Guide
  key s_code
  foreign_key(s_code) references Steakhouse
  foreign_key(pers_id) references Person )
{ attribute string s_code;
attribute set<string>type;
attribute integer pers_id;
};

interface Brasserie
( source relational Food_Guide
  key b_code )
{ attribute string b_code;
attribute string name;
attribute string address;
};
```

Appendice D

Tecnologia JavaBeans

D.1 Esempio introduttivo

In Figura D.1 ed in Figura D.2 viene mostrato un esempio di programmazione in un ambiente JavaBeans.

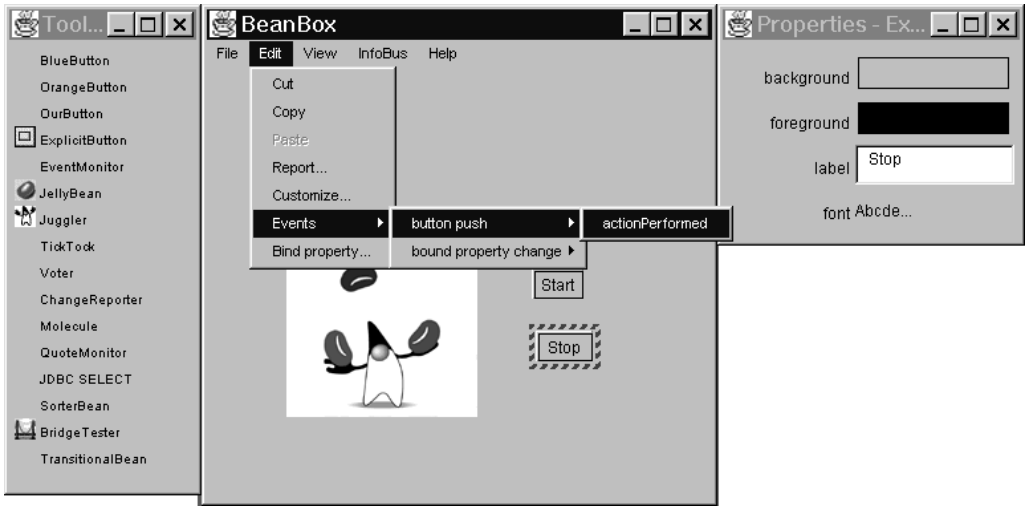


Figura D.1: Visione d'insieme dell'ambiente BDK

L'area di lavoro è il BeanBox: un pannello che mostra come si presenterà l'applicazione su cui si sta lavorando.

A sinistra si vede il ToolBox da cui si trascinano i componenti (JavaBeans) sul BeanBox per creare l'applicazione. Nell'esempio sono stati inseriti alcuni Bean: 2 `ExplicitButton` (un JavaBeans che incapsula un normale Button AWT) ed

un `Juggler` (un'animazione che mostra la mascotte di Java che gioca con dei chicchi di caffè).

A destra c'è la finestra delle proprietà con cui si può modificare l'aspetto o le funzionalità del JavaBean selezionato. In Figura è selezionato un `ExplicitButton` di cui si può modificare il colore del background e foreground, il font e l'etichetta.

Figura D.2: Collegamento di un evento

Per creare la logica dell'applicazione, o più in particolare per collegare un evento scatenato da un Bean ad un metodo (che implica un'azione) di un altro Bean, si seleziona dal menù Edit⟶Events⟶button push⟶actionPerformed (Figura D.1); appare una linea rossa che segue la punta del mouse; clickando, nell'esempio sul *Juggler*, appare una dialog (Figura D.2) da cui scegliere il metodo che deve essere attivato in risposta all'evento. In questo caso viene scelto il metodo che inizia l'animazione in risposta alla pressione (`actionPerformed`) del tasto "start".

D.2 Definizione

> Un JavaBean è un componente software riutilizzabile che può essere modificato visualmente con un tool di sviluppo.

L'esempio introduttivo è quello che più classicamente fa venire in mente la parola *Javabeans*. Mentre invece JavaBeans è più una metodologia ed un insieme di convenzioni che una libreria di funzioni.

Infatti, per sgomberare alcuni preconcetti, bisogna specificare che:

- Un JavaBean non è necessariamente visuale; ci possono essere dei JavaBean che non hanno interazione, per esempio un timer, un data factory, . . .

- Il linguaggio Java non è stato modificato per la tecnologia JavaBeans, cioè non c'è sintassi o grammatica finalizzata per i JavaBeans.

- Non è necessario estendere nessuna classe o implementare interfacce per creare un Bean.

- Non è neppure necessario importare i package (`java.beans`, `java.beans.beancontext`) per i Bean più semplici.

- Non è necessario il *BDK* (Bean Development Kit) o altri tool visuali. Infatti il BDK non contiene le librerie per i Bean o i JavaDoc che sono già nel JDK, ma solo 16 Demo commentati e il BeanBox.[1]

D.3 Obblighi

```
class name: any
superclass: any
contructor: Must have a no-argument contructor,
            or a serialized template file
packaging: JAR file manifest entry specifies Java-Bean: True
```

Le uniche imposizioni, dunque, sono:

- aver un costruttore senza parametri o la presenza di un file che contenga la versione serializzata del Bean. In sintesi questo permette ad un tool di sviluppo visuale di sapere come istanziare il Bean: o trova la versione serializzata di cui, dunque, è già stato chiamato il costruttore, oppure chiama il costruttore senza parametri.

- e che il Bean sia impacchettato in modo *furbo*. Il tool di sviluppo deve essere in grado di capire quali sono i Bean tra tutte le classi incluse in un file JAR (Java ARchive).

[1] Il BeanBox è un reference tool di sviluppo visuale, cioè dovrebbe servire per chi crea ambienti di sviluppo come esempio e non proprio per la sua funzione.

D.4 Convenzioni

```
Properties (property p of type T)
   getter: public T getP()
   setter: public void setP(T value)
Methods
   method name: any
   method args: any; some tools only recognize no-argument methods
Events (event name E)
   event class name: EEvent
   listener name: EListener
   listener methods: public void methodname(EEvent e)
   listener registration: public void addEListener(EListener l)
   listener removal: public void removeEListener(EListener l)
```

Queste convezioni permettono ad un tool di sviluppo visuale di sapere tutte le informazioni sul Bean, in modo che:

- tramite i getter/setter riempie la finestra delle proprietà del Bean; eventualmente possono essere di sola lettura o sola scrittura, se manca uno dei due.
- tramite i metodi, ed in particolare, per i tool più semplici come il BeanBox, quelli senza parametri, perché con i parametri non saprebbe come gestirli, riempie la lista dei *target* possibili che possono essere chiamati dagli eventi generati da altri Bean.
- tramite gli eventi riempie la lista degli eventi che il Bean può generare.

D.5 Introspezione

L'introspezione è già nativa in Java (tramite il package `java.lang.reflect`), ed è un metodo standard per esporre proprietà, metodi ed eventi a *run-time*.
I Bean nell'area di lavoro del tool di sviluppo visuale sono in esecuzione: vengono istanziati, e per esempio, tramite i metodi setter, vengono loro cambiate delle proprietà in tempo reale.

Un tool di sviluppo visuale per conoscere le proprietà, metodi ed eventi di un Bean ha due possibilità:

- tramite l'introspezione, se trova dei nomi di metodi/attributi che utilizzano le convenzioni.

- oppure tramite un oggetto BeanInfo. Una classe a parte rispetto al Bean con il solo scopo di descrivere le caratteristiche del Bean.

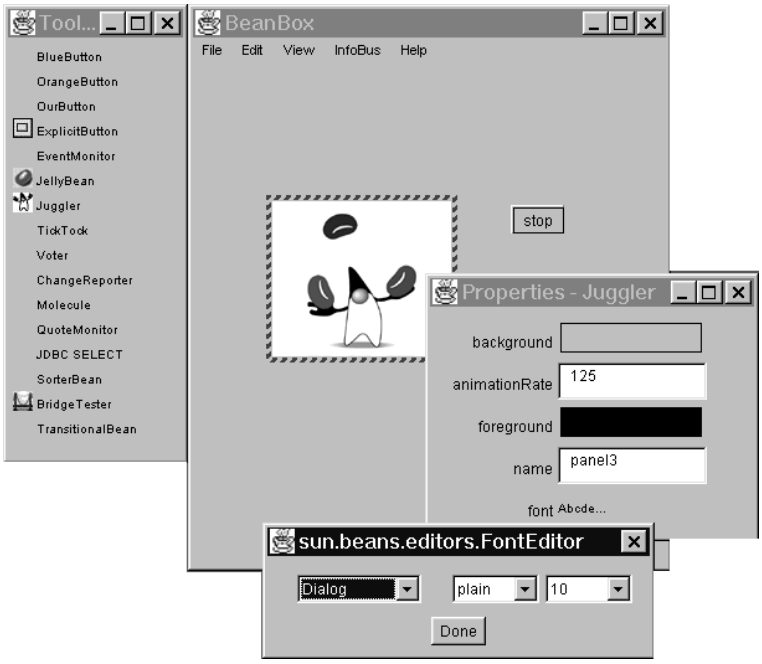


Figura D.3: Editor delle proprietà

D.6 Personalizzazione

Per un Bean si può fornire anche un editor delle proprietà da usare in fase di disegno; può essere utile per proprietà complicate oppure nell'ottica del riuso. In Figura D.4 un esempio.

D.7 Persistenza

Deve prendere parte al meccanismo di persistenza del suo contenitore.
Invece di creare un Bean con la `new`, si può usare il metodo statico `java.bean.Bean.instantiate()` che lo crea a partire da un file `.ser` con lo stesso nome del Bean (lo cerca nel percorso di `CLASSPATH` quindi anche all'interno del file JAR), se non lo trova usa il costruttore senza parametri.

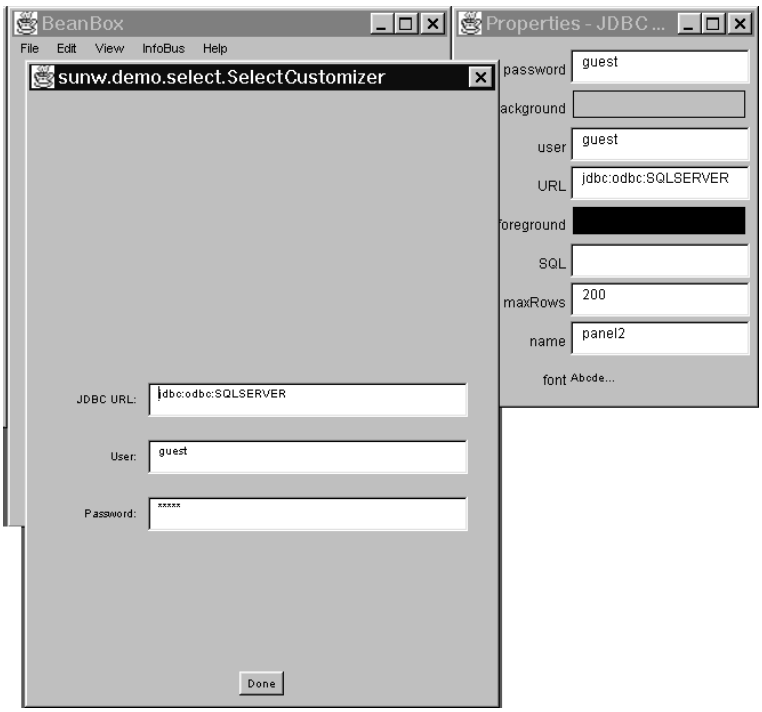


Figura D.4: Editor delle proprietà personalizzato

D.8 Significato della tecnologia JavaBeans

Cioè perché esistono i JavaBeans?

D.8.1 Premessa

In Java un oggetto descrive la sua interfaccia grafica chiamando delle funzioni (di solito) nel suo costruttore per:

- creare componenti: `mioBottone = new JButton("Bottone");`
- modificare le proprietà: `mioBottone.setFont(new Font("Serif", Font.BOLD, 14));`
- assegnargli una posizione: `mioPanel.add(mioBottone);`

Questo solo per descrivere la rappresentazione grafica, poi ci sarà dell'altro codice per la semantica.

D.8.2 Non è necessario un costruttore

Per istanziare un oggetto si usa:

```
SLIM mioSLIM = new SLIM(args, "CORBA_GlobalSchema");
```

Per un JavaBean invece si può anche usare:

```
SLIM mioSLIM=null;
try {
  mioSLIM = (SLIM)java.beans.Beans.instantiate(null, "SLIM");
} catch (Exception e) { }
```

Quindi ci sono 3 modi per costruire un oggetto SLIM:

- Costruttore con parametri:

```
public SLIM(String[] as, String CORBA_GSN) {
  args = as;
  CORBA_GSName = CORBA_GSN;
  initCORBA(); //inizializza oggetti CORBA
  initComponets(); //crea componenti grafici
}
```

- Costruttore senza parametri (usa valori di default):

```
public SLIM() {
  args = new String[] {"-ORBInitialHost", "sparc20.dsi.unimo.it"
, "-ORBInitialPort", "1050"};
  CORBA_GSName = "CORBA_GlobalSchema";
  initCORBA();
  initComponets();
}
```

- Funzione di lettura usata dalla deserializzazione:

```
private void readObject(java.io.ObjectInputStream in)
throws IOException, ClassNotFoundException {
  in.defaultReadObject();
  initCORBA(); //gli oggetti CORBA
               // si considerano transienti e vanno riottenuti
}
```

D.8.3 Non è più necessaria la funzione che crea gli oggetti grafici

`args` e `CORBA_GSName` sono ripristinati come erano stati impostati precedentemente. Da notare che in quest'ultimo caso non viene chiamata la funzione per creare i componenti grafici perché vengono ricostruiti dalla deserializzazione così come erano stati salvati. Dunque, visto che `instantiate` cerca prima il file `.ser`, dopo che si è fornito questo file, non è più necessaria la funzione che crea gli oggetti grafici.

Il BeanBox lavora in questo modo: permette di creare visivamente i componenti, poi li serializza. In questo modo mantengono tutte le proprietà e la posizione che avevano.

In pratica questo è un utilizzo improprio della persistenza. Persistenza significa salvare lo stato dell'oggetto, per esempio a SLIM basterebbe salvare i significati scelti, perché le altre informazioni possono essere ricostruite. In questo caso invece, il fatto importante è salvare i componenti grafici invece di ricrearli.

Ci sono molte analogie con Visual Basic (o simili); anche lì ci sono due componenti: un tipo di file per il codice, ed uno per le form, cioè per la descrizione dell'interfaccia. Quest'ultimo file si può modificare solo visualmente, i componenti grafici hanno le stesse caratteristiche che a run-time, le modifiche alle proprietà avvengono interattivamente e quando si salva viene serializzato.

I JavaBean sono talmente uguali a Visual Basic che tramite un ActiveX Bribge si può inserire un JavaBean in una form di Visual Basic perché il meccanismo di serializzazione è compatibile con quello (trasparente) di Visual Basic.

Quindi il vantaggio dei JavaBean è che possono essere utilizzati in tool di sviluppo visuale che salvano il lavoro tramite la serializzazione. Tool che sollevano il programmatore dal dover scrivere del codice per descrivere i componenti visuali.

Questo vantaggio può essere ottenuto, però, anche tramite ambienti di sviluppo che non usano la serializzazione, ma generano il codice al volo (per esempio Forte4j, http://www.netbeans.com). Cioè se, per esempio, viene modificato il font di una `JLabel` nella funzione `initComponets()` viene aggiunta la riga `jlblCurrentTime.setFont(new Font( ...` ; mentre utilizzando la tecnica sopra descritta non verrebbe aggiunto nessun codice, ma cambiata la proprietà font alla `JLabel`, proprietà che fa parte del suo stato e viene salvata, per essere ripristinata uguale.

D.9 JAR

`jar` ha una sintassi molto simile a `tar`, in particolare:

`jar cvf SLIM.jar *.class *.ser WordNetApp.class`	creare archivio
`jar tvf SLIM.jar`	visualizzare contenuto
`jar xvf SLIM.jar`	estrarre

Per segnalare che una classe è un JavaBean bisogna modificare il manifest file ed inserirlo dentro al jar con il parametro m

```
jar xvfm SLIM.jar SLIM.mf *.class *.ser WordNetApp\*.class
```

Il manifest file, in questo caso `SLIM.mf`, contiene solo:

```
Manifest-Version: 1.0


Name: SLIM.class
Java-Bean: True
```

Per le altri classi di ausilio del package non è necessario impostare che non sono JavaBean. Ottenuto `SLIM.jar` non servono altri file per eseguire l'applicazione:

```
java -cp .:SLIM.jar mContenitore
```

Il classpath deve contenere esplicitamente il file jar il quale può contenere una gerarchia di directory con le usuali convenzioni per i nomi.

Bibliografia

[1] Prof Aris M. Ouksel The University of Illinois at Chicago. Introduction to electronic commerce, ciclo di seminari sul commercio elettronico relativi al corso di basi di dati, Anno accademico 1999/2000. e-mail: aris@uic.edu.

[2] MOMIS staff. Momis (mediator environment for multiple information sources) project part of the interdata project. Available at http://sparc20.dsi.unimo.it/Momis.

[3] Andrea Zaccaria. Momis: Il componente query manager. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998. http://sparc20.dsi.unimo.it/tesi/index.html.

[4] Alberta Rabitti. Architettura di un mediatore per un sistema di integrazione di sorgenti distribuite ed autonome. Tesi di laurea, Università di Modena, Facoltà di ingegneria Informatica, 1997-1998. http://sparc20.dsi.unimo.it/tesi/index.html.

[5] Alberto Zanoli. Si-designer, un tool di ausilio all'integrazione di sorgenti di dati eterogenee distribuite: progetto e realizzazione. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998. http://sparc20.dsi.unimo.it/tesi/index.html.

[6] Simone Montanari. Un approccio intelligente all' integrazione di sorgenti eterogenee di informazione. Tesi di laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1996-1997. http://sparc20.dsi.unimo.it/tesi/index.html.

[7] Gianni Pio Grifa. Analisi di affinità strutturali fra classi ODL_{I^3} nel sistema momis. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998. http://sparc20.dsi.unimo.it/tesi/index.html.

[8] Maurizio Vincini. Utilizzo di tecniche di intelligenza artificiale nell'integrazione di sorgenti informative eterogenee. Tesi di dottorato, Università di Modena, Facoltà di Ingegneria, dottorato di ricerca in Ingegneria Informatica, 1997-1998. http://sparc20.dsi.unimo.it/tesi/index.html.

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

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

[11] J. Gilarranz, J. Gonzalo, and F. Verdejo. Using the eurowordnet multilingual semantic database. In *Proc. of AAAI-96 Spring Symposium Cross-Language Text and Speech Retrieval*, 1996.

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

[13] G. A. Miller. Wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4), 1990.

[14] G. A. Miller. Five papers on wordnet. *Cognitive Science Laboratory Report 43*, 1990, 1993. Available from ftp://ftp.cogsci.princeton.edu/wordnet/.

[15] C Fellbaum. Wordnet: An electronic lexical database, 1998.

[16] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Consistency checking in complex object database schemata with integrity constraints. *IEEE Transactions on Knowledge and Data Engineering*, 10:576–598, July/August 1998.

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

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

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

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

[21] Domenico Beneventano, Sonia Bergamaschi, and Claudio Sartori. Semantic query optimization by subsumption in OODB. In H. Christiansen, H. L. Larsen, and T. Andreasen, editors, *Flexible Query Answering Systems*, volume 62 of *Datalogiske Skrifter - ISSN 0109-9799*, Roskilde, Denmark, 1996.

[22] ODB-Tools staff. Odb-tools Project. Available at http://sparc20.dsi.unimo.it.

[23] N.V. Findler, editor. *Associative Networks*. Academic Press, 1979.

[24] Teseo staff. Teseo, an automatic classifier of web documents that exploits the context of links in web documents. Available at http://medialab.di.unipi.it/Project/Arianna/Teseo.

[25] Attardi G., Di Marco S., and Salvi D. Categorization by context. *Journal of Universal Computer Science*, 4(9):719–736, 1998. Available at http://www.iicm.edu/jucs_4_9/categorisation_by_context.

[26] D. Beneventano, S. Bergamaschi, A. Corni, R. Guidetti, and G. Malvezzi. Si-designer: un tool di ausilio all'integrazione intelligente di sorgenti di informazione. In *Ottavo Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBD2000, L'Aquila*, pages 123–137, 2000.