# D2I

**Integrazione, Warehousing e Mining di sorgenti eterogenee**
*Programma di ricerca (cofinanziato dal MURST, esercizio 2000)*


## Use of ontologies and extensional inter-schema properties for integration


Domenico Beneventano, Sonia Bergamaschi, Francesco Guerra, Silvana Castano, Maurizio Vincini



**D1.R2** **30 Aprile 2001**



### Sommario

In this report we detail and extend the method described in the previous project report **D1.R1** [10], evidencing the use of ontologies and extensional inter-schema properties in the integration process.

With respect to project report **D1.R1** we take into account the extensional intra and inter-schema properties by introducing some examples of extensional relationships and integrity constraint rules. Then we show as these properties influence the integration process by allowing, in particular, to infer new relationships and to force the inclusion of a class in a cluster.

From a teoretical point of view, we introduce the syntax and the semantics of the OLCD Description Logic and its inference capabilities. Then we describe how $\mathrm{ODL}_{I^3}$ source schema descriptions are translated into OLCD descriptions. Moreover we present a formalization of the extraction phase of the lexicon-derived inter-schema relationships based on the WordNet database.



| | |
|---|---|
| **Tema** | Tema 1: Applicazioni per basi di dati su Internet e Intranet |
| **Codice** | D1.R2 |
| **Data** | 30 Aprile 2001 |
| **Tipo di prodotto** | Rapporto tecnico |
| **Numero di pagine** | 18 |
| **Unità responsabile** | MO |
| **Unità coinvolte** | MI, MO |
| **Autore da contattare** | Domenico Beneventano<br>Dipartimento di Scienze dell'Ingegneria<br>Università di Modena e Reggio Emilia<br>via Vignolese 905<br>41100 Modena, Italia<br>beneventano.domenico@unimo.it |

# Use of ontologies and extensional inter-schema properties for integration

Domenico Beneventano, Sonia Bergamaschi, Francesco Guerra, Silvana Castano, Maurizio Vincini

30 Aprile 2001

**Abstract**

In this report we detail and extend the method described in the previous project report **D1.R1** [10], evidencing the use of ontologies and extensional inter-schema properties in the integration process.

With respect to project report **D1.R1** we take into account the extensional intra and inter-schema properties by introducing some examples of extensional relationships and integrity constraint rules. Then we show as these properties influence the integration process by allowing, in particular, to infer new relationships and to force the inclusion of a class in a cluster.

From a teoretical point of view, we introduce the syntax and the semantics of the OLCD Description Logic and its inference capabilities. Then we describe how ODL$_{I^3}$ source schema descriptions are translated into OLCD descriptions. Moreover we present a formalization of the extraction phase of the lexicon-derived inter-schema relationships based on the WordNet database.

## 1 Introduction

The report is organized as follows. In section 2.1 we present the ODL$_{I^3}$ language and in section 2.2 we consider the running example introduced in project report **D1.R1** by reporting the complete ODL$_{I^3}$ schemas representation of the `ED` and `FD` sources. In section 2.3 we introduce the syntax and the semantics of the OLCD Description Logics and its inference capabilities. Then in section 2.4 we describe how ODL$_{I^3}$ source schema descriptions are translated into OLCD descriptions.

In section 3 we consider and extend the method introduced in project report **D1.R1** by explaining the use of OLCD in the Common Thesaurus construction with particular reference to extensional knowledge.

In section 4 we describe the theoretical foundations of the techniques to discover affinity inter-schema relationships between ODL$_{I^3}$ classes in different schemas.

## 2 Preliminaries: ODL$_{I^3}$ and OLCD

### 2.1 The ODL$_{I^3}$ language

For a semantically rich representation of source schemas and object patterns associated with information sources to be integrated, we introduce an object-oriented language, called ODL$_{I^3}$. According to recommendations of ODMG and to the diffusion of $I^3$/POB [7, 5], the object data model ODL$_{I^3}$ is very close to the ODL language. ODL$_{I^3}$ is a source independent language used for information extraction to describe heterogeneous schemas of structured and semistructured data sources in a common way.

ODL$_{I^3}$ introduces the following main extensions with respect to ODL:

**Union constructor.** The union constructor, denoted by `union`, is introduced to express alternative data structures in the definition of an $\mathrm{ODL}_{I^3}$ class, thus capturing requirements of semistructured data. An example of its use will be shown in the following.

**Optional constructor.** The optional constructor, denoted by (?), is introduced for class attributes to specify that an attribute is optional for an instance (i.e., it could be not specified in the instance). This constructor too has been introduced to capture requirements of semistructured data. An example of its use will be shown in the following.

**Integrity constraint rules.** This kind of rule is introduced in $\mathrm{ODL}_{I^3}$ in order to express, in a declarative way, *if then* integrity constraint rules at both intra- and inter-source level.

**Intensional relationships.** They are *terminological relationships* expressing inter-schema knowledge for the source schemas. Intensional relationships are defined between classes and attributes, and are specified by considering class/attribute names, called terms. The following relationships can be specified in $\mathrm{ODL}_{I^3}$:

- SYN (Synonym-of), defined between two terms $t_i$ and $t_j$, with $t_i \neq t_j$, that are considered synonyms in every considered source (i.e., $t_i$ and $t_j$ can be indifferently used in every source to denote a certain concept).
- BT (Broader Terms), or hypernymy, defined between two terms $t_i$ and $t_j$ such as $t_i$ has a broader, more general meaning than $t_j$. BT relationship is not symmetric. The opposite of BT is NT (Narrower Terms), or hyponymy.
- RT (Related Terms), or positive association, defined between two terms $t_i$ and $t_j$ that are generally used together in the same context in the considered sources.

An intensional relationships is only a terminological relationship, with no implications on the extension/compatibility of the structure (domain) of the two involved classes (attributes). Consequently, our notion of intensional relationships is different from the one proposed by Catarci and Lenzerini [4], where an intensional relationships has some extensional import.

**Extensional relationships.** Intensional relationships SYN, BT and NT between two classes $C_1$ and $C_2$ may be "strengthened" by establishing that they are also *extensional* relationships. Consequently, the following extensional relationships can be defined in $\mathrm{ODL}_{I^3}$:

- $C_1$ $\mathrm{SYN}_{ext}$ $C_2$: this means that the instances of $C_1$ are the same of $C_2$.
- $C_1$ $\mathrm{BT}_{ext}$ $C_2$: this means that the instances of $C_1$ are a superset of the instances of $C_2$.
- $C_1$ $\mathrm{NT}_{ext}$ $C_2$: this means that the instances of $C_1$ are a subset of the instances of $C_2$.

Moreover, extensional relationships "constrain" the structure of the two classes $C_1$ and $C_2$, that is $C_1$ $\mathrm{NT}_{ext}$ $C_2$ is semantically equivalent to an "isa" relationship. As to summarize:

- an extensional relationship $C_1$ $\mathrm{NT}_{ext}$ $C_2$ is equivalent to an "isa" relationship $C_1$ ISA $C_2$ plus an intensional relationships $C_1$ NT $C_2$;
- an extensional relationship $C_1$ $\mathrm{BT}_{ext}$ $C_2$ is equivalent to an "isa" relationship $C_2$ ISA $C_1$ plus an intensional relationships $C_1$ BT $C_2$;
- an extensional relationship $C_1$ $\mathrm{SYN}_{ext}$ $C_2$ is equivalent to two "isa" relationships $C_1$ ISA $C_2$ and $C_2$ ISA $C_1$ plus an intensional relationships $C_1$ SYN $C_2$.

An "isa" relationships $C_1$ ISA $C_2$ is expressible in ODL$_{I^3}$ by the following integrity constraint rule:

```
rule Rule2 forall X in C1 then X in C2
```

**Mapping Rules.** This kind of rule is introduced in ODL$_{I^3}$ in order to express relationships holding between the integrated ODL$_{I^3}$ schema description of the information sources and the ODL$_{I^3}$ schema description of the original sources.

The extraction process has the goal of translating object patterns and source schemas into ODL$_{I^3}$ descriptions. Translation is performed by a wrapper. Moreover, the wrapper is also responsible for adding the source name and type (e.g., relational, semistructured). The translation into ODL$_{I^3}$, on the basis of the ODL$_{I^3}$ syntax (see Appendix A) and of the schema definition is performed by the wrapper as follows. Given a relation of a relational source or a pattern $\langle l, A\rangle$, translation involves the following steps: i) an ODL$_{I^3}$ class name corresponds to the relation name or to $l$, respectively, and ii) for each relation attribute or label $l' \in A$, an attribute is defined in the corresponding ODL$_{I^3}$ class. Furthermore, attribute domains are extracted. Structure extraction can be performed as proposed in [2, 9].

## 2.2 Running example

In this section we consider the running example introduced in project report **D1.R1**. We consider two sources in the Restaurant Guide domain, storing information about restaurants. The `Eating Source` guidebook (`ED`) is semistructured and contains information about fast foods of the west coast, their menu, quality, and so on. The `Food Guide` Database (`FD`) is a relational database containing information about USA restaurants from a wide variety of publications (e.g., newspaper reviews, regional guidebooks). The schema of this source is composed of four relations, namely, `Restaurant`, `Bistro`, `Person`, and `Brasserie`. Information related to restaurants is maintained into the `Restaurant` relation. `Bistro` instances are a subset of `Restaurant` instances and give information about the small informal restaurants that serve wine. Each `Restaurant` and `Bistro` is managed by a `Person`. Information about places where drinks and snacks are served on are stored in the `Brasserie` relation.

In the following we report the complete ODL$_{I^3}$ schemas representation of the `ED` and `FD` sources.

```
Eating_Source (ED):
interface Fast-Food
( source semistructured
   Eating_Source )
{ attribute string      name;
  attribute Address     address;
  attribute integer     phone?;
  attribute set<string> specialty;
  attribute string      category;
  attribute Restaurant  nearby?;
  attribute integer     midprice?;
  attribute Owner       owner?;};

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

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

```
Food_Guide_Source (FD):
interface Restaurant
( source relational Food_Guide
   key r_code
   foreign_key(pers_id)
        references Person )
{ attribute string      r_code;
  attribute string      name;
  attribute string      street;
  attribute string      zip_code;
  attribute integer     pers_id;
  attribute string      special_dish;
  attribute integer     category;
  attribute integer     tourist_menu_price;};
```

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

```
interface Bistro
( source relational Food_Guide
  key r_code
  foreign_key(r_code)
    references Restaurant,
  foreign_key(pers_id)
    references Person)
{ attribute string      r_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;};
```

To represent object patterns in $\mathrm{ODL}_{I^3}$, union and optional constructors are used. In particular, the `union` constructor is used to represent object patterns describing heterogeneous objects in the source. An example of use of the `union` constructor in the $\mathrm{ODL}_{I^3}$ class representing the `Address` pattern of the `ED` source is shown in Figure 1. The semantics of the `union` constructor and of optional attributes in $\mathrm{ODL}_{I^3}$ will be discussed in the next section, using the OLCD Description Logics.

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

Figure 1: An example of `union` constructor in $\mathrm{ODL}_{I^3}$

## 2.3 The OLCD Description Logic

$\mathrm{ODL}_{I^3}$ descriptions are translated into OLCD (*Object Language with Complements allowing Descriptive cycles*) descriptions in order to perform Description Logics inferences that will be useful for semantic integration.

In this section, we give the syntax and the semantics of OLCD. Readers interested in a more formal account can refer to [1].

### 2.3.1 Types and Schemas

We assume a countable set of symbols $\mathbf{A}$ of *attribute names* (denoted by $a, a_1, a_2, \dots$) and we assume a countable set $\mathbf{N}$ of *type names* (denoted by $N$, $N_1$,$N_2$, $\dots$), which includes the set $\mathbf{B} = \{\texttt{Integer}, \texttt{String}, \texttt{Bool}, \texttt{Real}\}$ of base-type designators (which will be denoted by $B$) and the symbols $\top$, $\bot$. A *path* $p$ is either the symbol $\epsilon$, or a dot-separated sequence of elements $e_1 . e_2 . \cdots . e_n$, where $e_i \in \mathbf{A} \cup \{\triangle, \exists\}$ $(i = 1, \dots, n)$. $\epsilon$ denotes the unique path of length 0. Let $\mathbf{W}$ denote the set of all paths.

$\mathbf{S}(\mathbf{A}, \mathbf{N})$ denotes the set of all *finite type descriptions* (denoted by $S$, $S_1$, $S_2$, $\dots$), also briefly called *types*, over given $\mathbf{A}, \mathbf{N}$, obtained according to the following abstract syntax rule, where $a_i \neq a_j$ for $i \neq j$ (in the sequel $p$, $p_1$, $p_2$, $\dots$ , denote a path, $d$ denotes a base value, $\theta$ denotes a relational operator):

$S \to N \mid S_1 \sqcup S_2 \mid S_1 \sqcap S_2 \mid \neg S \mid \{S\}_\forall \mid \{S\}_\exists \mid [a_1 : S_1, \dots, a_k : S_k] \mid \triangle S \mid p\theta d \mid p\uparrow$

$\top$ denotes the *top type*, $\bot$ denotes the *empty type*, $\{\}_\forall$ and $[\,]$ denote the usual type constructors of set and record (tuple), respectively. The $\{S\}_\exists$ construct is an existential set specification, where at least one element of the set must be of type $S$. The construct $\sqcap$ stands for *intersection*, the construct $\sqcup$ stands for *union*, the construct $\neg$ stands for *complement*, whereas $\triangle$ constructs class descriptions, i.e., is an object set forming constructor. $p\theta d$, $p\uparrow$ represent *atomic predicates*: $p\theta d$ is a *range restriction* and $p\uparrow$ expresses *path undefinedness*.

Given a set of type descriptions $\mathbf{S}(\mathbf{A}, \mathbf{N})$, a *schema* $\sigma$ over $\mathbf{S}(\mathbf{A}, \mathbf{N})$ is a total function $\sigma\colon \mathbf{N} \backslash (\mathbf{B} \cup \{\top, \bot\}) \to \mathbf{S}(\mathbf{A}, \mathbf{N})$, which associates type names to descriptions. $\sigma$ is partitioned into two functions: $\sigma_P$, which introduces the description of primitive type names whose extensions must be explicitly provided by the user; and $\sigma_V$, which introduces the description of virtual type names whose extensions can be recursively obtained from the extension of the types occurring in their description.

In OLCD cyclic type names are allowed: in fact, since a type name may appear in type descriptions, we can have *circular references*, that is, type names which make direct or indirect references to themselves. Giving a *type as set* semantics to type descriptions, Description Logics, and thus OLCD, allows one to provide relevant reasoning techniques: computing *subsumption* relations between types (i.e. "is-a" relationships implied by type descriptions), deciding *equivalence* between types, and detecting *inconsistent* (i.e., always empty) types.

### 2.3.2 OLCD : Interpretations and Database Instances

We assume the union of the integers, the strings, the booleans, and the reals as the set $\mathcal{D}$ of *base values*. To build *complex values*, we further assume a countable, set disjoint from $\mathcal{D}$, of *object identifiers* (denoted by $o, o_1, o_2, \dots$). The set $\mathcal{V}$ of all *values over* $O$ is defined as the smallest set containing $\mathcal{D}$ and $O$, such that, if $v_1, \dots, v_p$ are values, then the set $\{v_1, \dots, v_p\}$ is a value, and a partial function $t\colon \mathbf{A} \to \{v_1, \dots, v_p\}$ is a value. The function $t$ is the usual tuple value; the standard notation $[a_1\colon v_1, \dots, a_p\colon v_p]$ will be henceforth used.

Let $=, \neq, >, <, \geq, \leq$ be the equality, inequality and total order relations, denoted by $\theta$, defined as usual on $\mathcal{D}$. Equality and inequality can be extended from $\mathcal{D}$ to all $\mathcal{V}$: the equality operator $(=)$ has the meaning of *identity*, i.e., two objects are equal if they have the same identifier, two sets are equal iff they have equal elements, two tuples, say $v_a = [a_1\colon v_1, \dots, a_p\colon v_p]$ and $v_b = [a'_1\colon v'_1, \dots, a'_q\colon v'_q]$, are equal if they have the same attributes and equal attribute labels are mapped to equal values. Object identifiers are assigned values by a *total value function* $\delta$ from $O$ to $\mathcal{V}$.

Let $\mathbf{W}$ denote the set of all paths. Given a set of object identifiers $O$ and a value function $\delta$, let $\mathcal{J}\colon \mathbf{W} \longrightarrow 2^{\mathcal{V}\times\mathcal{V}}$ a function defined as follows:

- empty path: $\mathcal{J}[\epsilon] = \Big\{(v, v) \in \mathcal{V} \times \mathcal{V}\Big\}$

- single element path: $\mathcal{J}[a] = \left\{(v_1, v_2) \in \mathcal{V} \times \mathcal{V} \mid v_1 = [\ldots, a : v_2, \ldots]\right\}$
  $\mathcal{J}[\triangle] = \left\{(o, v) \in O \times \mathcal{V} \mid \delta(o) = v\right\}$
- multiple element path: $\mathcal{J}[e_1 . e_2 . \cdots . e_n] = \mathcal{J}[e_1] \circ \mathcal{J}[e_2] \circ \cdots \circ \mathcal{J}[e_n]$ where $\circ$ is the symbol of function composition.

Notice that, for all $p$, $\mathcal{J}[p]$ is undefined on set values. Let $v$ be a value and $p$ be a path. By $\mathcal{J}[p](v)$ we mean the unique value (when it exists) reachable from $v$ following $p$, that is the value of the partial function $\mathcal{J}[p]$ in $v$.

Let $\mathcal{I}_{\mathbf{B}}$ be the (fixed) standard interpretation function from $\mathbf{B}$ to $2^{\mathcal{D}}$. For a given object assignment $\delta$, each type expression $S$ is mapped to a set of values (its interpretation). An *interpretation function* is a function $\mathcal{I}$ from $\mathbf{S}$ to $2^{\mathcal{V}}$ satisfying the following equations:

$$
\begin{aligned}
\mathcal{I}[\top] &= \mathcal{V} \\
\mathcal{I}[\bot] &= \emptyset \\
\mathcal{I}[B] &= \mathcal{I}_{\mathbf{B}}[B]
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{I}[\{S\}_{\forall}] &= \{M \mid M \subseteq \mathcal{I}[S]\} \\
\mathcal{I}[\{S\}_{\exists}] &= \{M \mid M \cap \mathcal{I}[S] \neq \emptyset\} \\
\mathcal{I}\Big[[a_1 : S_1, \ldots, a_p : S_p]\Big] &= \{t \colon \mathbf{A} \to \mathcal{V} \mid t(a_i) \in \mathcal{I}[S_i], 1 \leq i \leq p\} \\
\mathcal{I}[S_1 \sqcap S_2] &= \mathcal{I}[S_1] \cap \mathcal{I}[S_2] \\
\mathcal{I}[S_1 \sqcup S_2] &= \mathcal{I}[S_1] \cup \mathcal{I}[S_2] \\
\mathcal{I}[\neg S] &= \mathcal{V} \setminus \mathcal{I}[S] \\
\mathcal{I}[\triangle S] &= \left\{o \in O \middle| \delta(o) \in \mathcal{I}[S]\right\} \\
\mathcal{I}[(p \theta d)] &= \left\{v \in \mathcal{V} \mid \mathcal{J}[p](v) \theta d\right\} \\
\mathcal{I}[(p\uparrow)] &= \{v \in \mathcal{V} \mid v \notin \text{ dom } \mathcal{J}[p]\}
\end{aligned}
$$

Note that the interpretation of tuples implies an open world semantics for tuple types similar to the one adopted by Cardelli [3], and that $(p\uparrow)$ selects objects which do not have the path $p$. It should be noted than an interpretation does not necessarily imply that the extension of a named type is identical to the type description associated with the type name via the schema $\sigma$. For this purpose, we have to further constrain the interpretation function: An interpretation function $\mathcal{I}$ is a *legal instance* of a schema $\sigma$ iff the set $O$ is finite, and for all $N \in \mathbf{N}$:

$$
\begin{aligned}
\mathcal{I}[N] &\subseteq \mathcal{I}[\sigma_P(N)] && \text{if } N \in \text{dom } \sigma_P \\
\mathcal{I}[N] &= \mathcal{I}[\sigma_V(N)] && \text{if } N \in \text{dom } \sigma_V
\end{aligned}
$$

From the above definition, we see that the interpretation of a primitive type name *is included* in the interpretation of its description, while the interpretation of a virtual type *is* the interpretation of its description. In other words, the interpretation of a primitive type name has to be provided by the user, according to the given description, while the interpretation of a virtual type name is drawn from its definition and from the interpretation of primitive type names, thus corresponding to a view in database context.

Given a type $S$ of a schema $\sigma$, we say that $S$ is *consistent* if and only if there is a legal instance $\mathcal{I}$ of $\sigma$ such that $\mathcal{I}[S] \neq \emptyset$. Given two types $S_1$, $S_2$ of a schema $\sigma$, we say that $S_1$ *subsumes* $S_2$ iff $\mathcal{I}[S_1] \supseteq \mathcal{I}[S_2]$ for all legal instances $\mathcal{I}$ of $\sigma$. Consistency and subsumption can be reduced to each other, according to the following rules: $S_1$ is subsumed by $S_2$ iff $S_1 \sqcap \neg S_2$ is inconsistent, and $S$ is consistent iff it is not subsumed by $\bot$. The consistency problem is PSPACE-hard; in [1], an algorithm for checking the consistency of a type (which can also be used for subsumption computation), based on the *tableaux calculus*, is given.

### 2.4 $\text{ODL}_{I^3}$ to OLCD translation

In this section, we describe how $\text{ODL}_{I^3}$ source schema descriptions are translated into OLCD descriptions.

**$\text{ODL}_{I^3}$ classes.** In general, a $\text{ODL}_{I^3}$ class is translated into a OLCD primitive class in a simple way: each attribute of the $\text{ODL}_{I^3}$ class becomes an attribute of the corresponding OLCD class.

For example, the `Restaurant` $\text{ODL}_{I^3}$ class is translated as follows:

$$\sigma_P(\texttt{ES.Restaurant}) = \triangle[\texttt{r\_code: String, name: String, street: String,} \\ \texttt{zip\_code : String, pers\_id : Integer, special\_dish : String,} \\ \texttt{category : Integer, tourist\_menu\_price : Integer}\,]$$

Some aspects of an $\text{ODL}_{I^3}$ class declaration, such as `key r_code` in the `Restaurant` $\text{ODL}_{I^3}$ class, are not translated into OLCD, but will be used in the semantic information integration.

**Union constructor.** The `union` constructor of $\text{ODL}_{I^3}$ is translated using the construct $\sqcup$ of OLCD; for example, the `Address` pattern of figure 1 is translated in OLCD as follows:

$$\sigma_P(\texttt{ES.Address}) = \triangle\Big(\texttt{String} \sqcup \\ [\texttt{city: String, street: String, zipcode: String}\,]\Big)$$

**Optional constructor.** The construct $\sqcup$ is also used to translate *optional* attributes into OLCD. In fact, an optional attribute `att` specifies that a value may exist or not for a given instance. This fact is expressed in OLCD as the union between the attribute specification (with its domain) and *attribute undefinedness*, denoted by$\uparrow$ operator: $([\texttt{att1: domain1}] \sqcup \texttt{att1}\uparrow)$. For example, in the `Fast_Food` interface, the optional attributes are translated as follows:

$$\sigma_P(\texttt{ES.Fast\_Food}) = \triangle\Big(\,[\,\texttt{name : String, address : ES.Address,} \\ \texttt{specialty : \{String\}, category : String}\,] \sqcap \\ ([\,\texttt{phone : Integer}\,] \sqcup \texttt{phone}\uparrow) \sqcap \\ ([\,\texttt{nearby : ES.Fast\_Food}\,] \sqcup \texttt{nearby}\uparrow) \sqcap \\ ([\,\texttt{midprice : Integer}\,] \sqcup \texttt{midprice}\uparrow) \sqcap \\ ([\,\texttt{owner : ES.Owner}\,] \sqcup \texttt{owner}\uparrow)\,\Big)$$

**Integrity constraint rules.** An *if then* integrity constraint rule is integrated into an OLCD class description, by using the $\sqcap$, $\sqcup$ and $\neg$ constructs. For example, the rule:

```
rule Rule1 forall X in Restaurant :
   (X.category > 5) then X.tourist_menu_price > 100;
```

is added to the `ES.Restaurant` description as follows:

$$\sigma_P(\texttt{ES.Restaurant}) = \triangle\Big(\,[\,\texttt{r\_code : String, name : String, street : String,} \\ \texttt{zip\_code : String, pers\_id : Integer, special\_dish : String,} \\ \texttt{category : Integer, tourist\_menu\_price : Integer}\,] \sqcap \\ (\neg(\texttt{category} > 5) \sqcup (\texttt{tourist\_menu\_price} > 100))\Big)$$

Then, in our framework, integrity constraints are statements about the world and not about the contents of the database. In other words, a schema is composed by classes + integrity constraints and we check the consistency of such a schema.

**Intensional relationships.** They are not translated.

**Extensional relationships.** An "isa" relationships $C_1$ ISA $C_2$ related to an Extensional relationships and expressed in ODL$_{I^3}$ by the rule:

```
rule Rule2 forall X in C1 then X in C2
```

is integrated in the $C_1$ class description, by using the $\sqcap$ construct: $\sigma_P(C_1) = C_2 \sqcap \ldots$

**Mapping Rules.** They are not translated.

# 3 Reasoning about ODL$_{I^3}$ schema descriptions to build a Common Thesaurus

This section repeat and extend the method adopted to buil the Common Thesaurus presented in project report **D1.R1**.

To develop intelligent techniques for semantic integration, inter-schema knowledge between information sources in the considered domain has to be identified and properly represented. For this purpose, we construct a *Common Thesaurus* of terminological intensional and extensional relationships, describing inter-schema knowledge about ODL$_{I^3}$ classes and attributes of source schemas. The Common Thesaurus provides a reference on which to base the identification of ODL$_{I^3}$ classes candidate to integration and subsequent derivation of their global represention.

In the Common Thesaurus, we express inter-schema knowledge in form of terminological relationships (SYN, BT, NT, and RT) and extensional relationships (SYN$_{ext}$, BT$_{ext}$, and NT$_{ext}$ between classes and/or attribute names.

The Common Thesaurus is constructed through an incremental process during which relationships are added in the following order:

1. *schema-derived relationships*
2. *lexical-derived relationships*
3. *designer-supplied relationships*
4. *inferred relationships*

All these relationships are added to the Common Thesaurus and thus considered in the subsequent phase of semantic information integration (see next section). Terminological relationships defined in each step hold at the intensional level by definition. Furthermore, in each of the above step the designer may "strengthen" a terminological relationships SYN, BT and NT between two classes $C_1$ and $C_2$ by establishing that they hold also at the extensional level, thus defining also an extensional relationship. The specification of an extensional relationship, on one hand, implies the insertion of a corresponding intensional relationship in the Common Thesaurus and, on the other hand, enable subsumption computation (i.e., inferred relationships) and consistency check between two classes $C_1$ and $C_2$.

## 3.1 Schema-derived relationships

In this step, we extract terminological and extensional relationships holding at intra-schema level by analyzing each ODL$_{I^3}$ schema separately. In particular, intra-schema RT relationships are extracted from the specification of foreign keys in relational source schemas.

**Example 3.1** Consider the `ED` and `FD` sources. A subset of intra-schema relationships automatically extracted is the following:
⟨`ED.Fast-Food` RT `ED.Owner`⟩,
⟨`ED.Fast-Food` RT `ED.Address`⟩,
⟨`ED.Fast-Food` RT `ED.Fast-Food`⟩,
⟨`FD.Restaurant` RT `FD.Person`⟩,
⟨`FD.Bistro` RT `FD.Person`⟩.

When a foreign key is also a primary key both in the original and in the referenced relation, a BT/NT relationship is extracted at the extensional level as in the case of ⟨ `FD.Bistro` $\mathrm{NT}_{ext}$ `FD.Restaurant` ⟩.

## 3.2 Lexical-derived inter-schema relationships

In this step, terminological and extensional relationships holding at inter- schema level are extracted by analyzing $\mathrm{ODL}_{I^3}$ schemas together. The extraction of these relationships is based upon the lexical relations holding between classes and attributes names, deriving from the mining of used words. This is a kind of knowledge which is not based on the rules of a data definition language but derives from the name assigned by the designer. It is a designer's task to assign descriptive/meaningful names or, at least, correctly interpretable names. An interpretation uncertainty is therefore inherent to the language ambiguity.

Anyway knowledge associated with schema names is an opportunity that must be exploited to extract relationships. As it is almost impossible to carry out this task manually when the number and dimensions of schema grows, it was decided to experiment the use of WordNet [8] lexical system to extract intensional inter-schema relationships and propose them to the designer.

### The WordNet database

WordNet is a lexical database which was developed by the Princeton University [8] Cognitive science Laboratory. WordNet is inspired by current psycholinguistic human lexical memory connected theories and it is regarded as the most important researcher's available resource in the fields of computational linguistics, textual analysis and other related areas. The lexical Wordnet database, in the current 1.6 version has 64089 lemma which are organized in 99757 synonym sets (*synset*).

The starting point of lexical semantics is the constatation of the existence of a conventional association between the words form (i.e., the way in which they are pronounced or written) and the concept/meaning they express; such association is of the many-to-many kind, giving rise to the following properties:

**Synonymy:** property of a concept/meaning which can be expressed with two or more words. A synonyms group is named *synset*. Note that one and only *synset* exists for each concept/meaning.

**Polysemy:** property of a single word having two or more meanings.

The correspondence between the words form and their meaning is synthesized in the so called *Lexical Matrix* $\mathcal{M}$, in which the words meaning are reported in rows (hence each row represents a *synset*) and columns represent the words form (form/base lemma).

Each matrix element is a(*entry*), $e = (f, m)$ definition, where $f$ is the *base form* and $m$ (*meaning*) is the meaning counter; for example (`address`, `2`) refers to the address where a person or a fast-food can be found; while (`address`, `1`) refers to a computer address in the informatics sphere. From here on the base form and the meaning of an element $e = (f, m)$ will be respectively indicated with $e.f$ and $e.m$. An element of the $\mathcal{M}$ matrix may be *null* or *indefinite*. As only one $\mathcal{M}$ row is associated to a *synset*, from here on we will use $s \in \mathbb{S}$ as a $\mathcal{M}$ row indicator. In other words the non null elements of the $\mathcal{M}[s]$ row, represent each and every $s$ element.

In the same way, as only one $\mathcal{M}$ column is associated to a base form, from here on we will use the base forms as $\mathcal{M}$ columns index.

### 3.2.1 Semantic relationships between schema terms

With the concept of *term* we associate a definition to each class or attribute name. A *term* is formed by the $t = (n, e)$ couple, where $n$ indicates a class or attribute name, and $e$ indicates a definition. A class or attribute name $n$ are qualified as follows a class name is qualified by the name of the source schema to whom the class belongs
(`source_name.class_name`), an attribute name is moreover qualified with the name of the class to whom it belongs
(`source_name.class_name.attribute_name`). The classes and attributes names set is indicated by $\mathbf{N}$; the set of words in $\mathbf{N}$ is indicated by $\mathbb{I}$. The relation between *synset* defined in Wordnet are the starting point to define semantic relations between words. Various relations are obtainable with the WordNet database; some of them are between single words others are between *synset*. In this context we will use the following relations between *synset*: Synonymy, Hypernymy, Hyponymy, Olonymy, Meronymy and Correlation[1] As hyponymy and meronymy are inverse relations to hypernymy and olonymy, respectively, the set of relations between *synset* is the following:

$\mathcal{W} = \{\mathbf{S}_{ynonymy}, \mathbf{H}_{ypernymy}, \mathbf{O}_{lonymy}, \mathbf{C}_{orrelation}\}$.

Given the *synset* $\mathbb{S}$ set and the $\mathcal{W}$ relations set, The function $\phi : \mathbb{S} \times \mathcal{W} \longrightarrow 2^{\mathbb{S}}$ is inserted giving for each *synset* $s$ the set of *synset* associated through the $r \in \mathcal{W}$ relation:

$$\phi(s, r) = \{s' \mid s' \in \mathbb{S}, r \in \mathcal{W}, \langle s' r s \rangle\}$$

Given a *synset* $\mathbb{S}$ set and a $\mathbb{I}$set of words, the function$\mathcal{H} : \mathbb{S} \longrightarrow 2^{\mathbb{I}}$ is defined associating, on the basis of the lexical matrix, a set of words to a given *synset* :

$$\mathcal{H}(s) = \{t = (n, e) | n \in \mathbf{N}, \mathcal{M}[s][t.e.f] = t.e\}$$

We can hence obtain the relations between the words using the relations existing between the *synset* that contain those words. Given a set of words $\mathbb{I}$, the set of relations between words $\mathcal{R}$, $\mathcal{R} \subseteq \mathbb{I} \times \mathcal{W} \times \mathbb{I}$, is defined as follows:

$$\mathcal{R} = \{\langle t_i r t_j \rangle | r \in \mathcal{W}, t_i, t_j \in \mathbb{I}, \exists s : t_i \in \mathcal{H}(s), t_j \in \phi(s, r), t_i \neq t_j\}$$

The relations deriving from are proposed as semantic relations to be inserted in the *Common Thesaurus* according to the following correspondence:
**Synonymy:** corresponds to a SYN relation.
**Hypernymy:** corresponds to a BT relation.
**Olonymy:** corresponds to a RT relation.
**Correlation:** corresponds to a RT relation.

**Example 3.2** Consider the `ED` and `FD` sources. The relationships derived using WordNet are the following:
⟨`FD.Restaurant` BT `FD.Brasserie`⟩,
⟨`FD.Person` BT `ED.Owner`⟩,
⟨`ED.Owner.name` BT `FD.Person.first_name`⟩,
⟨`ED.Owner.name` BT `FD.Person.last_name`⟩,
⟨`ED.Fast-Food.name` BT `FD.Person.first_name`⟩,
⟨`ED.Fast-Food.name` BT `FD.Person.last_name`⟩.

[1] Correlation is a relation which links 2 *synset* sharing the same hypernym, i.e. the same "father".

Note that, SYN relationships are also extracted for the attributes having the same name in the two sources, which are omitted from previous set for the sake of brevity.

### 3.3 Designer-supplied inter-schema relationships

In this step, new relationships can be supplied directly by the designer, to capture specific domain knowledge about the source schemas (e.g., new synonyms).
This is a crucial operation, because the new relationships are forced to belong to the Common Thesaurus and thus used to generate the global integrated schema. This means that, if a nonsense or wrong relationship is inserted, the subsequent integration process can produce a wrong global schema. The following Relationship validation section shows how our system help the designer in detecting wrong relationships.

**Example 3.3** In our example, the designer supplies the following relationships for classes and attributes:
⟨`ED.Fast-Food` $\text{SYN}_{ext}$ `FD.Restaurant`⟩,
⟨`ED.Fast-Food.category` BT `FD.Bistro.type`⟩,
⟨`ED.Fast-Food.specialty` BT `FD.Bistro.special_dish`⟩.

The definition of the relationship ⟨`ED.Fast-Food` $\text{SYN}_{ext}$ `FD.Restaurant`⟩ by the designer implies the automated definition of the relationship ⟨`ED.Fast-Food` SYN `FD.Restaurant`⟩ in the Common Thesaurus.

### 3.4 Relationships validation

In this step, ODB-Tools is employed to validate intensional relationships between attribute names and extensional relationships between class names.

The validation of intensional relationships between attribute names is based on the compatibility of the domains associated with the attributes. This way, *valid* and *invalid* intensional relationships are distinguished. In particular, let $a_t = \langle n_t, d_t \rangle$ and $a_q = \langle n_q, d_q \rangle$ be two attributes, with a name and a domain, respectively. The following checks are executed on intensional relationships defined for attribute names in the Common Thesaurus:

- $\langle n_t$ SYN $n_q \rangle$: the relationship is marked as valid if $d_t$ and $d_q$ are equivalent, or if one is a specialization of the other;
- $\langle n_t$ BT $n_q \rangle$: the relationship is marked as valid if $d_t$ contains or is equivalent to $d_q$;
- $\langle n_t$ NT $n_q \rangle$: the relationship is marked as valid if $d_t$ is contained in or is equivalent to $d_q$.

When an attribute domain $d_t$ ($d_q$) is defined using the `union` constructor, as in the `Address` example (see Figure 1), a *valid relationship* is recognized if at least one domain $d_t$ ($d_q$) is compatible with $d_q$ ($d_t$).

**Example 3.4** Referring to our Common Thesaurus resulting from Examples 3.1 to 3.3, the output of the validation phase is the following (for each relationship, control flag [1] denotes a valid relationship while [0] an invalid one):

| | |
|---|---|
| ⟨`ED.Fast-Food.category` BT `FD.Bistro.type`⟩ | [0] |
| ⟨`ED.Owner.name` BT `FD.Person.first_name`⟩ | [1] |
| ⟨`ED.Owner.name` BT `FD.Person.last_name`⟩ | [1] |
| ⟨`ED.Fast-Food.specialty` BT `FD.Bistro.special_dish`⟩ | [1] |
| ⟨`ED.Fast-Food.name` BT `FD.Person.first_name`⟩ | [1] |

⟨`ED.Fast-Food.name` BT `FD.Person.last_name`⟩ [1]
⟨`ED.Fast-Food.category` SYN `FD.Restaurant.category`⟩ [0]

As an extensional relationship between two classes $C_1$ and $C_2$ is integrated in the description of the class $C_1$, its validation is performed by checking the consistency of the class $C_1$. For example, the extensional relationship ⟨`FD.Restaurant` $\text{BT}_{ext}$ `FD.Bistro` ⟩ stated before by the designer is expressed in the `FD.Bistro` class description as follows:

$$\sigma_P(\texttt{FD.Bistro}) = \texttt{FD.Restaurant} \sqcap \triangle\, [\, \texttt{r\_code} : \texttt{String}, \texttt{type} : \texttt{String}, \texttt{pers\_id} : \texttt{Integer}\,]$$

Since the `FD.Bistro` class description is consistent, the relationship between `FD.Bistro` and `FD.Restaurant` is validated. On the other hand, the extensional relationship ⟨`ED.Fast-Food` $\text{SYN}_{ext}$ `FD.Restaurant`⟩ is rejected, as the class description:

$$\sigma_P(\texttt{ED.Fast-Food}) = \texttt{FD.Restaurant} \sqcap \ldots$$

is inconsistent (the attribute `category` is defined in both the classes but on disjoint domains). In the presence of integrity rules less intuitive incoherencies may arise.

In this case, the designer modifies his statement and only the terminological relationship ⟨`ED.Fast-Food` SYN `FD.Restaurant`⟩ is kept in the Common Thesaurus.

### 3.5 Inferring new relationships

In this step, inference capabilities of ODB-Tools are exploited to infer new relationships, in order to set up a rich Common Thesaurus to support the identification of semantically similar $\text{ODL}_{I^3}$ classes in different sources, as will be shown in next section.

**Example 3.5** Relationships inferred in this step are the following:
⟨`FD.Bistro` RT `ED.Owner`⟩,
⟨`FD.Bistro` RT `ED.Address`⟩,
⟨`FD.Brasserie` RT `ED.Address`⟩,
⟨`FD.Brasserie` RT `FD.Person`⟩,
⟨`FD.Restaurant` RT `ED.Address`⟩,
⟨`ED.Fast-Food` BT `FD.Brasserie`⟩,
⟨`ED.Fast-Food` BT `FD.Bistro`⟩,
⟨`FD.Restaurant` RT `ED.Fast-Food`⟩,
⟨`FD.Restaurant` RT `ED.Owner`.⟩

Note that, due the simplicity of the adopted example, many of the discovered relationships are trivial. In order to show an example of inference due to extensional relationships, suppose to introduce a new pattern into the `Eating Source`:

`New-Food-pattern = (New-Food,{ name,specialty,category*})`

The related $\text{ODL}_{I^3}$ class is :

```
interface ED.New-Food
  ( source semistructured Eating_Source )
{  attribute string      name;
   attribute set<string> specialty;
   attribute string      category*;  };
```

Moreover, suppose that the designer states the following extensional relationship:

⟨`ED.New-Food` $\text{BT}_{ext}$ `FD.Restaurant`⟩.

This extensional relationship is validated as consistent; in fact the `category` attribute is optional in `ED.New-Food`. By exploiting subsumption computation ODB-Tools obtains the following inferred relationship: ⟨`ED.New-Food` $\text{BT}_{ext}$ `FD.Bistro` ⟩.

The inferred relationships obtained by subsumption computation are less trivial at the presence of integrity constraint rules and/or *views* (which are not considered in this report). In ODL$_{I^3}$ we can express, in a declarative way, *if then* integrity constraint rules at both intra- and inter-source level. For example, let us consider the following inter-source integrity constraint rules:

```
rule Rule2 forall X in FD.Restaurant :
   (X.pers_id in  FD.Person) then X in ED.Fast-Food;
```

By exploiting subsumption computation ODB-Tools obtains the following inferred relationship: ⟨FD.Bistro $\mathrm{NT}_{ext}$ ED.Fast-Food⟩.
In fact, we have ⟨ FD.Bistro $\mathrm{NT}_{ext}$ FD.Restaurant ⟩ (see subsection 3.1) and, due to the specification of foreign keys,

```
foreign_key(pers_id) references Person
```

in the FD.Bistro source, we have that the FD.Bistro class satisfies the antecedent of the integrity rules Rule2, then FD.Bistro is a ED.Fast-Food.

# 4 Exploiting ontology knowledge to discover affinity relationships among ODL$_{I^3}$ classes

In this section, we describe the theoretical foundations of the techniques to discover affinity inter-schema relationships between ODL$_{I^3}$ classes in different schemas. Affinity relationships express the fact that ODL$_{I^3}$ classes are semantically related, that is, they express the same or similar information in diverse schemas and, as such, they can be integrated. Discovering affinity relationships is based on two functions: a *similarity function*, called $AFFINITY()$, and a *clustering function*, called $GROUP()$. The purpose of the similarity function is to determine the level of semantic similarity of pairs of ODL$_{I^3}$ classes in their respective schemas. The purpose of the clustering function is to group all ODL$_{I^3}$ classes that are semantically similar in the analyzed schemas and group them into *clusters*. In the following, we describe the theoretical foundations of these two functions. Techniques for their implementation are described in [10].

## 4.1 Affinity function

Let $C$ be the set of ODL$_{I^3}$ classes to be analyzed. The $AFFINITY()$ function is defined as follows:
$$AFFINITY() : C \times C \rightarrow [0, 1]$$
$AFFINITY()$ is evaluated on ODL$_{I^3}$ classes with respect to *comparison features*. Different kinds of comparison features can be selected for ODL$_{I^3}$ classes (e.g.,names of the classes, attributes). Let us denote by $CF(c_i)$ the set of comparison features of a ODL$_{I^3}$ class $c_i$. Not all possible pairs of comparison features of two ODL$_{I^3}$ classes are relevant for the evaluation of $AFFINITY()$, but only the pairs that have a *semantic correspondence*. Two comparison features have a semantic correspondence if they describe the same real-world information. Let $cf \in CF(c_i)$ be a comparison feature of $c_i$. We denote by $\sim$ the existence of a semantic correspondence between comparison features of different elements. Let $CF(c_i) \cap CF(c_j) = \{(cf, cf') \mid cf \in CF(c_i), cf' \in CF(c_j), cf \sim cf'\}$ be the set composed of the pairs of comparison features that have a correspondence in $c_i$ and $c_j$.

The following properties are defined for $AFFINITY()$.

($\mathbf{P_1}$) *Nonnegativity.* The semantic similarity of two ODL$_{I^3}$ classes is nonnegative and is at most 1. $\forall c_i, c_j \in C, AFFINITY(c_i, c_j) \geq 0$ **and** $AFFINITY(c_i, c_j) \leq 1$.

($\mathbf{P_2}$) *Null value.* $AFFINITY()$ for two ODL$_{I^3}$ classes $c_i$ and $c_j$ is null if they do not have comparison features with semantic correspondence.
$CF(c_i) \cap CF(c_j) = \emptyset \Rightarrow AFFINITY(c_i, c_j) = 0.$

($\mathbf{P_3}$) *Identity.* The comparison of a ODL$_{I^3}$ class with itself always returns the greatest semantic similarity value.
$AFFINITY(c_i, c_i) = 1.$

($\mathbf{P_4}$) *Commutativity.* The semantic similarity of two ODL$_{I^3}$ classes is independent of their comparison order.
$AFFINITY(c_i, c_j) = AFFINITY(c_j, c_i)$

($\mathbf{P_5}$) *Monotonicity.* Adding comparison features with semantic correspondence to a pair of ODL$_{I^3}$ classes cannot decrease their semantic similarity.
$(CF(c_i) \cap CF(c_j)) \subset (CF'(c_i) \cap CF'(c_j)) \Rightarrow AFFINITY_{CF}(c_i, c_j) \leq AFFINITY_{CF'}(c_i, c_j)$, with $CF(c_i) \subset CF'(c_i)$ **and** $CF(c_j) \subset CF'(c_j)$[2].

($\mathbf{P_6}$) *Maximum value.* $AFFINITY()$ has value 1 for two ODL$_{I^3}$ classes $c_i$ and $c_j$ if all features of $c_i$ have a semantic correspondence with features of $c_j$ and vice versa, that is,
$\forall cf \in CF(c_i) \exists cf' \in CF(c_j), (cf, cf') \in (CF(c_i) \cap CF(c_j))$ **and** $\forall cf \in CF(c_j) \exists cf' \in CF(c_i), (cf, cf') \in (CF(c_i) \cap CF(c_j)) \Rightarrow AFFINITY(c_i, c_j) = 1.$

Criteria for the establishment of semantic correspondences between comparison features depend on the kind of feature under consideration. For example, using names as comparison features, semantic correspondences can be established using a criterion based on terminological relationships in the reference ontology, i.e., the Common Thesaurus. The similarity technique described in [10] performs ODL$_{I^3}$ class comparison at the level of attribute domain, by considering compatibility of domains with respect to type and structure in different ODL$_{I^3}$ schemas. In particular, the following features are considered for ODL$_{I^3}$ classes:

- the name of the classes. ODL$_{I^3}$ classes are compared with respect to their names. In fact, names are generally considered the first, heuristic indicator of the semantic similarity of different schema classes. A *Name Affinity* coefficient is defined and calculated to reflect the level of similarity of two ODL$_{I^3}$ classes based on their names.

- *Attributes* of ODL$_{I^3}$ classes. ODL$_{I^3}$ classes are compared with respect to their attributes, to conclude about their similarity on the basis also of their structure, in terms of class properties and referenced classes. In fact, class names alone provide only a partial indicator of semantic similarity, which should be complemented by the analysis of the structure. Classes having the same real world semantics, besides showing a terminological relationship, are also generally characterized by a semantically similar structure. ODL$_{I^3}$ classes are compared with respect to names and domains of both structural attributes (i.e., attributes with pre-defined domains) and *reference attributes* (i.e., attributes with a class domain referencing another class in the schema). A *Structural Affinity* coefficient is defined and calculated to reflect the level of similarity of two ODL$_{I^3}$ classes based on their attributes.

The $AFFINITY()$ value is obtained by combining the Name Affinity coefficient and the Structural Affinity coefficient into a comprehensive Global Affinity coefficient.

The establishment of affinity coefficients relies on the knowledge in the Common Thesaurus. A detailed description of the metrics adopted for computation of the affinity coefficients is given in [10]. To make the numerical evaluation of $AFFINITY()$ possible, each terminological

[2] Notation $AFFINITY_{CF}(c_i, c_j)$ is used to indicate that $AFFINITY()$ is evaluated with respect to features in $CF$.

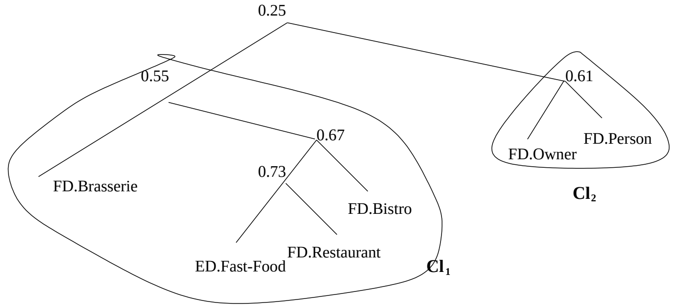


Figure 2: Example of affinity tree

relationship $\Re$ in the Common Thesaurus is properly strengthened. The strength $\sigma_{\Re} \in (0,1]$ of a terminological relationship $\Re$ expresses its implication for similarity. Different types of relationships have different implications for semantic similarity. In particular, we have $\sigma_{SYN} \geq \sigma_{BT} \geq \sigma_{RT}$. We assign the highest strength to the $SYN$ relationship, since synonymy indicates class similarity more precisely than remaining terminological relationships. As for $BT/NT$ and $RT$ strengths, we consider the semantic similarity implication of schema links represented by these relationships. Motivations to set $\sigma_{BT} \geq \sigma_{RT}$ are related to the fact that "is-a" links express a higher semantic connection between classes than relationships. In our experimentation, we used $\sigma_{SYN} = 1$, $\sigma_{BT} = \sigma_{NT} = 0.8$, and $\sigma_{RT} = 0.5$.

## 4.2 Clustering function

The $GROUP()$ function is defined as follows: $GROUP() : C \rightarrow 2^C$,
where $2^C$ is the powerset of $C$. $GROUP()$ starts from the set of local ODL$_{I^3}$ classes to be analyzed and returns sets (i.e., clusters) of semantically related classes, on the basis of the $AFFINITY()$ values for pairs of classes. Let $Cl_k \in 2^C$ be a cluster of semantically similar ODL$_{I^3}$ classes. The following property holds for $GROUP()$:

($\mathbf{P_7}$) *Homogeneity.* The value of $AFFINITY()$ between each possible pair of ODL$_{I^3}$ classes in a given cluster $Cl_k$ is always greater than the $AFFINITY()$ value between a ODL$_{I^3}$ class outside $Cl_k$ and any classes belonging to $Cl_k$.
$c_h \notin Cl_k \Rightarrow AFFINITY(c_i, c_j) \geq AFFINITY(c_i, c_h)$, **forall** $Cl_k$, **forall** $c_i, c_j \in Cl_k$.

This property ensures that in a given cluster we can find the most similar classes among all possible classes of $C$. To cluster semantically similar ODL$_{I^3}$ classes, we adopt classical clustering techniques of hierarchical type [6]. The general hierarchical clustering procedure is described in [10]. As the result of clustering, a *tree* is obtained, where several clusters, with an associated $AFFINITY()$ value, can be identified. In the similarity tree, leaves are ODL$_{I^3}$ classes and other nodes are virtual classes which abstract the commonalities of their children classes[3] and an associated $AFFINITY()$ value. The root represents the centroid of all classified classes, and its $AFFINITY()$ value can be null, if no commonalities are identified among all analyzed ODL$_{I^3}$ classes. Several clusters can be identified in the tree by specifying a threshold value of $AFFINITY()$. The number of classes in a cluster depends on the selected value of $AFFINITY()$. Once clusters have been selected, ODL$_{I^3}$ classes that have an extensional terminological relationship with at least one class in the cluster and not yet included in it (if any), are forced to belong to the cluster anyway, to define an integrated global ODL$_{I^3}$ class that is representative of all possible semantically related ODL$_{I^3}$ source classes.

As an example, considering the ODL$_{I^3}$ classes of example 2.2, using a threshold $T = 0.5$, two clusters are selected, namely $Cl_1$ and $Cl_2$, as shown in Figure 2. Cluster $Cl_1$ contains all

[3]Virtual schema elements are called "centroids" in the literature [6]).

ODL$_{I^3}$ classes describing different kinds of eating place, while cluster $Cl_2$ contains all ODL$_{I^3}$ classes describing persons. These two cluster are highly homogeneous and contain all classes characterized by an affinity relationship with a high value with other classes of the cluster and a low value with classes outside. Moreover, such two clusters contain all involved classes also with respect to extensional relationships.

# A The ODL$_{I^3}$ description language

The following is a BNF description for the ODL$_{I^3}$ description language. We included the main syntax fragments which differ from the original ODL grammar
(see `http://sparc20.dsi.unimo.it/Momis/documents/odli3_syntax.pdf` for the complete syntax)

| | | |
|---|---|---|
| ⟨interface_dcl⟩ | ::= | ⟨interface_header⟩ |
| | | **{**[⟨ interface_body⟩] [**union** ⟨interface_body⟩]**};** |
| ⟨interface_header⟩ | ::= | **interface** ⟨identifier⟩ |
| | | [⟨inheritance_spec⟩] [⟨type_property_list⟩] |
| ⟨inheritance_spec⟩ | ::= | **:** ⟨scoped_name⟩ [,⟨inheritance_spec⟩] |

Local schema pattern definition: the wrapper must indicate the kind and the name of the source of each pattern.

| | | |
|---|---|---|
| ⟨type_property_list⟩ | ::= | **(** [⟨source_spec⟩] [⟨extent_spec⟩] [⟨key_spec⟩] [⟨f_key_spec⟩] **)** |
| ⟨source_spec⟩ | ::= | **source** ⟨source_type⟩ ⟨source_name⟩ |
| ⟨source_type⟩ | ::= | **file** \| **relational** \| **nfrelational** |
| | | \| **object** \| **semistructured** |
| ⟨source_name⟩ | ::= | ⟨identifier⟩ |
| ⟨extent_spec⟩ | ::= | **extent** ⟨extent_list⟩ |
| ⟨extent_list⟩ | ::= | ⟨string⟩ \| ⟨string⟩,⟨extent_list⟩ |
| ⟨key_spec⟩ | ::= | **key**[s] ⟨key_list⟩ |
| ⟨f_key_spec⟩ | ::= | **foreign_key** **(**⟨f_key_list⟩**)** |
| | | **references** ⟨identifier⟩[,⟨f_key_spec⟩] |

Global pattern definition rule, used to map the attributes between the global definition and the corrisponding ones in the local sources.

| | | |
|---|---|---|
| ⟨attr_dcl⟩ | ::= | [**readonly**] **attribute** [⟨domain_type⟩] |
| | | ⟨attribute_name⟩ [*] [⟨fixed_array_size⟩] |
| | | [⟨mapping_rule_dcl⟩] |
| ⟨mapping_rule_dcl⟩ | ::= | **mapping_rule** ⟨rule_list⟩ |
| ⟨rule_list⟩ | ::= | ⟨rule⟩ \| ⟨rule⟩,⟨rule_list⟩ |
| ⟨rule⟩ | ::= | ⟨local_attr_name⟩ \|'⟨identifier⟩' |
| | | ⟨and_expression⟩ \|⟨union_expression⟩ |
| ⟨and_expression⟩ | ::= | **(** ⟨local_attr_name⟩ **and**⟨and_list⟩ **)** |
| ⟨and_list⟩ | ::= | ⟨local_attr_name⟩ \| ⟨local_attr_name⟩ **and** ⟨and_list⟩ |
| ⟨union_expression⟩ | ::= | **(**⟨local_attr_name⟩ **union** ⟨union_list⟩ **on** ⟨identifier⟩**)** |
| ⟨union_list⟩ | ::= | ⟨local_attr_name⟩ \| ⟨local_attr_name⟩ **union** |
| | | ⟨union_list⟩ |
| ⟨local_attr_name⟩ | ::= | ⟨source_name⟩**.**⟨class_name⟩**.**⟨attribute_name⟩ |

Relationships used to define the Common Thesaurus.

```
⟨relationships_list⟩          ::=  ⟨relationship_dcl⟩; | ⟨relationship_dcl⟩;
                                   ⟨relationships_list⟩
⟨relationships_dcl⟩           ::=  ⟨local_name⟩ ⟨relationship_type⟩
                                   ⟨local_name⟩
⟨local_name⟩                  ::=  ⟨source_name⟩. ⟨local_class_name⟩
                                   [.⟨local_attr_name⟩]
⟨relationship_type⟩           ::=  ⟨intensional_relationship⟩ |
                                   ⟨extensional_relationship⟩
⟨intensional_relationship⟩    ::=  SYN | BT | NT | RT
⟨extensional_relationship⟩    ::=  SYN_E | BT_E | NT_E
```

OLCD integrity constraint definition: declaration of rule (using *if then* definition) valid for each instance of the data; mapping rule specification (*or* and *union* specification rule).

```
⟨rule_list⟩        ::=  ⟨rule_dcl⟩; | ⟨rule_dcl⟩; ⟨rule_list⟩
⟨rule_dcl⟩         ::=  rule ⟨identifier⟩ ⟨rule_spec⟩
⟨rule_spec⟩        ::=  ⟨rule_pre⟩ then ⟨rule_post⟩ | {⟨case_dcl⟩}
⟨rule_pre⟩         ::=  ⟨forall⟩ ⟨identifier⟩ in ⟨identifier⟩ : ⟨rule_body_list⟩
⟨rule_post⟩        ::=  ⟨rule_body_list⟩
⟨case_dcl⟩         ::=  case of ⟨identifier⟩ : ⟨case_list⟩
⟨case_list⟩        ::=  ⟨case_spec⟩ | ⟨case_spec⟩ ⟨case_list⟩
⟨case_spec⟩        ::=  ⟨identifier⟩ : ⟨identifier⟩ ;
⟨rule_body_list⟩   ::=  (⟨rule_body_list⟩) | ⟨rule_body⟩ |
                        ⟨rule_body_list⟩ and ⟨rule_body⟩ |
                        ⟨rule_body_list⟩ and (⟨rule_body_list⟩)
⟨rule_body⟩        ::=  ⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨literal_value⟩ |
                        ⟨dotted_name⟩ ⟨rule_const_op⟩
                        ⟨rule_cast⟩ ⟨literal_value⟩ |
                        ⟨dotted_name⟩ in ⟨dotted_name⟩ |
                        ⟨forall⟩ ⟨identifier⟩ in ⟨dotted_name⟩ :
                        ⟨rule_body_list⟩ | exists ⟨identifier⟩ in
                        ⟨dotted_name⟩ : ⟨rule_body_list⟩
⟨rule_const_op⟩    ::=  = | ≥ | ≤ | > | <
⟨rule_cast⟩        ::=  (⟨simple_type_spec⟩)
⟨dotted_name⟩      ::=  ⟨identifier⟩ | ⟨identifier⟩. ⟨dotted_name⟩
⟨forall⟩           ::=  for all | forall
```

# References


[1] 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.

[2] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proc. of ICDT 1997*, pages 336–350, Delphi, Greece, 1997.

[3] L. Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 51–67. Springer-Verlag, 1984.

[4] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[5] R. G. G. Cattell, editor. *The Object Database Standard: ODMG93.* Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[6] B. Everitt. *Computer-Aided Database Design: the DATAID Project.* Heinemann Educational Books Ltd, Social Science Research Council, 1974.

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

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

[9] S. Nestorov, S. Abiteboul, and R. Motwani. Inferring structure in semistructured data. *SIGMOD Record*, 26(4):39–43, 1997.

[10] Luigi Palopoli, Domenico Rosaci, Giorgio Terracina, Domenico Ursino, Ilario Benetti, Sonia Bergamaschi, Domenico Beneventano, Francesco Guerra, Federica Mandreoli, Maurizio Vincini, Silvana Castano, Valeria De Antonellis, and Michele Melchiori. Methodologies and techniques for the extraction, the representation and the integration of structured and semi-structured information sources. Technical Report D1.R1, Integrazione, Warehousing e Mining di sorgenti eterogenee, March 2001.