NeP4B
Networked Peers for Business

WP4
Task T4.2
Deliverable D4.2.1

Design of procedures for building inter-peer semantic mapping

(FINAL, 30/04/2009)

Abstract –
This deliverable describes matching technics necessary to rewrite a query and to address it towards the peer which most probably keep most relevant information. Such information can be of semantic or multimedia nature, so we have to perform two different matching technics.

IsGroup
Unimo

Document information

Document ID code	D4.2.1		
Keywords	semantic matching, semantic peer, multimedia indexing		
Classification	**FINAL**	Date of reference	**30/04/2009**
Distribution level	**NeP4B Consortium**		

Editor	IsGroup	Unimo
Authors	Claudio Gennaro	ISTI-CNR
	Rita Lenzi	Unimo
	Federica Mandreoli	Unimo
	Riccardo Martoglia	Unimo
	Matteo Mordacchini	ISTI-CNR

Version history		
Date	*Version*	*Description*
30/04/2009	FINAL 1.0	Final version
27/01/2009	DRAFT 0.1	Draft version

Contents

1 Executive summary

This deliverable describes matching technics necessary to rewrite a query and to address it towards the peer which most probably keep most relevant information. Such information can be of semantic or multimedia nature, so we have to perform two different matching technics. The first one, concerning semantic aspect, has been developed by IsGroup of University of Modena and Reggio Emilia while the second one, concerning multimedia contents has been worked up by ISTI-CNR.

2 Introduction

Information and communication technologies (ITCs) over the Web have become a strategic asset in the global economic context. The Web fosters the vision of an Internet-based global marketplace where automatic cooperation and competition are allowed and enhanced. This is the stimulating scenario of the outgoing Italian Council co-founded NeP4B (Network Peers for Business) Project whose aim is to develop an advanced technological infrastructure for small and medium enterprizes (SMEs) to allow them to search for partners, exchange data and negotiate without limitations and constraints.

According to the recent proposal of Peer Data Management Systems (PDMSs), the project infrastructure is based on independent and interoperable semantic peers who behave as nodes of a virtual peer-to-peer (P2P) network for data and service sharing. In this context, a semantic peer can be a single SME, as well as a mediator representing groups of companies, and consists of a set of data sources placed at the P2P network disposal through an OWL ontology. These data sources include multimedia objects, such as the description/presentations of the pruducts/services extracted from the companies' Web sites. This information is represented by means of appropriate multimedia attributes in the peers' ontologies (e.g. *image* in Peer1's ontology of Figure 1 that are exploited in the searching process by using a SPARQL-like language properly extended to support similarity predicates.

Each peer is connected to its neighbors through semantic mappings, appropriately extended with scores expressing their strength, which are exploited for query processing purpose: in order to query a peer, its own ontology is used for query formulation and semantic mappings are used to reformulate the query over its immediate neighbors, then over their immediate neighbors, and so on, following a semantic path of mappings. For instance, in Figure 1 the concepts *product*, *origin* and *image* of the sample query must be reformulated in *item*, *provenance* and *photo* when the query is forwarded to Peer2.

Scores are very important because of the heterogeneity of schemas as they give a measure of the semantic compatibility occurring between involved concepts. Moreover, as such scores reflect the relevance of peer's data to a query, we deem that semantic mappings can be exploited in the searching phase to suggest a direction towards the semantic paths which better satisfy the query conditions.

In this document we present:

- in Section 3 our semantic mapping technic in order to be able to perform a query reformulation: when a querying peer p_i forwards the query q to one of its neighbors, say p_j, q must be reformulated into q' so that it refers to concepts in the p_j's schema. To this end, p_i uses the semantic mapping between source and target schema;
- in Section 4 our extension of semantic mapping, relying on a fuzzy set theory, to obtain semantic paths associated with a score;
- in Section 5 a support for similarity search over sets of multimedia attributes for content-based retrieval;
- in Section 6 some experiments relative to our matching algorithm.

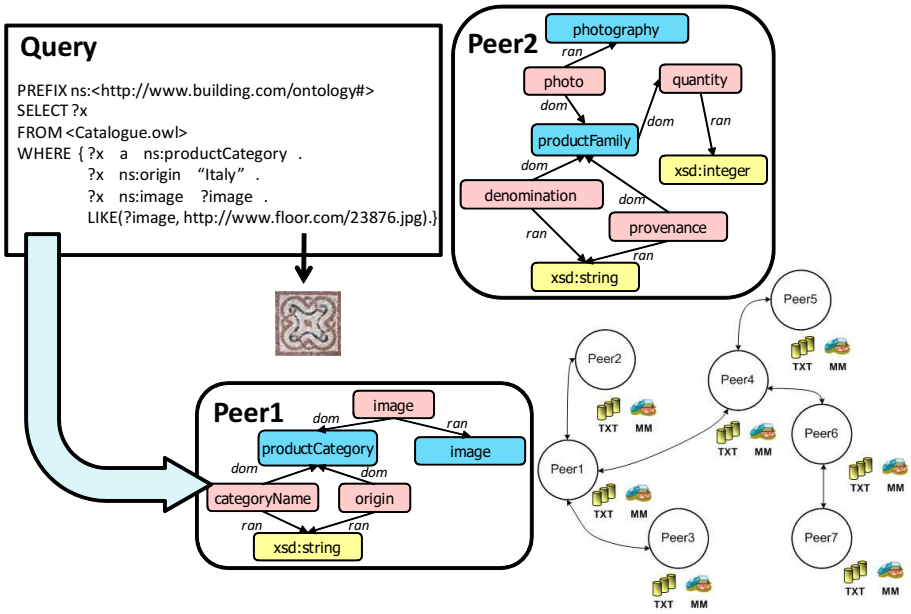


Figure 1: Reference scenario

3 Semantic Matching

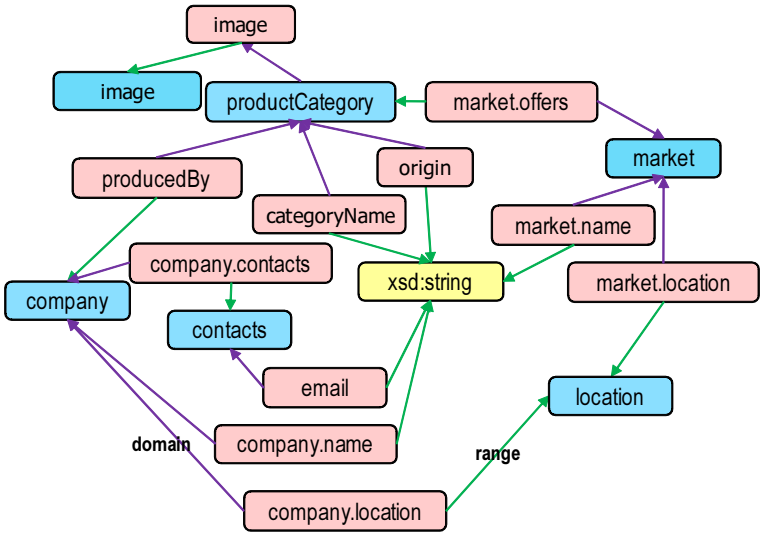
In this section, we describe our proposal of Semantic Matching which goal is to find correspondences between concepts, i.e. classes and properties, defined in the RDF/OWL schemas.

We denote with P a set of peers. Each peer $p_i \in P$ stores local data, modeled upon local schema S_i, like those shown in Figure 2. This makes a peer p_i a *semantic peer* [6], in that its local schema S_i describes the semantic content of its underlying data. A peer schema, as we work with RDF/OWL standard, is represented by a graph $SG = (C, P, H_c, H_p, Lc, Lp)$ where C is a class set, P is a property set, H_c and H_p are sets of standard constructs used to define hierarchies between, respectively, classes and properties and Lc and Lp are label sets. A single class $c_i \in C$ is associated to a unique label $l_c \in Lc$. A property $p_i \in P$ is defined as (l_p, c_d, c_r) where $l_p \in Lp$ is its label and c_d and c_r are respectively its domain and range classes. We further define a path on the schema as sequence of properties $(p_i, ...p_j)$ such that the range class of p_k, $i \leq k \leq j$, corresponds to the domain class of p_{k+1}. Notice that a path whose length is 1, represents a single property.

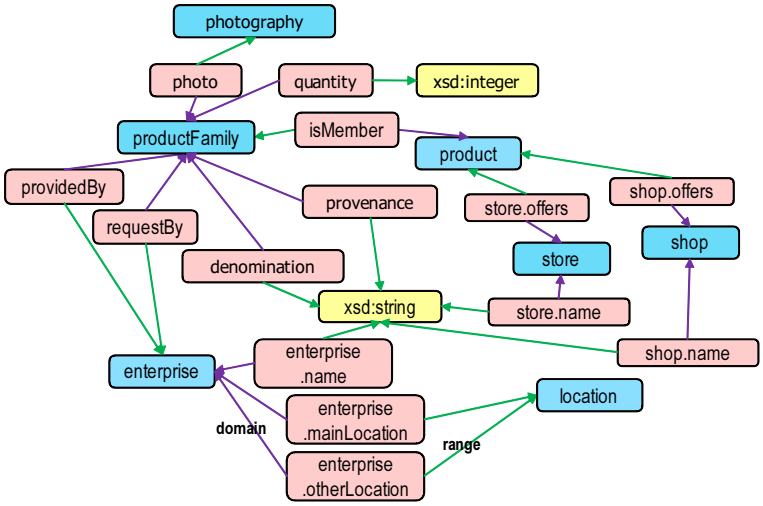
Peers are pairwise connected in a semantic network through semantic mappings between their schemas. A semantic mapping $M(S_i, S_j)$ can be established from a source schema S_i and a target schema S_j, and it defines how to represent S_i in terms of S_j's vocabulary. In particular, it associates each class and each property in S_i to a corresponding class or path, respectively, in S_j according to a $score \in [0, 1]$, denoting the *degree of semantic similarity* between two concepts ($\mu(C, C')$ or simply $sim(C, C')$).

The proposed matching algorithm consists of 5 different sections, schematically shown in Figure 3:

1. *parsing* of the input schemas;
2. *pre-processing* to write schemas in such a format that maximize the matching effec-



(a) Peer1 schema fragment



(b) Peer2 schema fragment

Figure 2: Peers' schema

tiveness;

3. *basic matching computation*, which concerns single concepts, i.e. single classes and single properties, performed through a Similarity Flooding [7] inspired algorithm;

4. *refinement* to extract relevant results coming from the previous phase;

5. *path matching computation*, i.e. a specifically devised semi-automatic process aimed at finding correspondences between unmatched properties and paths, exploiting different techniques based on label similarity, automatic semantic path categorization and user feedback learning.

Such algorithm output is an xml file containing the semantic mapping for the involved couple

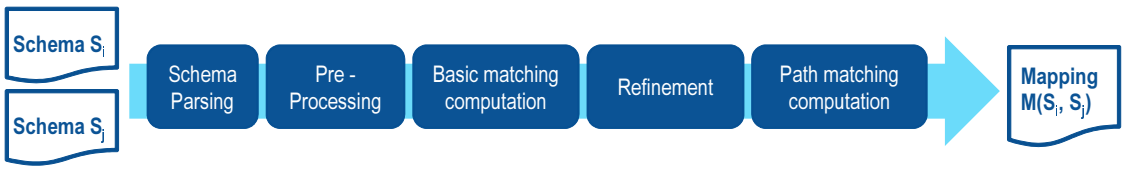


Figure 3: Schema matching phases

of schemas.

3.1 Schema parsing

The matching phase input is represented by OWL or RDF *annotated* schemas. This means that, for each class or property, a further information, represented by one or more senses in a reference ontology, is required in order to point out their meaning. We extract annotations from the English lexical database *WordNet* (WN) [1] using a completely automated approach [5]. WordNet is organized conceptually in synonym sets of synsets, representing different meaning or senses. Each term in WN is usually associated to more than one synset, signifying it is polysemic, i.e. has more than one meaning.

In our computation a schema element can be composed by more terms, one of which is marked as the "main" one, i.e. the one bearing most of the semantics and each term can be associated with one or more synsets. Formally we define, for each element the set of terms associated to as $\{t_1, ...t_n\}$ and for each term t_i the corresponding set of senses as $\{s_{i_1}, ...s_{i_h}\}$. Figure 4 schematically represents this situation: different annotation levels for a generic element of one schema.

Example 3.1 *Following our reference Example (Figure 2), concepts* `productCategory` *and* `productFamily` *are both composed by two terms, "product" and "category" and "product" and "family", respectively, with "category" and "family" as main terms. Further we will suppose they are both annotated using two synsets. In particular* `category` *is annotated as "a collection of things sharing a common attribute" and as "a general concept that marks divisions or coordinations in a conceptual scheme", while* `family` *is annotated as "a social unit living together" and as "a collection of things sharing a common attribute".*

In the schema parsing phase it is necessary to extract all the different annotations and properly keep them to be used during the similarity calculation between concepts.

3.2 Schema pre-processing

The phase of schema pre-processing is very simple and consists of two different steps:

1. *graph extraction*;

2. *property inference*;

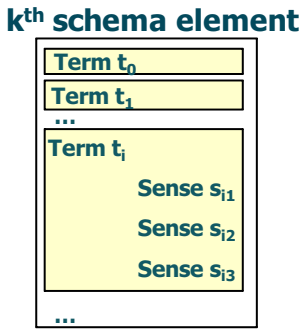


Figure 4: Annotation format for the k_{th} schema element

3.2.1 Graph extraction

This phase consists of the extraction of a direct labeled graph from each of the considered schemas. If they are well-formed RDF/OWL schemas, this step is very simple. In fact what we want is a graph structure as general as possible in which both classes and user-defined properties are nodes while edges can be only standard RDF constructs.

Single nodes have not a real meaning if considered as lonely entity but they are associated to their right names using a *label* arc: if we analyze only the node we can not distinguish if it represents a class or a user defined property. To define relationships between schema elements we use RDF/OWL construction, like, for example, *rdfs:subClassOf*, *rdfs:subPropertyOf*, *domain* and *range* which became the graph edges. This is the reason why if we have a well-formed RDF schema this phase is so simple. Figure 5 shows a portion of a original schema format (Figure 5(a)), as we can represent it just read from a file, translated into the correspondent standard labeled graph (Figure 5(b)) just described.

As the big amount of RDF construction types we performed tests to estimate matching effectiveness using only subsets of them. In particular we noted that best performances were obtained using only:

- subPropertyOf;
- subClassOf;
- label;
- type;
- domain;
- range;

For this reason our graph can have at most these arcs types. It is possible, in fact to restrict the range of possible properties choosing them before running the algorithm.

3.2.2 Property inference

Before skip to the basic matching computation, we perform inferences on the schema properties. In particular, subClassOf and subPropertyOf arcs are analyzed because, having a

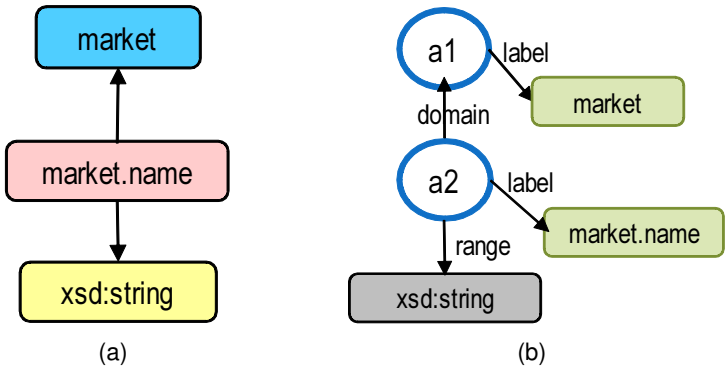


Figure 5: Representation of a portion of the original schema, as we can read from a generic input file (a), with the corresponding portion of the direct labeled graph created (b).

such that chain like that in Figure 6(a) in the schema, it is possible to define relationships in order to connect all the involved chain concepts.

As we can see in the Figure 6(b) we created three new arcs that connect class c_3 with class c_1 and class c_4 with classes c_2 and c_1.

These new relationships don't provide any additional information but they can be useful in the similarity calculation between two concepts because, as we can see in the next section, it is performed analyzing similarity between near elements (*neighbors*). As subClassOf and subPropertyOf are relations expressing concept specialization it is absolutely rightful to consider as neighbor of one chain element all the other ones.

A further pre-processing operation that is a part of the property inference procedure, concerns only subPropertyOf arcs. Imagine a situation like that of Figure 7(a) in which property p has class c as domain and represents a super-property for p_1, p_2 and p_3. This means that relations

1. (p_1, subPropertyOf, p);
2. (p_2, subPropertyOf, p);
3. (p_3, subPropertyOf, p).

are defined in the schema. Even if it should not appear formally correct, it is allowed not to define the sub-property domain. In this case, it is possible to perform an inference operation by which three new domain arcs, that connect respectively p_1, p_2 and p_3 to c, are created.

Two clarifications must be provided about that. The first one is that this inference is not allowed in the opposite case wherein a domain relationship is defined for sub-properties but not for super-property. Second the procedure does not chance if we know the range instead of the domain or even both of them.

3.3 Matching computation

In this phase our goal is to find all the possible matches between single classes and properties starting from the expanded annotated schemas.

Our matching technique is inspired by the Similarity Flooding approach by Garcia-Molina et al [7]. The algorithm works starting from two direct labeled graphs that, as said in Sec.

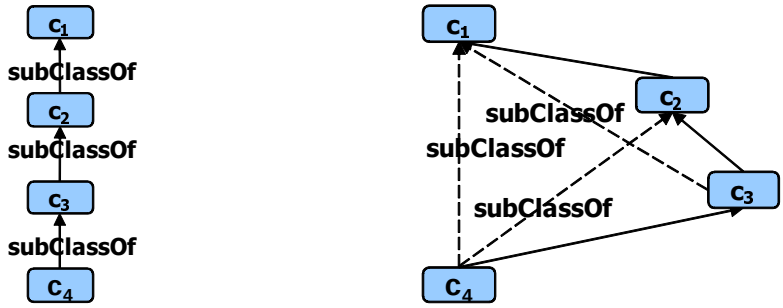


Figure 6: Property inference for a subClassOf arc chain

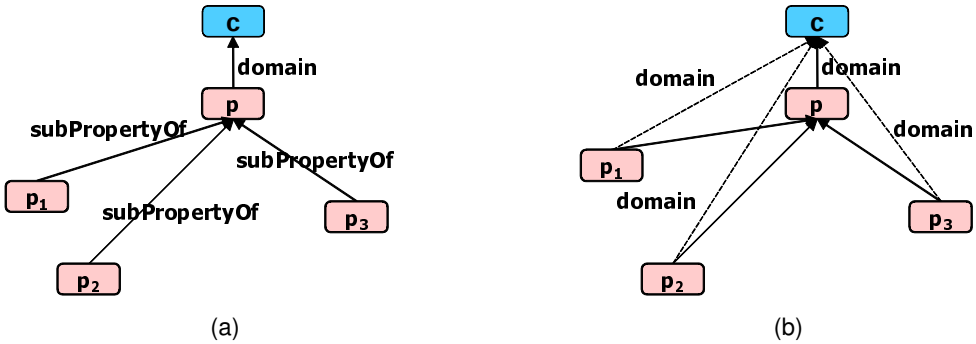


Figure 7: Property inference related to subPropertyOf arcs with undefined domain or range

3.2.1, represent both classes and user-defined properties as nodes. Graphs are used in a iterative fixpoint calculation whose results tell us which nodes in the first one are similar to nodes in the second one. The two direct labeled graphs are respectively:

$$G_1 = (V_1, E_1) \qquad G_2 = (V_2, E_2)$$

where V_1 and V_2 are vertex sets and E_1 and E_2 are edge sets.

Starting from G_1 and G_2 an auxiliary structure is created: the *pairwise connectivity graph* (PCG). A PCG is defined as:

$$((x, y), p, (x', y')) \in PCG(G_1, G_2) \Leftrightarrow (x, p, x') \in G_1 \, and \, (y, p, y') \in G_2$$

Each node in the connectivity graph is an element from the cartesian product between nodes from G_1 and nodes from G_2. We call such nodes *map pairs*. The intuition behind arcs that connect map pairs *(x, y)* and *(x', y')* is that if *x* is similar to *y* then probably *x'* is somewhat similar to *y'*. This is evidenced and captured by the property *p* that represent an edge in the connectivity graph leading from *(x, y)* and *(x', y')*. For this reason *(x, y)* and *(x', y')* are *neighbors*.

The PCG is then extended: for every edge a new one going in the opposite direction is added. Weights are placed on each arcs in order to indicate how well the similarity of a given map pair propagates to its neighbors and back. These so-called *propagation coefficients* $\in [0, 1]$ can be calculated in many different ways. For example, most of the time in our calculation, we use an average approach based on the intuition that each edge type makes

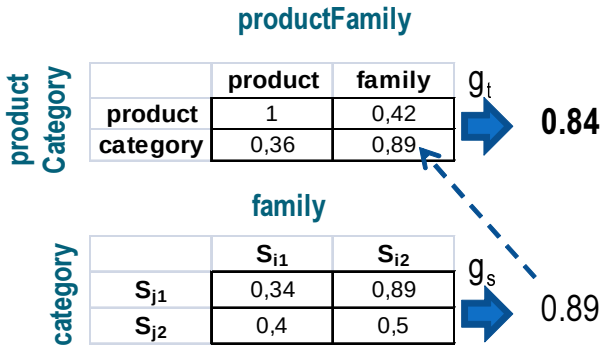


Figure 8: Similarity table for concepts `productCategory` and `productFamily` and for single terms `category` and `family`

an equal contribution of 1.0 to spreading of similarities from a given map pair. For this reason the weight of a certain type *p*-edge leaving from a map pair *(x, y)* is given by:

$$\frac{1.0}{number\,of\,that\,type\,p_i\,edge\,leaving\,from\,(x,y)}$$

Each PCG node composed of two literals is assigned an initial score quantifying the semantic similarity between the involved resources. As said in Section 3.1 given $n_1 \in G_1$ and $n_2 \in G_2$, we can have:

- $\{t_1^1, ...t_n^1\}$: term set associated to n_1 and $\{t_1^2, ...t_m^2\}$: term set associated to n_2
- $\{s_{i_1}, ...s_{i_h}\}$: sense set associated to t_i^1, $i = 1, ...n$ and $\{s_{j_1}, ...s_{i_k}\}$: sense set associated to t_j^2, $j = 1, ...m$

The idea is to define a way to compute the similarity between two senses s_{i_l} and $s_{j_{l'}}$, then to perform a two stage aggregation in order to obtain first similarity between terms (t_i^1 and t_j^2) and then between relative nodes.

Three methods can be used to compute initial similarity between different senses:

1. *Path-based similarity*: exploits WordNet hypernym/hyponym hierarchies between synsets;
2. *PageRank based similarity* [A. Esuli & F. Sebastiani, 2007];
3. *WordNet::similarity*.

In our computation we use the first one: scores for each pair of senses (s_{i_l} $s_{j_{l'}}$) are obtained by computing the depths of the synsets in the WN hypernyms hierarchy and the length of the path connecting them as follows:

$$\frac{2 \cdot depth\,of\,the\,least\,common\,ancestor}{depth\,of\,s_{i_l} + depth\,of\,s_{j_{l'}}}$$

The first aggregation stage is characterized by the function g_s which allows to pass from the single sense similarities $\{sim(s_{i_l}, s_{j_{l'}})\}$, $l = 1, ...h$, $l' = 1, ...k$, $i = 1, ...n$ and $j = 1, ...m$, to term similarity $sim(t_i^1, t_j^2)$. The second step, otherwise, involves the function g_t that, starting

from terms, returns the similarity between nodes ($sim(n_1, n_2)$). In our computation we chose the *max* for g_s and a *weighted mean* for g_t but any aggregation function is possible.

Focus on g_t the weight are represented by:

1. a highest coefficient used for similarity between main terms to give them the most importance;
2. a lowest coefficient used for similarities involving only one of the two main terms because most probably such couples are not significant;
3. an intermediate coefficient used for all the other similarities which can help in identifying concepts context.

Example 3.2 *Figure 8 shows an exemplification of the process for computing the initial similarity between the concepts* `productCategory` *and* `productFamily` *considered in Example 3.1. The final value (0.84) derives from the aggregation g_t of the similarities between all term pairs (the values shown in the upper table). For instance, the similarity between the main terms, which in this case are* `family` *and* `category`*, is 0.89. This derives in turn from the aggregation g_s of all the different senses associated to such terms (lower part of the figure).*

Initial similarities are then refined by a *iterative fixpoint calculation* as the basic assumption is that whenever any two nodes in G_1 and G_2 are found to be similar, the similarity of their adjacent elements increases. On the other hand that similarity decreases if they have very different neighbors. Thus, over a number of iterations, the initial similarity of any two nodes propagates through the graphs. Let $\sigma(x, y) \geq 0$ be the similarity measure of nodes $x \in G_1$ and $y \in G_2$. We refer to σ as a *mapping* whose values are calculated in a iterative computation: Let σ^i denote the mapping at the i^th iteration while σ^0 represents initial similarities, in every step, the σ-values for a map pair (x,y) are incremented by the σ-values of its neighbor pairs in the propagation graph multiplied by the propagation coefficients on the edges going from the neighbor pairs to (x,y). In general, mapping σ^{i+1} is computed from mapping σ^i as follows:

$$\sigma^{i+1}(x,y) = \sigma^i(x,y) + \sum_{(a_u,p,x)\in G_1,\,(b_u,p,y)\in G_2} \sigma^i(a_u,b_u)\cdot w((a_u,b_u),(x,y)) + \sum_{(x,p,a_v)\in G_1,\,(y,p,b_v)\in G_2} \sigma^i(a_v,b_v)\cdot w((a_v,b_v),(x,y))$$

where $w((a_u, b_u), (x, y))$ is the propagation coefficient on the graph edge that goes from (a_u, b_u) to (x, y). In each of such steps it is required to normalize σ^i for example divided all for the maximal σ^i-value of the current iteration.

This computation is performed iteratively until the Euclidean length of the residual vector $\Delta(\sigma^n, \sigma^{n-1})$ becomes less than a certain ϵ for some $n > 0$. If the computation does not converge, we terminate it after some maximal number of iteration or a certain time.

The resulting similarities of the PCG nodes give rise to a *multimapping* representing all the possible matches between nodes so it contains many potentially useful mappings as subsets. Single matches are expressed as $\{(n_1, n_2, sim) \mid n_1 \in G_1, n_2 \in G_2, sim \in [0, 1]\}$ where the score sim is fundamental to extract the more relevant mapping. Each node of

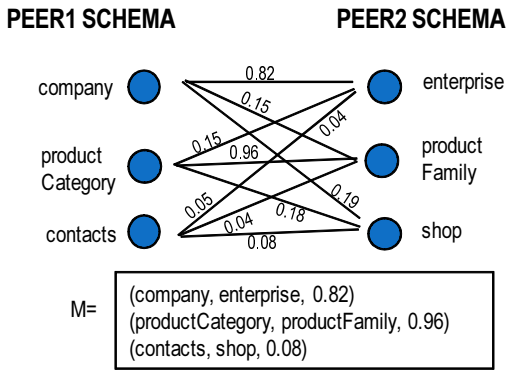


Figure 9: Example of multimapping

one graph can be associated to more than one node of the other but they must be of the same category, i.e. both classes or both properties. Despite it could be too limitative in our computation we exclude the class-property match because otherwise a lot of problems would be coming out during the query rewriting procedure that would have loose its general features. We have decided so as the matching phase is not an end in itself but it is targeted for the query rewriting that must be consistent and efficient.

Such "basic" matches will be refined to the final mapping in the subsequent refinement phase (Section 3.4). They will also constitute the starting points for the final matching phase (Section 3.5), which will be able to discover advanced matches, specifically the ones where properties match compositions of concepts (paths) rather than single properties. In this way, through the different phases we are able to deal with gradually more complex matching problems, thus allowing the more difficult and possibly error-prone matching evaluations to be supported by all the previously computed results.

Example 3.3 *The upper part of Figure 9 depicts a small portion of the multimapping for source classes* `company`, `productCategory` *and* `contacts`. *From these correspondences, a mapping M (lower part of the figure) will be extracted in the next matching phase.*

3.4 Refinements

The refinements phase takes as input the multimapping returned from the previous phase and, using the list of ranked map pairs, it extracts a consistent and correct mapping between, first, single classes and, second, single properties. Such order is fundamental because, as in our Semantic Web project scenario classes bear the largest part of the schema meaning, their correspondences will became the bases on which to find the following property and path matchings. This also allows query reformulation, which represents the major final goal of our matching computations, to be kept light while always performing naturally consistent and correct rewritings.

3.4.1 Class - class matching

The goal of the first refinement step is to finalize the correspondences between classes. The fundamental step in this phase is represented by the stable marriage filter which produces

the best matching for our schemas taking as input the multimapping. It bases on the intuition provided by the so-called *Stable marriage problem*. In one of its instances, each of m men and w women lists the members of the opposite sex in order of preference. A stable marriage is defined as a complete matching of men and women with the property that there are no two couples (x, y) and (x', y') such that x prefers y' to y and y' prefers x to x'. For obvious reasons, such a situation would be regarded as unstable. Naturally, in our computation, men represent source schema elements and women represent target schema elements.

The problem of this approach is that the stable marriage is able to find only 1:1 matches, while, in our experience we often need to have 1:N correspondences. In fact, as we can see in Figure 10(a) it is possible, and even frequent, that a class in one schema corresponds to more classes in the other one. This is evident in the multimapping because those pairs present very close, or ever equal, similarity values as appear for `(market, store)` and `(market, shop)` of the example. For this reason a threshold τ_{DH} (*Dead-heat*)is defined in order to create equivalence classes that share a very close similarity. Taking a (c_1, c_2, sim) that is a class-class match, if an element (c_1, c_2', sim') exists such that $sim - sim' \leq \tau_{DH}$, they are put in the same equivalent class that presents as comprehensive similarity value the maximum one. This became the real stable marriage input, that so, allow to find 1:N correspondences.

At this point of the computation we have the best matching but it may contains couples of concepts with a vary low similarity and so, not relevant for our purpose. For this reason a *pruning threshold filter* can be applied in order to extract only significant results. It simply consists of the definition of a pruning threshold value τ_P and analyzing the stable marriage output: only couples that have $sim > \tau_P$ are selected for the final mapping.

Example 3.4 *With regards to Example 3.3, the output mapping fragment M is shown in the lower part of Figure 9. In particular, the two couples are the direct output of the stable marriage filter. Notice that* `contacts`*, which has no correspondence in the target schema, correctly does not appear in M due to the application of the threshold filter (threshold 0.2).*

Refinements parameters can be changed to run different tests, but they especially can be excluded from the computation in order to measure how much they are essential in the matching computation effectiveness.

3.4.2 Property - property matching

The second refinement step involves the property matching results extraction. As already discussed, at this point we assume correctness of the class mappings, so we can start from this to define the property mappings. Such procedure is absolutely analogue to the class one but in this case not all the property are accepted. In particular a property $p_1 = (l_p^1, c_d^1, c_r^1)$ can be mapped to a semantically related property $p_2 = (l_p^2, c_d^2, c_r^2)$ only if (c_d^1, c_d^2) and (c_r^1, c_r^2) are both class mappings.

Example 3.5 *Let us have a look at Figure 10(b) and in particular at properties* `company.location` *and* `market.location` *(source schema on the left) and* `enterprise.mainLocation` *(target one). Even if their main term ("location") and annotations are the same, the only right map pair is* `(company.location, enterprise.mainLocation)` *and this is also the only one which is allowed, as evidenced by the final mapping. In particular, the presence of* `(company, enterprise)`

and `(location, location)` *in the class mapping satisfies the check, while, for instance, the absence of* `(market, enterprise)` *in the mapping rejects* `(market.location, enterprise.mainLocation)`.

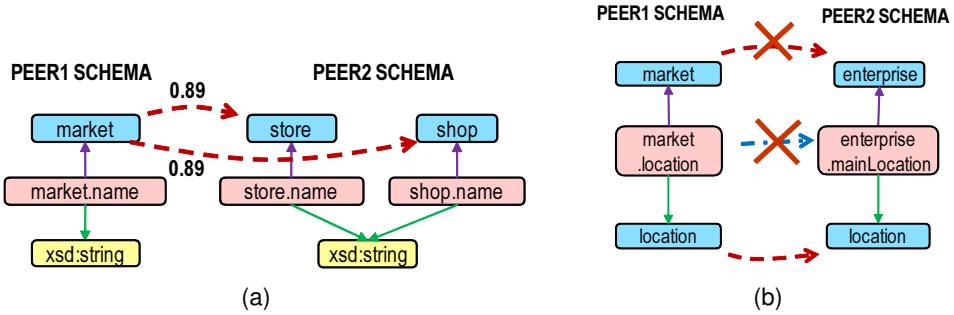


Figure 10: The left figure shows an example of 1:N matches while the right one shows a previously property exclusion

3.5 Path Matching Computation

The goal of this last phase is to enhance the final matching completeness by enriching it with possibly advanced and "mixed" matches which could not be discovered by the basic matching algorithm. In particular, in the most general case, we are now ready to compute correspondences between a property and a path, i.e. a composition of concepts. To do that we assume the completeness and correctness of both classes and properties mappings obtained from previous phases but we know that this could not be enough: there may still be unmatched properties, since there may be more subtle correspondences between schemas. This is a challenge for OWL schema matching right now, so we propose a specifically devised semi-automatic approach exploiting different techniques based on label similarity, automatic semantic path categorization and user feedback learning. Such approach can provide more and more precise suggestions about possible complex mappings.

The first step for this method consists of the identification of all those properties which have been excluded during previous phases. For each unmatched property $p^1 = (l_p^1, c_d^1, c_r^1)$ in source schema it is necessary to:

1. extract from the mapping, if exist, classes c_d^2 and c_r^2 of the target schema, related to c_d^1 and c_r^1;
2. find all the possible paths that connect c_d^2 and c_r^2.

Having a property and a series of possible corresponding paths we apply two independent methods:

- *Label similarity*;
- *Category confidence* from category pattern learning.

They are run independently from each other and both return a ranking of the schema paths which is often different from the other one. For this reason a ranking fusion is necessary in order to have the best correspondences considering both results.

3.5.1 Label similarity

The label similarity technique provides a first method to coarsely evaluate path correspondences without the need of any previous user intervention. It consists of the global evaluation of all the similarities between the source schema property p^1 and each single component (class or property) of a target schema path. We denote with S the sum of all these similarities normalized by the length of the path. In particular, given the k^{th} path $(p_1^2, ...p_h^2)$ going from c_d^2 and c_r^2 we can define an S^k label similarity value for each of them as:

$$S^k = \frac{\sum_{i=1,h} sim(p^1, p_i^2) + \sum_{i<h} sim(p^2, c_{r_i}^2)}{l}$$

with l the number of properties and classes of that path, excluding c_d^2 and c_r^2. Similarities are extracted directly from the basic matching computation output as we need information that can not be contained into results coming out refinement procedure. Notice that normalizing the S-value using the length of the considered path helps us in avoiding that very long chains of concepts, which rarely are significant for our scope, become part of the mapping. Once having obtained a S-value for each of the target schema path it is necessary to create the ranking for them. To do that we simply order S-values, and consequently paths related to, from the bigger to the smaller one. It is possible also to apply a sort of threshold filter in order to immediately exclude from the ranking paths with too low similarity with p^1 to be considered relevant. The obtained result is the *Similarity label ranking*.

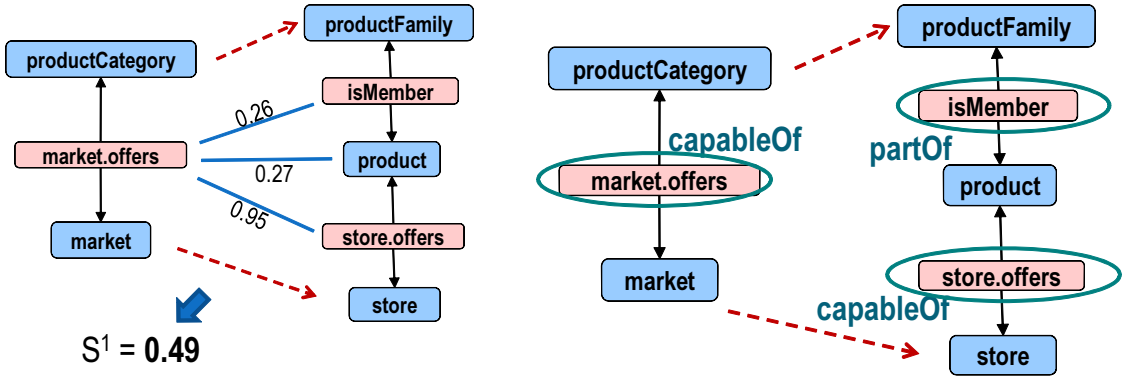
Example 3.6 *Let us consider property* `marketOffers` *in the source schema (Figure 2(a)), for which no corresponding single property exists in the target schema. Indeed, it should correspond to the path* `isMember-product-storeOffers` *or equivalently to* `isMember-product-shopOffers` *of the target schema (Figure 2(b)). Figure 11(a) shows the label similarity computation involved in this case.*

3.5.2 Category confidence

In order to enhance the matching suggestions' effectiveness, we complement the label similarity approach with a learning one, which is able to learn from users feedback. The initial aim is to classify user-defined properties using a defined set of categories as we are interested in limiting the infinite set of possible properties so to be able to create significant statistics.

In our project scenario, we perform categorization w.r.t. the *Global Mind knowledge base* (GM) [3], a database containing lots of current English language sentences expressed as triples in the form of *subject + verb + complement* and categorized into 20 groups, as, for example:

- LocationOf
- CapableOf
- UsedFor
- PartOf
- ...



(a) Computation of the label similarity value S related to property p_a and one of the possible corresponding paths.

(b) Example of property categorization using Global Mind.

Figure 11: Different steps in the semi-automatic approach to find matching between a property and a path.

In our Category confidence approach we first have to categorize the source schema property p^1 using Global Mind, analyzing its similarity with one of the sentences contained. Then this is done also for all concepts composing all the target schema paths but in this case we keep an internal representation of each path as a sequence of Global Mind categories, call them *category paths*. We also introduce a sort of approximation in the categorization process. In particular, a maximum score is assigned if a sentence exists corresponding exactly to (c_d, p, c_r), while a lower score is given if only two or, possibly, one of the three concepts are present. Further, a path-based similarity like the one used in the basic matching computation can be applied to further enhance the categorization effectiveness.

In correspondence of a given property category and of the possible path categories found out in the source and target schemas, the algorithm internally keep up to date some statistics accounting the number of times such correspondence appears (*Nfind*) and the number of times it has been effectively chosen by user (*Nright*).

Let us now analyze how path matching is performed for a given source schema property p^1. The first time all $Nright = 0$ but they are updated as user make a decision on which match is the right one. The algorithm starts presenting all the possible corresponding target schema paths ordered according only to the label similarity ranking. User must choose one of them (if any) for the final mapping and, basing on his decision, statistics are updated increasing the related Nright value. During the computation for a subsequent property of the same category, a so called *category confidence level* (CL) is computed as

$$CL = Nright/Nfind$$

for each associated category pattern found in the statistics. On the basis of CL, paths are sorted on a *category confidence ranking* and this means that paths are ordered upon the matching most probable candidate for that property category.

This is a learning procedure as the algorithm is able to get knowledge from user's decision, to applying it to category patterns and to provides more and more precise mapping suggestions as path ranking. On the other hand this absolutely remains a semiautomatic approach as the algorithm returns just tips and user must take the final mapping decision.

PEER1 SCHEMA	PEER2 SCHEMA	Nright	Nfind
capableOf	partOf capableOf	8	11
…	…	…	

CL = **0.73**

Figure 12: Category confidence level computation

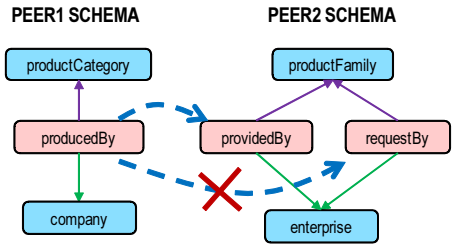


Figure 13: Two possible path that connect two concepts in Schema B. The algorithm is able to decide which of them is the right correspondence for the Schema A property.

3.6 Output result

At the end of the computation the matching algorithm produces a result similar to the one shown in table 1. This is the final mapping which contains all the relevant couples of similar concepts from the two schemas. As we can see in the first and in the third row of table 1, this algorithm is able to find semantic correspondences between terms annotated only in a similar way, for example using synonyms. This is very important in order to provide an efficient and robust mechanism able to hack it heterogeneity of schemas. Moreover if we have a situation like that in Figure 13 where we have two couples of classes in the final mapping but, during the property matching we find that target schema classes are connected by two or more paths (composed of only one property). The algorithm can extracts only the correct match between for the source schema property.

The matching algorithm produces also an xml file containing all the information about the final mapping. This will be read by the query rewriter in order to understand concept similarities to perform its job.

The file is composed by different sections: the first one is a description of the original source and target schemas as a simple list of classes and properties:

SCHEMA A	SCHEMA B	SIM
company	enterprize	0.81838
location	location	0.95719
market	store	0.81556
market	shop	0.81556
producedBy	proviededBy	0.76328
company.location	enterprize.mainLocation	0.89398
company.location	enterprize.otherLocation	0.89398
market.offers	isMember product store.offers	0.65285
...	...	...

Table 1: Example of matching algorithm result representing only correct and more significative mapping couples. First three rows correspond to class matching while the other are about property matching. The last column (SIM) shows similarity values between concepts.

$< schemaInfo >$
$< schema >$ Schema1.rdf $< /schema >$
$< nodeInfo >$
$< path >$ http://www.nep4b.owl#company $< /path >$
$< /nodeInfo >$
$< nodeInfo >$
$< path >$ http://www.nep4b.owl#market $< /path >$
$< /nodeInfo >$

Then it follows the class mapping set:

$< schemaMatch >$
$< schema1 >$ Schema1.rdf $< /schema1 >$
$< schema2 >$ Schema2.rdf$< /schema2 >$
$< nodeMatch >$
$< node1 >$ http://www.nep4b.owl#company $< /node1 >$
$< node2 >$ http://www.nep4b.owl#enterprise $< /node2 >$
$< sim >$ 0.81838 $< /sim >$
$< /nodeMatch >$

and the property mapping set:

$< nodeMatch >$
$< node1 >$ http://www.nep4b.owl#company.location $< /node1 >$
$< node2 >$ http://www.nep4b.owl#enterprise.mainLocation $< /node2 >$
$< sim >$ 0.89398 $< /sim >$
$< /nodeMatch >$

4 Semantic Peers

In this section, we introduce the basic concepts routing mechanism relies on, although routing is not properly a point of this document. We do that because we are interested in defining some aspects, like semantic path and generalized semantic mapping, directly related to what we have already discuss about.

These concepts are defined in a fuzzy theoretical framework. Fuzzy set theory has been widely applied in contexts where uncertainty of description is intrinsic in the nature of data, most notably in the case of multimedia data [4, 8]. We deem that these principles can provide a valid support to deal with the semantic approximation originated by the heterogeneity of the schemas in a PDMS.

4.1 Semantic path

A semantic path [6] is a chain of semantic mappings connecting a given pair of peers. Through the reformulation of a query along the mappings composing a semantic path, the PDMS can access data on remote peers. As a local semantic mappings may involve semantic approximations, the semantic approximation given by a semantic path can be obtained by composing the fuzzy relations understood by the involved mappings. This relies on the notion of *generalized composition* of binary fuzzy relations [9]: given a t-norm[1] I and the semantic mappings, $M(S_i, S_j) \subseteq S_i \times S_j$ and $M(S_j, S_k) \subseteq S_j \times S_k$, the I-composition of $M(S_i, S_j)$ and $M(S_j, S_k)$ is the semantic mapping $M(S_i, S_j) \circ^I M(S_j, S_k) \subseteq S_i \times S_k$. This is defined by: $[M(S_i, S_j) \circ^I M(S_j, S_k)](C, C'') = I[M(S_i, S_j)(C, C'), M(S_j, S_k)(C', C'')], \forall C \in S_i, C'' \in S_k, with C' \in S_j$.

After the definition of composition of mapping we can pin down on a semantic path. Given a t-norm I and $< M(S_1, S_2), ..., M(S_{k-1}, S_k) >$ that is a sequence of mappings connecting peer p_1 with peer p_k, the path $P_{p_1 ... p_k} \subseteq S_1 \times S_k$ is the *semantic mapping* $M(S_1, S_2) \circ^I ... \circ^I M(S_{k-1}, S_k)$. The composition function should capture the intuition that the longer the chain of mappings, the lower the grades, thus denoting the accumulation of semantic approximations given by a sequence of connecting peers. In order to obtain such effect of semantic attenuation due to the chain of mappings from C_1 to C_k in the schema of a peer p_k which is far away form p_1, a possible choice for the t-norm I is the $algebraic\ product\ I(\mu, \mu') = \mu * \mu'$. In fact, given that the arguments are grades in [0,1], their algebraic product is still in [0,1] and it is lower than or at most equal to its arguments.

4.2 Generalized semantic mappings

The query execution process starts from the querying peer which reformulates the query over its immediate neighbors, than over their neighbors and so on. Thus, from a multi-step reformulation point of view, whenever a query posed over peer p_i is reformulated over p_j, the query is moving from p_i to the subnetwork rooted at p_j and it might follow any of the semantic paths originating at p_j. In order to model the semantic approximation of the p_j's subnetwork w.r.t. the p_i's schema, the semantic approximation given by each path in the p_j's subnetwork are aggregated into a measure reflecting the relevance of the subnetwork as a whole.

[1] A t-norm I is a binary operation on [0,1] that is monotone, commutative, associative and it satisfies the boundary condition $I(a, 1) = a \forall a \in [0, 1]$.

To this end, the notion of semantic mapping is generalized as follows [6]. Let $p_j^{\triangle}$ denote the set of peers in the subnetwork rooted in p_j, $S_j^{\triangle}$ the set of schemas $\{S_{j_k}|p_{j_k} \in p_j^{\triangle}\}$ and $P_{p_i...p_j^{\triangle}}$ the set of paths from p_i to any peer in $p_j^{\triangle}$. The generalized mapping relates each concept C in S_i to a set of concepts $C^{\triangle}$ in $S_j^{\triangle}$ taken from the mappings in $P_{p_i...p_j^{\triangle}}$, according to an *aggregation score* which expresses the semantic similarity between C and $C^{\triangle}$.

Given two peers p_i and p_j, not necessarily distinct, and an aggregation function g, we can formally define a *generalized semantic mapping* between p_i and p_j as a fuzzy relation $M(S_i, S_j^{\triangle})$ where each instance $(C, C^{\triangle})$ is such that:

- $C^{\triangle}$ is a set of concept $\{C_1, ..., C_h\}$ associated with C in $P_{p_i...p_j^{\triangle}}$
- $\mu(C, C^{\triangle}) = g(\mu(C, C_1), ..., \mu(C, C_h))$.

As to the function g, the following properties, which express the essence of the notion of aggregation [9], must hold:

1. *g* is *monotonic increasing* in all its arguments;
2. *g* is a *continuous* function;
3. *g* respects the *boundary conditions* $g(0, ..., 0) = 0$ and $g(1, ..., 1) = 1$;
4. *g* is a *symmetric* function for all its arguments;
5. *g* is a *idempotent* function, that is, $g(a, ..., a) = a \forall a \in [0, 1]$.

The last two conditions are usually expected when one refers to aggregating operations on fuzzy sets. Several choices are possible for g, for instance function such as the min, the max, any generalized mean or any ordered weighted averaging function.

5 Multimedia Data Indexing

5.1 Multimedia Routing Index

In this section, we describe our proposal of Multimedia Routing Index that supports similarity search over sets of multimedia attributes for content-based retrieval.

This problem of similarity searching can be formalized by the mathematical notion of *metric space*, so data elements are assumed to be objects from a metric space domain where only pairwise distances between the objects can be determined by a respective distance function.

More formally, a metric space is defined by a domain of objects $\mathcal{D}$ (elements, points) and a distance function d – a *non-negative* and *symmetric* function which satisfies the *triangle inequality* $d(X,Y) \leq d(X,Z) + d(Z,Y), \forall Z, Y, Z \in \mathcal{D}$. The focus of this paper is on similarity-based *Range Queries* over multi-feature metric objects, defined as follows: given a database $D \subset \mathcal{D}$ of objects, a query object $Q \in \mathcal{D}$, and a positive range $r \in \mathbb{R}$, a query $\mathcal{Q} = (Q, r)$ has to retrieve the set $\{X \in D \mid d(X,Q) \leq r\}$.

In the rest of this paper we use the following conventions. We consider a P2P network composed by N peers. Each peer P knows a a set of other peers directly connected to it. This set is called the *neighborhood* of P and is denoted with *Nb*(P). Moreover, P owns a set of *data objects*, called *local repository*. We denote it with *Data*(P). We use the qualifier "data" in order to distinguish it from metric objects. Every data object O of such repositories is characterized by a set of multimedia features. A feature is a metric object extracted from the data object. For instance a data object could be a jpeg image, from which we can extract two features, e.g. the texture and the color-histogram. O^F indicates the value of feature F for O, and $\mathcal{D}^F$ its corresponding domain of metric objects. As done by other systems, we index data objects with the help of some reference objects, or $pivots$. Each feature has its own set of reference points. The cardinality of these sets may differs for different features. Supposing that we have m reference objects, we denote with $R^F = \{R_1^F, \ldots, R_m^F\}$ the set of such objects. Finally, in case of a single feature scenario, for sake of simplicity, we omit the subscript symbol F since it is clear form the context.

5.2 Network Organization

Routing Indices are thought for unstructured P2P networks. No organization in special shapes are required, in contrast to the DHT-based networks. Nonetheless, there exist one problem for RIs related the P2P system topology.

Routing Indices collect and summarize the information about objects located in a given portion of the network. If the network present a loop, the information about the peers involved in the loop could be duplicated. This behavior can be considered correct, since a node P involved int he loop can reach the objects owned by another peer P' following the loop in one sense or or in the other. Anyway, since RIs store only the information about object values and not about object's owners, a request could be sent along both sides of the loop only to reach the same node twice.

In order to avoid such a problem, we decided to keep inside each node a cache of the IDs of already received updates. Each information is maintained for a given time and is used to reject updates that are received twice or more from different neighbors. The result is that

updates about newly available objects are stored by a peer only for one side of a loop, thus avoiding to replicate object information and then improving the query forwarding process.

Given this mechanism for updating the indices, the information collected by each RI comes from a loopless part of the network. Thus, the index associated with a peer P_j of a node P represent the values of objects stored in a *logical* tree rooted on P_j. From now on, we refer to this logical tree as $\mathcal{T}(P \rightarrow P_j)$

5.3 Data Indexing

The data indexing process is intended to produce a summarized description of all the features of the objects owned by each peer in order to produce efficient multi-feature Routing Indices. The aim of these indices is to provide a concise but yet sufficiently detailed description of the resources available in a given network area. This information is then used at query-resolution time to prune entire system zones from searching, thus easing the search process.

Since we are considering objects in a metric space, we use reference objects and the triangle inequality as the base for building our Routing Indices. When dealing with metric objects, the triangle inequality property is exploited to build indices that allow us to select only some subsets of *candidate* objects, among which the objects relevant to the query can appear. These indices permit to save work: only for the candidate objects X we need to compute the distances $d(Q, X)$ in order to determine the relevant objects with respect to query $\mathcal{Q} = (Q, r)$.

Several indices have been proposed so far [2]. We will refer to the the well known technique of *reference objects* (or pivots) to transform each indexed object and query object into a vector of distances from the m reference objects $R = \{R_1, \ldots, R_m\}$. In particular, given $X \in D$, function $T : D \times R \rightarrow \mathbb{R}^m$ transforms x as follows:

$$T(X, R) = (d(X, R_1), \ldots, d(X, R_m))$$

Due to the triangular inequality property of metric spaces, using L^∞ we obtain a *contractive* mapping in the new vector space, such that $L^\infty(T(X), T(Y)) \leq d(X, Y)$.

Therefore, given $\mathcal{Q} = (Q, r)$, we have to check regions of our mapped space that include objects X such that
$L^\infty(T(Q, R), T(X, R)) \leq r$, i.e., such that $\max_{i=1}^{m}(|d(X, R_i) - d(Q, R_i)|) \leq r$.

Then $L^\infty(T(Q, R), T(X, R)) \leq r$ is a *necessary* condition for $d(Q, X) \leq r$, and thus for X to be a relevant objects for query $\mathcal{Q} = (Q, r)$. Unfortunately it is not a sufficient condition too. Therefore, for all the objects X such that $L^\infty(T(Q, R), T(X, R)) \leq r$, we need to check whether $d(Q, X) \begin{matrix} \leqq \\ \geqq \end{matrix} r$. However, thanks to the triangular inequality, we can prune all the objects for which
$L^\infty(T(Q, R), T(X, R)) > r$.

In conclusion, we need only to check regions of our mapped space $\mathbb{R}^m$ centered in $(d(Q, R_1), \ldots, d(Q, R_m))$, such that $[d(Q, R_i) - r, d(Q, R_i) + r]$ for each reference object R_i (i.e., for each dimension i of $\mathbb{R}^m$).

Practically , first we compute the vector
$(d(Q, R_1), \ldots, d(Q, R_m)) \in \mathbb{R}^m$, and then select all objects X, already mapped in $\mathbb{R}^m$, such that each component $d(X, R_i)$ of the vector associated with X falls into interval $[d(Q, R_i) - r, d(Q, R_i) + r]$.

The situation depicted above refers to object with a single feature. In presence of a multi-feature objects, we have to take into account that each feature is characterized by its own set of reference objects and, consequently, its own distance definition. Usually, the metric distance d between multi-feature objects is defined as a *linear combination* of the various metric distances d_j, where each d_j is defined on the basis of the *j–th* feature extracted from each object:

$$d(X,Y) = \alpha_1 d_1(X^{F_1}, Y^{F_1}) \ldots + \alpha_n d_n(X^{F_n}, Y^{F_n})$$

where $\alpha_j \in \mathbb{R}$, $\alpha_j \geq 0$, are scalar coefficients, and $X^{F_j}, Y^{F_j} \in \mathcal{D}^{F_j}$. If each component d_j of this linear combination is metric, it can easily be shown that also distance d is metric.In this way, we obtain a new (combined) metric space $\mathcal{D}$ that is the result of the fusion of n metric spaces (features) $\mathcal{D}^{F_1}, \ldots, \mathcal{D}^{F_n}$.

While we can always define an index on the basis of the total distance d, thus considering all the features extracted from the multimedia objects, one can also be interested in making query definition more flexible for the users. For example, to allow them to specify similarity-based queries that only consider a subset of all the features extracted from the multimedia objects, or to change the coefficient α_j of the linear combination defined above.

Unfortunately, in order to provide such flexibility, we have to define a separate index for each of the component distances d_j of the linear combination of d, thus mapping each single feature object into a separate vector of distances from m reference objects $R = \{R_1, \ldots, R_m\}$. Note that the reference objects, used to map the feature objects onto a vectorial space $\mathbb{R}^m$, can be the same for all the features, or can be different for each of the features considered. Since query $\mathcal{Q} = (Q, r)$ has to retrieve the set $Obj(Q,r) = \{X \in D \mid d(X,Q) \leq r\}$, but we do not have an index defined over d, we can only try to extract the candidate objects of each set $Obj_j(Q^{F_j}, r) = \{X^{F_j} \in D^{F_j} \mid \alpha_j d_j(X^{F_j}, Q^{F_j}) \leq r\}$. Only objects that are found to belong to all sets $Obj_j(Q^{F_j}, r)$ can belong to $Obj(Q,r)$.

Theorem 1 *$X \in Obj(Q,r)$, where $Obj(Q,r) = \{X \in D \mid d(X,Q) \leq r\} \Rightarrow \forall j$, $X^{F_j} \in Obj_j(Q^{F_j}, r)$, where*
$Obj_j(Q^{F_j}, r) = \{X^{F_j} \in D^{F_j} \mid \alpha_j d_j(X^{F_j}, Q^{F_j}) \leq r\}$.

Proof Since $d(X,Q) \leq r$, where $d(X,Q) =$
$\alpha_1 d_1(X^{F_1}, Q^{F_1}) + \ldots + \alpha_j d_j(X^{F_j}, Q^{F_j}) + \ldots + \alpha_n d_n(X^{F_n}, Q^{F_n})$, and each single member of this summation must not be negative ($\alpha_j \geq 0$, and also $d_j(X^{F_j}, Q^{F_j}) \geq 0$, due to the positiveness property of metric distance), then we can deduce that for all $j \in \{1, \ldots, n\}$, $\alpha_j d_j(X^{F_j}, Q^{F_j}) \leq r$ (i.e., $X^{F_j} \in Obj_j(Q^{F_j}, r)$). ■

Theorem 1 states that a necessary condition to have $d(X,Q) \leq r$ for a given object $X \in D$ is that $d_j(X^{F_j}, Q^{F_j}) \leq r$ for all feature objects $j \in \{1, \ldots, n\}$. Obviously, the last condition is not a sufficient one, so that, when we select all the objects $X \in D$, such that $\forall j$, $X \in Obj_j(q,r)$, many of them will not be relevant for the range query $\mathcal{Q} = (Q,r)$, because the linear combination $d(x,q)$ will result greater than r. Therefore, by building a separate index for each feature objects, we trade *flexibility*, since we allow users to determine which combination of features to consider when formulating query Q, for *additional work* in selecting the real relevant objects.

All the facts stated above are used to build the indices used in our system. As stated in the previous sections, we make use of Routing Indices for forwarding queries toward network

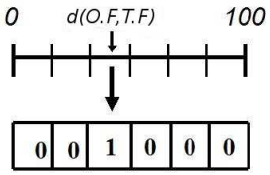


Figure 14: Creation of a single local index

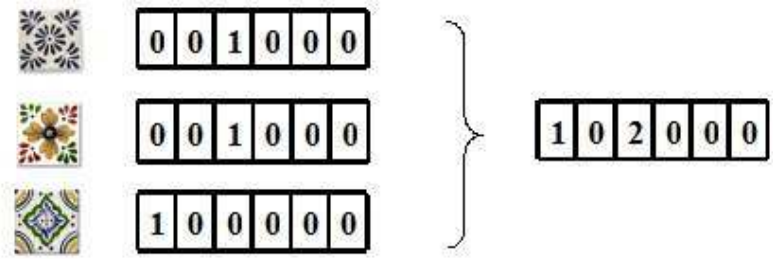


Figure 15: A peer's histogram for a reference object R_j

areas of potential interest. Routing Indices provide a concise representation of the objects available in a given zone of a P2P network. In order to achieve this goal, for each object O we need a different representation of O's features values. For each feature F_j and its set of associated reference objects $R^{F_j} = \{R_1^{F_j}, \ldots, R_m^{F_j}\}$, we encode the value $d_j(O, R_i^{F_j})$, $i = 1, \ldots, m$, with a k bit binary vector

$$\textit{DataIdx}(O)_{R_j^{F_j}} = (b_0, b_1, \ldots, b_{k-1}) \tag{1}$$

such that $b_c = 1$ if and only if $d(O^{F_j}, R_i^{F_j}) \in [a_c, a_{c+1})$. An example of such a representation is given in Figure 14. Both the parameter k (number of elements of the vector) and the division points $a_0, a_1, \ldots, a_k$ may be different for each reference object of each feature type. Thus, each feature is now characterized by a set of m bit vectors, one for each of the feature's reference objects. Each of these vectors shows which is the interval in which falls the real distance of the data object from a reference object.

Such simple object indices become useful to construct a concise description of all the data owned by a peer. The peer index, wrt a reference object R_i, is obtained by simply summing the indices of all the objects of the local repository of P:

$$\textit{NodeIdx}(P, F_j)_{R_j} \equiv \sum_{O \in \textit{Data}(P)} \textit{DataIdx}(O)_{R_i^{F_j}} \tag{2}$$

The final result is a vector that represent how the objects of the peer are distributed with respect to the feature of the reference object R_j, as illustrated in Figure 14.

The next step allows the construction of Routing Indices and thus the spreading of object information along the network. RIs are characterized by the fact that they represent a summarization of all the objects values that can be found in a given zone of a P2P network. Since the index of a peer's objects are already in a condensed form, we can exploit them to obtained the final RIs.

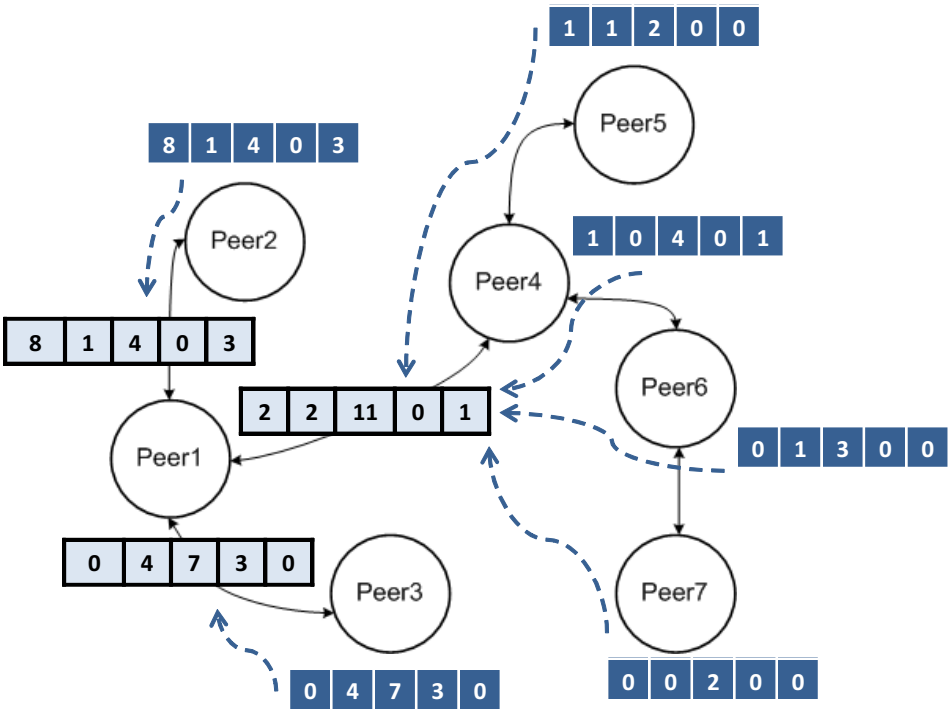


Figure 16: Construction process of the Multimedia Routing Index.

Let us consider a generic peer P_i. For each neighbor $P_j \in \mathit{Nb}(P_i)$, P_i keeps information on the data items which can be found by following the link $\{P_i, P_j\}$ on the overlay network. For this purpose, P_i maintains an index called $\mathit{LinkBitIdx}(P_i \rightarrow P_j, F)_{R_k}$ for each reference object R_k of each feature F of the data items in $\mathcal{T}(P_i \rightarrow P_j)$. Since P_i can receive information only from its neighborhood, the index is calculated recursively in the following way:

$$\mathit{LinkBitIdx}(P_i \rightarrow P_j, F_j)_{R_k^{F_j}} \equiv \mathit{NodeIdx}(P_j, F_j)_{R_k} + \sum_{P \in \mathit{Nb}(P_j) - P_i} \mathit{LinkBitIdx}(P_j \rightarrow P, F_j)_{R_k}$$

The final result is that the index contains the sum of all the vector indices $\mathit{NodeIdx}(P, F)_{R_k}$ associated with every peer in $\mathcal{T}(P_i \rightarrow P_j)$.

As it is clarified in the Deliverable D.4.3.1, this index is exploited during the query evaluation in order to drive the query to the peer that best match the query. In fact, since the MRI is composed of histograms of the distribution of objects with reference to every feature, it is possible to know the number of potentially matching objects. This possibility gives us the ability to perform different query forwarding strategies. One of this strategies consist in allowing a peer p, that has to forward a query, to select its neighbors on the basis of the number of potential matchings they have in their subnetworks. The selection is made using what we call the *requested coverage*, i.e. the neighbors the query is to be forwarded to must have a number of matchings that is equal or above a given percentage of all the possible matchings given by the routing indices. The aim of this process is to further reduce the number of query messages, while trying to collect the largest number of results.

6 Matching Experiments

In this section we present a selection of the results we obtained through the experimental evaluation performed on the matching procedure.

In the first subsection we discuss results relative to the current reference scenario for the NEP4B project while in the next one we analyze two different couples of schemas which are been devised ad hoc to precisely evaluate the behavior of different features of our approach.

6.1 Tourism Scenario

Tourism scenario is the current reference one for the NEP4B project. We have at our disposal three peers shown, respectively, in Figures 17, 18 and 19. They all contains a lot of concepts but their structure is very light. In the reported figures we decided to leave out arc names for sick of clarity but we have used different colors to distinguish them. In particular a purple arc represents a domain relationship, a green one shows a range relationship and a blue one is a subClassOf relationship. Object properties can be distinguished because they present both the domain and range arcs while for datatype properties, which have always a range in the rdf class "http://www.w3.org/2001/XMLSchema#string", we have decided to shown only domain arcs.

Most significant results are about matching between Peer1 and Peer3 and they are shown in Tables 2. Keeping constant the Dead-heat threshold value, we have varied the Pruning threshold in order to maintain only most significant results. It is clear that, as we can see in Table 1(a), without any pruning filter all matching couples are returned: both good ones and bad ones, although their similarity is too low. Let's see, for example, all inconsistent mappings involving Peer1 concept `hotels.email`: they present the same small similarity value and they are all wrong. As we can see later in Section 6.2, this is due to schemas structure. On the other hand good results are represented by, for example, the matching between the two classes (first two rows of the table) and, in general, can be recognized thanks to a similarity value grater than 0.13 in the third column of the table.

Let's look to Tables 1(b) and 1(c) to see that results are very good. In particular in the first case we applied a pruning threshold of 0.10 while in the second one its value was of 0.15. Those values must not be compared to those reported in the third column of tables because those final results are obtained by the computation of a relative similarity using results coming from the self matching after the filter application.

In Table 1(b) not all results are really significative because we can see two mapping involving Peer1 concept `hotels.fax` which are wrong together with matching between `hotels.web_site` and `hotels.single_room` and between `hotels.telephone` and `hotels.code`. Totaly we find 16 correct concept couples as in Table 1(a). This is a good result as it means that pruning threshold filter has excluded only wrong mappings keeping all the write ones. This is not true for the third case (see Table 1(c)) in which not relevant results just said are rule out but also with a significant couple represented by `restaurants.locality` and `restaurant.province`.

For this reason it must be careful to increase threshold filter because good results can be pruned although it is clear that a mapping refinement is necessary with this kind of schemas.

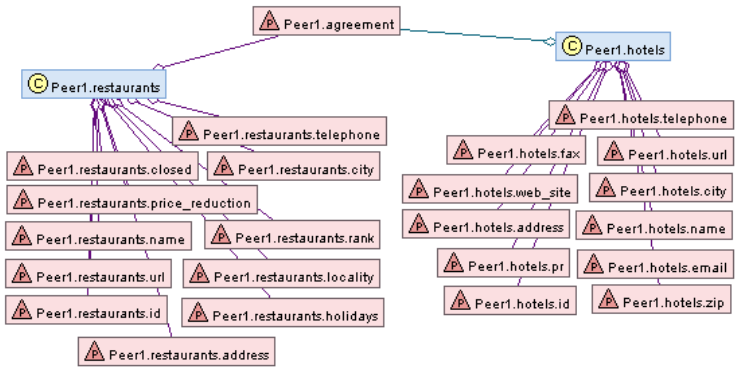


Figure 17: Peer1 schema

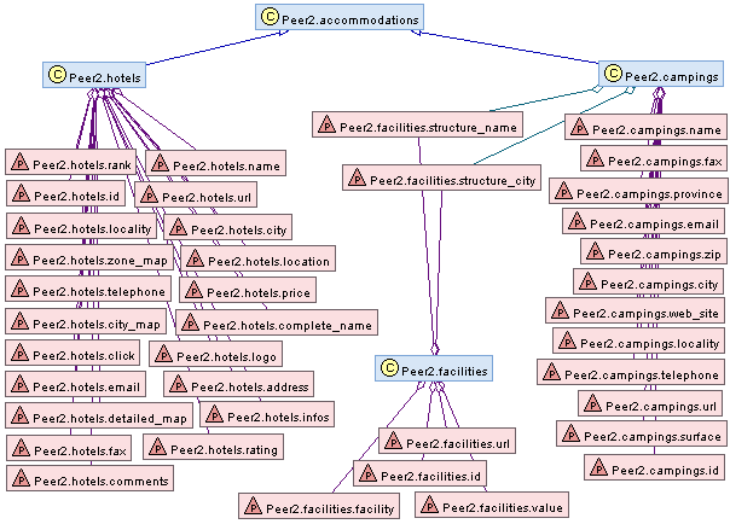


Figure 18: Peer2 schema

In appendix **??** analogues results are reported. In this case the matching computation involves Peer1 and Peer2 and results are not as good as in the previous description. If we look to Figure 17 and 18 we can see that Peer1 presents classes `hotels` and `restaurants` while Peer2 has `hotels`, `campings`, `accommodations` and `facilities` so, it's evident they have very different schema structure. Matching computation feels the effect of this difference and so results are those shown in Tables 5, 6 and 7. Experiments have been done exactly in the same manner as in the previous description three tables refers respectively to proves done without a pruning threshold, with a pruning threshold of 0.10 and of 0.15.

Results coming from the first proof are naturally the worst but the real problem is that classes `restaurants` and `campings` are matched together while it is not right. Unfortunately this appears in all results, although the increase of the pruning threshold, so while mapping is gradually refined, wrong results are keeping because of this uncorrect matching. Let's look to Table 7: all couples involving class `hotels` are right and all the other property matching are reasonable because they express, for example, the name, the url or the telephone but they refer to different aspects. This is a big problem in rewriting phase because it

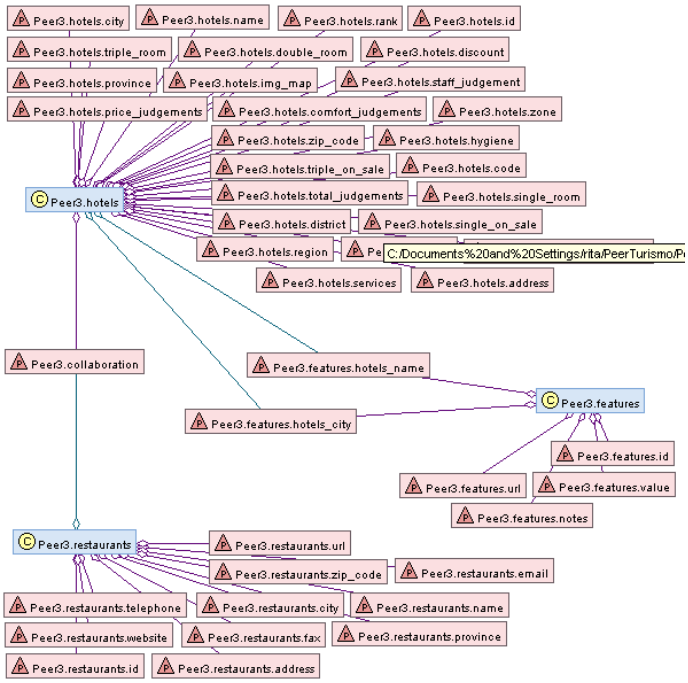


Figure 19: Peer3 schema

is not possible to distinguish consistent or not information.

6.2 Other examples

In this section we analyze two different couples of schemas which are been devised ad hoc to precisely evaluate the behavior of the matching algorithm when more complex situation are present in schema structure.

The first example involves rdf schemas presented respectively in Figures 20 and 21 of Appendix B and matching results are in Table 3. First 10 rows refer to class mapping while the others are about of property mapping. Differently from the Section 6.1, we present only mappings obtained without a pruning threshold filter because we found the same results both applying and not applying it.

The final mapping is very good for these two schemas also because they don't have a simple structure. All the relevant correspondences are found and none of them is wrong.

The second example involves schemas shown in Figures 22 and 23 of Appendix C. These are very simple and light but they present all the possible situations the matching algorithm must be robust to; for example:

- in Schema A both concepts `name` and `secondName` must correspond to the same concept `playerName` in Schema B;
- class `photo` in Schema A must match to `picture` although they are annotated with synonymous terms;

(a) $T_P = 0$ and $T_{DH} = 1e-3$

PEER1	PEER3	SIM
hotels	hotels	0.29493
restaurants	restaurants	0.30679
hotels.id	hotels.id	0.34542
hotels.zip	hotels.zip_code	0.37050
hotels.email	hotels.region	0.12430
hotels.email	hotels.district	0.12430
hotels.email	hotels.price_judgements	0.12430
hotels.email	hotels.single_on_sale	0.12430
hotels.email	hotels.double_on_sale	0.12430
hotels.email	hotels.services	0.12430
hotels.email	hotels.triple_room	0.12430
hotels.email	hotels.discount	0.12430
hotels.email	hotels.triple_on_sale	0.12430
hotels.email	hotels.rank	0.12430
hotels.email	hotels.zone	0.12430
hotels.email	hotels.total_judgements	0.12430
hotels.url	hotels.url	0.34542
hotels.fax	hotels.double_room	0.21877
hotels.fax	hotels.img_map	0.21877
hotels.fax	hotels.comfort_judgements	0.12430
hotels.city	hotels.city	0.34619
hotels.web_site	hotels.hygiene	0.12430
hotels.web_site	hotels.single_room	0.21877
hotels.web_site	hotels.staff_judgement	0.12430
hotels.address	hotels.address	0.34610
hotels.telephone	hotels.code	0.25746
hotels.name	hotels.name	0.37050
hotels.pr	hotels.province	0.37050
restaurants.price_reduction	restaurants.fax	0.11876
restaurants.name	restaurants.name	0.35053
restaurants.url	restaurants.url	0.55893
restaurants.closed	restaurants.fax	0.11876
restaurants.id	restaurants.id	0.34974
restaurants.locality	restaurants.province	0.20973
restaurants.address	restaurants.address	0.35050
restaurants.city	restaurants.city	0.58517
restaurants.telephone	restaurants.telephone	0.35207
restaurants.holidays	restaurants.website	0.11876
restaurants.holidays	restaurants.zip_code	0.12053
restaurants.holidays	restaurants.email	0.11876

(b) $T_P = 0.10$ and $T_{DH} = 1e-3$

PEER1	PEER3	SIM
hotels	hotels	0.29493
restaurants	restaurants	0.30679
hotels.id	hotels.id	0.34542
hotels.zip	hotels.zip_code	0.37050
hotels.url	hotels.url	0.34542
hotels.fax	hotels.double_room	0.21877
hotels.fax	hotels.img_map	0.21877
hotels.city	hotels.city	0.34619
hotels.web_site	hotels.single_room	0.21877
hotels.address	hotels.address	0.34610
hotels.telephone	hotels.code	0.25746
hotels.name	hotels.name	0.37050
hotels.pr	hotels.province	0.37050
restaurants.name	restaurants.name	0.35053
restaurants.url	restaurants.url	0.55893
restaurants.id	restaurants.id	0.34974
restaurants.locality	restaurants.province	0.20973
restaurants.address	restaurants.address	0.35050
restaurants.city	restaurants.city	0.58517
restaurants.telephone	restaurants.telephone	0.35207

(c) $T_P = 0.15$ and $T_{DH} = 1e-3$

PEER1	PEER3	SIM
hotels	hotels	0.29493
restaurants	restaurants	0.30679
hotels.id	hotels.id	0.34542
hotels.zip	hotels.zip_code	0.37050
hotels.url	hotels.url	0.34542
hotels.city	hotels.city	0.34619
hotels.address	hotels.address	0.34610
hotels.name	hotels.name	0.37050
hotels.pr	hotels.province	0.37050
restaurants.name	restaurants.name	0.35053
restaurants.url	restaurants.url	0.55893
restaurants.id	restaurants.id	0.34974
restaurants.address	restaurants.address	0.35050
restaurants.city	restaurants.city	0.58517
restaurants.telephone	restaurants.telephone	0.35207

Table 2: Matching between Peer1 schema and Peer3 schema, using different pruning thresholds

- class `playerImageSource` in Schema B must match only to class `photoSource` although they present the same annotation of `instanceSource`;
- classes `player` and `internationalCups` in Schema B are connected by two different paths but only the one involving property `wins` must match to property `victories` of Schema A.

Results are shown in Table 4. As for the example of Section 6.1 we have tested software

CAMERA	PHOTOCAMERA	SIM
Viewer	OpticalDevice	0.26675
Lens	Lens	0.50459
Body	Structure	0.11953
Money	Money	0.78429
SingleLensReflex	SingleLensReflex	0.55450
Range	Range	0.52054
BodyWithNonAdjustableShutterSpeed	FixedShutterSpeed	0.36257
PurchaseableItem	Components	0.15704
Camera	Model	0.16063
LargeFormat	LargeFormat	0.44927
focalLength	focalLength	0.88099
LargeFormatBody	characteristics	0.41670
aperture	aperture	0.88099
min	min	0.92078
size	property	0.38807
units	units	0.91276
fStop	fStop	0.88099
currency	currency	0.95589

Table 3: Matching between Camera schema and Photocamera schema, using Pruning threshold = 0.0 and Dead-heat threshold = 1e-3

varying the pruning threshold. In particular Table 3(a) gives the mapping found without it while Table 3(b) gives the mapping relatives to a pruning threshold value of 0.10. In this case the best result is the first one because it presents all the possible right matching couples including in the solution only one wrong mapping: `height` and `stateOfBirth`. If we look at the second table we can see that the situation is massively get worse because together with the uncorrect match, a lot of correct ones are excluded. This is a case in which schemas structure helps to find right mappings while, for the example of Section 6.1, it makes the results less precise and it makes a final couples refinement necessary.

(a) Pruning threshold=0.0 and Dead-heat threshold=1e-3

FIFA	UEFA	SIM
dataset	dataset	0.64194
player	player	0.70374
internationalCups	internationalCups	0.56073
photo	playerImage	0.49150
position	position	0.70019
currentClub	currentTeam	0.31813
photoSource	playerImageSource	0.05464
victories	wins	0.68999
picture	picture	0.72463
nationality	stateOfBirth	0.32506
hasPosition	role	0.72943
height	stateOfBirth	0.32506
name	playerName	0.49380
secondName	playerName	0.49380
club	team	0.48532
photoFrom	from	0.27994

(b) Pruning threshold=0.10 and Dead-heat threshold=1e-3

FIFA	UEFA	SIM
dataset	dataset	0.64194
player	player	0.70374
internationalCups	internationalCups	0.56073
photo	playerImage	0.49150
position	position	0.70019
currentClub	currentTeam	0.31813
victories	wins	0.68999
picture	picture	0.72463
hasPosition	role	0.72943
secondName	playerName	0.49380
club	team	0.48532

Table 4: Matching between Fifa schema and Uefa schema, using different pruning threshold

7 Conclusion

This deliverable proposed a detail design of two different matching technics, applicable respectively to obtain a semantic mapping and a sort of similarity between multimedia contents. Several issues are still open especially about the combination of these two approaches because of their differences and about the matching between a single property and a path as concerns lonely semantic aspect. In the future we are intended to study further these two aspects.

References

[1] *http://wordnet.princeton.edu/*.

[2] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[3] Hyemin Chung. Global mind - bridging the gap between different cultures and languages with common-sense computing. Master's thesis, Massachussets Institute of Technology, 2006.

[4] Ronald Fagin. Combining fuzzy information: an overview. *SIGMOD Record*, 31(2):109–118, 2002.

[5] F. Mandreoli, R. Martoglia, and E. Ronchetti. Versatile Structural Disambiguation for Semantic-aware Applications. In *Proc. of CIKM*, 2005.

[6] Federica Mandreoli, Riccardo Martoglia, Simona Sassatelli, and Wilma Penzo. Sri: exploiting semantic information for effective query routing in a pdms. In *Eigth ACM International Workshop on Web Information and Data Management (WIDM 2006), Arlington, Virginia, USA, November 10, 2006*, pages 19–26, 2006.

[7] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, San Jose, CA*, pages 117–128, 2002.

[8] Wilma Penzo. Rewriting rules to permeate complex similarity and fuzzy queries within a relational database system. *IEEE Trans. Knowl. Data Eng.*, 17(2):255–270, 2005.

[9] Jiri Pospichal. Fuzzy sets and fuzzy logic: Theory and applications. by george j. klir and bo yuan. prentice hall: Upper saddle river, nj, 1995, 574 pp, isbn 0-13-101171-5. *Journal of Chemical Information and Computer Sciences*, 36(3):619, 1996.

A Peer1-Peer2 Matching Results

PEER1	PEER2	SIM
hotels	hotels	0.26541
restaurants	campings	0.23086
hotels.id	hotels.id	0.33308
hotels.zip	hotels.comments	0.20412
hotels.zip	hotels.logo	0.19203
hotels.zip	hotels.infos	0.21156
hotels.email	hotels.email	0.33308
hotels.url	hotels.url	0.33308
hotels.fax	hotels.fax	0.33308
hotels.city	hotels.city	0.33308
hotels.web_site	hotels.city_map	0.20412
hotels.web_site	hotels.detailed_map	0.20412
hotels.web_site	hotels.zone_map	0.20412
hotels.address	hotels.address	0.33308
hotels.telephone	hotels.telephone	0.33308
hotels.name	hotels.name	0.33308
hotels.name	hotels.complete_name	0.33308
hotels.pr	hotels.locality	0.19768
restaurants.price_reduction	campings.zip	0.09699
restaurants.name	campings.name	0.33028
restaurants.url	campings.url	0.33028
restaurants.closed	campings.web_site	0.09699
restaurants.id	campings.id	0.33028
restaurants.locality	campings.locality	0.33028
restaurants.address	campings.province	0.14700
restaurants.rank	campings.email	0.09699
restaurants.rank	campings.web_site	0.09699
restaurants.rank	campings.fax	0.09699
restaurants.city	campings.city	0.33028
restaurants.telephone	campings.telephone	0.33028

Table 5: Pruning threshold = 0 and Dead-heat threshold = 1e-3

PEER1	PEER2	SIM
hotels	hotels	0.26541
restaurants	campings	0.23086
hotels.id	hotels.id	0.33308
hotels.zip	hotels.comments	0.20412
hotels.zip	hotels.logo	0.19203
hotels.zip	hotels.infos	0.21156
hotels.email	hotels.email	0.33308
hotels.url	hotels.url	0.33308
hotels.fax	hotels.fax	0.33308
hotels.city	hotels.city	0.33308
hotels.web_site	hotels.city_map	0.20412
hotels.web_site	hotels.detailed_map	0.20412
hotels.web_site	hotels.zone_map	0.20412
hotels.address	hotels.address	0.33308
hotels.telephone	hotels.telephone	0.33308
hotels.name	hotels.complete_name	0.33308
hotels.name	hotels.name	0.33308
hotels.pr	hotels.locality	0.19768
restaurants.name	campings.name	0.33028
restaurants.url	campings.url	0.33028
restaurants.id	campings.id	0.33028
restaurants.locality	campings.locality	0.33028
restaurants.city	campings.city	0.33028
restaurants.telephone	campings.telephone	0.33028

Table 6: Pruning threshold = 0.10 and Dead-heat threshold = 1e-3

PEER1	PEER2	SIM
hotels	hotels	0.26541
restaurants	campings	0.23086
hotels.id	hotels.id	0.33308
hotels.email	hotels.email	0.33308
hotels.url	hotels.url	0.33308
hotels.fax	hotels.fax	0.33308
hotels.city	hotels.city	0.33308
hotels.address	hotels.address	0.33308
hotels.telephone	hotels.telephone	0.33308
hotels.name	hotels.name	0.33308
hotels.name	hotels.complete_name	0.33308
restaurants.name	campings.name	0.33028
restaurants.url	campings.url	0.33028
restaurants.id	campings.id	0.33028
restaurants.locality	campings.locality	0.33028
restaurants.city	campings.city	0.33028
restaurants.telephone	campings.telephone	0.33028

Table 7: Pruning threshold = 0.15 and Dead-heat threshold = 1e-3

B Camera Scenario

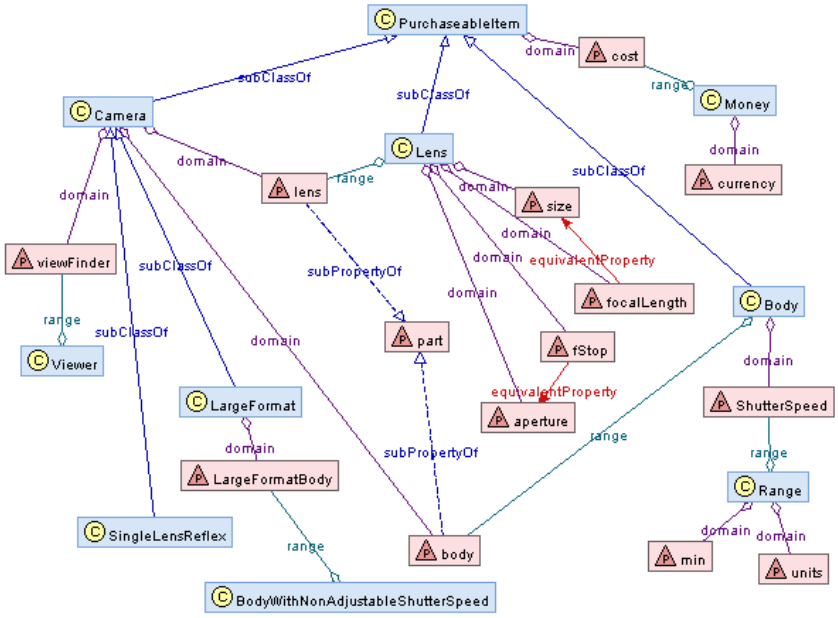


Figure 20: Schema A

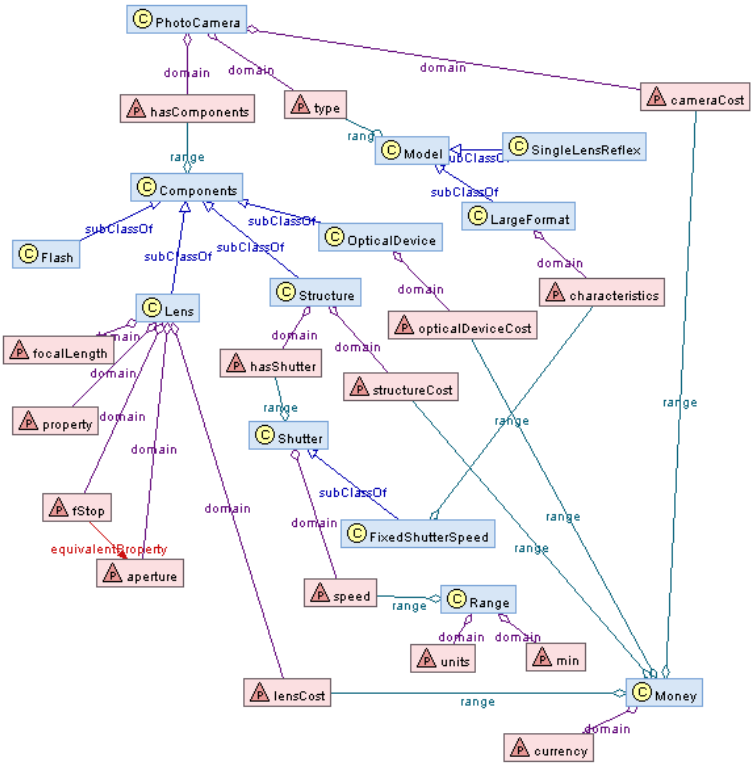


Figure 21: Schema B

C Soccer Scenario

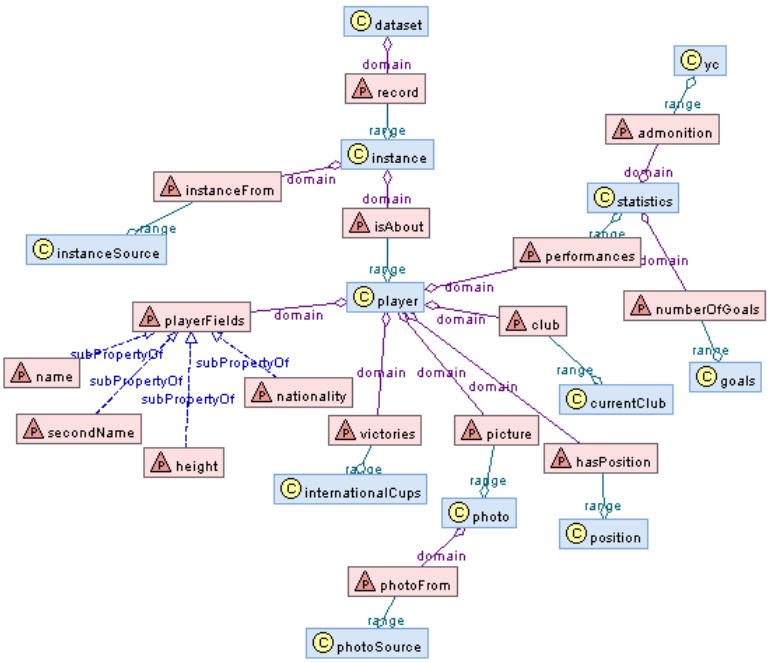


Figure 22: Schema A

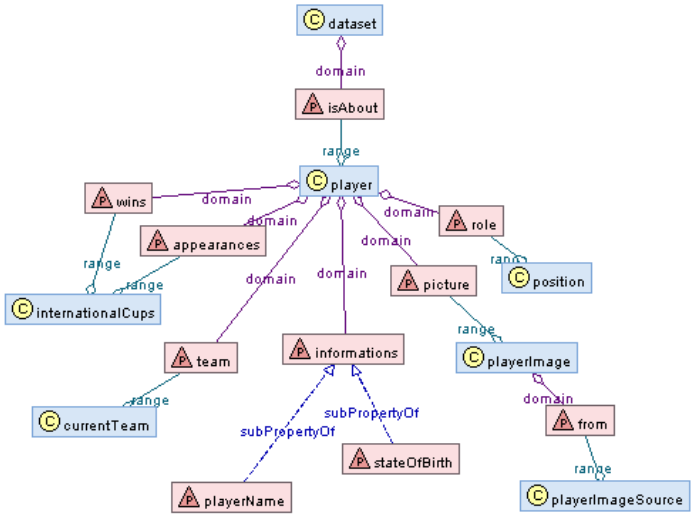


Figure 23: Schema B