Indexing Source Descriptions based on Defined Classes

Ralph Lange Frank Dürr Kurt Rothermel
Institute of Parallel and Distributed Systems
Universitätsstraße 38
70569 Stuttgart, Germany
<firstname.lastname>@ipvs.uni-stuttgart.de

ABSTRACT

Scaling heterogeneous information systems (HIS) to thousands of sources poses particular challenges to source discovery. It requires a powerful formalism for describing the contents of the sources in a concise manner and for formulating compatible queries as well as a suitable structure for indexing and retrieving the source descriptions efficiently.

We propose an extended logic-based description formalism for large-scale HIS with structured sources and a shared ontology. The formalism refines existing approaches that describe the sources by constraints on the attribute value ranges in several ways: It allows for complex, nested descriptions based on *defined classes*. It supports alternative descriptions to express that a source may be discovered by different combinations of constraints. Finally, it allows to adjust between positive matching, similar to keyword-based discovery, and negative matching, as used in existing logic-based approaches.

We further propose the SDC-Tree for indexing such source descriptions. To allow for efficient discovery, the SDC-Tree features multidimensional indexing capabilities for the different attributes and the IS-A hierarchy of the shared ontology, but also incorporates the existence or absence of constraints. For this purpose, it supports three different types of node split operations which exploit the expressiveness of the description formalism. Therefore, we also propose a generic split algorithm which can be used with arbitrary ontologies.

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous Databases; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, design, management, performance

Keywords

Heterogeneous information systems, source descriptions, indexing of source descriptions, defined classes, tree-based index structure

1. INTRODUCTION

Heterogeneous information systems (HIS) consist of a number of information sources, possibly distributed around the globe. The sources provide information on different aspects and clippings of the overall domain of discourse – usually using different local schemas. There exist various types of HIS such as federated DBMS, mediator-based IS, and peer data management systems (PDMS). Exemplary application areas for such systems include logistics, stock exchange, earth sciences, and scalable context information management [13].

One problem in HIS is how to efficiently discover the sources that are relevant for a certain application task or query. A number of systems (e.g. [2,8,16]) utilize the schema mappings between the sources or to some shared schema to exclude those sources from processing that do not provide *any* information about the queried relations or classes. This approach, however, does not scale to large HIS with thousands of sources, where each source may provide a comparatively small clipping of information for a certain class or relation, and applications likewise are interested in information about few entities of a certain class or relation only.

As an example, consider a distributed spatial context information system like Nexus [17]. The system aims at providing an integrated view on spatial context information in form of street maps, 3D building plans, sensor data maps, etc. given by commercial and non-commercial information sources all over the globe. Because of the diversity of contextual information and spatial restrictions, most sources represent small clippings of the physical world only. For example there may exist thousands of sources providing building plans, i.e. information on rooms, corridors, stairways and other building parts, from different buildings all over the world. This information may be used for a number of purposes such as indoor navigation, remote visualization, and context-aware museum guidance. However, such an application only requires one or few plans of certain buildings at a time.

Therefore, large-scale HIS require an approach for source discovery that utilizes explicit descriptions for the sources' contents rather than or in addition to schema mappings. Such an approach generally comprises two parts:

1. A formalism for describing the contents of the sources,

for formulating compatible queries, and for deciding whether a description *matches* a query or not, i.e. whether the corresponding source is relevant for the query.

2. An index structure for efficiently retrieving those descriptions (and thus sources) that match a given query. Of course, the designing of an index structure highly depends on the description formalism and its matching semantics.

Different description formalisms are imaginable – ranging from simple keyword lists, which are particularly used for text sources, to logic-based approaches for structured sources using constraints, e.g. to exclude a source about expensive hotels (price $\geq$ \$200) from a query for hostels (price $\leq$ \$30).

At a first glance, those two extremes seem to imply different matching semantics: Keywords imply *positive* semantics in view of the fact that a description matches a given query only if both share one or more explicitly stated keywords. Thus, a query issuer has to know the "right" keywords to discover a certain source. Logic-based formalisms for source discovery, on the contrary, imply *negative* semantics in view of the fact that a description is considered to match a query unless the converse can be proven by means of correspondent constraints.

In this paper, we propose a novel logic-based description formalism enabling positive *and* negative matching semantics as well as a compatible index structure. Our approach is targeted on large-scale HIS with structured sources, where the sources share a (simple) ontology. Note that such an ontology can be easily derived if the sources share some (mediated) object-oriented schema.

The proposed description formalism is based on *defined classes* and refines and extends the idea of describing the sources by constraints in several ways: It allows for conjunctive and generally nested constraints over not only all attributes but also all relations of the classes of the shared ontology. The formalism enables to distinguish between different perspectives on the contents of a source and thus alternative descriptions. For example, it allows to express that a source providing a 3D building model of the British Museum can be discovered either just by the location of the museum *or* just by the famous and unique name, whereas a less famous museum (with ambiguous name) may have to be discovered by its name *and* the coarse location. Finally, the formalism allows to adjust between positive and negative matching semantics for each query separately, which we suppose to be important for effective source discovery in large-scale HIS.

We further propose the *Source Description Class Tree* (SDC-Tree) for indexing source descriptions by means of their defined classes. To enable efficient discovery, the SDC-Tree features multidimensional indexing capabilities for the different attributes, relations, and the IS-A hierarchy of the shared ontology – but it also incorporates the existence or absence of constraints and the ranges of the constraints. For this purpose, it supports three different types of node split operations exploiting the expressiveness of the description formalism.

Due its flexibility regarding node splitting, the SDC-Tree can be easily adapted to a wide range of ontologies and source descriptions. Therefore, we further present a generic split algorithm suitable for arbitrary ontologies.

The remainder of the paper is structured as follows: We discuss related work in Section 2, before we present the description formalism in Section 3. In Section 4 we present the SDC-Tree and the corresponding algorithms. Simulation results showing the efficiency of the SDC-Tree are given in Section 5. Finally, the paper is concluded in Section 6 with a summary.

2. RELATED WORK

The SDC-Tree and the proposed source description formalism are related to integration systems in general and discovery services for information sources in particular.

Integration systems include federated DBMS and mediator-based approaches such as Information Manifold [18], SIMS [3], Infomaster [10], DISCO [29], MOMIS [8], Object Globe [9], Amos II [26], InfoSleuth [22], and Quete [16]. These systems are based on formal mappings from the sources' local schemas to some shared global schema or ontology, also named *reference schema/ontology.* The classes or relations specified in the mappings give a coarse estimation of the entities represented by a source. However, a source may only represent a small subset of the entities belonging to these classes. Several works (e.g. [14, 16, 18, 19]) therefore enhance these kinds of source descriptions with constraints on the attribute ranges, as illustrated in the example $V_2(c) \subseteq \text{CarForSale}(c)$, $\text{Price}(c, p)$, $p \geq 20000$ from [18] to specify a database with cars priced above \$20,000.

We argue that source discovery and mapping between schemas or ontologies are two concerns and focus on the former in this paper. This differentiation is particularly important for large-scale HIS since there may be thousands of sources providing information about different entities of a certain class using one and the same local schema.

This argumentation similarly applies to peer data management systems (PDMS)[1] such as WebFINDIT [25], PeerDB [21], Hyperion [2], and Piazza [14], which consist of a number of sources named *peers* and a collection of mappings between the peers' local schemas – instead of a shared schema. According to Halevy et al. [14], a PDMS can be viewed as a strict generalization of data integration systems.

There exist several dedicated approaches for source discovery in large-scale HIS. Text-source discovery services such as GlOSS [11] use statistical summaries of the textual contents to rank the sources with respect to a query of keywords.

Semantic Context Space (SCS) [12] is a P2P overlay network for context information search. It assumes a shared upper ontology and a large number of sources providing context information by means of RDF triples to different clippings of the domain of discourse. SCS clusters all sources providing triples on the same class by creating overlay network links between them and organizes the clusters into a ringlike overlay network similar to the distributed hash table (DHT) *Chord* [28]. Thus, SCS allows discovering all sources providing information about the queried class, but does not take constraints on attributes or relations into account.

The authors of [15] propose a similar approach based on an extension of the Chord protocol. The sources describe each relation of their local schemas and the belonging attributes by means of a shared ontology. These descriptions are stored

[1]PDMS must not be mistaken with structured peer-to-peer systems and distributed hash tables (DHT) such as Chord [28], which partition a given data set to a number of nodes.

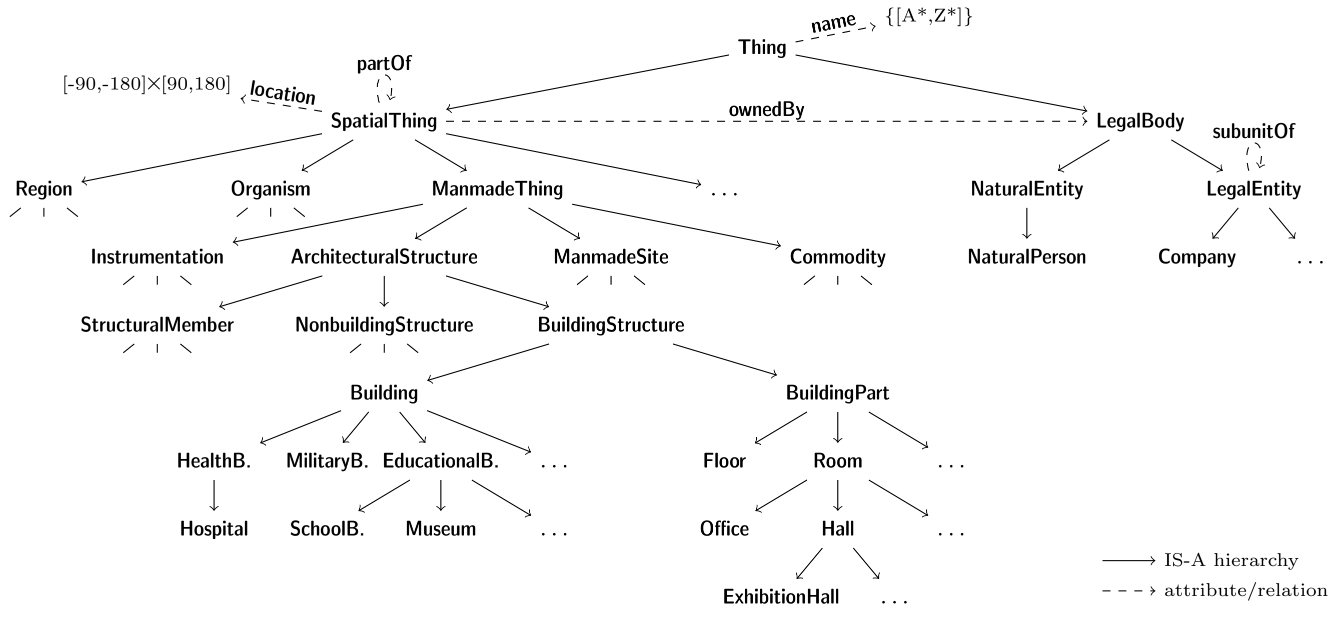


Figure 1: Spatial context ontology created on the basis of [4, 27, 30].

in a Chord ring, using a predefined mapping from classes to integer keys. This approach allows retrieving all sources that provide information about a queried class, while taking the attributes of interest into account. However, it does not support constraints on the ranges of attributes or relations.

GloServ [1] also is a distributed discovery service based on a shared ontology. The nodes of the service are organized according to the IS-A hierarchy of the ontology. Sources are registered at those nodes that correspond to classes for which they provide information. GloServ supports constraints on predefined attributes by refining the IS-A hierarchy with respect to the ranges of these attributes.

The description formalism presented in this paper extends the idea of using constraints on attributes to precisely describe the contents of the sources in several ways: It allows for alternative descriptions, constraints on relations, and nested constraints, as well as positive and negative matching semantics. With the SDC-Tree, we further propose a flexible indexing approach for such source descriptions, incorporating not only the IS-A hierarchy, as the above-mentioned works, but also the existence or absence of constraints and the ranges of the constraints.

3. DESCRIPTION FORMALISM

In this section, we introduce our assumptions and notations, before we explain how to describe sources by defined classes. Based on the definition of source descriptions, we finally discuss how to formulate compatible queries and how to match source descriptions against such queries.

3.1 Assumptions and Notations

We consider an HIS where the sources share some ontology, providing a conceptualization of the overall domain of discourse. Such an ontology may be hand-crafted for the purpose of source discovery only, or may be automatically generated from some shared (object-relational) schema.

For the purpose of source descriptions and discovery, we consider three ontology components only:

1. Classes such as **ExhibitionHall**, **Building**, and **BuildingPart** organized in an IS-A hierarchy, i.e. where each class C has a unique *parent class* $\mathcal{Prnt}(C)$, except for the top class $C_\top$. The expression $C \prec C'$ states that C is a subclass of C'.
2. Attributes such as **name** and **location** with primitive value ranges such as the set of all strings, the integers from -50 to 70, or the points (latitude and longitude) on the sphere. We denote the range of an attribute a by $\mathcal{Rng}(a)$, e.g. $\mathcal{Rng}(a) = \{[-50,70]\}$. The *domain* of a, i.e. the class C and subclasses of C it belongs to, is denoted by $\mathcal{Dom}(a) = C$.
3. Relations such as **partOf** and **ownedBy** between entities of certain classes. Similar to attributes, we refer to the *domain* of a relation r as $\mathcal{Dom}(r) = C$ and to the *range* of r as $\mathcal{Rng}(r) = C'$.

Note that the ontology particularly enables *concrete domains*, i.e. attribute value ranges with (partial) orders and relational operators like $\leq$ and $\geq$, in addition to $=$ and $\neq$.

We do not make any assumptions on how applications interact with the HIS since all the different interaction patterns (e.g. query/response, data streams, events) require a mechanism to discover or decide on the relevant sources in advance.

Our examples throughout the paper are inspired by a distributed spatial context information system using the ontology given in Figure 1. The range [-90,-180]×[90,180] of the attribute **location** denotes the latitudes and longitudes of the world. This ontology has been created on the basis of the Alexandria Digital Library Feature Type Thesaurus [30], the Suggested Upper Merged Ontology (SUMO) [4], and the PROTON Ontology (PROTo ONtology) [27].

It has been also used for the evaluation of the SDC-Tree described in Section 5.

3.2 Describing Sources by Defined Classes

Sources generally provide information about one or few coherent clippings of the domain of discourse. Thus, a source represents one or more sets of entities sharing some characteristic properties that can be described concisely by the class and one or more constraints. For example, the entities represented by a museums database for London simply share the properties that they all belong to the class museum and are located in London and thus can be described by

$$\langle \textsf{Museum} : \textsf{location} \in \{\text{London, UK}\} \rangle \ ,$$

where we assume that the range $\{\text{London, UK}\}$ is given as polygon on the globe.[2]

Such a description is a complete formal specification of a subclass of Museum and therefore is referred to as *defined class* or *complex concept* (e.g. [5, 20]).

The characteristic properties may be also relations to one or few entities, which again can be described by defined classes. For example, the entities represented by a 3D building plan of the British Museum share the property that they are part of the British Museum and hence can be described by

$$D_1 = \langle \textsf{BuildingPart} : \textsf{partOf} \in \langle \textsf{Museum} : \textsf{name} \in \{\text{“British Museum”}\} \rangle \rangle$$

using the famous and unique name of the museum.

Therefore, we propose to describe a source by one or more defined classes of the following form.

Definition 1 [Defined class]: A defined class D is specified by a class C followed by a conjunction of constraints on zero or more attributes and relations whose domains comprise C:

$$D = \langle C : a_1 \in X_1 \wedge a_2 \in X_2 \wedge \ldots \wedge r_1 \in \bar{D}_1 \wedge r_2 \in \bar{D}_2 \wedge \ldots \rangle$$

We refer to the class C as *base* of the defined class and introduce the operator $\mathcal{Base}(D)$ to retrieve C. The base specifies that the source provides information about entities of the class C or of a subclass of C.

A constraint $a_i \in X_i$, with $X_i \subseteq \mathcal{Rng}(a_i)$, states that the entities represented by the source have at least one value $x_i \in X_i$ for the attribute a_i. A range X_i can be given by a set- or interval-based expression depending on the data type. To retrieve X_i, we introduce the operator $\mathcal{Con}_{a_i}(D)$. To test whether D actually has a constraint on attribute a, we introduce the operator $\textsc{isCon}_a(D)$ returning $\textsc{True}$ or $\textsc{False}$, respectively.

Analogously, an expression $r_j \in \bar{D}_j$ specifies a constraint on the relation r_j to the range $\bar{D}_j$, where $\bar{D}_j$ again is a defined class of the above form with $\mathcal{Base}(\bar{D}_j) \preceq \mathcal{Rng}(r_j)$. We refer to $\bar{D}_j$ as *nested* defined class of D. The constraint states that the entities represented by the source have a relation r_j to an entity belonging to $\bar{D}_j$. The operator $\mathcal{Con}_{r_j}(D)$ returns the nested defined class $\bar{D}_j$. The operator $\textsc{isCon}_r(D)$ queries whether D has a constraint on the relation r or not.

As mentioned above, a source may represent multiple coherent clippings of the domain of discourse. These can be described independently of each other. Without loss of generality, we therefore consider only one clipping per source in the following. Nevertheless, even a single clipping may be described by multiple defined classes. For example, the building plan of the British museum can be also described by

$$D_2 = \langle \textsf{BuildingPart} : \textsf{location} \in \{\text{44 Gt Russell St, London, UK}\} \rangle \ .$$

Of course, the constraints of D_1 and D_2 could be merged into a single defined class. However, this would hide the fact that both, the location and the unique name of the British Museum, each suffice as unambiguous description for the museum and the corresponding building plan. As discussed in the next section, to realize source discovery with positive matching semantics, such alternative *perspectives* on a source should be reflected by separate defined classes as exemplified by $\{D_1, D_2\}$.

A defined class may nevertheless consist of multiple conjunctive constraints to describe the contents of a source distinctively. For example, this applies to museums with ambiguous names such as “Local Heritage Museum”. A source about a certain museum with that name has to be either described by the exact location or by the name and the coarse location together.

3.3 Matching Source Descriptions against Discovery Queries

Analogously to source descriptions, we propose to specify discovery queries by defined classes. However, in contrast to a source description, a query should consist of one defined class Q (*query class*) only, as argued below.

Intuitively, a given source description with *source classes* $\{D_1, \ldots, D_n\}$ should *match* a query class Q if Q refers to one or more entities of the domain of discourse that are described by $\{D_1, \ldots, D_n\}$. However, since the individual entities are not known for source discovery, different matching semantics are imaginable. As indicated in Section 1, we can distinguish between two extremes. Considering defined classes, these extremes can be specified as follows:

- *Positive matching:* A source description $\{D_1, \ldots, D_n\}$ matches a query Q only if there exists a D_i such that D_i and Q have constraints on the same attributes and relations and that the ranges of correspondent constraints overlap.

  Thus, the query issuer has to “know” the combinations of attributes and relations that are reasonably used in source descriptions.

- *Negative matching:* A source description $\{D_1, \ldots, D_n\}$ matches a query Q unless the converse can be proven by the ranges of two correspondent constraints.

  More precisely, the source description *dismatches* Q if there exists a D_i such that D_i and Q have disjoint constraint ranges for a certain attribute or relation.

On the one hand, we suppose that negative matching generally returns too many irrelevant sources in large-scale HIS with thousands of sources. On the other hand, the positive matching semantics are very strict since they require the query issuer to know or presume the combinations of attribute and relations being (typically) used in source classes. Therefore, we propose a flexible approach between both extremes.

Our matching approach is based on two predicates: The *query matching predicate* $\rightsquigarrow_{Q}$ reflects the positive matching

[2]There exist hybrid location models that allow determining the geographic area (polygon of latitude-longitude pairs) for a given location name and vice-versa [6].

semantics, while the *query dismatching predicate* $/\!\!/_{\mathrm{Q}}$ reflects the negative extreme. The predicates $\rightsquigarrow_{\mathrm{Q}}$ and $/\!\!/_{\mathrm{Q}}$ together yield three possible states between a pair D_i and Q. Based on these states it then is decided whether a source description $\{D_1, \ldots, D_n\}$ matches Q or not.

Definition 2 [Query matching predicate $\rightsquigarrow_{\mathrm{Q}}$]: Given a source class D_i and a query class Q, it holds $D \rightsquigarrow_{\mathrm{Q}} Q$ iff the following three conditions hold:

1. $(\mathcal{B}ase(D_i) \succeq \mathcal{B}ase(Q)) \vee (\mathcal{B}ase(D_i) \preceq \mathcal{B}ase(Q))$
2. $\forall$ attrib. a with $(\mathcal{D}om(a) \succeq \mathcal{B}ase(Q)) \wedge (\mathcal{D}om(a) \succeq \mathcal{B}ase(D_i))$: $\textsc{isCon}_a(D_i) \Rightarrow (\textsc{isCon}_a(Q) \wedge (\mathcal{C}on_a(D_i) \cap \mathcal{C}on_a(Q) \neq \emptyset))$
3. $\forall$ relat. r with $(\mathcal{D}om(r) \succeq \mathcal{B}ase(Q)) \wedge (\mathcal{D}om(r) \succeq \mathcal{B}ase(D_i))$: $\textsc{isCon}_r(D_i) \Rightarrow (\textsc{isCon}_r(Q) \wedge (\mathcal{C}on_r(D_i) \rightsquigarrow_{\mathrm{Q}} \mathcal{C}on_r(Q)))$

Thus, D_i and Q have to base on classes that are equal or sub-/superclass of each other, and Q has to specify correspondent constraints with overlapping ranges for all constraints specified in D_i.

Note that Q may specify constraints on more attributes and relations than D_i. Therefore, $\rightsquigarrow_{\mathrm{Q}}$ is not commutative. This alleviates the strict positive semantics and is also the reason why a source should be described by multiple alternative defined classes – representing different perspectives as explained in Section 3.2 – with as few constraints as possible, whereas a query should consist of only one defined class, containing all known constraints.

Definition 3 [Query dismatching predicate $/\!\!/_{\mathrm{Q}}$]: Given a source class D_i and a query class Q, it holds $D /\!\!/_{\mathrm{Q}} Q$ iff *one* of the following two conditions holds:

1. $\exists$ attrib. a with $(\mathcal{D}om(a) \succeq \mathcal{B}ase(Q)) \wedge (\mathcal{D}om(a) \succeq \mathcal{B}ase(D))$: $\textsc{isCon}_a(D) \wedge \textsc{isCon}_a(Q) \wedge (\mathcal{C}on_a(D) \cap \mathcal{C}on_a(Q) = \emptyset)$
2. $\exists$ relat. r with $(\mathcal{D}om(r) \succeq \mathcal{B}ase(Q)) \wedge (\mathcal{D}om(r) \succeq \mathcal{B}ase(D))$: $\textsc{isCon}_r(D) \wedge \textsc{isCon}_r(Q) \wedge (\mathcal{C}on_r(D) /\!\!/_{\mathrm{Q}} \mathcal{C}on_r(Q))$

This predicate directly implements the negative matching semantics. Moreover, it is contrary to $\rightsquigarrow_{\mathrm{Q}}$. Hence, $D_i \rightsquigarrow_{\mathrm{Q}} Q$ implies that D_i and Q do *not* dismatch.

Definition 4 [Matching]: Based on these predicates, we define that a source description $\{D_1, \ldots, D_n\}$ matches a query Q only if there exists a D_i with $D_i \rightsquigarrow_{\mathrm{Q}} Q$ but no D_j with $D_j /\!\!/_{\mathrm{Q}} Q$.

Thus, $\rightsquigarrow_{\mathrm{Q}}$ is a necessary condition for matching, while $/\!\!/_{\mathrm{Q}}$ is a sufficient condition for a dismatch.

For example, consider the source description $\{D_1, D_2\}$ given in Section 3.2, and the following query:

$$Q_1 = \langle \textsf{ExhibitionHall} : \textsf{location} \in \{\text{32-50 Gt Russell St, London, UK}\}\rangle$$

It queries for sources providing information about exhibition halls – which are a subclass of **BuildingPart**, cf. Figure 1 – located between 32 and 50 Great Russell Street. The source description matches Q_1 since $D_2 \rightsquigarrow_{\mathrm{Q}} Q_1$. However, it does not match

$$Q_2 = \langle \textsf{ExhibitionHall} : \textsf{partOf} \in \langle \textsf{Museum} : \textsf{name} \in \{[\text{E*,F*}]\}\rangle\rangle$$

as neither D_1 nor D_2 match Q_2 according to $\rightsquigarrow_{\mathrm{Q}}$. Also, the source description does not match

$$Q_3 = \langle \textsf{ExhibitionHall} : \textsf{partOf} \in \langle \textsf{Museum} : \textsf{name} \in \{[\text{E*,F*}]\}\rangle \wedge \textsf{location} \in \{\text{London, UK}\}\rangle$$

since $D_1 /\!\!/_{\mathrm{Q}} Q_3$, even though $D_2 \rightsquigarrow_{\mathrm{Q}} Q_3$.

Negative matching semantics can be realized (completely or partially) by extending Q with *pseudo constraints* of the form $a \in *$ or $r \in *$. For an attribute a, $*$ simply evaluates to $\mathcal{R}ng(a)$. In case of a relation r, $*$ has to be applied recursively to all attributes and relations whose domains comprise $\mathcal{R}ng(r)$. However, for evaluating $\rightsquigarrow_{\mathrm{Q}}$ or $/\!\!/_{\mathrm{Q}}$ it suffices to know that $*$ overlaps with any nested defined class $\mathcal{C}on_r(D_i)$.

The more pseudo constraints are specified in a query Q, the more likely there exists a $D_i \in \{D_1, \ldots, D_n\}$ where $D_i \rightsquigarrow_{\mathrm{Q}} Q$. Thus, the influence of the predicate $\rightsquigarrow_{\mathrm{Q}}$ decreases, whereas the influence of $/\!\!/_{\mathrm{Q}}$ increases.

Based on $\rightsquigarrow_{\mathrm{Q}}$ we can directly define a subsumption predicate $\succeq_{\mathrm{Q}}$, allowing determining whether a certain defined class not only overlaps but *completely comprises* another defined class. This predicate is also essential for the SDC-Tree.

Definition 5 [Subsumption predicate $\succeq_{\mathrm{Q}}$]: A defined class D subsumes another defined class Q, denoted by $D \succeq_{\mathrm{Q}} Q$ iff the following three conditions hold:

1. $\mathcal{B}ase(D) \succeq \mathcal{B}ase(Q)$
2. $\forall$ attribute a with $\mathcal{D}om(a) \succeq \mathcal{B}ase(D)$: $\textsc{isCon}_a(D) \Rightarrow (\textsc{isCon}_a(Q) \wedge (\mathcal{C}on_a(D) \supseteq \mathcal{C}on_a(Q)))$
3. $\forall$ relation r with $\mathcal{D}om(r) \succeq \mathcal{B}ase(D)$: $\textsc{isCon}_r(D) \Rightarrow (\textsc{isCon}_r(Q) \wedge (\mathcal{C}on_r(D) \succeq_{\mathrm{Q}} \mathcal{C}on_r(Q)))$

4. SOURCE DESCRIPTION CLASS TREE

In this section we present the *Source Description Class Tree* (SDC-Tree), a powerful tree-based structure for indexing source descriptions by means of their defined classes and for efficiently retrieving those descriptions matching a given query Q.

Given a source description $\{D_1, \ldots, D_n\}$, the SDC-Tree uses each source class D_i as index key and stores a corresponding entry, mapping the source class to the source description, at one or more leaf nodes. The SDC-Tree considers the necessary matching condition only, given by $\rightsquigarrow_{\mathrm{Q}}$. Source descriptions with a source class $D_i \rightsquigarrow_{\mathrm{Q}} Q$ that do not match because of an additional $D_j /\!\!/_{\mathrm{Q}} Q$ are filtered in a post-processing step.

The SDC-Tree supports different operations for splitting a leaf node (*split types*) to exploit the expressiveness of the description formalism completely.

Next, we describe the tree's structure and the split types, before we give the corresponding formal definitions. That followed, based on the formal definitions, we prove the completeness of indexing. Finally, an algorithm for selecting and parameterizing the split types is presented.

4.1 Structure of SDC-Tree

Each node of the SDC-Tree is associated with a defined class named *node class* whose form slightly extends the above definition for defined classes. The formal definition for node classes is given below in Section 4.2.

The tree reflects the subsumption hierarchy of the node classes. Thus, let N be the node class of an inner node and N' the node class of one of its child nodes, it holds $N \succeq_{\mathrm{Q}} N'$. The node class of the root node is $\langle C_\top, \textsc{True} : \rangle$, matching every defined class, where the role of $\textsc{True}$ is explained below.

Figure 2 shows a potential SDC-Tree for indexing source descriptions that are based on the spatial context ontology given at the beginning of Section 3. More precisely, the fig-

$$\begin{tikzcd}[column sep=tiny]
 & & N_1 = \langle \textsf{Thing}, \textsc{True} : \rangle \arrow[dll] \arrow[d] \arrow[drr] & & \\
N_2 = \langle \textsf{Thing}, \textsc{False} : \rangle & & N_3 = \langle \textsf{Spatial}, \textsc{True} : \rangle \arrow[dl] \arrow[dr] & & N_4 = \langle \textsf{LegalBody}, \textsc{True} : \rangle \\
 & N_5 = \langle \textsf{Spatial}, \textsc{True} : \textsf{location} \in \text{[-90,-180]}\times\text{[90,180]} \rangle \arrow[dl] \arrow[dr] & & N_6 = \langle \textsf{Spatial}, \textsc{True} : \textsf{location} \in \textsc{Null} \rangle & \\
N_7 = \langle \textsf{Spatial}, \textsc{True} : \textsf{location} \in \text{[-90,-180]}\times\text{[0,180]} \rangle & & N_8 = \langle \textsf{Spatial}, \textsc{True} : \textsf{location} \in \text{[0,-180]}\times\text{[90,180]} \rangle & &
\end{tikzcd}$$

Figure 2: Potential SDC-Tree with three splits of different types at N_1, N_3, and N_5.

ure depicts the node classes and the subsumption hierarchy of the tree. The root node class of the tree is $\langle \textsf{Thing}, \textsc{True} : \rangle$ since the top class of the ontology is **Thing**.

A source class D to be indexed is passed down iteratively from the root to each child node whose node class N' matches D by the *index matching predicate* $\leadsto_{\mathrm{I}}$, formally defined in Section 4.2. This predicate is a special case of the query matching predicate $\leadsto_{\mathrm{Q}}$ such that $N' \leadsto_{\mathrm{I}} D$ implies $N' \leadsto_{\mathrm{Q}} D$. Finally, an entry mapping from D to the source description it belongs to is stored at every leaf node with node class $N \leadsto_{\mathrm{I}} D$. For example, in Figure 2 the source class

$$D = \langle \textsf{BuildingPart} : \textsf{location} \in \{\text{44 Gt Russell St, London, UK}\} \rangle$$

is passed from N_1 via N_3 and N_5 to N_8 as D's base **BuildingPart** is a subclass of **Spatial** and the constraint on **location** (polygon of latitude-longitude pairs) is completely contained in the Northern Hemisphere specified by the constraint range [0,-180]×[90,180] of N_8.

A query Q, on the contrary, is passed down using $\leadsto_{\mathrm{Q}}$. Thus, Q is distributed to every leaf node with node class $N \leadsto_{\mathrm{Q}} Q$. Next, each entry of those leaf nodes is evaluated against Q using $\leadsto_{\mathrm{Q}}$ and the source descriptions of those entries with source class $D \leadsto_{\mathrm{Q}} Q$ are collected and duplicates are removed. Then, those source descriptions that do not match Q because of another source class $D \not\leadsto_{\mathrm{Q}} Q$ are removed. Finally, the remaining set of source descriptions is returned to the query issuer.

Since $\leadsto_{\mathrm{I}}$ implies $\leadsto_{\mathrm{Q}}$, it generally holds $\{N : N \leadsto_{\mathrm{I}} D\} \subset \{N : N \leadsto_{\mathrm{Q}} D\}$. This property results from non-commutativity of $\leadsto_{\mathrm{Q}}$. For example, in Figure 2, the query class

$$Q = \langle \textsf{ExhibitionHall} : \textsf{location} \in \{\text{32-50 Gt Russell St, London, UK}\} \rangle ,$$

being matched by D from the above example, is passed from N_1 to the three leaf nodes N_2, N_6, and N_8 using $\leadsto_{\mathrm{Q}}$.

To reduce the number of source classes $\mathbb{D}$ indexed by a leaf node with node class N, it can be split into two or more new nodes with node classes $N'_1, N'_2, \ldots$

We distinguish three split types, one for each aspect of how defined classes can differ from one another.

Base Split. This type of split operation performs a split by means of the IS-A hierarchy of the shared ontology. Given a leaf node with node class N and source classes $\mathbb{D}$, it specifies $1 + l$ new child nodes with nodes classes N'_1 to N'_{1+l}, where $l = |\{C' : \mathscr{Prnt}(C') = \mathscr{Base}(N)\}|$, which must be > 0.

N'_1 equals N except that is does not match source classes D with $\mathscr{Base}(D) \prec \mathscr{Base}(N)$ by $\leadsto_{\mathrm{I}}$. For this purpose, a node class extends the form of defined classes by a Boolean parameter given after the base, as shown in Figure 2. This parameter indicates whether the node class matches source classes whose base is a proper subclass of its own base (True) or not (False). Hence, with N'_1 this parameter is set to False and N'_1 cannot be further split by means of its base.

N'_2 to N'_{1+l} are equal to N, i.e. the mentioned parameter is True, except that each of them has different new base C' with $\mathscr{Prnt}(C') = \mathscr{Base}(N)$.

After a base split operation, each $D \in \mathbb{D}$ can be relocated unambiguously to one of the new nodes according to $\leadsto_{\mathrm{I}}$. A query class Q with $N \leadsto_{\mathrm{Q}} Q$, however, is matched by two or all new node classes.

A base split can be useful for two reasons: First, it may partition $\mathbb{D}$ to different nodes and thus allow for pruning irrelevant source classes for a query class Q with $\mathscr{Base}(Q) \prec \mathscr{Base}(N)$. Second, it enables further node splits by means of attributes and relations that are specific to child classes of $\mathscr{Base}(N)$. Figure 2 exemplifies a base split at N_1.

Existence Split. This type of split operation performs a split by specifying the existence or non-existence of an attribute constraint. Given a leaf node with source classes $\mathbb{D}$ and node class N without constraint on attribute a, it is useful to differentiate between source classes that have a constraint on a and those that have no constraint. Furthermore, existence splits are needed to enable range splits (see below) on the former set of source classes.

The split operation creates two new child nodes with node classes N'_1 and N'_2 as follows: N'_1 equals N but additionally specifies a constraint $\mathscr{Con}_a(N'_1) = \mathscr{Rng}(a)$. N'_2 equals N but explicitly specifies that a source class D must *not* have a constraint on a in order to $N'_2 \leadsto_{\mathrm{I}} D$. For this purpose, node classes extend the definition of defined classes by the possibility to prevent a constraint on a, indicated by $a \in \textsc{Null}$. Such a prevention is shown in Figure 2 at N_6, resulting from an existence split at N_3. As a consequence, a source class D with $N \leadsto_{\mathrm{I}} D$ is matched by either N'_1 or N'_2 only.

Since $\leadsto_{\mathrm{Q}}$ does not consider the prevention of constraints, a query Q with $N \leadsto_{\mathrm{Q}} Q$ is matched by either N'_1 and N'_2, or only N'_2.

An existence split may be also performed by a relation r, where N'_1 extends N by a $\mathscr{Con}_r(N'_1) = \langle \mathscr{Rng}(r), \textsc{True} : \rangle$, accordingly.

Range Split. Given a leaf node whose node class N has a constraint on attribute a, this operation performs a split by partitioning $\mathcal{Con}_a(N)$. This split type is especially useful if the source classes $\mathbb{D}$ indexed by the node significantly differ regarding $\mathcal{Con}_a(N)$.

A range split specifies two or more node class N'_1, N'_2, ... with ranges $\mathcal{Con}_a(N'_i)$ such that $\mathcal{Con}_a(N'_1) \cup \mathcal{Con}_a(N'_2) \cup \ldots = \mathcal{Con}_a(N)$ and $\mathcal{Con}_a(N'_i) \cap \mathcal{Con}_a(N'_j) = \emptyset$, $\forall i \neq j$.

A source class D with $N \rightsquigarrow_{\mathrm{I}} D$ whose constraint on a overlaps a certain $\mathcal{Con}_a(N'_i)$ but no other $\mathcal{Con}_a(N'_j)$ $(j \neq i)$ is indexed at the new child node with node class N'_i only. Otherwise, D is passed to multiple of the new nodes. Similarly, a query class may be passed to one or multiple of the new nodes.

The SDC-Tree does not pose any restrictions on how the ranges $\mathcal{Con}_a(N'_1), \mathcal{Con}_a(N'_2), \ldots$ are determined. This also depends on the data type of a. The Generic Split Algorithm proposed in Section 4.4 uses the split algorithm of the R*-Tree [7] for this purpose.

Figure 2 gives an example of a range split with the node class N_5 by means of the constraint range [-90,-180]×[90,180] (the latitudes and longitudes of the whole world) into two ranges specifying the Southern and Northern Hemisphere, respectively.

Each of the above types of split operations can be also performed by means of a nested node class. For example, the node class

$$\langle \textsf{BuildingPart}, \textsc{True} : \textsf{partOf} \in \langle \textsf{Museum}, \textsc{True} : \textsf{name} \in \{[\text{A*, Z*}]\}\rangle\rangle$$

with a nested class for **partOf** may be split into

$$\langle \textsf{BuildingPart}, \textsc{True} : \textsf{partOf} \in \langle \textsf{Museum}, \textsc{True} : \textsf{name} \in \{[\text{A*, M*}]\}\rangle\rangle$$

and

$$\langle \textsf{BuildingPart}, \textsc{True} : \textsf{partOf} \in \langle \textsf{Museum}, \textsc{True} : \textsf{name} \in \{[\text{N*, Z*}]\}\rangle\rangle$$

using a range split by the constraint on the attribute **name**.

For every pair of source class D and query class Q, each split type guarantees that given an inner node with node class N there exists a child node with node class N' such that $(D \rightsquigarrow_{\mathrm{Q}} Q) \wedge (N \rightsquigarrow_{\mathrm{I}} D) \wedge (N \rightsquigarrow_{\mathrm{Q}} Q)$ implies $(N' \rightsquigarrow_{\mathrm{I}} D) \wedge (N' \rightsquigarrow_{\mathrm{Q}} Q)$. This property is essential for the completeness of indexing of the SDC-Tree, i.e. that the SDC-Tree correctly determines all source classes D with $D \rightsquigarrow_{\mathrm{Q}} Q$ for a given query Q. A proof of this property is given below in Section 4.3.

The SDC-Tree gives a lot of freedom regarding the splitting of nodes, despite the fact that base and existence splits generally are first required to enable range splits. Therefore, different strategies for the selection and parameterization of split operations are feasible.

For example, a thinkable approach is to first perform a series of base splits until no more base splits are possible, then to perform existence splits on all attributes and relations, and then to perform range splits where reasonable. However, in case that the source classes mainly differ in terms of nested classes on a certain relation this approach would generate many unnecessary nodes.

Furthermore, one may think of tailored split algorithms for each ontology taking the semantics and characteristics of the different classes, attributes, and relations into account. For example, in a context ontology the characteristic attribute **location** is likely to be used in many source classes and therefore an obvious candidate for splitting.

Before we propose the Generic Split Algorithm, which is independent of the semantics of the shared ontology and thus HIS, we next give formal definitions for node classes and the index matching predicate $\rightsquigarrow_{\mathrm{I}}$ and prove the completeness of indexing.

4.2 Formalism for Node Classes and Index Predicates

In the following, we give formal definitions for node classes and $\rightsquigarrow_{\mathrm{I}}$ and introduce an *index subsumption predicate* $\succeq_{\mathrm{I}}$.

Definition 6 [Node class]: A node class N is a defined class according to the definition in Section 3.2 with an additional Boolean parameter b as follows:

$$N = \langle C, b : a_1 \in X_1 \wedge a_2 \in X_2 \wedge \ldots \wedge r_1 \in \bar{N}_1 \wedge r_2 \in \bar{N}_2 \wedge \ldots \rangle$$

The parameter b denotes whether N may match other node classes or defined classes with base $C' \prec C$ by $\rightsquigarrow_{\mathrm{I}}$ and can be queried by the unary operator $\textsc{inclSub}(N) = b$.

A range X_i may be $\textsc{Null}$ to indicate that N prevents constraints on the corresponding attribute a_i. This property can be queried by a Boolean unary operator $\textsc{isPrev}_{a_i}(N)$. Hence, N may either

1. specify no constraint $(\neg\textsc{isCon}_{a_i}(N) \wedge \neg\textsc{isPrev}_{a_i}(N))$, or
2. specify a constraint $(\textsc{isCon}_{a_i}(N) \wedge \neg\textsc{isPrev}_{a_i}(N))$, or
3. prevent a constraint $(\neg\textsc{isCon}_{a_i}(N) \wedge \textsc{isPrev}_{a_i}(N))$.

The same applies to constraints on relations.

Definition 7 [Index matching predicate $\rightsquigarrow_{\mathrm{I}}$]: A node class N matches a source class D, denoted by $N \rightsquigarrow_{\mathrm{I}} D$, iff the following three conditions hold:

1. $(\mathcal{Base}(N) = \mathcal{Base}(D)) \vee (\textsc{inclSub}(N) \wedge (\mathcal{Base}(N) \succ \mathcal{Base}(D)))$
2. $\forall$ attribute a with $\mathcal{Dom}(a) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_a(N) \Rightarrow (\textsc{isCon}_a(D) \wedge (\mathcal{Con}_a(N) \cap \mathcal{Con}_a(D) \neq \emptyset))$,
   $\textsc{isPrev}_a(N) \Rightarrow \neg\textsc{isCon}_a(D)$
3. $\forall$ relation r with $\mathcal{Dom}(r) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_r(N) \Rightarrow (\textsc{isCon}_r(D) \wedge (\mathcal{Con}_r(N) \rightsquigarrow_{\mathrm{I}} \mathcal{Con}_r(D)))$,
   $\textsc{isPrev}_r(N) \Rightarrow \neg\textsc{isCon}_r(D)$

Since each of the three conditions implies the respective condition of $\rightsquigarrow_{\mathrm{Q}}$, it follows that $N \rightsquigarrow_{\mathrm{I}} D$ implies $N \rightsquigarrow_{\mathrm{Q}} D$.

Analogous to the definition of $\rightsquigarrow_{\mathrm{I}}$, a transitive *index subsumption predicate* $\succeq_{\mathrm{I}}$ implying $\succeq_{\mathrm{Q}}$ can be defined – between pairs of node classes as well as pairs of node classes and source classes.

Definition 8 [Index subsumption predicate $\succeq_{\mathrm{I}}$]: A node class N subsumes another node class N', denoted by $N \succeq_{\mathrm{I}} N'$, iff the following three conditions hold:

1. $(\textsc{inclSub}(N) \wedge (\mathcal{Base}(N) \succeq \mathcal{Base}(N'))) \vee$
   $(\neg\textsc{inclSub}(N) \wedge \neg\textsc{inclSub}(N') \wedge (\mathcal{Base}(N) = \mathcal{Base}(N')))$
2. $\forall$ attribute a with $\mathcal{Dom}(a) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_a(N) \Rightarrow (\textsc{isCon}_a(N') \wedge (\mathcal{Con}_a(N) \supseteq \mathcal{Con}_a(N')))$,
   $\textsc{isPrev}_a(N) \Rightarrow \textsc{isPrev}_a(N')$
3. $\forall$ relation r with $\mathcal{Dom}(r) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_r(N) \Rightarrow (\textsc{isCon}_r(N') \wedge (\mathcal{Con}_r(N) \succeq_{\mathrm{I}} \mathcal{Con}_r(N')))$,
   $\textsc{isPrev}_r(N) \Rightarrow \textsc{isPrev}_r(N')$

For the subsumption of a source class D, it holds: $N \succeq_{\mathrm{I}} D$

iff the following three conditions hold:

1. $(\mathcal{Base}(N) = \mathcal{Base}(D)) \vee (\textsc{inclSub}(N) \wedge (\mathcal{Base}(N) \succ \mathcal{Base}(D)))$
2. $\forall$ attribute a with $\mathcal{Dom}(a) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_a(N) \Rightarrow (\textsc{isCon}_a(D) \wedge (\mathcal{Con}_a(N) \supseteq \mathcal{Con}_a(D)))$,
   $\textsc{isPrev}_a(N) \Rightarrow \neg\textsc{isCon}_a(D)$
3. $\forall$ relation r with $\mathcal{Dom}(r) \succeq \mathcal{Base}(N)$:
   $\textsc{isCon}_r(N) \Rightarrow (\textsc{isCon}_r(D) \wedge (\mathcal{Con}_r(N) \succeq_{\text{I}} \mathcal{Con}_r(D)))$,
   $\textsc{isPrev}_r(N) \Rightarrow \neg\textsc{isCon}_r(D)$

It holds that the node classes of the SDC-Tree form a subsumption hierarchy with respect to $\succeq_{\text{I}}$ and thus $\succeq_{\text{Q}}$.

In summary, the query predicates $\rightsquigarrow_{\text{Q}}$, $/\!\!/_{\text{Q}}$, and $\succeq_{\text{Q}}$ ignore the two extensions of node classes by $\textsc{inclSub}$ and $\textsc{isPrev}$, whereas the index predicates $\rightsquigarrow_{\text{I}}$ and $\succeq_{\text{I}}$ take both extensions into account. The resulting implications between the five predicates can be depicted by a graph as follows:

$$\begin{array}{ccccc} \succeq_{\text{Q}} & \Rightarrow & \rightsquigarrow_{\text{Q}} & \Rightarrow & \text{not } /\!\!/_{\text{Q}} \\ \Uparrow & & \Uparrow & & \\ \succeq_{\text{I}} & \Rightarrow & \rightsquigarrow_{\text{I}} & & \end{array}$$

4.3 Proof for the Completeness of Indexing

To show that the SDC-Tree correctly determines all source classes $D \rightsquigarrow_{\text{Q}} Q$ for a query Q, we prove that for each pair D and Q there exists a leaf node with node class $N_k \rightsquigarrow_{\text{I}} D$ and $N_k \rightsquigarrow_{\text{Q}} Q$ and that this leaf node is found when matching the node classes from the root downwards to the leaf nodes against Q by $\rightsquigarrow_{\text{Q}}$.

Theorem 1 [Completeness of indexing]: For every pair of source class D and query class Q with $D \rightsquigarrow_{\text{Q}} Q$, there exists a sequence of nodes with node classes $N_1, \ldots, N_k$ from the root with node class $N_1 = \langle C_\top, \textsc{True} : \rangle$ to a leaf node with node class N_k such that

$$\forall N_i \in \{N_1, \ldots, N_k\} : (N_i \rightsquigarrow_{\text{I}} D) \wedge (N_i \rightsquigarrow_{\text{Q}} Q) \ .$$

Proof: Induction starting at node class $N_1 = \langle C_\top, \textsc{True} : \rangle$ of the root node.

Base case: Since N_1 has base $C_\top$ and no constraints, it holds $N_1 \rightsquigarrow_{\text{I}} D$ and $N_1 \rightsquigarrow_{\text{I}} Q$, independent of whether D matches Q by $\rightsquigarrow_{\text{Q}}$ or not.

Inductive step: For each split type, we show that given an inner node with node class N_i $(1 \leq i < k)$, there exists a child node with node class N_{i+1} such that $(D \rightsquigarrow_{\text{Q}} Q) \wedge (N_i \rightsquigarrow_{\text{I}} D) \wedge (N_i \rightsquigarrow_{\text{Q}} Q)$ implies $(N_{i+1} \rightsquigarrow_{\text{I}} D) \wedge (N_{i+1} \rightsquigarrow_{\text{Q}} Q)$. For readability, we do not consider splits by means of nested classes of N_i in the following, as the argumentations can be applied to them analogously.

1. *Base split:* It is $\textsc{inclSub}(N_i) = \textsc{True}$ and $\mathcal{Base}(N_i) \succeq \mathcal{Base}(D)$ since $N_i \rightsquigarrow_{\text{I}} D$.

   If $\mathcal{Base}(N_i) = \mathcal{Base}(D)$, then N_i and the node class N_{i+1} of the first child only differ in the value of $\textsc{inclSub}$. Thus, it holds $N_{i+1} \rightsquigarrow_{\text{I}} D$ and $N_{i+1} \rightsquigarrow_{\text{Q}} Q$.

   Otherwise, if $\mathcal{Base}(N_i) \succ \mathcal{Base}(D)$, then there exists a child with node class N_{i+1} with $\mathcal{Prnt}(\mathcal{Base}(N_{i+1})) = \mathcal{Base}(N_i)$ and $\mathcal{Base}(N_{i+1}) \succeq \mathcal{Base}(D)$. Hence, it holds $N_{i+1} \rightsquigarrow_{\text{I}} D$, and thus also $N_{i+1} \rightsquigarrow_{\text{Q}} Q$ since $D \rightsquigarrow_{\text{Q}} Q$.

2. *Existence split:* Let be a the attribute that has been used for the existence split. Thus, it is $\textsc{isCon}_a(N_i) = \textsc{False}$ and $\textsc{isPrev}_a(N_i) = \textsc{False}$.

   If D has a constraint on a, then the node class N_{i+1} of the child node defining a constraint to $\mathcal{Rng}(a)$ will match D by $\rightsquigarrow_{\text{I}}$. As $D \rightsquigarrow_{\text{Q}} Q$ either requires Q to also have a constraint on a or $\mathcal{Base}(Q) \succ \mathcal{Dom}(a)$, it also holds $N_{i+1} \rightsquigarrow_{\text{Q}} Q$.

   If D does not have a constraint on a, the node class N_{i+1} of the other child preventing constraints on a will match D by $\rightsquigarrow_{\text{I}}$. Clearly, it also holds $N_{i+1} \rightsquigarrow_{\text{Q}} Q$ since $\rightsquigarrow_{\text{Q}}$ ignores $\textsc{isPrev}_a(N_{i+1})$.

   The same argumentation applies to an existence split by means of a relation r.

3. *Range split:* Let a be the attribute that has been used for the range split. Next, we distinguish whether $\mathcal{Base}(Q) \succ \mathcal{Dom}(a)$ or $\mathcal{Base}(Q) \preceq \mathcal{Dom}(a)$.

   In the former, $N_i \rightsquigarrow_{\text{Q}} Q$ implies $N_{i+1} \rightsquigarrow_{\text{Q}} Q$ as $\rightsquigarrow_{\text{Q}}$ evaluates neither the value range $\mathcal{Con}_a(N_i)$ nor $\mathcal{Con}_a(N_{i+1})$ of the node class N_{i+1} of any child node.

   In the latter, we again have to distinguish whether $\mathcal{Con}_a(D)$ and $\mathcal{Con}_a(Q)$ overlap within the range $\mathcal{Con}_a(N_i)$ or not.

   If they overlap, i.e. $(\mathcal{Con}_a(D) \cap \mathcal{Con}_a(Q)) \cap \mathcal{Con}_a(N_i) \neq \emptyset$, then there exists at least one child with node class N_{i+1} where they overlap within $\mathcal{Con}_a(N_{i+1})$ since $\mathcal{Con}_a(N_i)$ is completely partitioned. Thus, it holds $N_{i+1} \rightsquigarrow_{\text{I}} D$ and $N_{i+1} \rightsquigarrow_{\text{Q}} Q$.

   The converse, i.e. that $\mathcal{Con}_a(D)$ and $\mathcal{Con}_a(Q)$ are disjoint within $\mathcal{Con}_a(N_i)$, cannot happen due to the inductive construction of $N_1, \ldots, N_i$: The first range split on a at node class N_f $(f < i)$ was performed on $\mathcal{Con}_a(N_f) = \mathcal{Rng}(a)$ definitely comprising $\mathcal{Con}_a(D) \cap \mathcal{Con}_a(Q)$. At this range split and any further range split on a, we always chose the child with node class N_j $(f < j < i)$ such that $\mathcal{Con}_a(N_j)$ also comprises $\mathcal{Con}_a(D) \cap \mathcal{Con}_a(Q)$, as just explained.

□

4.4 Generic Split Algorithm

The SDC-Tree gives a lot of freedom regarding the splitting of nodes, despite the fact that base and existence splits generally are first required to enable range splits. Therefore, different strategies for node splitting are imaginable.

In the following, we propose the *Generic Split Algorithm* (GSAlg), which is independent of the semantics of the shared ontology. GSAlg splits a leaf node once the number of entries stored in the leaf reaches a certain *split size threshold* n_{split}.

For deciding on the child node classes, GSAlg proceeds as follows: First, it computes all possible splits. Then, it determines how the source classes are distributed to the different node classes of each split and rates the splits accordingly. Finally, GSAlg chooses the split with the best rating.

For computing all possible splits of a leaf node with node class N storing the source classes $\mathbb{D}$, GSAlg uses a recursive function given in Figure 3. First, it computes the base split of N if possible (lines 2 to 4), before it computes all possible existence splits (lines 5 to 10). GSAlg then computes one possible range split, with two subranges X_1 and X_2, for each attribute constraint (lines 11 to 16). For this purpose, GSAlg uses different variants of the split algorithm of the R*-Tree [7], adapted to the different primitive data types. This algorithm takes the ranges of the constraints

```
1: $\mathbb{S} \leftarrow \emptyset$    ▷ Result set of all possible splits.
2: if $\textsc{inclSub}(N)$ and $|\{C' : \mathcal{Prnt}(C') = \mathcal{Base}(N)\}| > 0$ then
3:     $\mathbb{S} \leftarrow \{\{N$ with $\textsc{inclSub}(N) \leftarrow \textsc{False}$,
               $N$ with $\mathcal{Base}(N) \leftarrow$ first $C', \ldots\}\}$
4: end if
5: for all attribute $a$ where $\mathcal{Dom}(a) \succeq \mathcal{Base}(N)$ do
6:     if $\neg\textsc{isCon}_a(N)$ and $\neg\textsc{isPrev}_a(N)$ then
7:         $\mathbb{S} \leftarrow \mathbb{S} \cup \{\{N$ with $\mathcal{Con}_a(N) \leftarrow \mathcal{Rng}(a)$,
                   $N$ with $\textsc{isPrev}_a(N) \leftarrow \textsc{True}\}\}$
8:     end if
9: end for
10: [...]    ▷ Same for each relation $r$ where $\mathcal{Dom}(r) \succeq \mathcal{Base}(N)$
11: for all attribute $a$ with $\mathcal{Dom}(a) \succeq \mathcal{Base}(N)$ do
12:     if $\textsc{isCon}_a(N)$ and $|\mathcal{Con}_a(N)| > 1$ then
13:         $(X_1, X_2) \leftarrow$ compute partitioning of $\mathcal{Con}_a(N)$
14:         $\mathbb{S} \leftarrow \mathbb{S} \cup \{\{N$ with $\mathcal{Con}_a(N) \leftarrow X_1$,
                   $N$ with $\mathcal{Con}_a(N) \leftarrow X_2\}\}$
15:     end if
16: end for
17: for all relation $r$ where $\mathcal{Dom}(r) \succeq \mathcal{Base}(N)$ do
18:     if $\textsc{isCon}_r(N)$ then
19:         $\bar{\mathbb{D}} \leftarrow \{\mathcal{Con}_r(D) : \forall D \in \mathbb{D}\}$ ▷ Get nested source classes.
20:         $\bar{\mathbb{S}} \leftarrow$ recursive call with $\mathcal{Con}_r(N)$ and $\bar{\mathbb{D}}$
21:         for all splits $\bar{\mathbb{N}} \in \bar{\mathbb{S}}$ do
22:             $\mathbb{S} \leftarrow \mathbb{S} \cup \{N$ with $\mathcal{Con}_r(N) \leftarrow \bar{N} : \forall \bar{N} \in \bar{\mathbb{N}}\}$
23:         end for
24:     end if
25: end for
26: return $\mathbb{S}$
```

Figure 3: Algorithm for computing all possible splits of a node with node class N, storing the source classes $\mathbb{D}$.

of the source classes $\mathbb{D}$ into account such that X_1 and X_2 are optimized to divide $\mathbb{D}$ into equal-sized halves as possible. Finally, a recursive call is performed for each nested node class $\bar{N} = \mathcal{Con}_r(N)$ (lines 17 to 25).

The number of possible splits is limited by the number of attributes and relations of the different classes of the shared ontology and therefore widely independent of the size of the SDC-Tree. Furthermore, it is largely independent of the number of classes in the ontology, i.e. the size of the IS-A hierarchy, which can be seen from the fact that the function in Figure 3 adds at most one base split to $\mathbb{S}$ during each (recursive) call. Only for the very first splits, i.e. small tree sizes, there exist less possible splits since base splits are generally required to enable other types of splits.

For rating the possible splits, we propose three tailored rating functions – one for each split type – based on a uniform metric from 0 (bad) to 1 (good).

A base split is only useful if there exists a significant number of source classes $D \in \mathbb{D}$ with $\mathcal{Base}(D) \prec \mathcal{Base}(N)$. The rating function therefore simply is

$$\frac{|\{D \in \mathbb{D} : \mathcal{Base}(D) \prec \mathcal{Base}(N)\}|}{n_{\text{split}}} .$$

The utility of an existence split on attribute a increases with the number of source classes having a constraint on a since the split is required to enable range splits. However, it may be also useful to differentiate between source classes that have a constraint on a and those that have not. Therefore, the rating function is

$$\min\left(1,\ \frac{2\,|\{D \in \mathbb{D} : \textsc{isCon}_a(D)|}{n_{\text{split}}}\right) .$$

The same applies to existence splits on relations.

A range split on $\mathcal{Con}_a(N)$ by GSAlg is particularly useful if it partitions $\mathbb{D}$ into roughly equal-sized halves. Also, the number of source classes that are matched by both new node classes N_1' and N_2' should be small. We refer to the former property as *distribution* and to the latter as *selectivity.* The selectivity is computed as

$$\frac{|\{D \in \mathbb{D} : (N_1' \rightsquigarrow_{\text{I}} D) \neq (N_2' \rightsquigarrow_{\text{I}} D)\}|}{n_{\text{split}}}$$

and the distribution as

$$\min\left(1,\ \frac{2\,|\{D \in \mathbb{D} : N_1' \rightsquigarrow_{\text{I}} D\}|}{n_{\text{split}}},\ \frac{2\,|\{D \in \mathbb{D} : N_2' \rightsquigarrow_{\text{I}} D\}|}{n_{\text{split}}}\right) .$$

The rating of a range split is the product of both. This normally ensures that a bad selectivity or distribution each serve as exclusion criterion for being the best possible split, i.e. the split with the best rating. Our simulation results in Section 5.3 show that the ratings of the best possible range splits are close to 1, despite the multiplication of selectivity and distribution.

All ratings can be easily computed along with the possible splits in the above algorithm.

5. EVALUATION

As proof of concept for describing sources and to show the efficiency of the SDC-Tree and GSAlg, we implemented the SDC-Tree as a main-memory index and simulated it with source descriptions derived from the OpenStreetMap (OSM) database [24]. Next, we first describe the evaluation setup, followed by the results on search costs and tree sizes. Finally, we give results on insertion costs and node splitting.

5.1 Setup

The components of the shared ontology and the source descriptions could generally be expressed by a subset of OWL [32] and SPARQL [33]. For simplicity and direct support of 2D geometries, however, we developed an easily readable *Simple Ontology Language* (SOL) inspired by the notation for defined classes used in the previous sections, and implemented an appropriate ontology framework. For example, the defined class

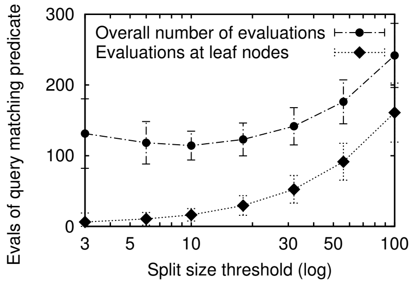
$$\langle \textsf{BuildingPart} : \textsf{partOf} \in \langle \textsf{Museum} : \textsf{name} \in \{\text{“British Museum”}\}\rangle\rangle$$

is specified as

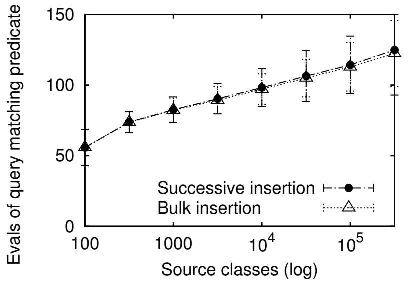
```
<BuildingPart : partOf IN <Museum : name IN
                          {String:"British Museum"}>>
```

in SOL. 2D geometries simply are specified by Well-Known Text (WKT) [23].

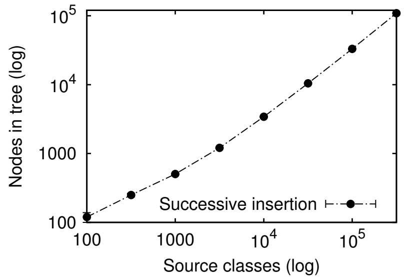
The ontology framework supports strings, integers and 2D geometries as primitive data types and provides common relational operators on their values (e.g., $<$ and $>$) and on sets of their values (e.g., $\subseteq$ and $\supseteq$). For managing 2D geometries, the framework uses the Java Topology Suite [31]. Regarding strings, the framework supports arbitrary intervals and sets of intervals on the lexicographical order of the strings. Hence, it not only allows specifying intervals of strings sharing a certain prefix but *any* finite or infinite interval such as



(a) Search costs depending on split size threshold.



(b) Search costs depending on source classes.



(c) Tree size in nodes depending on source classes.

Figure 4: Evaluation results concerning search costs and tree sizes.

[Museum, Museum] and [MA, MC), specifying the single string *Museum* and the set of all strings starting with *MA* or *MB*, respectively.

In accordance with the examples of the previous sections, we evaluated the SDC-Tree with potential source descriptions and discovery queries in a large-scale spatial context information system. To simulate realistic descriptions as possible, we used the OSM database as follows:

1. Since the OSM database uses a flat tagging system, we created a spatial context ontology on the basis of three established top-level ontologies – namely the Alexandria Digital Library Feature Type Thesaurus [30], the Suggested Upper Merged Ontology (SUMO) [4], and the PROTON Ontology (PROTo ONtology) [27] – and mapped the most frequent tags to the corresponding classes, relations, and attributes. Figure 1 in Section 3 shows an excerpt of this ontology.

2. We created nine templates for source descriptions of possible information sources, e.g. of sources providing information about the building elements of a public building or about the transportation structures of a town. Each such description template consists of two or more templates for defined classes.

3. Using these templates and the OSM database extract for Europe, we generated $5 \cdot 10^5$ source descriptions with more than $1.1 \cdot 10^6$ source classes.

We simulated the SDC-Tree with different numbers of source descriptions (and thus source classes) as well as different values for n_{split}. For querying, we randomly selected 1000 of the source descriptions, merged the source classes of each description, and used the resulting 1000 defined classes as queries. In our simulations these queries matched up to 100 source descriptions. Each measurement was repeated 25 times with different subsets of the $5 \cdot 10^5$ source descriptions.

To be able to assess the performance of GSAlg with *successive* insertion and splitting as described in Section 4.4, we also created the SDC-Tree by *bulk* insertion as follows: The simulated number of descriptions is inserted at once into a SDC-Tree consisting of a single leaf node. Then, the leaf node with the most entries is split repeatedly using GSAlg until the number of nodes in the tree is equal the number of nodes generated by successive insertion. For the selection and parameterization of the split operations, the algorithm for computing all possible splits and the rating functions of GSAlg are used again – except that n_{split} is replaced by the actual number of entries in the leaf node.

5.2 Search Costs and Tree Sizes

First, we consider the search costs measured by the number of evaluations of $\rightsquigarrow_{\mathrm{Q}}$ per query and the tree size measured by the number of nodes.

Figure 4a shows the average search costs for a tree with 10^5 source classes depending on n_{split}. The upper line gives the overall number of $\rightsquigarrow_{\mathrm{Q}}$ evaluations, whereas the lower line denotes the evaluations with source classes only. The latter increases monotonously with n_{split} due to the larger number of source classes per leaf. The overall number of evaluations, however, also increases with decreasing $n_{\text{split}} < 10$ due to the resulting large numbers of nodes. Therefore, we used $n_{\text{split}} = 10$ in the following.

Figure 4b shows that the search cost depend logarithmically on the number of index source classes and thus demonstrates the effectiveness of the SDC-Tree. For 1000 source classes, the SDC-Tree reduces the number of $\rightsquigarrow_{\mathrm{Q}}$ evaluations to less than 10% compared to plain search over all descriptions. For 10^5 source classes, it even saves 99.9%. Besides, Figure 4b shows that the search costs with bulk insertion are only about 1% less than with successive insertion. Thus, successive insertion with GSAlg achieves the same indexing performance as bulk insertion to a great extent, although the latter considers the whole (but fixed) set of source descriptions for the split decisions.

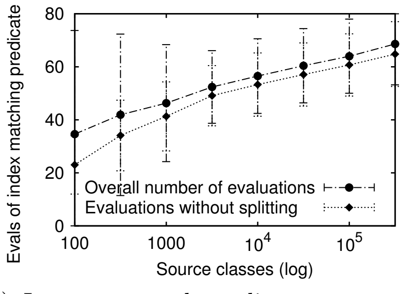
These results coincide with the linear dependency between the number of indexed source classes and the size of the SDC-Tree illustrated in Figure 4c. Note the marginal standard deviations in these measurements.

Furthermore, to verify the choice of the uniform rating metric in GSAlg, we conducted experiments with split-type specific weights. Our results show that weight factors are not necessary: For example, when underweighting range splits the search cost slightly increase, whereas overweighting does not yield any noticeable changes.

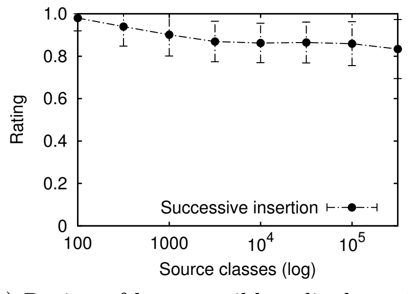
5.3 Insertion and Splitting

Next, we consider the insertion costs measured by the number of evaluations of $\rightsquigarrow_{\mathrm{I}}$ per source class and analyze the splitting with GSAlg. For this purpose, we recorded the costs and split statistics of the last 1000 successive insertions during each measurement.

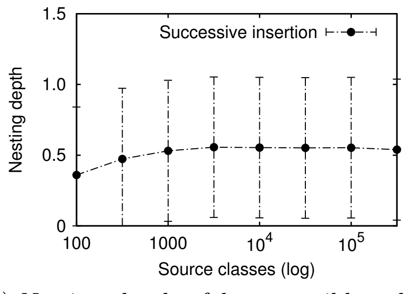
Figure 5a shows the average insertion costs depending on the number of indexed source classes. The lower line only



(a) Insert costs depending on source classes.



(b) Rating of best possible split depending on source classes.



(c) Nesting depth of best possible split depending on source classes.

Figure 5: Evaluation results concerning insertion and splitting.

denotes the $\rightsquigarrow_{\mathrm{I}}$ evaluation for determining the leaf nodes to index a given source class. The upper line additionally includes the average number of $\rightsquigarrow_{\mathrm{I}}$ evaluations for splitting. It again shows the effectiveness of the SDC-Tree: The costs for determining the relevant leaf nodes depend logarithmically on the number of source classes. The costs for splitting are widely independent of the number of source classes and only amount to about four evaluations of $\rightsquigarrow_{\mathrm{I}}$ per insertion.

Figure 5b gives the average rating of the best possible split depending on the number of indexed source classes. It starts with a value of about 1.0 since the first possible splits are base and existence splits only and all source classes have proper subclasses of $C_{\top}$ as bases. Then, the possibility of range splits increases more and more, and the average rating slightly decreases as the constraint ranges of the source classes often overlap such that GSAlg cannot determine an optimal partitioning with maximum selectivity and uniform distribution. With very large numbers of source classes this effect may increase, causing a slight decline in the average rating of the best possible split.

Figure 5c gives the nesting depth of the best possible split depending on the number of indexed source classes. A nesting depth of one specifies a split by means of a nested node class $\mathcal{C}on_r(N)$, a depth of two specifies a split by means of $\mathcal{C}on_{r'}(\mathcal{C}on_r(N))$, and so on. The simulated source classes have nesting depths of zero or one only, as we expect nesting depths of two or more to be rarely used in source description. This causes a standard deviation of 0.5 in our measurements. Nevertheless, the figure shows that GSAlg performs splits by means of nested node classes even for a tree indexing only 100 source classes. Thus, GSAlg rapidly adapts to source descriptions whose source classes mainly differ in terms of nested ones.

6. CONCLUSIONS

For scaling ontology-based HIS to thousands of information sources, we proposed a powerful formalism allowing to precisely describe the contents of a source by multiple *defined classes*. The formalism refines the idea of describing information sources by constraints in several ways: It enables alternative descriptions, and it features complex, nested constraints on arbitrary attributes and relations. Finally, it allows adjusting between positive and negative matching semantics.

To enable efficient source discovery in large-scale HIS, we further proposed the SDC-Tree for indexing of source descriptions by means of their defined classes. Also, we presented a generic algorithm for splitting of leaf nodes during insertions and showed the efficiency of the SDC-Tree in evaluations.

The proposed approach provides the foundation for realizing efficient, precise source discovery services in large-scale HIS – either as centralized, possibly replicated services or as distributed services by partitioning the SDC-Tree to sets of servers.

7. ACKNOWLEDGMENTS

The work described in this paper was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center (SFB) 627.

8. REFERENCES

[1] K. Arabshian and H. Schulzrinne. An Ontology-based Hierarchical Peer-to-Peer Global Service Discovery System. *JUCI*, 1(2):133–144, Dec. 2007.

[2] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. *SIGMOD Record*, 3(3):53–58, Sept. 2003.

[3] Y. Arens, C. A. Knoblock, and W.-M. Shen. Query Reformulation for Dynamic Information Integration. *J. of Intelligent Inf. Sys.*, 6(2–3):99–130, June 1996.

[4] Articulate Software. Suggested Upper Merged Ontology. http://www.ontologyportal.org/.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, Cambridge, UK, 2003.

[6] C. Becker and F. Dürr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9(1):20–31, Jan. 2005.

[7] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. of SIGMOD '09*, pages 322–331, Atlantic City, NJ, May 1990.

[8] S. Bergamaschi, S. Castano, and M. Vincini. Semantic Integration of Semistructured and Structured Data Sources. *ACM SIGMOD Record*, 28(1):54–59, Mar. 1999.

[9] R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker. ObjectGlobe:

Ubiquitous query processing on the Internet. *VLDB J.*, 10(1):48–71, 2001.

[10] M. R. Genesereth, A. M. Keller, and O. M. Duschka. Infomaster: An Information Integration System. In *Proc. of 1997 SIGMOD*, pages 539–542, Tucson, AZ, May 1997.

[11] L. Gravano, H. García-Molina, and A. Tomasic. GlOSS: Text-Source Discovery over the Internet. *ACM TODS*, 24(2):229–264, June 1999.

[12] T. Gu, H. K. Pung, and D. Zhang. A Peer-to-Peer Overlay for Context Information Search. In *Proc. of 14th ICCCN*, pages 395–400, San Diego, CA, Oct. 2005.

[13] P. Haase. *Semantic Technologies for Distributed Information Systems*. Karlsruhe University Press, Karlsruhe, Germany, Jan. 2007.

[14] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer Data Management System. *IEEE TKDE*, 16(7):787–798, July 2004.

[15] R. A. King, A. Hameurlain, and F. Morvan. Ontology-Based Data Source Localization in a Structured Peer-to-Peer Environment. In *Proc. of 12th IDEAS*, pages 9–18, Coimbra, Portugal, Sept. 2008.

[16] H. Kondylakis, A. Analyti, and D. Plexousakis. Quete: Ontology-Based Query System for Distributed Sources. In *Proc. of 11th ADBIS*, pages 359–375, Varna, Bulgaria, Sept. 2007.

[17] R. Lange, N. Cipriani, L. Geiger, M. Großmann, H. Weinschrott, A. Brodt, M. Wieland, S. Rizou, and K. Rothermel. Making the World Wide Space Happen: New Challenges for the Nexus Context Platform. In *Proc. of 7th PerCom*, pages 300–303, Galveston, TX, Mar. 2009.

[18] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of 22th VLDB*, pages 251–262, Mumbai (Bombay), India, Sept. 1996.

[19] C. Li. Using Constraints to Describe Source Contents in Data Integration Systems. *IEEE Intelligent Systems*, 18(5):49–53, Sept. 2003.

[20] G. Meditskos and N. Bassiliades. Structural and Role-Oriented Web Service Discovery with Taxonomies in OWL-S. *IEEE TKDE*, 22(2):278–290, Feb. 2010.

[21] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. In *Proc. of 19th ICDE*, pages 633–644, Bangalore, India, Mar. 2003.

[22] M. M. Nodine, A. H. H. Ngu, A. Cassandra, and W. G. Bohrer. Scalable Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth. *IEEE TKDE*, 15(5):1082–1098, Sept. 2003.

[23] Open Geospatial Consortium, Inc. OpenGIS Implementation Specification for Geographic information. http://www.opengeospatial.org/, Oct. 2006.

[24] OpenStreetMap Project. OpenStreetMap. http://www.openstreetmap.org/.

[25] M. Ouzzani, B. Benatallah, and A. Bouguettaya. Ontological Approach for Information Discovery in Internet Databases. *Dist. and Parallel Databases*, 8(3):367–392, July 2000.

[26] T. Risch, V. Josifovski, and T. Katchaounov. *Functional Approach to Data Management*, chapter Functional Data Integration in a Distributed Mediator System. Springer, 2003.

[27] SEKT Project. PROTON Ontology. http://proton.semanticweb.org/.

[28] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Trans. on Networking*, 11(1):2003, Feb. 2003.

[29] A. Tomasic, L. Raschid, and P. Valduriez. Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE TKDE*, 10(5):808–823, Sept. 1998.

[30] University of California. Alexandria Digital Library Feature Type Thesaurus. http://www.alexandria.ucsb.edu/.

[31] Vivid Solutions Inc. Java Topology Suite. http://sourceforge.net/projects/jts-topo-suite/.

[32] W3C. OWL Web Ontology Language: Reference. http://www.w3.org/TR/owl-ref/, Feb. 2004.

[33] W3C. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, Jan. 2008.