Schema Extraction for Semi-Structured Data

M.S. Hacid†, F.Soualmia‡ and F.Toumani‡

† LISI-INSA, 20 avenue Albert Einstein - 69621 Villeurbanne - France

Email: mshacid@lisi.insa-lyon.fr

‡ LIMOS, ISIMA CAMPUS des Cezeaux - B.P.125

Email: soualmia@libd2.univ-bpclermont.fr, ftoumani@isima.fr,

Abstract

The emerging field of semistructured data leads to new ways of representing data as *'schemaless'* or *'self-describing'*. However, in many applications data has often some regularity and ignoring the (possibly partial) structure hinders the abilities to interpret the data and to access them efficiently. In this paper we investigate a knowledge-based approach for discovering (partial) implicit structures from semistructured data. We show that semistructured data, represented in the form of labeled directed graphs can be typed using description logics.

1 Introduction

Semistructured data is characterized by a structure that can be absent, irregular, implicit or partial [2, 5]. In many application areas such as digital libraries, genome databases and electronic commerce, data reside in *semistructured* forms and cannot be constrained by a rigid schema as in conventional database systems. Recently, new kind of data models, less constraining than traditional ones, has been proposed for representing semistructured data [2, 5]. In these models, data is described by means of a labeled directed graph in which the vertices represent objects and the labels on the edges convey semantic information about the relationship between objects [1].

Such an approach for modeling semistructured data is called *minimalist* since schematic information on data is always missed, even when it is partially or implicitly available [2]. However, very often, data is not fully unstructured and exploiting, even a partial structure, can be very beneficial, e.g., for browsing and querying the data, optimizing query evaluation or improving storage [5, 1]. Therefore, discovering structures, even partial, from semistructured data is significant [2, 5, 12, 6, 14].

In this paper, we investigate a novel approach for discovering implicit structures from semistructured data. We show that semistructured data, represented in the form of labeled directed graphs, can be typed using description logics. The idea of typing semistructured data is somewhat novel and lack of foundations [1]. We resort to a formal framework, based on graph simulation [5], to study the appropriateness of description logics to describe semistructured data.
Since a semistructured database can have more than one type, we consider first the extraction of the schema that best describes the data. For each object we compute a single type description reflecting relevant assertions about the object in the database. The computed description is the *most specific concept*, expressible in our type description language, of which the object is an instance. The resulting schema is the *most specific* one abstracting the database content and corresponds to perfect typing.
Second, due to the irregularity of structures in semistructured databases, perfect typing may lead to large sets of descriptions [1]. We show that it is possible to identify similar, and not only identical, objects, and providing an approximate typing. For that, by computing least common subsumers of some appropriate descriptions in the most specific schema of the previous step, we build a compressed version of this most specific schema. The ability of description logics to handle partial information in conjunction with an open world assumption makes them suitable for approximate typing.
The rest of this paper is organized as follows. Background knowledge on semistructured data is provided in Section 2. Appropriateness of the used description logic for typing semistructured data is discussed in Section 3. The problem of discovering *typical* structural information of semistructured data is addressed in Section 4. We conclude in Section 5.

2 Semistructured data

We review here the basic concepts in semistructured databases and typing. Informally, a semistructured data model describes data using a graph, called *data graph*, with objects as vertices and labels on the edges. Each object has a unique *identifier* from the type oid.

Definition 1 (Data graph) A data graph $DB = (V_a \cup V_c, E, r)$ is a labeled rooted graph, where V_a and V_c are disjoint sets of oid's corresponding respectively to *atomic* and *complex* objects. $V_a \cup V_c$ forms a finite set of nodes. $r \in V_c$ is a root node; $E \subseteq V_c \times L \times (V_a \cup V_c)$ is a set of labeled edges, where L is an infinite set of strings indicating labels. (a, l, b) will denote an edge going from the node a to the node b and labeled l. We assume that all the nodes in $V_a \cup V_c$ are reachable from the root node.

In this paper we consider only semistructured data whose graph is acyclic. An example of a data graph in the style of OEM [15] is shown in figure 1. Vertices without outgoing edges in the graph represent atomic objects and have values from one of the disjoint basic types, e.g., Integer, String, Real, etc. Each other vertices represents a complex object which has labeled outgoing edges to other objects. In the sequel, we denote by $type(o)$ the type of an atomic object o. For example, in the data graph of figure 1, $type(n_3) =$ String.

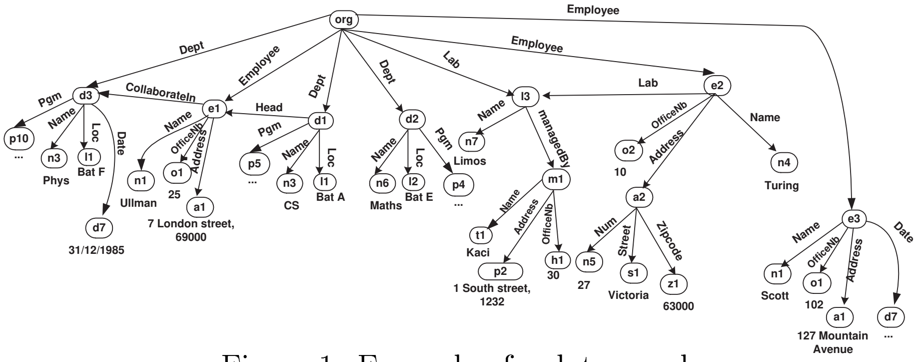

Figure 1: Example of a data graph

Recently, a new notion of schema appropriate for semistructured data has been proposed [5, 1]. Informally, a data graph is described using a *graph schema* which lists all *allowed* labels for each kind of object. In the data graph of figure 1, major kind of objects are, for example, department objects d_1, d_2 and d_3. A corresponding schema graph will describe this kind of objects by a vertices, let us say Department, with outgoing edges Pgm, Name, Loc, Head and Date. Note that, not all these edges are required in every department object.

A notion of *conformance* between a data graph and a schema graph is then defined based on *simulation* relation [5, 1].

Definition 2 (Simulation)
Let $G_1 = (V_1, E_1, r_1)$ and $G_2 = (V_2, E_2, r_2)$ be two graphs, where r_1 and r_2 are, respectively, the roots of G_1 and G_2. A relation R on V_1, V_2 is a simulation if it satisfies: (1) $r_1 R r_2$, and (2) $\forall x_1, y_1 \in V_1, \forall x_2 \in V_2 \; ((x_1, l, y_1) \in E_1 \wedge x_1 R x_2 \Rightarrow \exists y_2 \in V_2((x_2, l, y_2) \in E_2 \wedge (y_1 R y_2))$.

Informally, a simulation from a graph G_1 to a graph G_2 means that whenever there is an edge in G_1, there is a corresponding edge with the same label in G_2.

A data graph DB conforms to a schema graph S, in notation $DB \leq S$, if there exists a simulation from DB to S.

The notion of conformance allows to define an ordering on graph schemas. Informally, a schema graph S is a *refinement* of a schema graph S' iff whenever a data graph DB conforms to S it conforms to S' [5]. Refinement of schema graph can also be defined by means of simulation relation: Let S and S' be two schema graphs. We say that S' is a refinement of S, noted $S \leq S'$, if there exists a simulation from S to S'.

Since simulation relation is transitive [1], it is easy to see that if S is a refinement of S' (i.e., $S \leq S'$), then every data graph DB that conforms to S (i.e, $DB \leq S$)

also conforms to S' (i.e., $DB \leq S'$).

3 Description logics and semistructured data

Let us first introduce the $\mathcal{EL}$ description logic which will be used in this paper.

3.1 The $\mathcal{EL}$ language

The set of constructors allowed by the language $\mathcal{EL}$ are given below.
Let $\mathcal{C}$ and $\mathcal{R}$ be two pairwise disjoint sets of concept names and role names respectively. Let $A \in \mathcal{C}$ be a concept name and $R \in \mathcal{R}$ be a role name. Concepts C, D are defined inductively by the rules: $C, D \longrightarrow \top \mid \bot \mid A \mid C \sqcap D \mid \exists R.C$
The semantics of these constructors as well as subsumption (respectively, equivalence) of $\mathcal{EL}$-concepts ($C \sqsubseteq D$) (respectively, $C \equiv D$) is defined in the usual way.
A knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ built by means of the $\mathcal{EL}$ description logic is formed by two components: a TBox $\mathcal{T}$, the *intensional* one, and an ABox $\mathcal{A}$, the *extensional* one. A TBox (or a terminology) $\mathcal{T}$ is a (finite) set of *concept definitions* ($B = D$, where B is a concept name and D a concept). An ABox is a finite set of assertions of the form $R(a, b)$ (role assertion) and $C(a)$ (concept assertion), where a, b denote *object names* (or simply *objects*), R is a role name, and C is an $\mathcal{EL}$-concept.
The semantics of TBoxes and ABoxes is defined as usual.

Reasoning services Various kind of reasoning services (e.g., subsumption, concept satisfiability, ...) provided by knowledge representation systems based on description logics are described in the literature (see among others [11, 8]). In the following, we mainly resort to non standard inference services, namely the computation of the *least common subsumer* (lcs) of a set of descriptions and the *most specific concept* (msc) of an object.
Let $C_1, \ldots, C_n$ and E be concept descriptions in $\mathcal{EL}$. The concept description E is a least common subsumer of $C_1, \ldots, C_n$ (noted $E = lcs(C_1, \ldots, C_n)$ iff: *(1)* $C_i \sqsubseteq E$ for all $1 \leq i \leq n$, and *(2)* E is the least concept description with this property, i.e., if E' is a concept description satisfying $C_i \sqsubseteq E'$ for all $1 \leq i \leq n$, then $E \sqsubseteq E'$ [4].
The msc is a process that abstract an object o by constructing a very specific description $msc(o)$ that characterize o. Let $\mathcal{A}$ be an ABox, o be an object in $\mathcal{A}$, and C be a concept description. C is the *most specific concept* for o in $\mathcal{A}$, noted $C = msc(o)$, iff $o \in C^{\mathcal{J}}$ and if C' is a concept description satisfying $o \in C'^{\mathcal{J}}$, then $C \sqsubseteq C'$.
Techniques for computing least common subsumers and most specific concepts are described in [11, 7, 3, 4]. Subsumption in $\mathcal{EL}$ can be decided in polynomial time. Computing the *lcs* of two $\mathcal{EL}$-descriptions C, D can be done in polynomial time and the size of the *lcs* is polynomial in the size of C and D[4].

3.2 Description logics for typing semistructured data

In order to show how description logics can be used to effectively define an abstract layer on top of a semistructured database, we make use of simulation. For that, we first represent a TBox by means of a graph based representation. Such a representation is already used for developing subsumption algorithms [4]. Following this approach, our TBoxes can be viewed as a graph: a TBox $\mathcal{T}$ can be represented as a labeled directed graph $\mathcal{G}_\mathcal{T} = (\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T})$, where $\mathcal{V}_\mathcal{T}$ is the set of concept names appearing in $\mathcal{T}$ and $\mathcal{E}_\mathcal{T} \subseteq \mathcal{V}_\mathcal{T} \times \mathcal{R} \times \mathcal{V}_\mathcal{T}$ is a set of directed labeled edges, with $\mathcal{R}$ the set of role names appearing in $\mathcal{T}$. An edge $(N_i, l, N_j) \in \mathcal{E}_\mathcal{T}$ if N_i contains $\exists l.N_j$ in its definition. In the sequel, we call such a graph an $\mathcal{EL}$ graph.

Example 1 The TBox

$$Employee = \exists Name.String \sqcap \exists WorksIn.Department$$
$$Department = \exists Pgm.String \sqcap \exists Name.String \sqcap \exists Loc.String$$

yields the graph shown figure 2.

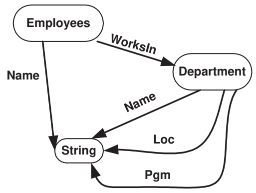

Figure 2: An $\mathcal{EL}$ graph

According to the semantics of the existential constructor ($\exists$), $\mathcal{EL}$ graphs list all *required* edges for a given node rather than the *allowed* edges as in schema graphs. This kind of graph is called *dual schema graph*[1]. For example, in the dual schema graph of figure 2, an object d is in class Department if it has at least the attributes Pgm, Name and Loc.

The semantics of dual schema graphs can also be described in terms of simulation, but now in reversed direction [1]. A data graph DB conforms to a dual schema graph S if $S \leq DB$.

According to this semantics, the notion of *most specific dual schema graph* can be defined as follows: S is the most specific dual schema graph of a data graph DB if whenever DB conforms to a dual schema graph S', $S' \leq S$. One can show that if a dual schema graph S is a *refinement* of a dual schema graph S' then whenever a graph data DB conforms to S, it conforms to S'.

4 Discovering structures

We consider the problem of discovering *typical* structural information of semistructured data. This problem, referred to as *schema discovery*, can be formulated as follows: given a data graph DB, find the corresponding TBox (and hence the $\mathcal{EL}$ graph) that describes the common substructures within DB.

In this section, we propose a bottom up approach to deal with the schema discovery problem. We consider first the problem of discovering the most specific dual schema graph of a data graph DB. Then, we show how this most specific schema can be compressed to obtain an approximate typing of the input data graph.

4.1 Extracting the most specific schema

We propose an algorithm, called Gen$\mathcal{K}$, that allows to *abstract* objects of a data graph.

Algorithm 1 Gen$\mathcal{K}$

Input: a data graph $DB = (V_a \cup V_c, E, L, v, r)$
Output: a knowledge base $\mathcal{K} = (\mathcal{T_S}, \mathcal{A})$.

`// Step 1 : Generating` $\mathcal{A}$
for each $(o_i, l, o_j) \in E$ **do**
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{l(o_i, o_j)\}$
end for
`// Step 2 : Generating` $\mathcal{T_S}$
`// Initialization of specific descriptions and atomic types`
for each o_i in V_a **do**
  `// If an object` o_j `is atomic we denotes by` $type(o_j)$ `its atomic type`
  $N_{o_j} \leftarrow type(o_j)$
end for
for each o_i in V_c **do**
  $\delta_{o_i} \leftarrow \emptyset$
end for
`// Computation of the exact structure for each object`
for each $l(o_i, o_j) \in \mathcal{A}$ **do**
  $\delta_{o_i} \leftarrow \delta_{o_i} \sqcap \exists l.N_{o_j}$
end for
for each generated δ_{o_i} **do**
  $\mathcal{T_S} \leftarrow \mathcal{T_S} \cup \{N_{o_i} = \delta_{o_i}\}$
end for
`// Step 3 : Merging equivalent concepts`
Merge together equivalent concepts in $\mathcal{T_S}$
return $\mathcal{K} = (\mathcal{T_S}, \mathcal{A})$

The algorithm Gen$\mathcal{K}$ works as follows:

- *Step 1*: It maps a data graph DB to an ABox $\mathcal{A}$. We view a semistructured database as a knowledge base $\mathcal{K} = (\emptyset, \mathcal{A})$ with an empty TBox. The translation of a data graph into an ABox is straightforward.
- *Step 2*: A TBox, called $\mathcal{T_S}$, is derived. For each object o occurring in $\mathcal{A}$, the TBox $\mathcal{T_S}$ contains an axiom of the form $N_o = \delta_o$, where N_o is a concept name and δ_o is a concept term corresponding to $msc(o)$. This assertion reflects the knowledge in the ABox $\mathcal{A}$ concerning the object o.
- *Step 3*: It merges together equivalent concepts in $\mathcal{T_S}$.

Objects	Concept name
org	org
e_1, e_2, e_3, m_1	Employees
e_1, e_3, m_1	Employees1
e_3	Employees2
e_2	Employees3
e_1	Employees4
d_1, d_2, d_3	Department
d_1	Department2
d_3	Department3
l_3	Laboratory
a_2	Address

Table 1: Retrieving objects instances of each concept

Example 2 The ABox corresponding to the data graph of figure 1 contains, among others, the following assertions about objects d_1 and d_2 :
$\mathcal{A}= \{\ldots, Name(d_1,n_3), Head(d_1,e_1), Pgm(d_1,p_5), Loc(d_1,l_1), Name(d_2,n_6), Pgm(d_2,p_4), \ldots\}$
A fragment of the most specific TBox corresponding to the graph of figure 1 is:

$$\begin{array}{lcl} N_{d_1} & = & \exists Head.N_{e_1} \sqcap \exists Pgm.String \sqcap \exists Name.String \sqcap \exists Loc.String \\ N_{d_2} & = & \exists Pgm.String \sqcap \exists Name.String \sqcap \exists Loc.String \\ N_{e_1} & = & \exists Name.String \sqcap \exists officeNb.Integer \sqcap \exists Address.String \sqcap \exists CollaborateIn.N_{d_3} \\ \ldots & & \end{array}$$

The obtained $\mathcal{T}_{\mathcal{S}}$ is shown in figure 3, with the following simplifications: predefined concepts corresponding to basic types, e.g., String, Integer, $\ldots$, are not displayed, and concept names occurring in the left hand side in the above axioms are replaced with meaningful names.

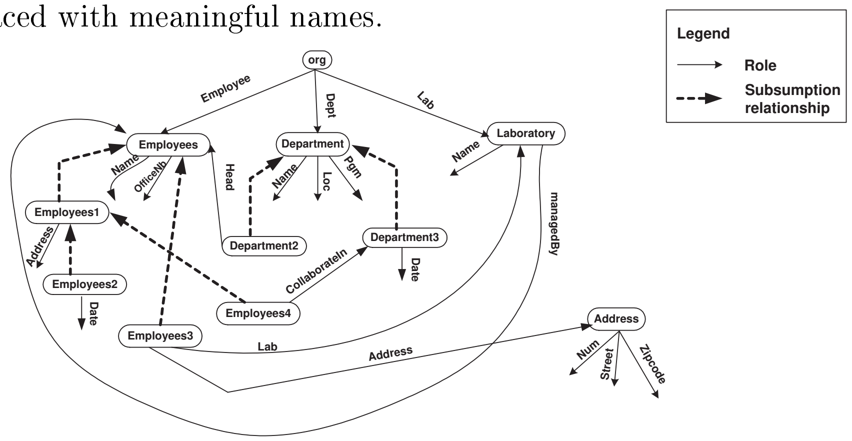

Figure 3: The generated TBox

Once a TBox $\mathcal{T}_{\mathcal{S}}$ that describes a given data graph DB has been generated, the *retrieval*[1] reasoning mechanism can be used for recasting original data within the generated types. Table 1 shows instance-of relationships between objects of the data graph of figure 1 and concepts in the TBox of figure 3.
The following proposition states the correctness of the algorithm Gen$\mathcal{K}$.

[1]The retrieval mechanism consists in retrieving all objects, instances of a given concept [11, 13].

Proposition 1 (Correctness) Let $\mathcal{K} = (\mathcal{T}_{\mathcal{S}}, \mathcal{A})$ be the knowledge base generated by applying the algorithm $\mathsf{Gen}\mathcal{K}$ on a data graph $DB = (V_a \cup V_c, E, r)$, and $\mathcal{G}_{\mathcal{T}_{\mathcal{S}}} = (\mathcal{V}_{\mathcal{T}_{\mathcal{S}}}, \mathcal{E}_{\mathcal{T}_{\mathcal{S}}})$ be the description graph corresponding to $\mathcal{T}_{\mathcal{S}}$.
(1) The data graph DB conforms to $\mathcal{G}_{\mathcal{T}_{\mathcal{S}}}$.
(2) The $\mathcal{EL}$ graph $\mathcal{G}_{\mathcal{T}_{\mathcal{S}}} = (\mathcal{V}_{\mathcal{T}_{\mathcal{S}}}, \mathcal{E}_{\mathcal{T}_{\mathcal{S}}})$ is the most specific schema graph of DB.

(Proof is given in [10]).

4.2 Approximate typing

In the previous section we provided an algorithm for extracting the most specific schema abstracting a data graph. Although, the number of concepts (in other words the number of axioms in the TBox) is reduced by grouping together equivalent concepts, the most specific schema may remain too large.

Example 3 Consider the objects e_1, e_2 and e_3 of the data graph given figure 1. These objects are respectively described by the following three concepts (see figure 3):

$$\begin{aligned} Employees4 &= \exists Name.String \sqcap \exists officeNb.Integer \sqcap \exists Address.String \sqcap \\ &\quad \exists CollaborateIn.Department3 \\ Employees3 &= \exists Name.String \sqcap \exists officeNb.Integer \sqcap \exists Address.N_{a_2} \sqcap \\ &\quad \exists Lab.Laboratory \\ Employees2 &= \exists Name.String \sqcap \exists officeNb.Integer \sqcap \exists Address.String \sqcap \\ &\quad \exists Date.Date \end{aligned}$$

The differences between the structures of the concepts describing objects e_1, e_2 and e_3 just illustrate the heterogeneity in terms of structure between (possible) instances of a same general class (here the class employee).
In general, we do not expect to find strict regular structures in semistructured data. Indeed, in a data graph many of the most interesting structures show up in a slightly different forms. So, we can be often satisfied by a compact graph schema which roughly describes the input data graph.
In the following, we sketch an approach for reducing the size of the most specific schema by grouping together the concepts that have similar structures. To do that, we replace sets of concept descriptions by their least common subsumer in the TBox.

Example 4 Consider again the objects e_1, e_2, e_3 and their corresponding concepts *Employees*2, *Employees*3 and *Employees*4 respectively . If we only care about approximate descriptions of these objects, we can generalize these three concepts (i.e., *Employees*2, *Employees*3 and *Employees*4) into one concept describing their common substructure. Such a concept, let us say *employee*, can be obtained by computing the least common subsumer of these three concepts. The description of *Employee* is: $Employee = \exists Name.String \sqcap \exists officeNb.Integer \sqcap \exists Address.\top$
Note that, *Employee* encodes the largest expressible set, in the language $\mathcal{EL}$, of similarities between *Employees*2, *Employees*3 and *Employees*4. The parts of

Concept name	Equivalent concepts	Associated description
Department	$lcs(N_{d_1}, N_{d_2}, N_{d_3})$	$\exists Pgm.String \sqcap \exists Name.String \sqcap \exists Loc.String$
Employee1	$lcs(N_{e_1}, N_{e_2}, N_{e_3})$ $lcs(N_{e_2}, N_{e_3}, N_{m_1})$	$\exists Name.String \sqcap \exists officeNb.Integer \sqcap \exists Address.Top$
Employee2	$lcs(N_{e_1}, N_{e_2}, N_{m_1})$,	$\exists Name.String \sqcap \exists officeNb.Integer \sqcap \exists Address.String$
Other	$lcs(N_{e_1}, N_{e_3}, N_{m_1})$, $lcs(N_{d_1}, N_{d_2}, N_{e_1})$, $lcs(N_{d_1}, N_{d_2}, N_{e_2})$, ...	$\exists Name.String$

Table 2: Generated concepts

the descriptions of these concepts (e.g., the role $CollaborateIn$ of $Employees4$) which do not appear in the description of $Employee$ can still be managed by an appropriate semistructured approach.
We describe here a simple approach for extracting *frequent* substuctures from an input data graph. We define the notion of *frequency* of a substructure with respect to a threshold k. k stands for the minimum number of objects in the data graph which must satisfy the structure so that the structure can be considered as relevant. The primary motivation is to find substructures that enhance the level of interpretation of the data. The discovered approximate structures allow abstraction over detailed irregular structures in the original data and provide only relevant attributes for interpreting the data.

Computing common subsumers Given a data graph DB and a threshold k, our goal is to compute frequent substructures in DB. Our approach is based on the following proposition.
Proposition 2 Let $DB = (V_a \cup V_c, E, r)$ be a data graph. A description D is verified by k objects in DB iff $D \sqsubseteq lcs(msc(o_{i_1}), ..., msc(o_{i_k}))$, for some o_{i_j} in $V_c, j \in [1, k]$.
So far, the most specific schema (i.e., the content of a TBox) of a data graph is composed of most specific concepts. Therefore, given a most specific schema $\mathcal{T_S}$[2] of a data graph DB and a threshold k, we can infer the frequent descriptions by computing the least common subsumers of each k concepts in $\mathcal{T_S}$. The computation terminates since $\mathcal{T_S}$ is finite.

Example 5 Table 2 shows the result of applying the algorithm Comp$\mathcal{T}$ on the data graph DB of figure 1, with a threshold $k = 3$.
Note that, since the lcs is associative and commutative (e.g., $lcs(N_1, N_2, N_3) = lcs(lcs(N_1, N_2), N_3))$, one can use a levelwise approach to compute the frequent substructures. The idea is to compute the least common subsumers of n concepts level by level, starting in level 1 by pairs of concepts. At each level, the descriptions equivalent to $\top$ are pruned. Moreover, this technique allows to adapt the discovery process to sequentially find a good schema.

[2]In fact, $\mathcal{T_S}$ is computed by a slightly modified version of the algorithm Gen$\mathcal{K}$ from which we remove the Step 3 which consists in merging together equivalent concepts.

5 Conclusion

We have proposed an approach for approximate typing of semistructured data. The extracted schema is expressed in the form of terminological axioms in a small description logic (equipped with tractable reasoning services). The semantics associated to the extracted schema is that of dual schema graphs (i.e., listing required labels), which is much closer to the usual notion of schema in databases. The proposed approach makes use of well studied reasoning services provided by description logic systems.
An experimentation of our approach is an ongoing work in the framework of the MKBEEM[3] project [9].

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML.* Morgan Kaufmann, 2000.

[2] Serge Abiteboul. Querying semi-structured data. In *ICDT'97, Delphi, Greece*, pages 1–18, January 1997.

[3] F. Baader and R. Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In O. Herzog and A. Günter, editors, *KI-98*, volume 1504 of *LNCS*, pages 129–140, Bremen, Germany, 1998. Springer.

[4] F. Baader, R. Küsters, and R. Molitor. Computing Least Common Subsumer in Description Logics with Existential Restrictions. In T. Dean, editor, *IJCAI'99*, pages 96–101. Morgan Kaufmann, 1999.

[5] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *ICDT'97*, pages 336–351. Springer, 1997.

[6] A. Deutsch, M. F. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *ACM SIGMOD, Philadephia*, pages 431–442, 1999.

[7] F. Donini and A. Era. Most Specific Concepts Technique for Knowledge Bases with Incomplete Information. In *CIKM'92, Baltimore*, pages 545–551. Springer, 1992.

[8] F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Deduction in concept languages: From subsumption to instance checking. *JLC*, 4(4):423–452, 1994.

[9] Alain Lger et al. Mkbeem : Ontology Domain Modeling Support for Multi-lingual services in E-Commerce. *To appear in the Workshop on Applications of Ontologies and Problem-Solving Methods, ECAI'2000*, 2000.

[10] M.S. Hacid, F. Soualmia, and F. Toumani. Substructures discovery from semistructured data. Technical report, LIMOS, Université Clermont-Ferrand II, France, April 2000.

[11] B. Nebel. Reasoning and Revision in Hybrid Representation Systems. volume 422 of *LNCS*, page 300. Springer-Verlag, New York, 1990.

[12] R. Motwani S. Nestorov, S. Abiteboul. Extracting schema from semistructured data. In *ACM SIGMOD*, pages 295–306, 1998.

[13] A. Schaerf. Reasoning with Individuals in Concept Languages. *DKE*, 13(2):141–176, 1994.

[14] D. Suciu T. Milo. Type inference for queries on semistructured data. In *PODS'99*, pages 215–226, June 1999.

[15] J. Widom Y. Papakonstantinou, H. Garcia Molina. Object exchange across heterogeneous information sources. In *ICDE'95, Tape, Tawan*, pages 251–260, March 1995.

[3]IST-1999-10589 (http://www.linglink.lu/hlt/projects/mkbeem/).