Formalizing the XML Schema Matching Problem as a Constraint Optimization Problem

Marko Smiljanić[1], Maurice van Keulen[1], and Willem Jonker[1,2]

[1] University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
[2] Philips Research, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
{m.smiljanic, m.vankeulen, w.jonker}@utwente.nl

Abstract. The first step in finding an efficient way to solve any difficult problem is making a complete, possibly formal, problem specification. This paper introduces a formal specification for the problem of *semantic XML schema matching*. Semantic schema matching has been extensively researched, and many matching systems have been developed. However, formal specifications of problems being solved by these systems do not exist, or are partial. In this paper, we analyze the problem of semantic schema matching, identify its main components and deliver a formal specification based on the constraint optimization problem formalism. Throughout the paper, we consider the schema matching problem as encountered in the context of a large scale XML schema matching application.

1 Introduction

Schema matching is a process of identifying semantically similar components within two schemas. Schemas, e.g., database schemas, are designed by humans; they are the product of human creativity. Therefore, it is said that a schema matching problem is an *AI-complete* problem [3], i.e., to solve this problem a system must implement human intelligence. The demand for schema matching is great in data exchange, and data analysis applications. Many semi-automatic schema matching systems have been developed [12] claiming various levels of usefulness [4]. These systems use heuristics to combine various syntactic features of schemas in order to estimate which schema components are similar. Machine learning, reuse of previous results, and user interaction, to name a few, are techniques used to improve the quality of matching.

In current schema matching systems, focus is placed on effectiveness. Heuristics are being used only to provide semantically relevant results. On the other hand, the efficiency of schema matching is mostly ignored. Systems are designed and tested on small scale problems in which efficiency is not an issue. New large scale schema matching applications emerge making the need for efficient schema matching imminent. Large scale schema matching approaches are being developed [13]. To understand and handle the complexity of the schema matching problem and to be able to devise an efficient algorithm to solve the matching problem, a *formal problem specification* must be acquired. To the best of our

knowledge, none of the existing schema matching system was built on the basis of a *complete formal specification* of the schema matching problem.

In this paper, we focus on understanding, modeling and formalizing the problem of semantic XML schema matching. The scope of this paper, within the main line of our research, is indicated in Fig. 1.

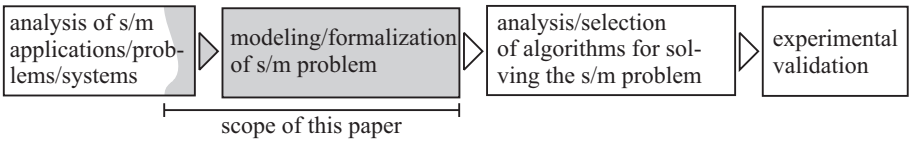

Fig. 1. The scope of this paper within the main line of schema matching (s/m) research

Our research is guided by a large scale schema matching application – a structured-data search engine called *Bellflower*. In Bellflower, the user defines his own XML schema called *personal schema*. A personal schema embodies the user's current information need and expectation with respect to the structure of the desired information. For example, a user looking for information on books would create a personal schema as the one shown in Fig. 2 (A). The task of Bellflower is to match the personal schema against a large *schema repository*, possibly containing all the XML schemas found on the Internet. A fragment of such a repository is shown in Fig. 2. Mappings that Bellflower retrieves are shown as shadowed subgraphs (B) and (C) in the repository. These mappings are presented to the user. The user then selects one mapping to be used by Bellflower to retrieve the actual data. Additionally, Bellflower allows the user to ask XPath queries over his personal schema, e.g., book[contains(author,"Verne")]/title, to filter the retrieved data.

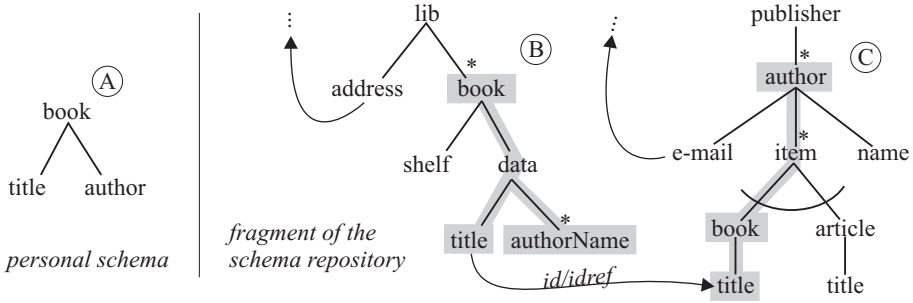

Fig. 2. Personal schema and a fragment of schema repository

The main contribution of this paper is a first thorough formal specification of the semantic XML schema matching problem. Another contribution is a comprehensive framework for the analysis and modeling of the semantic XML schema matching problem, which includes the description of the approximation of semantic matching that must be performed in order to build a system for schema matching.

The paper is organized as follows. Sec. 2 presents the model of semantic matching. Sec. 3 introduces the approximations of semantic matching in an XML schema matching system. Sec. 4 formalizes the matching problem using the constraint optimization problem formalism. Related research is discussed in Sec. 5 followed by the conclusion in Sec. 6.

2 Model of a semantic matching problem

To come to a formal specification of a semantic XML schema matching problem, we devised a model of a *semantic matching problem* [14]. This section describes the main components of the model.

In a generic matching problem, a *template object* T is matched against a set of *target objects* $R = \{\tau_1, \ldots, \tau_k\}$. If a template object is related to a target object through some *desired relation*, i.e., $T \approx \tau_i$, it is said that they match and the (T, τ_i) pair forms one *mapping*. The *solution of a matching problem* is a list of mappings. In some matching problems, an objective function $\Delta(T, \tau_i)$ can be defined. The objective function evaluates to what extent the desired relation between the matching objects is met. In such problems, the objective function is used to rank, i.e., order, the mappings in the solution.

A semantic matching problem differs from the generic matching problem in that objects are matched based on their semantics. Semantics is commonly defined as the *meaning* of data. Therefore, in semantic matching the template and the target objects are matched based on their meanings. The desired semantic relation is a relation between meanings.

For example, in a semantic matching problem a person is looking for a book similar to Verne's book "20,000 Leagues Under the Sea". The person is matching its mental perception of Verne's book against a set of mental impressions about target books, e.g., books in his personal library. In this problem, the desired semantic relation is *similarity of mental impressions about books*.

The semantics, i.e., the ability to generate meanings about objects and to reason about these meanings, is the privilege of humans. Building a computer system that performs true semantic matching is in principle impossible. In practice, computer systems only *approximate* semantic matching [6].

The model of semantic matching that we are to show is a practical simplification of what is really happening in human mind. In our model of semantic matching, the desired relation is divided into a *semantic predicate function* and the *semantic objective function*.

Parts of the desired semantic relation that must necessarily be satisfied are captured within the semantic predicate function. For example, a person is looking for a book; it must be true that the target object is what this person thinks is a book. Further, the person might reason that books are similar if they have the same author; another predicate. The semantic objective function is a model of human's ability to establish ranking among meanings. E.g., the person will think that book A is more similar to Verne's book than book B if book A has a more similar topic; the person's opinion on topic similarity ranks the books.

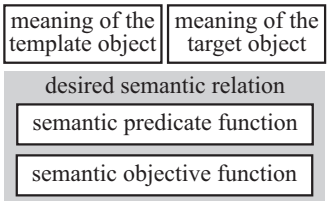

Fig. 3. Decl. components of a semantic matching problem

We have described a model of semantic matching in terms of four declarative components (rectangles in Fig. 3). In order to specify a semantic matching problem in a way that can be used by an automated computer system, all the components of the semantic model must be approximated. With approximation, semantic problem components are expressed using syntactic constructs. Sec. 3 describes these approximations in the context of a semantic XML schema matching problem.

3 Approximations in semantic XML schema matching

The approximation of the components of the model of semantic matching is not a straightforward process. It is a design process burdened with trade-off decisions. In this section we describe approximations which are tailored to meet the needs of the XML schema matching encountered in the Bellflower system. Other schema matching systems would decide to approximate components differently. Nevertheless, the common point of all systems is that these semantic components are approximated.

3.1 Approximating the meaning of template and target objects

In *semantic XML schema matching*, the meanings of XML schemas are being matched against each other. XML schemas are created by humans in a design process. This design process creates a syntactic representation of some semantic concept. For example, a librarian designs an XML schema that models his understanding, i.e., semantics, of a library. This means that every XML schema is already an approximation, a model, of some meaning, i.e., *XML schemas are approximations of their own meaning*. Given an XML schema, no further approximation of its meaning is needed.

In practice, problems come from a different direction; from the heterogeneity of languages and techniques used by men to create schemas. To tame this syntactic diversity, schema matching systems always provide a generic, unified, data model. Different schema representations are then captured within this model. For XML schema matching in Bellflower, we use *directed graphs* enriched with *node and edge properties*; a model similar to ones used in other schema matching systems.

Definition 1. *A directed graph G with properties is a 4-tuple $G = (N, E, I, H)$ where*

- $N = \{n_1, n_2, ..., n_i\}$ *is a nonempty finite set of* nodes*,*
- $E = \{e_1, e_2, ..., e_j\}$ *is a finite set of* edges*,*
- $I : E \rightarrow N \times N$ *is an* incidence function *that associates each edge with a pair of nodes to which the edge is* incident*. For $e \in E$, we write $I(e) = (u, v)$ where u is called the* source node *and v the* target node*.*

- $H : \{N \cup E\} \times \mathcal{P} \rightarrow \mathcal{V}$ *is a property set function where* $\mathcal{P}$ *is a set of properties, and* $\mathcal{V}$ *is a set of values including the null value. For* $n \in \{N \cup E\}$, $\pi \in \mathcal{P}$, *and* $v \in \mathcal{V}$ *we write* $\pi(n) = v$ *(e.g.,* $name(n_1) =$ *'book').*

A walk p *in a directed graph is any alternating sequence of nodes and edges, i.e.,* $p = (n_1, e_1, n_2, e_2, \ldots, e_{k-1}, n_k)$ *such that an edge in* p *is incident to its neighboring nodes. Nodes* n_1 *and* n_k *are said to be the* source *and the* target *nodes of the walk* p*. In this paper,* path *and* walk *are synonyms.*

A graph G' *is a* partial subgraph *of graph* G *(i.e.,* $G' \sqsubset G$*), if* G' *can be constructed by removing edges and nodes from graph* G*.*

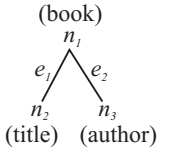

Fig. 4. Model of personal schema

The book personal schema from Fig. 2 is modeled as shown in Fig. 4.

- $N = \{n_1, n_2, n_3\}$
- $E = \{e_1, e_2\}$
- $I(e_1) = (n_1, n_2),\ I(e_2) = (n_1, n_3)$
- $name(n_1) =$ *'book'*, $name(n_2) =$ *'title'*, ...

We call a graph that models an XML schema a *schema graph*. In schema graphs, XML schema components are represented with either a *node*, an *edge*, or a node's or an edge's property. For example, relationships defined in XML schema by means of *id/idref* pairs, or similar mechanisms, are modeled as edges that we call *explicit edges* (edges with an arrow in Fig. 2). More details on how we represent an XML schema using a directed graph can be found in [14]. Schema graphs are a complete representation of an *XML schema*, i.e., the one can be converted into the other and vice versa without loss of information.

3.2 Approximating the semantic predicate function

In semantic matching, the desired semantic relation is a relation between the meanings of a template and a target object. As shown in Sec. 3.1, schema graphs are used to approximate these meanings. Consequently, the desired semantic relation will be approximated as a relation between schema graphs.

We approximate the semantic predicate function with a predicate function $C(T, \tau)$, where T, τ are the template and the target schema graphs. C is a composition of a number predicates c_i, e.g., a conjugation.

$$C(T, \tau) = \bigwedge_{i=1}^{k} c_i(T, \tau)$$

Functions c_i specify various aspects of the relation between the template and the target schema graph that must be true in a mapping. We describe a few in the sequel.

Our schema repository comprises many XML schemas, or rather schema graphs, collected from the Internet. Such a repository can be treated in two different ways: as a set of independent schema graphs, or as one large schema graph. This distinction influences the definition of the target object in a matching task. The matching task is either:

1. for a template schema T, *find the most similar target schema graph* τ_i in the repository $R = \{\tau_1, \ldots, \tau_k\}$. The output of this matching approach is a list of concrete schemas from R, namely the ones most similar to T, or
2. for a template schema T, *find the most similar partial subgraphs* τ_i in R (i.e., $\tau_i \sqsubset R$, see Def. 1). The output of this matching approach is a list of subgraphs of the repository schema graph R. Such subgraphs can in general be composed of nodes and edges from different concrete schema graphs participating in R.

In our research, we adopt the second matching goal. This allows a personal schema to be matched to a target schema obtained by joining fragments of several distinct schemas, or to be matched to a fragment of only one schema, as well. A predicate function $c_1(T, \tau) := (\tau \sqsubset R)$, where R is the schema repository, can be used to specify this matching goal. Predicate c_1 is not very strict and does not consider the properties of the personal schema. For the book personal schema in Fig. 2, the c_1 predicate would be satisfied, for example, for τ being any single node of the repository. This seldom makes any sense. We therefore provide a stricter relation between the template and the target schema graph as follows.

The target schema graph $\tau = (N_\tau, E_\tau, I_\tau, H_\tau)$, where $\tau \sqsubset R$ for repository R, can form a mapping with a template schema graph $T = (N_T, E_T, I_T, H_T)$ if the following set of rules is satisfied.

1. for each node $n \in N_T$, there exists one and only one *match node* $n' \in N_\tau$, depicted as $n' = \mathit{Match}(n)$. E.g., in Fig. 2 the node 'authorName' in (B) is the match node for the node 'author' in (A).
2. for each edge $e \in E_T$, there exists one and only one match path $p' \in \tau$, depicted as $p' = \mathit{Match}(e)$, where $\mathit{source}(p') = \mathit{Match}(\mathit{source}(e))$, and $\mathit{target}(p') = \mathit{Match}(\mathit{target}(e))$. E.g., in Fig. 2 the path 'book-data-title' in (B) is the match path for the edge 'book-title' in (A).
3. the fact that schemas in a mapping (T, τ) meet the conditions 1 and 2 is depicted as $\tau = \mathit{Match}(T)$.

A new predicate can be defined as $c_2(T, \tau) := (\tau = \mathit{Match}(T))$.

Note that the first rule restricts node mappings to what is known as $1:1$ node mapping, i.e., each personal schema node is mapped to only one target node. Other systems [7] need different set of restrictions to be able to accommodate the $1:N$ or $M:N$ node mappings.

In principle, the set of predicate functions is complete when it precisely defines the search space within which the match for the template object is to be found. For example, a schema matching system that does not handle cyclic data structures should include the predicate $c_3(T, \tau) := (T \textit{ is non cyclic}) \wedge (\tau \textit{ is non cyclic})$.

3.3 Approximating the semantic objective function

The semantic objective function is approximated with an objective function $\Delta(T, \tau) \in \mathbb{R}$, where T is a template schema graph, τ is a target schema graph

taken from the repository R. It is common to normalize the value of the objective function to the $[0, 1]$ range. Furthermore, $\Delta(T, \tau)$ is *undefined* if the predicate function $C(T, \tau) = false$.

It has been shown that a number of different heuristics have to be used in order to acquire higher matching quality [5, 12]. The heuristics exploit different schema properties as datatypes, structural relations, and documentation, to name a few. Each of the heuristics is implemented using a separate ranking function $\delta_i(T, \tau) \in \mathbb{R}$. These ranking functions are composed to define the objective function Δ. This composition can be algebraic, AI based, or hybrid, often involving additional heuristics. A comprehensive survey by Rahm and Bernstein [12] discusses different approaches for building the Δ function.

In this section, we have shown how to approximate declarative components of semantic XML schema matching problem. The result is a set of declarative syntactic components. In our research, we came to understand that these components are almost identical to components of a known class of problems called *constraint optimization problems (COP)* [1, 10]. In the sequel, we show one way to specify a schema matching problem in the COP framework. We also discuss benefits that schema matching can draw from being treated as a COP.

4 Formal specification of the problem

In this section, we first describe the formalism for representing constraint optimization problems. We then show one way to specify semantic XML schema matching problem using this formalism.

4.1 Constraint optimization problems

Constraint programming (i.e., CP) is a generic framework for problem description and solving [1, 10]. CP separates the declarative and operational aspects of problem solving. CP defines different classes of problems, of which we solely focus on the declarative aspects of *constraint optimization problems*.

Definition 2. *A* constraint optimization problem *(i.e., COP) P is a 4-tuple $P = (X, D, C, \Delta)$ where*

- $X = (\mathsf{x}_1, \ldots, \mathsf{x}_n)$ *is a list of* variables*,*
- $D = (D_1, \ldots, D_n)$ *is a list of finite* domains*, such that variable* x_i *takes values from domain D_i. D is called the* search space *for problem P.*
- $C = \{c_1, \ldots, c_k\}$ *is a set of* constraints*, where $c_i : D \rightarrow \{true, false\}$ are predicates over one or more variables in X, written as $c_i(X)$.*
- $\Delta : D \rightarrow \mathbb{R}$ *is a the* objective function *assigning a numerical quality value to a solution (*solution *is defined below).*

COP is defined in terms of *variables* X, taking values from search space D. A complete variable assignment is called *valuation*, written as Θ. Θ is a vector in D, thus assigning a value in D_i to each variable x_i, $i = 1, n$. A valuation Θ for which *constraints* $C(X)$ hold, i.e., $C(\Theta) = true$, is called a *solution*. The quality of a solution is determined by the value of the objective function, i.e., $\Delta(\Theta)$.

4.2 Semantic XML schema matching as COP

This section presents one possible way for specifying an XML schema matching problem as COP. The approach is based on rules defined as a part of the $c_2(T, \tau)$ predicate in Sec. 3.2. To support the explanation in this section, we will use the book personal schema, and the schema repository shown in Fig. 2, as well as the personal schema graph given in Fig. 4.

Definition 3. *A semantic XML schema matching problem with a template schema graph T, a repository $R = (N_R, E_R, I_R, H_R)$ of target schema graphs τ_i, where $\tau_i \sqsubset R$, a predicate function $C(T, \tau)$, and an objective function $\Delta(T, \tau)$ is formalized as a constraint optimization problem $P = (X, D, C, \Delta)$. The following four rules construct P in a stepwise manner.*

Rule-1. For a template schema graph $T = (N_T, E_T, I_T, H_T)$, the *repository of target schema graphs* R is formalized in a COP problem P, as follows:

1. for each node $n_i \in N_T$, a node variable x_{n_i} and a domain N_R are added to P (see section 3.2, rule 1),
2. for each edge $e_i \in E_T$, a path variable x_{p_i} and a domain $\mathcal{L}_R$ are added to P, where $\mathcal{L}_R$ is the set of all paths in repository R (see section 3.2, rule 2),
3. for each edge $e_k \in E_T$, where $I_T(e_k) = (n_i, n_j)$, a constraint $ic_k(X) := source(x_{p_i}) = Match(n_i) \wedge target(x_{p_i}) = Match(n_j)$ is added to P. We denote the conjunction of all such constraints as $IC(X)$ – the incidence constraint. This constraint ensures that target paths are connected in the same way as template edges are connected.

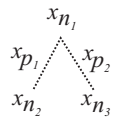

Fig. 5. Target object variables

For the book example, P is, so far, defined as

$$X = (x_{n_1}, x_{n_2}, x_{n_3}, x_{p_1}, x_{p_2})$$
$$D = (N_R, N_R, N_R, \mathcal{L}_R, \mathcal{L}_R)$$
$$C = \{IC(X)\}$$
$$\Delta(X) = \textit{not yet defined}$$

Fig. 5 illustrates *Rule-1*; node and edge variables are assigned based on the shape of the book template schema graph.

Rule-2. The predicate function $C(T, \tau)$ is formalized in P by adding a constraint $C(X)$ – a constraint function identical to predicate $C(T, \tau)$. Function $C(T, \tau)$ can be directly transformed to $C(X)$; node and path variables found in X replace τ, nodes and edges of T (n_1, n_2, n_3, e_1, and e_2 in the book example) appear as constants in $C(X)$.

For example, $\mathsf{c_4(X)} := (\mathsf{datatype}(n_2) = \mathsf{datatype}(x_{n_2}))$ is a constraint responsible for ensuring that the 'title' node and its match have the same datatype.

Rule-3. The objective function $\Delta(T, \tau)$ is formalized in P using the objective function $\Delta(X)$ – a function identical to $\Delta(T, \tau)$.

Rule-4. Template schema graph T is constant in P, i.e., for two different schema graphs T_1 and T_2, two different COP problems must be declared. As already indicated, T is represented through constants in $C(X)$ and $\Delta(X)$.

For the schema matching problem, the specification of P is now complete.

$$X = (x_{n_1}, x_{n_2}, x_{n_3}, x_{p_1}, x_{p_2})$$
$$D = (N_R, N_R, N_R, \mathcal{L}_R, \mathcal{L}_R)$$
$$C = \{IC(X), C(X)\}$$
$$\Delta(X) = \textit{as defined in the approximation phase (see Sec. 3.3)}$$

4.3 The benefits of using COP framework

The benefit of formalizing a schema matching problem as a COP is that a schema matching problem can now be regarded as a combinatorial optimization problem with constraints. COP problems have been largely investigated and many [non]exhaustive techniques for efficient solving have been proposed [11, 10]. Branch and bound, clustering methods, simulated annealing, tabu search, to name a few, can be investigated and adopted to schema matching problems represented as COPs. Important issues that influence the efficiency, e.g., variable ordering, value ordering, constraint simplification are also discussed in the COP framework.

In Bellflower, we are currently investigating the combination of clustering methods and the branch and bound algorithm for efficient search space traversal.

5 Related research

Schema matching attracts significant attention as it finds application in many areas dealing with highly heterogeneous data. A survey by Rahm and Bernstein [12] identifies *semantic query processing* as an application domain where schema matching is used as a part of query evaluation. This is similar to how we use schema matching in Bellflower.

Representatives of automated schema matching systems include COMA [5], Cupid [9], and LSD [6], to name a few. These systems formalize only the behavior of the objective function, or use no problem formalization at all. The COP framework and the approach that we have used in the formalization can be used to complement the existing partial formalisms to a complete problem specification. For our approach, we found inspiration in the work of Bergholz [2]. Bergholz addresses the problem of querying semistructured data and provides a formal specification in the form of a constraint satisfaction problem. In his work, querying is treated as strict database querying. Structural relaxations (e.g., matching an edge to a path) have to be accounted for by the user, and specified in the query. Ranking of results is not supported. As such, his formalism is not suitable for describing a semantic schema matching problem.

To come to a full specification of a schema matching problem, we touch several areas. First, we model semantic matching. In [8] a different way to model semantic mappings and to reason about these is given. Second, we model XML schemas as a graph data structure. This is similar to how most other systems model schemas, e.g., Cupid [9]. Finally, we use the constraint optimization problem framework [10] as a base for the formalization.

6 Conclusion

In this paper, we have described an approach to formally specify semantic XML schema matching problems. The formalism is developed to support research related to a large scale schema matching system – Bellflower.

We gave a model of semantic matching followed by the approximations for describing a semantic problem in a syntactic domain. Finally, the constraint optimization framework was identified as a suitable framework for capturing all the declarative syntactic components of the problem.

With this formalism, the goal of this part of our research (see Fig. 1) was achieved: a clear and unambiguous specification of the problem – a good starting point for the exploration of efficient algorithms for solving.

We are currently investigating the combination of clustering methods and the branch and bound algorithm to achieve efficient schema matching in a large scale schema matching application.

References

[1] R. Bartak. Constraint programming: In pursuit of the holy grail. In *Proceedings of the Week of Doctoral Students (WDS)*, pages 555–564, June 1999.

[2] A. Bergholz and J. C. Freytag. Querying Semistructured Data Based on Schema Matching. *Lecture Notes in Computer Science*, 1949, 2000.

[3] P. A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-strength schema matching. *SIGMOD Rec.*, 33(4):38–43, 2004.

[4] H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the 2nd Int. Workshop on Web Databases*, 2002.

[5] H. H. Do and E. Rahm. COMA — A system for flexible combination of schema matching approaches. In P. A. Bernstein et al., editors, *Proc. Intl. Conf. VLDB 2002*. Morgan Kaufmann Publishers.

[6] A. Doan. *Learning to Map between Structured Representations of Data*. PhD thesis, University of Washington, 2002.

[7] B. He and K. C.-C. Chang. A holistic paradigm for large scale schema matching. *SIGMOD Rec.*, 33(4):20–25, 2004.

[8] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Y. Halevy. Representing and reasoning about mappings between domain models. In *Proc. Conf. (AAAI/IAAI-02)*, pages 80–86.

[9] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases(VLDB '01)*, pages 49–58, Orlando, Sept. 2001. Morgan Kaufmann.

[10] K. Marriott and P. J. Stuckey. *Programming with Constraints: an Introduction*. MIT Press, 1998.

[11] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer Verlag, December 1999.

[12] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, Dec. 2001.

[13] E. Rahm, H.-H. Do, and S. Mamann. Matching large xml schemas. *SIGMOD Rec.*, 33(4):26–31, 2004.

[14] M. Smiljanić, M. van Keulen, and W. Jonker. Defining the XML Schema Matching Problem for a Personal Schema Based Query Answering System. Technical Report TR-CTIT-04-17, Centre for Telematics and Information Technology, Apr. 2004.