Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany.

Automatic Generation of Mediated Schemas Through Reasoning Over Data Dependencies

Xiang Li, Christoph Quix, David Kensche, Sandra Geisler and Lisong Guo
Informatik 5 (Databases and Information Systems)
RWTH Aachen University, Germany
Email: LastName@dbis.rwth-aachen.de

***Abstract*—Mediated schemas lie at the center of the well recognized data integration architecture. Classical data integration systems rely on a mediated schema created by human experts through an intensive design process. Automatic generation of mediated schemas is still a goal to be achieved. We generate mediated schemas by merging multiple source schemas interrelated by tuple-generating dependencies (tgds). Schema merging is the process to consolidate multiple schemas into a unified view. The task becomes particularly challenging when the schemas are highly heterogeneous and autonomous. Existing approaches fall short in various aspects, such as restricted expressiveness of input mappings, lacking data level interpretation, the output mapping is not in a logical language (or not given at all), and being confined to binary merging. We present here a novel system which is able to perform native n-ary schema merging using P2P style tgds as input. Suited in the scenario of generating mediated schemas for data integration, the system opts for a minimal schema signature retaining all certain answers of conjunctive queries. Logical output mappings are generated to support the mediated schemas, which enable query answering and, in some cases, query rewriting.**

I. Introduction

Data integration systems involve a multitude of data sources. A unified user view called the *mediated schema* lies at the center of a data integration architecture. Logical mappings, e.g., tuple-generating dependencies (tgds), from the source to the mediated schema are the key to enable query processing over the mediated schema. Classical data integration systems [1] nowadays still rely on a mediated schema created by an intensive manual design process by human experts, which is costly and inflexible in a dynamic evolving world. Automatic generation of mediated schemas is still a goal to be achieved.

Schema merging is the process of consolidating multiple related heterogeneous input schemas to produce a merged schema. It is widely applied to different scenarios, such as view integration (designing a storage schema from a set of desired user views) and data integration (generating mediated schemas). In vision of the importance of schema merging, Merge is proposed as one of the major operators in *Model Management* [2]. As retrospected by Bernstein and Melnik in [3], the original vision of Model Management 1.0 is not semantic but structural, i.e., not relating schema and data. That is, operators are interpreted in terms of schemas, while lacking the connection to the underlying data of the schemas. A semantic merge is in need, not only for expressiveness reasons but also for executability reasons. Merging using logical schema mappings is inevitable for realizing a model management engine to address real-world data programmability problems. We demonstrate a system, which makes use of expressive mapping constraints in tgds and produces mediated schemas with provable qualities, i.e., minimal signature with complete certain answers. Our work is a follow-up in the direction of constraint driven merging, extending the work in [4], [5], [6].

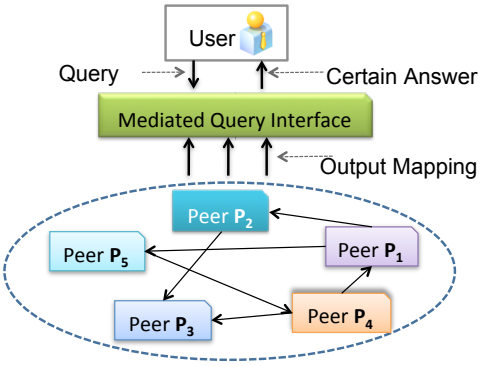

Fig. 1. An N-ary Schema Merging Scenario

Most of the early merging approaches surveyed by Batini et al. [7] are binary, i.e., merging two schemas at a time. We deem a native n-ary merge (e.g., as in [8]) as advantageous for generating mediated schemas for a multitude of data sources. Though binary merging algorithms can be applied iteratively to merge multiple schemas, the process needs a complete human supervision, i.e., in each iteration an expert is required to generate mappings between a new source schema and the intermediate merged result from previous steps. Moreover, in a scenario of ad hoc P2P environments, only some particular mappings are available and nobody has the complete knowledge to produce arbitrary mappings. Therefore, a native n-ary merging algorithm is more suited to exploit all the available mappings for multiple data sources.

Batini et al. [7] contribute the popularity of binary merge to a complexity reason, that is, by involving less input the problem of merging will become less complicated. However, as we have already described, constructing schema mappings is also quite expensive and requires a lot of human supervision, which compensates for the increased computation efforts of a native n-ary merge.

In this demo, we present a novel prototype performing native n-ary merging to generate mediated schemas automatically. The prototype is backed by a solid semantic interpretation relating schemas and the underlying data. The schema merging problem is modeled as searching for minimal schema signatures that are able to retain all certain answers of conjunctive queries. We propose a novel algorithm that is able to test whether a given mediated schema, resulting from a transformation of the joint union of the source schemas, is complete with respect to certain answers of conjunctive queries [9]. We employ an A-priori variant to enumerate candidate mediated schemas to reduce unnecessary reasoning. Even taking unambiguous logical constraints as input, alternatives of mediated schemas can still arise, since the same piece of information can be represented differently. In recognition of this, our system returns a series of minimal mediated schemas. Practical schemas are usually accompanied by integrity constraints which clarify the intra-schema structures. Therefore, source integrity constraints are treated as first-class citizens in our algorithm. To the best of our knowledge, our prototype is the first system that is able to incorporate source integrity constraints (in the general form of tgds and egds) seamlessly.

The rest of the paper is organized as follows. Section II introduces briefly the modeling of the semantic merge framework and the merging algorithm opting for minimized mediated schemas. Section III describes the system architecture and query processing strategies over the mediated schemas. Section IV details the demo scenario we are proposing.

II. Schema Merging Through Reasoning

This section summarizes the modeling and the algorithm behind our merging prototype, presented in [10] and [9].

A. The N-ary Merging Problem

Consider the scenario of merging multiple data sources, such as merging the employee databases of Sun and Oracle. Since the data sources are independently developed, their extensions, i.e., explicitly stored data, usually do not conform to any inter-schema logical constraints. This is one main reason why Pottinger and Bernstein [6] interprete their input mapping as specification of overlap between schemas instead of direct logical constraints over data instances. In this paper, we take a different direction by interpreting the input mapping as constraints that are expected to hold over the integrated global database, which is inline with the common assumption in data integration that sources are incomplete.

Given an incomplete database I of schema S, the *semantics* of I wrt. a set of data dependencies Σ over S is $Sem_\Sigma(I) = \{I' : I' \in Inst(S) \wedge I \subseteq I' \wedge I' \models \Sigma\}$. When the dependencies are clear from context, we simply write $Sem(I)$ for brevity.

Consider n source schemas $S_1, S_2, \ldots, S_n$ with no two relations sharing the same name. Each source schema S_i has a set of integrity constraints Σ_i as a union of tgds and egds. The input mapping is specified by a set of inter-schema tgds Σ_{in} among the source schemas. The *joint source schema* is the disjoint union of the n source schemas, while a *joint source instance* is the disjoint union of n instances with one instance for each source schema. The *merge input* is then a pair (S, Σ), with S being a joint source schema and $\Sigma = \Sigma_{in} \cup \bigcup_i \Sigma_i$. The semantics of a joint source instance I is then $Sem_\Sigma(I)$. In the following, we only consider sound data sources, that is, they are consistent with the specified data dependencies and the possible world set they present is not empty.

For a merge input (S, Σ), a merge output is a binary mapping $\mathcal{M}_o = (S, G, \Sigma_o)$, called the *output mapping*, in which G is the *mediated schema* and Σ_o is a set of data dependencies.

B. Semantics for Schema Merging

In [11], Miller et al. use formal schema equivalence results to characterize the semantics of schema merging. However, as we understand it today, the semantics of a mediated schema is better characterized using not only the signature of the schema but also the mapping relating the mediated schema to data sources that host base data. Based on such an observation, the semantics of schema merging in our approach is characterized as query answering properties of the output mapping system between the joint source schema and the mediated schema.

An output mapping is *complete* if for every conjunctive query q, there exists another conjunctive query over the mediated schema, which has an equivalent certain answer to q over the incomplete source, for every merge input. Our definition of completeness differs from the one in [6] in that under the open world assumption, we focus on certain answers (of CQs) instead of data source extensions.

Since data sources are incomplete, two different joint source instances may have the same semantics, i.e., the same possible world set. Therefore, we require the output mapping not to distinguish these equivalent joint sources. This is achieved by requiring that an incomplete joint source instance is mapped to the union of images of each possible world in its semantics. To put it more formally, under the output mapping, the solution space of a joint source instance I should be the union of the solution spaces of each source instance in $Sem(I)$. Since it has an effect of mixing equivalent data, we call it *integratedness*.

For creating a mediated query interface for data integration systems, we take the assumption that a smaller query interface (still retaining all the query capabilities) is better and head for a minimal schema with no redundant column. Redundancy of columns is defined wrt. a given output mapping. *Minimality* states that an output mapping system cannot be transformed via a family of mappings to achieve another output mapping with a smaller mediated schema without losing a given property. We are interested in properties such as completeness and integratedness raised earlier.

A complete and formal treatment of the semantics can be found in [10] and [9].

C. Algorithm Overview

Since the desired properties such as completeness and integratedness are in general undecidable [10], we consider a class of output mappings that have a mediated schema as

a transformation result from the joint source schema. The algorithm has two stages: a constraint repairing stage and a mediated schema minimization stage.

In the first stage, we construct an initial canonical output mapping trying to "repair" the joint source instance. The canonical output mapping consists of two parts: a set of copy tgds from the joint source schema to one replica, and all the input data dependencies encoded as target dependencies. It is straightforward to show that the canonical output mapping is both complete and integrated [10]. Interestingly, Melnik describes in Theorem 4.2.4 of [12] a straightforward algorithm creating a mediated schema for view integration, which corresponds to our initial output mapping but in a "reversed" direction. However, since the size of the signature does not matter for view integration, schema minimization is not performed.

During the second stage, we enumerate candidate mediated schemas, and test whether a candidate is still complete. A candidate is minimal if it cannot be reduced further in size and will then be output as a possible result. This stage is composed of two main components: completeness test and enumeration strategy. In [9], we provide a procedure based on chase that is able to test completeness for unions of CQs (UCQs) and show that it is in PTIME if each data dependency involved has a bounded length. Since we are trying to reduce redundant columns in the mediated schema, we consider transformations that project out some columns of a given schema. This family of transformations strictly reduce the size of the mediated schema and hence termination of the minimization process is ensured.

III. System Overview

The system diagram is shown in Figure 2. It is based on Eclipse Rich Client Platform and it makes use of a Prolog Knowledge Base Engine extensively for reasoning tasks, e.g, completeness test and query rewriting. On top of the Prolog engine, there are two layers:

- an algorithmic layer takes care of schema merging, query processing, and I/O among disk files, Java and Prolog;
- a workbench layer deals with presenting schema and mapping structures, browsing data sources (in the form of Prolog source files), receiving user queries, keeping lineage of source mappings and merged results, etc.

We currently support only source data as files. An implementation for supporting JDBC is in progress. The following subsections reveals some major implementation decisions and our query processing strategies over mediated schemas.

A. Implementation

The section details various important design/implementation choices during the development of the system.

A-priori Enumeration: the most costly part in our merging algorithm is repeating completeness tests for various candidates. A characteristic we exploit to reduce unnecessary reasoning is that the redundant columns in the schemas possess the A-priori property, i.e., any subset of a set of redundant

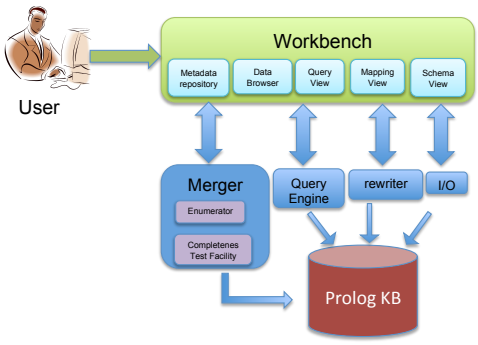

Fig. 2. System Architecture

columns is also redundant. A variant of the A-priori algorithm [9] is employed to prune unnecessary reasoning.

Chase implementation: we implemented a parallel chase on top of the Prolog engine. Labelled nulls (or variables) are represented by ground skolem terms. An interesting lesson we learnt is that the skolem term should not be nested further, due to performance reasons. In an early version, we nested skolem terms in the hope of recording the lineage of chasing, which soon lead to an explosion in expression complexity.

Value Conversion Functions: real world data sources are usually incompatible in data formats, which is a main source for the necessity of value conversion functions. We distinguish invertible functions and non-invertible functions. The former is treated similarly as skolem functions used in chasing, while the latter needs some auxiliary rule rewriting to avoid recursive nesting [9].

Static Analysis of Mappings: we have implemented a syntax checker which tests input whether mappings admit terminating chase. Currently, only weak acyclicity is supported. More general conditions such as stratification are also in our research agenda. Our weak acyclicity tests already take into account the use of value conversion functions in the mapping.

Mapping Syntax: the input mappings used by our system are plain tgds and egds in a Prolog syntax [9]. Output mappings are a result of a composition of a standard schema mapping (s-t tgds with target dependencies) and a set of full s-t tgds, which cannot be represented even by second order tgds. However, a recent result by Arenas et al. [13] shows that composition of two mappings specified by s-t tgds with target dependencies is able to be expressed using source-to-target second order dependencies (s-t SO dependencies). In our current implementation, the output mapping is represented as egds and tgds, using helper predicates from the canonical mediated schema.

Batch Completeness Test: completeness tests for a given candidate mediated schema is consolidated into one test, by assigning disjoint domains for the data. This is due to a consideration of reducing the communication costs between Java and Prolog.

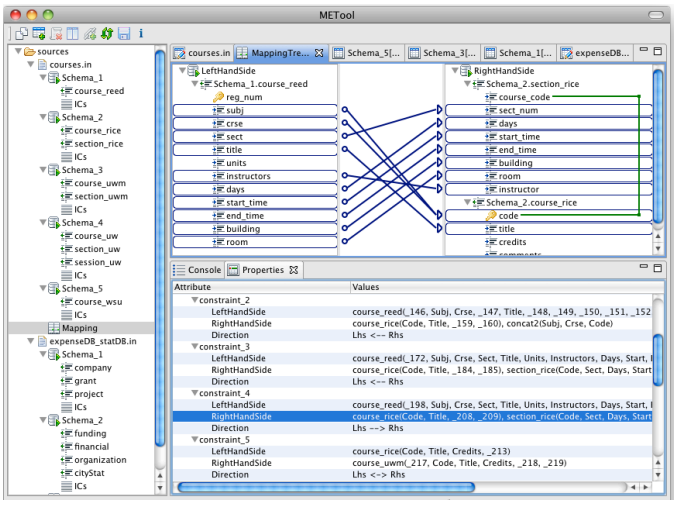

Fig. 3. Example of N-ary Mapping Visualization

B. Query Processing

Given a user query against the mediated schema, there are two strategies for query processing. The first way is query answering using a materialized instance called the universal solution [14], while the second is query rewriting.

When the input data dependencies admit a terminating chase, the output mapping generated by our system is guaranteed to admit also a terminating chase. Therefore, query answering can be performed by chasing the source instances against the output mapping and then perform query evaluation over the materialized instance of the mediated schema.

However, the expressive mapping language we allow imposes challenges on query rewriting, since the output mapping may involve recursion of relations. We detail here an algorithm which is able to rewrite a conjunctive query into a Datalog program when the input mapping consists of weakly acyclic tgds [14]. The rewriting algorithm is an extension of the inverse rule algorithm for query rewriting in Local-As-View systems [15]. The rewriting algorithm proceeds in three stages. In the first stage, bottom-up generation of functional patterns of predicates are performed until a fixpoint is reached. This stage can be shared by rewriting of different queries against a given mediated schema. In the second stage, for a given user query, we check the reachability of patterned predicates backward from the given query. The third stage takes as input the reachable patterned rules obtained in the previous stage and employs the predicate-split [15] technique to produce a function-free Datalog program.

IV. Demo Scenario

First, we will show and analyze an n-ary mapping input, taken from the real world data set Illinois Semantic Integration Archive (http://pages.cs.wisc.edu/~anhai/wisc-si-archive/). The goal is to show the flexibility of our system to model real world P2P style mappings and our novel n-ary mapping visualization. Recognizing that a constraint in an an n-ary mapping may involve multiple schemas, visualization is done constraint-wise. Each constraint has two sides, which are two conjunctive queries over multiple schemas.

Second, we will demonstrate the merging process. Our system supports multiple configuration options, such as minimal schema vs. minimal-sized schema and enum-one-schema vs. enum-all-schemas. During the merging process, the reasoning procedure would be visible via our log console view. At the end of merging, the user is able to navigate the multiple choices of the alternative mediated schemas with corresponding output mappings.

The third part will demonstrate the query processing capability of our mediated schemas. The user will be allowed to attach data sources to the source schemas and a data browser is provided for introspecting the contents. A query view can be opened on a mediated schema and users are able to enter conjunctive queries over the mediated schema. Our query engine is able to compute the maximal certain answer and present the result in a tabular form to the user. Furthermore, we will demonstrate the query rewriting function over weakly acyclic inputs. When no egds are involved, a maximally contained rewriting in Datalog will be produced.

Acknowledgment

The work is supported by the Research Cluster on Ultra High-Speed Mobile Information and Communcation UMIC (http://www.umic.rwth-aachen.de).

References

[1] M. Lenzerini, "Data integration: A theoretical perspective," in *PODS*, 2002, pp. 233–246.
[2] P. A. Bernstein, A. Y. Halevy, and R. Pottinger, "A vision for management of complex models," *SIGMOD Record*, vol. 29, no. 4, pp. 55–63, 2000.
[3] P. A. Bernstein and S. Melnik, "Model management 2.0: Manipulating richer mappings," in *Proc. SIGMOD*, Beijing, China, 2007, pp. 1–12.
[4] J. Biskup and B. Convent, "A formal view integration method," in *Proc. SIGMOD*, Washington, D.C., 1986, pp. 398–407.
[5] M. A. Casanova and V. M. P. Vidal, "Towards a sound view integration methodology," in *PODS*. Atlanta, GA: ACM, 1983, pp. 36–47.
[6] R. Pottinger and P. A. Bernstein, "Schema merging and mapping creation for relational sources," in *Proc. EDBT*, 2008.
[7] C. Batini, M. Lenzerini, and S. B. Navathe, "A comparative analysis of methodologies for database schema integration," *ACM Computing Surveys*, vol. 18, no. 4, pp. 323–364, 1986.
[8] *Proc. SIGMOD*, 2008.
[9] X. Li, C. Quix, D. Kensche, and S. Geisler, "Automatic schema merging using mapping constraints over incomplete sources," in *CIKM*, 2010.
[10] X. Li, "Towards a unified framework for schema merging," in *VLDB PhD Workshop*, 2010.
[11] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan, "The use of information capacity in schema integration and translation." in *Proc. VLDB*. Morgan Kaufmann, 1993, pp. 120–133.
[12] S. Melnik, *Generic Model Management: Concepts and Algorithms*, ser. LNCS. Springer, 2004, vol. 2967.
[13] M. Arenas, R. Fagin, and A. Nash, "Composition with target constraints," in *ICDT*, 2010.
[14] R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa, "Data exchange: Semantics and query answering," *Theoretical Computer Science*, vol. 336, pp. 89–124, 2005.
[15] O. M. Duschka and M. R. Genesereth, "Answering recursive queries using views," in *PODS*, 1997, pp. 109–116.