Schema Management

Periklis Andritsos[*] Ronald Fagin[†] Ariel Fuxman[*] Laura M. Haas[†]
Mauricio A. Hernández[†] Ching-Tien Ho[†] Anastasios Kementsietsidis[*]
Renée J. Miller[*] Felix Naumann[†] Lucian Popa[†] Yannis Velegrakis[*]
Charlotte Vilarem[*] Ling-Ling Yan[†]

Abstract

Clio is a management system for heterogeneous data that couples a traditional database management engine with additional tools for managing schemas (models of data) and mappings between schemas. In this article, we provide a brief overview of Clio and place our solutions in the context of the rich research literature on data integration and transformation. Clio is the result of an on-going collaboration between the University of Toronto and IBM Almaden Research Center in which we are addressing both foundational and systems issues related to heterogeneous data, schema, and integration management.

1 Introduction

Heterogeneous data sets contain data that have been represented using different data models, different structuring primitives, or different modeling assumptions. Such sets often have been developed and modeled with different requirements in mind. As a consequence, different schemas may have been used to represent the same or related data. To manage heterogeneous data, we must be able to manage these schemas and mappings between the schemas.

The management of heterogeneous data is a problem as old as the data management field itself. However, its importance and the variety of contexts in which it is required has increased tremendously as the architectures for exchanging and integrating data have increased in sophistication. Traditionally, heterogeneous data sets were managed within a *federated* architecture where a virtual, global schema (or view) is created over a set of heterogeneous sources (or local schemas) [HM85]. A mapping (in this case a set of view definitions) is established between each source and the global view. The mapping may define each component of the global schema in terms of the source schemas (*global-as-view* or GAV), or alternatively, the mapping may define the components of each source schema in terms of the global (*local-as-view* or LAV) [LRO96]. A federated architecture is appropriate for closely-coupled applications where queries on the virtual global view can be answered using data stored in the sources [Len02]. Notice that in federated systems, data is only exchanged at run-time (that is, query-time). However, the Web has inspired a myriad of more loosely-coupled, autonomous applications where

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[*]University of Toronto, Computer Science, www.cs.toronto.edu/db/clio
[†]IBM Almaden Research Center, www.almaden.ibm.com/software/km/clio/index.shtml

data must be shared, but where run-time coupling of the applications is not possible. These applications include data grids and peer-to-peer applications [BGK$^+$02]. In such applications, data must be exchanged, replicated, or migrated. To accomplish this, mappings between pairs of schemas are established (between a source and target schema). However, unlike in more integrated federated environments, the target data must be materialized so that target queries can be answered locally (even when the source data is unavailable). Furthermore, this target data must reflect the source data as accurately as possible. In federated environments, no such exchange of data is required. In general, we would like to build and manage schema mappings that permit both virtual data integration (as in federated environments) and the exchange of data between independent cooperating peers.

The architecture of Clio is described elsewhere [MHH$^+$01]. In this work, we focus on the specific solutions employed within Clio as they relate to schema and mapping management. We consider our solutions to old, well-established research problems such as generating *schema correspondences* [BLN86] (more recently called *schema matching* [RB01]) and data integration [Len02]. We also identify new foundational and systems issues related to the problem of exchanging data between heterogeneous schemas. In Section 2, we describe our solutions for creating and managing schemas. We consider the creation and management of correspondences between schemas (Section 3) and how correspondences can be used to create mappings (Section 4). Finally, we present our solutions for using mappings in both data integration and data exchange settings.

2 Creating and Managing Schemas

For the purposes of this overview, a schema is a set of structuring primitives (predicates) and a set of constraints (logical statements) over these primitives. Within Clio, we use a nested-relational model over which a powerful set of constraints, including nested referential constraints, can be expressed.

A schema is a model of a data set and it is the primary vehicle we use for both querying the data and understanding the data. To manage heterogeneous data sets, each modeled by a different schema, we create mappings between their schemas. Within Clio, schemas are the primary vehicle for understanding and creating this mapping. Many data management solutions assume that we are given a schema that accurately reflects the data. Yet, often, this is not the case. We may be using data from a legacy database management system with little support for representing schemas, particularly constraints. Alternatively, we may be using data from a system where the data and schema are managed by different loosely-coupled, or perhaps uncoupled, software tools. So the data may be dirty and the schema may not be an accurate model of the data.

To overcome these challenges, we provide data mining tools for discovering constraints (and approximate constraints) that hold on a specific data set [Vil02]. Given a schema, we mine the data (within the given structure) for constraints. For example, in the expenseDB schema depicted on the left side of Figure 1, we could mine the data to discover the two inclusions r_1 and r_2. These inclusions represent the fact that in the given database instance, all `grant.grantee` values are a subset of `company.cid`. An inclusion is a candidate inclusion dependency. However, from a single database instance, we cannot determine whether the inclusion will continue to hold as the data changes. In addition, we may discover that an approximate inclusion holds from `grant.sponsor` to `company.cid` (not shown in the Figure) indicating that most sponsors are companies. Constraint mining looks to find a set of constraints (logical statements) that hold on a given data set and structure. The set of predicates used in these constraints is fixed – that is, the set of predicates corresponds exactly to the set of tables (or nested tables) defined in the schema.

Schemas are the result of a data design process (performed by a human designer or by an automated tool). The choices made in data design are known to be highly subjective. A schema is inherently one out of many possible choices for modeling a data set. To understand and reconcile heterogeneous data, we may need to understand (and explicitly represent) some of the alternative design choices. Our current research focuses on the development of schema discovery techniques for finding such alternative designs.

Consider again the expenseDB schema of Figure 1. Suppose that some but not all values in `grant.grantee`

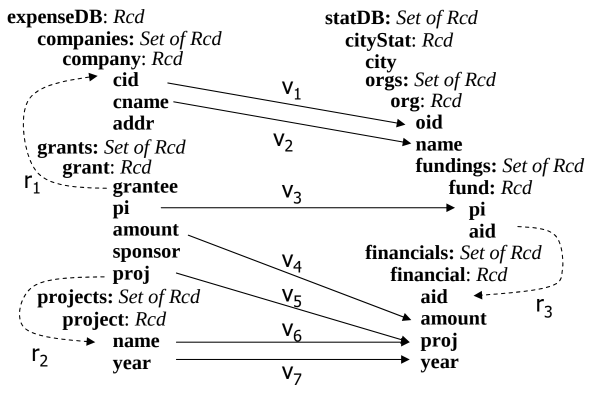

Figure 1: Two example schemas: expenseDB (left) & statDB (right).

are in `company.cid`, so that no inclusion holds between the attributes of `grants` and `companies`. Instead, `grant.grantee` may be a combination of companies, non-profit foundations and government agencies. (To keep our example simple, assume that foundations and government agencies are each modeled by a separate relational table.) Furthermore, suppose the `grants` given by `companies` (almost) always have a PI, while those given by both `foundations` and `governmentAgencies` never have a PI (the grants are given to an institution, not to an individual). These facts suggest that an alternative design for this information would be to horizontally decompose the `grants` table into three tables: `companyGrants`, `foundationGrants`, and `governmentGrants`. Such a decomposition is suggested because each subgroup of tuples shares structural characteristics (the presence or lack of a PI along with three distinct inclusion relationships into the three tables `companies`, `foundations` and `governmentAgencies`) that the group as whole does not share. We are developing mining algorithms to help find such decompositions by identifying groups of data that share common structural characteristics.

Our work can be distinguished from other structure mining techniques, including Bellman [DJMS02], in that we are searching for models that best fit the data. However, like Bellman we are performing a form of schema and data analysis. Such analysis can be very valuable in practice for large schemas that are hard for a DBA to fully understand. The tool can help a user abstract away the specific (given) design choices and better understand the schemas and the data they structure.

3 Creating and Managing Correspondences

To integrate or reconcile schemas we must understand how they correspond. If the schemas are to be integrated, their corresponding information should be reconciled and modeled in one consistent way. If data is to be exchanged (or translated) between two schemas, we must first understand *where* two schemas overlap or correspond. Methods for automating the discovery of correspondences use linguistic reasoning on schema labels [BHP94] (including ontologies and thesauri) and the syntactic structure of the schema [RB01]. Such methods have come to be referred to as *schema match operators*. The output of a schema match is an uninterpreted *correspondence relation* between elements of two schemas [RB01]. As such, a match can be viewed as a similarity join [Coh00] between two relations containing the names or descriptions (types, annotations, properties, etc.) of two separate schemas [RB01].

Within Clio, we make use of correspondences between attributes or atomic elements [NHT+02]. Our work uses not only attribute labels but also syntactic clues in the data itself to classify attributes and suggest attribute correspondences. We analyze data values and derive a set of characteristic features. We then use a classifier to group attributes with similar features. Through experimentation, we have derived a small set of domain-independent features that capture most of the structural information embedded in the attribute data. For the two example schemas of Figure 1, our techniques will, for example, suggest the attribute correspondences v_1 through v_7. Correspondence v_4 is suggested not solely based on the name similarity between attribute labels, but rather because the data values within these two attributes have similar characteristics (for example, the range of the amounts is similar and they perhaps have many 0's in them). The values themselves do not need to overlap, rather they simply need to have similar characteristics.

Note that all of the correspondence techniques we have described make use of little, if any, semantic knowledge about the data. Constraint annotations (for example, the declaration of an attribute as a key) may be used as syntactic clues to suggest a correspondence (two key attributes are more likely to correspond than a key and non-key attribute). However, these correspondence or matching approaches do not, in general, attempt to attach a semantics or meaning to specific correspondences [RB01]. Many of the matching approaches return not only attribute correspondences, but also correspondences between other schema components including relations. These correspondences may be suggested by name similarity or by correspondences among attributes of the relations. For example, the correspondences v_1 and v_2 suggest that `company` and `org` correspond. Similarly, v_3 and v_4 suggest that `grant` corresponds to both `fund` and `financial`. Note however, that the techniques for finding such "structure" correspondences are inherently dependent on the specific logical design choices made in a schema (for example, the grouping of attributes into relations or the nesting of relations) [RB01]. Hence, applying such a technique on a schema and an equivalent normalized version of the schema may yield different results. Rather than relying on such syntactic structure correspondences, we use only attribute correspondences and then apply semantic reasoning to derive a mapping [MHH00].

4 Creating and Managing Mappings

To be able to use correspondences to map data, we must associate a meaning to the correspondence relation. Typically, this semantics is expressed as inter-schema constraints. Following this convention, we express the mapping semantics as referential constraints. Since we are using a nested relational model, the mappings are then *nested referential constraints* [PVM$^+$02]. Each constraint in the mapping asserts that a query q_S over the source is associated with a query q_T over the target. In Clio, we consider *sound* mappings [Len02] where the association is a containment relation: $q_S \subseteq q_T$. Our mappings are generalizations of two common types of mappings: GAV (global-as-view) and LAV (local-as-view) mappings [Len02]. In GAV, a query over the source is asserted to be associated with a single relation of the target (global) schema. In other words, q_T is restricted to be a single relation. In LAV, a single source is asserted to be associated with a query over the target. So in LAV, q_S is restricted to be a single relation. The mappings produce by Clio have been referred to as GLAV since they have neither the GAV nor the LAV restriction [FLM99].

To create a mapping, we must take a correspondence relation R over schemas S and T and create a set of constraints, each of which has the form $q_S \subseteq q_T$. We have referred to this process as *query discovery* in that we must discover the queries used in the mapping [MHH00]. Our approach is to use the semantics of the data model to determine a set of source and target queries that produce sets of semantically related attributes. We illustrate our approach with an example.

Consider the correspondences depicted in Figure 1. In creating a mapping for the `financial` relation, we must determine which combinations of values from `grant.amount`, `project.name` and `project.year` to use in creating `financial` tuples. It is unlikely that all combinations of values are semantically meaningful. Rather, we can use the schema to determine that by taking an equi-join of `grant` and `project` on `grant.proj = project.name`, we can create triples of `grant.amount`, `project.name` and `project.year` values that are semantically related. The queries we use in our mappings are based on this idea. We use the nested structure of the schema and constraint inference to determine a set of queries that each return a set of semantically related attributes.

Often, we will find alternative semantic relationships. For example, in addition to the foreign key join between `grants` and `projects` on r_2, it may also be possible to associate `grants` with `projects` that are led by the `grant.pi`. We use carefully selected data values to illustrate and explain to a user the various mapping alternatives [YMHF01].

5 Using Mappings

The mappings produced by Clio can be used for both *data integration* (where the target schema is virtual) and for *data exchange* (where the target schema is materialized).

5.1 Data Integration

In data integration, we are given one (or more) source schemas and a single target (or global) schema. Mappings are established between each of the source schemas and the target schema. The target schema is a virtual view and as such is usually limited to have no constraints [Len02].[1] Queries posed on the target are answered by reformulating the query as a set of queries over the sources. Notice that the sound mappings that we produce do not determine a unique target instance that satisfies the mappings for a given set of source instances. Hence, it is not clear what the "right" answer to a query on the target is. The query semantics adopted most often in data integration is that of computing the *certain answers*. An instance I of the sources will produce a set of possible instances of the target J such that I and J collectively satisfy the mappings. The answer to a target query q is then the intersection of $q(J)$ as J varies over all of these possible target instances. In data integration, a target query is rewritten as a source query and the certain answers are computed *using the source instances*.

5.2 Data Exchange

A data exchange setting is very similar to that of data integration [FKMP03]. We are given a source schema (or a set of source schemas), a target schema and a mapping (or mappings) between each source and the target. As in data integration, mappings may not determine a unique target instance for a given instance I of the sources. The problem in data exchange is to materialize a single target instance (among the set of all possible target instances) that reflects the source data as accurately as possible. Furthermore, in data exchange, the target often has constraints, so we are not guaranteed that any target instance exists that satisfies the mappings and the target constraints. Given an instance I of the sources, a solution to the data exchange problem is a single target instance J that satisfies the target constraints and such that I and J collectively satisfy the mapping. In data exchange, we are faced with the problem of both determining if there is a solution to the data exchange problem and determining which solution, that is, which target instance, is the best one to exchange.

To perform data exchange, we have given an algebraic specification of a special class of *universal* solutions [FKMP03]. A universal solution has no more and no less data than needed in data exchange. In particular, for relational schemas and for target queries q that are unions of conjunctive queries, then the certain answers for q are exactly the tuples in $q(J)$ if and only if J is a universal solution. We are able to compute a canonical universal solution efficiently for a large and practical set of target constraints and mappings. We have also investigated the computational complexity of computing the certain answers in data exchange [FKMP03].

[1]Note that recent work has begun to consider constraints expressed on the target view [CGL$^+$01].

6 The Clio System

In our system, we provide support for creating mappings between combinations of relational or XML schemas (including DTDs) [PHVM02]. In addition to mappings (the constraints described in Section 4), Clio also produces data translation queries. These source queries produce target data satisfying the constraints and (possibly nested) structure of the target schema. If the source and target have different data content, then the translation queries may also invent values for non-null target attributes that have no correspondence to the source [PVM$^+$02]. For XML sources, these queries are produced in XQuery or XSLT. For relational sources, these queries are produced in SQL (if the target is relation) or SQL with XML extensions (if the target is XML). These translation queries are used for data exchange to produce a materialized target instance. They may also be used for query composition in data integration settings.

Our technology is already being transferred into the DB2 product line. Clio has been used to create a sophisticated query builder (SQL Assist) for relational schemas. In SQL Assist, a user specifies the schema of the desired query (the attributes) and the tool suggests the query structure. Constraint reasoning is used to suggest queries that are semantically meaningful. In addition, Clio is being used in an XML to relational translator. The tool provides a default translation of an XML document into relations. A user may then modify this relational schema to meet her own requirements. Clio automatically produces and maintains a translation query as the target schema is modified.

References

[BGK$^+$02] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In *Proc. of the Int'l Workshop on the WEB and Databases (WebDB)*, 2002.

[BHP94] M. W. Bright, A. R. Hurson, and S. Pakzad. Automated Resolution of Semantic Heterogeneity in Multidatabases. *ACM Trans. on Database Sys. (TODS)*, 19(2):212–253, June 1994.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[CGL$^+$01] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data integration in data warehousing. *Int'l Journal of Cooperative Information Systems*, 10(3):237–271, 2001.

[Coh00] W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Office Information Systems*, 18(3):288–321, 2000.

[DJMS02] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. How to Build a Data Quality Browser. In *ACM SIGMOD Int'l Conf. on the Management of Data*, 2002.

[FKMP03] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. To appear, 2003.

[FLM99] M. Friedman, A. Levy, and T. Millstein. Navigational Plans for Data Integration. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 67–73. AAAI Press/The MIT Press, 1999.

[HM85] D. M. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Office Information Systems*, 3:253–278, 1985.

[Len02] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 251–262, Bombay, India, 1996.

[MHH00] R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 77–88, Cairo, Egypt, September 2000.

[MHH$^+$01] R. J. Miller, M. A. Hernández, L. M. Haas, L.-L. Yan, C. T. H. Ho, R. Fagin, and L. Popa. The Clio Project: Managing Heterogeneity. *SIGMOD Record*, 30(1):78–83, March 2001.

[NHT$^+$02] Felix Naumann, Ching-Tien Ho, Xuqing Tian, Laura Haas, and Nimrod Megiddo. Attribute classification using feature analysis. In *Proc. of the Int'l Conf. on Data Eng.*, San Jose, CA, 2002. Poster.

[PHVM02] L. Popa, M. A. Hernández, Y. Velegrakis, and R. J. Miller. Mapping XML and Relational Schemas with CLIO, *System Demonstration*. In *Proc. of the Int'l Conf. on Data Eng.*, San Jose, CA, February 2002.

[PVM$^+$02] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 598–609, Hong Kong SAR, China, August 2002.

[RB01] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The Int'l Journal on Very Large Data Bases*, 10(4):334–350, 2001.

[Vil02] C. Vilarem. Approximate Key and Foreign Key Discovery in Relational Databases. Master's thesis, Department of Computer Science, University of Toronto, 2002.

[YMHF01] L.-L. Yan, R. J. Miller, L. Haas, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. *ACM SIGMOD Int'l Conf. on the Management of Data*, 30(2):485–496, May 2001.