The Clio Project: Managing Heterogeneity

Renée J. Miller[1] Mauricio A. Hernández[2] Laura M. Haas[2] Lingling Yan[2]
C. T. Howard Ho[2] Ronald Fagin[2] Lucian Popa[2]

[1]Univ. of Toronto [2]IBM Almaden Research Center
miller@cs.toronto.edu {mauricio,laura,lingling,ho,fagin,lucian}@almaden.ibm.com

Abstract

Clio is a system for managing and facilitating the complex tasks of heterogeneous data transformation and integration. In Clio, we have collected together a powerful set of data management techniques that have proven invaluable in tackling these difficult problems. In this paper, we present the underlying themes of our approach and present a brief case study.

1 Introduction

Since the advent of data management systems, the problems of data integration and transformation have been recognized as being ubiquitous and critically important. Despite their importance and the wealth of research on data integration, practical integration tools are either impoverished in their capabilities or highly specialized to a limited task or integration scenario. As a result, integration and transformation remain largely manual, time-consuming processes. However, a careful examination of both integration tools and research proposals reveals an interesting commonality in the basic data management techniques that have been brought to bear on these problems. This is a commonality that we believe has not been sufficiently exploited in developing a general purpose integration management tool. In the Clio project, a collaboration between IBM Almaden Research Center and the University of Toronto, we have built a tool that automates the common, even routine, data and structure management tasks underlying a wealth of data integration, translation, transformation and evolution tasks.

We begin in Section 2 with a brief overview of some of the problems we are addressing. In Section 3, we present an overview of the Clio architecture and discuss how we support common requirements for managing heterogeneous data. We present a brief overview of how Clio can be used in a case study in Section 4 and conclude in Section 5 with a brief description of the current direction of our work.

2 Data Integration, Transformation, and Evolution

Many modern data applications in data warehousing and electronic commerce require merging, coalescing and transforming data from multiple diverse sources into a new structure or schema. Many of these applications start with an understanding of how data will be used and viewed in its new form. For instance, in a data exchange scenario, the exchange format may have been standardized (perhaps in the form of a standard XML schema or DTD). Other applications may involve an integration step in which the transformed or integrated structure is created. The activities involved in many of these integration or transformation scenarios can be grouped into three broad categories.

Schema and Data Management At the core of all integration tasks lies the representation, understanding and manipulation of schemas and the data that they describe and structure. The specifics of different integration proposals can vary dramatically. However, all approaches require reasoning about schemas, data and constraints. Often legacy schemas are underspecified or have not been maintained to accurately and completely model the semantics of a perhaps evolving data set. Since integration methodologies depend on the accuracy and completeness of structural and semantic information, they are best employed in an environment where specified (and unspecified) schema information, constraints and relationships can be learned, reasoned about and verified.

Correspondences Management A second step common to all integration tasks is the understanding of how different, perhaps independently developed, schemas (and data) are related. To support this activity, a set of correspondences or "matches" between schemas must be determined. In the schema integration literature, this process is referred to as determining "inter-schema" relationships [RR99]. In model management, it is referred to as model matching [BHP00]. For example in the schemas of Figure 2, different terminology is used within the schemas (and perhaps within the data). Before we can integrate the two schemas (or before we can translate data from one representation to the other), we must have some understanding of how the schemas correspond.

Integration tasks that involve matching often cannot be fully automated since the syntactic representation of schemas, metadata and data may not completely convey the semantics of different data sets. As a result, we must rely on an outside source (either a user or a knowledge discovery technique) to provide some information about how different schemas (and data) correspond. There have been a host of techniques developed for (partially) automating the matching task developed both for the specific problem of schema integration [RR99] and for the broader model management task [BHP00]. While differing in the knowledge used and the reasoning or knowledge discovery techniques employed, at their core all these techniques learn or propose associations or correspondences between components of different schemas. For example, in Figure 2, there may be a correspondence between `Calls.Caller` and `References.Artifact` (perhaps `Calls` contains information about program functions which call each other and functions are considered to be program artifacts in the warehouse schema). Similarly, there may be a separate correspondence between the file in which a function is defined (`Function.File`) and the source of a program artifact (`References.Source`). Such correspondences may be entered by a user, perhaps using a graphical interface such as the one supported in Clio [MHH00], or learned using a machine learning technique applied to the data or schema names. These correspondences may in turn be given different interpretations. Perhaps a correspondence means that one attribute is a subset of the other. Or perhaps it means that the two attributes are semantically related.

However, regardless of interpretation, there are some characteristics that all matching approaches share. First, no approach, whether manual or automated, is always complete, nor always accurate for all possible schemas. As a result, it is important to permit verification of the correspondences, either manually or using a knowledge discovery technique. Second, the sheer number of correspondences can be enormous. No successful matching technique has employed solely relation or class level correspondences. Rather, matching requires a fine grain specification of correspondences at the attribute (or even data) level. As a result, any technique that uses manual specification or verification of correspondences, must necessarily be incremental in nature to permit a user to work with large schemas and large sets of correspondences without being overwhelmed. An incremental approach also facilitates the correction and refinement of correspondences.

Mapping Management The third common step is that of creating an operational mapping between schemas. Such a mapping is a program or set of queries than can be used to translate data between the schemas. Creating and maintaining such mappings is today a largely manual (and extremely complex) process. In some integration scenarios, the mapping (perhaps a view definition) may be an artifact of the transformation used. However, the mappings produced by a series of transformations and merging steps must be integrated and composed. In many other applications, the integrated schema is created independently of the source schemas. Hence, mapping must be done independently [MHH00]. For instance, before a data warehouse can be loaded, DBAs and consultants spend months determining what types of queries will be asked, and then designing a schema that will readily support those queries. To load the warehouse, they then must create mappings between the warehouse schema and the underlying data sources' schemas. To deploy a global information system, experts first determine what information it will present to the world, that is, what logical structure (the transformation process), and then create the view definitions (the mapping creation process) that map between the new schema and the data sources. The focus in these schema mapping applications is on the discovery of a set of queries that realizes the mapping [MHH00].

Mapping management, like schema and correspondence management, requires reasoning about matches and schemas. The correspondence process may not be sufficient to fully convey the semantics of how schemas are related. In the example mentioned above, if `Calls.Caller` and `References.Artifact` have been matched and if `Function.File` and

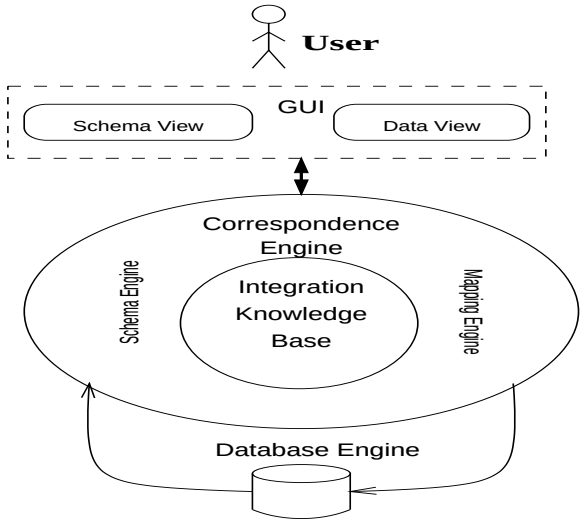

Figure 1: Clio's Logical Architecture

`Reference.Source` have been matched (Figure 2), these correspondences alone are not sufficient to uniquely determine how data from the source will appear in the target. In particular, details about how source entities are paired or joined are left unspecified as are details of which source entities should be included. Resolving such ambiguities requires reasoning about schemas and constraints and may result in constraints or correspondences being modified or added.

3 An Overview of Clio

Clio is a system for managing and facilitating the complex tasks of heterogeneous data transformation and integration. Note that Clio does not perform schema integration *per se*. Rather, Clio supports the generation and management of schemas, correspondences between schemas and mappings (queries) between schemas. The logical architecture of Clio is depicted in Figure 1. Each management and reasoning component makes use of a database management system for storing knowledge gained about schemas and integrations. Clio provides schema and data browsers to elicit and obtain feedback from users and to allow user to understand the results produced by each component.

Schema Engine A typical session with Clio starts with the user loading one or more schemas into the system. These schemas are read from either an underlying Object-Relational database, a legacy source that has been wrapped with a Garlic Object-Relational wrapper [TS97], or from an XML file with an associated XML schema. The schemas may be legacy schemas or they may include an integrated schema produced manually or by an integration tool. The schema engine is used to augment the schema with additional constraint information, if necessary. Currently, Clio makes use of metadata, including query workloads (if available) and view definitions, along with data. For example, in the absence of declared constraints, we mine the data for possible keys and foreign keys. Finally, the schemas are verified by the user to ensure validity of generated information. For example, a discovered foreign key or inclusion dependency may hold on the current instance by accident, that is to say, it may not necessarily hold for all, or even most, instances. Clio permits such corrections to be made by a user.

To facilitate this process, Clio makes use of a graphical user interface to communicate information to the user [YMHF01]. In the **Schema View** mode, users see a representation of the schemas including any generated information. This view may be used to edit or further augment the schema. In addition, we provide a **Data View** mode, through which users may see some example data from the schemas to further help them understand the schemas. The data view can be invaluable in helping users understand opaque schema labels.

Correspondence Engine Given a pair of schemas, the correspondence engine generates and manages a set of candidate correspondences between the two schemas. Currently, we make use of an attribute classifier to learn possible correspondences [HT01]. Clio could (and may in the future) be augmented to make use of dictionaries, thesauri, and other matching techniques. The generated correspondences can be augmented, changed or rejected by a user using a graphical user interface through which users can draw value correspondences between attributes. Entering and manipulating value correspondences can be done in two modes. In the **Schema View** mode, users see a representation of the schemas and create value correspondences by selecting schema objects to be included in a correspondence. The alternative **Data View** mode offers a WYSIWYG interface that displays example data for the attributes used in the correspondences [YMHF01]. The data view helps a user check the validity of generated and user entered information. Users may add and delete value correspondences and immediately see the changes reflected in the example data.

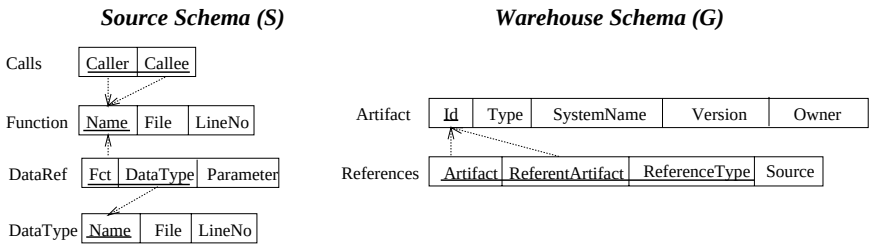

Figure 2: Schema of a source database and a target Software Engineering Warehouse

Mapping Engine The mapping engine supports the creation, evolution and maintenance of mappings between pairs of schemas. A mapping is a set of queries from a source schema to a target schema that will translate source data into the form of the target schema. Clio produces a mapping (or set of alternative mappings) that are consistent with the available correspondences and schema information. The mapping engine is therefore using information gathered by both the schema engine and the correspondence engine. As with the correspondences and schemas constructs suggested by Clio, mappings are verified using the data view to help users understand alternative mappings. Users see example data from selected source tables and the contents of the target as they would appear under the current mapping. Examples are carefully chosen to both illustrate a given mapping (and the correspondences it uses) and to illustrate the perhaps subtle differences between alternative mappings [YMHF01]. For example, in some cases, changing a join from an inner join to an outer join may dramatically change the data produced by the mapping. In other cases, the same change may have no effect due to constraints that hold on the schemas.

To permit scalability and incremental invocation of the tool, we also permit (partial) mappings to be read and modified. Such mappings may be created by a former session with Clio or by another integration tool. For example, a user may have used Clio to map a source and target schema. At a later time, after the source schema has evolved, the user may again invoke Clio to create a mapping from the modified source to the target. The old mappings may be read in and used as a starting point for the mapping process. Modification is done using operations on data examples, in the data view [YMHF01].

The mapping creation process is inherently interactive and incremental. Clio stores the current mapping within its knowledge base and, through an incremental mapping discovery algorithm, allows users to extend and refine mappings one step at a time [MHH00]. For example, when value correspondences are added, deleted or modified within the correspondence engine, the mapping engine uses the new correspondences or modification to update the mapping. Similarly, in order to verify a particular mapping, the mapping engine may invoke the schema engine to verify whether a specific constraint holds in a schema.

4 A Data Warehouse Example

To illustrate our approach, we present an example based on a proposed software engineering warehouse for storing and exchanging information extracted from computer programs [BGH99]. Such warehouses have been proposed both to enable new program analysis applications, including data mining applications [MG99], and to promote data exchange between research groups using different tools and software artifacts for experimentation [HMPR97]. Figure 2 depicts a portion of a warehouse schema for this information. This schema has been designed to represent data about a diverse collection of software artifacts that have been extracted using different software analysis tools. The warehouse schema was designed to be flexible and uses a very generic representation of software data as labeled graphs. Conceptually, software artifacts (for example, functions, data types, macros, *etc.*) form the nodes of the graph. Associations or references between artifacts (for example, function calls or data references) form the edges. Two of the main tables for artifacts and references are depicted in the figure.

As new software analysis tools are developed, the data from these tools must be mapped into this integrated schema. In Figure 2, we also give a relational representation of an example source schema. This schema was imported using a wrapper built on top of output files produced by a program analysis tool. The wrapper produces a flat schema with no constraints. Clio's schema engine is used to suggest a set of keys and foreign keys that hold on the data. Foreign keys are depicted by dashed lines. Key attributes are un-

derlined. The user may use Clio's Schema View to browse, edit or augment this schema information.

To start the correspondence process, if the warehouse is populated with data, our correspondence engine will apply an attribute matching algorithm to determine potential correspondences based on the characteristics of the values of different attributes. For example, if values in `Calls.Caller` in the source and `References.Artifact` are all UNIX file pathnames (that is, a sequence of mainly alphanumeric tokens separated by '/'s with perhaps one token having a '.' extension), our mining algorithm would suggest that these attributes match [HT01]. If the warehouse is not populated with data, and the user cannot provide a few example values, then correspondences may be entered using our Schema View as suggested in Figure 3. It is unlikely that a mining technique based solely on schema labels will be effective for this example since it is not obvious, even using ontology or dictionary based techniques, that `Calls.Caller` should be matched with `References.Artifact`.

Suppose the user had only entered the first four correspondences ($f_1 - f_4$) indicating how function call information corresponds to the warehouse schema. Using the discovered schema information together with these correspondences, Clio's mapping engine may produce the following two mappings. The first populates the `Source` attribute of the target with the `File` attribute of the caller function (Mapping S_1). The second populates the `Source` attribute of the target with the `File` attribute of the called function (Mapping S_2). Note that correspondence f_4 maps the relation name into the ReferenceType value, effectively transforming schema to data [Mil98]. The left-outer join is used to ensure information is not lost in the mapping. That is, Clio will prefer mappings that map every function (whether it participates in the Calls relation or not) to a target value.

S_1:
```
SELECT C.Caller, C.Callee, relname(C), F.File
FROM   Function F left outer join Calls C
WHERE  C.Caller = F.Name
```
S_2:
```
SELECT C.Caller, C.Callee, relname(C), F.File
FROM   Function F left outer join Calls C
WHERE  C.Callee = F.Name
```

To validate and choose among these mappings, Clio will illustrate this mapping using the data view. Example data will be selected and displayed for the user. The examples will illustrate the differences in the join paths used in each mapping selected by Clio. Hence, to illustrate S_1, an example will be used of a function that calls at least one function but is itself not called (such a function will appear in the target associated with its callee using S_1, but not using S_2). Similarly, an example will be used of a function that is called by at least one function but does not call another function. Of course, such examples are used only if they are available in the data source. Once a join path is selected (perhaps S_1 is selected), examples are also used to determine if the mapping should be a left-outer join or if the user only wishes References to be populated with functions that appear in Calls (an inner join) [YMHF01].

Once the user is happy with this (partial) mapping, she may proceed incrementally by entering more value correspondences, by using operations on data examples to refine the current mapping [YMHF01], or by reinvoking the correspondence engine. Due to space limits we only illustrate this final option. Clio may make use of a (partial) mapping to deduce additional correspondences. Given the mapping S_1 and the discovered constraints on S, Clio can infer that `DataRef.Fct` also may correspond to `Reference.Artifact`. Similarly, if the attribute matching algorithm was applied to the source attributes alone, a value correspondence between `Function.File` and `DataType.File` may have already been deduced *within* the source schema. This correspondence is not based on the two attributes containing the same values necessarily. Rather, it is an indication that the values share similar characteristics and therefore may (possibly) have a semantic relationship. Based on this information, Clio may propose f_6 and f_8 as potential correspondences. The mapping engine may then search for potential join paths to use in mapping data reference information to the warehouse.

5 Conclusion

We have discussed Clio, a system for managing data transformation and integration under development at IBM Almaden. Clio's Integration Engine is composed of three components (Schema, Correspondence, and Mapping Engine) that interact with our internal mapping knowledge base and with the user to produce the desired mapping. Our initial implementation of Clio includes most of the functionalities of the Correspondence Engine and the Mapping Engine described in this paper. Using the GUI's Schema View, users can draw correspondences among the selected source and target schemas and review the resulting mapping query (which is currently expressed as an SQL View). The initial implementation of the Schema Engine includes schema readers for relational and XML Schema sources. Augmentation of these

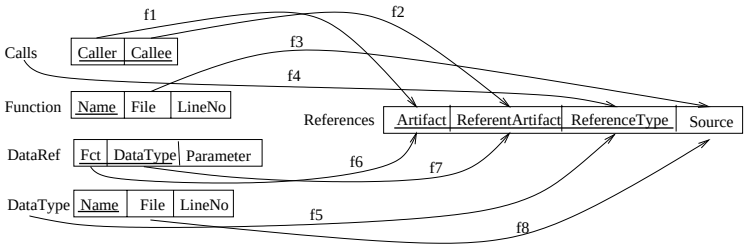

Figure 3: Value correspondences used to map between the source schema the warehouse schema

schemas is possible through a knowledge discovery module that searches for keys and referential constraints using the underlying data.

We envision a number of extension to the Mapping Engine. The mapping algorithm used in our prototype can only handle correspondences from flat relational source schemas into either relational or nested target schemas. We are working on generalizing this algorithm to handle correspondences from *nested* source schemas into nested target schemas. The next version of the system will be able to use source and target data constraints in combination with the input correspondences to validate the mapping (i.e., detect inconsistencies) and logically infer new mappings. If inconsistencies arise after the user enters a value correspondence, Clio should be able to explain the problem (e.g., violation of a key constraint) and suggest fixes (e.g., modifying the correspondence or relaxing a constraint). We are also looking into the ability to invert mappings (when possible) which would allow Clio to be used for bi-directional exchange of data.

Ultimately, we view Clio as an extensible management platform on which we can build a host of new integration and transformation techniques including perhaps a robust query facility for schemas, correspondences and mappings (for example, to permit users to ask questions about mappings and their properties). An important theme in Clio, which we expect to continue, has been the use of data to help users to understand the results produced by each reasoning component.

References

[BGH99] I. T. Bowman, M. W. Godfrey, and R. C. Holt. Connecting Software Architecture Recovery Frameworks. In *Proceedings of the First International Symposium on Constructing Software Engineering Tools (CoSET'99)*, Los Angeles, May 17-18 1999.

[BHP00] P. A. Bernstein, A. Y Halevy, and R. A. Pottinger. A Vision for Management of Complex Models. *SIGMOD Record*, 29(4):55–63, 2000.

[HMN+99] L. M. Haas, R. J. Miller, B. Niswonger, M. Tork Roth, P. M. Schwarz, and E. L. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *IEEE Data Engineering Bulletin*, 22(1):31–36, 1999.

[HMPR97] M. J. Harrold, R. J. Miller, A. Porter, and G. Rothermel. A Collaborative Investigation of Program-Analysis-Based Testing and Maintenance. In *International Workshop on Experimental Studies of Software Maintenance*, pages 51–56, Bari, Italy, October 1997.

[HT01] H. C. T. Ho and X. Tian. Automatic Classification of Database Columns Using Feature Analysis. Submitted for publication, 2001.

[MG99] R. J. Miller and A. Gujarathi. Mining for Program Structure. *International Journal on Software Engineering and Knowledge Engineering*, 9(5):499–517, 1999.

[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.

[Mil98] R. J. Miller. Using Schematically Heterogeneous Structures. *ACM SIGMOD Int'l Conf. on the Management of Data*, 27(2):189–200, June 1998.

[RR99] S. Ram and V. Ramesh. Schema Integration: Past, Current and Future. In A. Elmagarmid, M. Rusinkiewicz, and A. Sheth, editors, *Management of Heterogeneous and Autonomous Database Systems*, pages 119–155. Morgan Kaufmann Publishers, 1999.

[TS97] M. Tork Roth and P. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 266–275, Athens, Greece, August 1997.

[YMHF01] L. Yan, R. J. Miller, L. Haas, and R. Fagin. Data-Drivien Understanding and Refinement of Schema Mappings. Submitted for publication, 2001.