EIRENE: Interactive Design and Refinement of Schema Mappings via Data Examples *

Bogdan Alexe
UCSC
abogdan@cs.ucsc.edu

Balder ten Cate
UCSC
balder.tencate@gmail.com

Phokion G. Kolaitis
UCSC & IBM Research - Almaden
kolaitis@cs.ucsc.edu

Wang-Chiew Tan
IBM Research - Almaden & UCSC
wangchiew@us.ibm.com

ABSTRACT

One of the first steps in the process of integrating information from multiple sources into a desired target format is to specify the relationships, called schema mappings, between the source schemas and the target schema. In this demonstration, we showcase a new methodology for designing schema mappings. Our system Eirene interactively solicits data examples from the mapping designer in order to design a schema mapping between a source schema and a target schema. A data example, in this context, is a pair consisting of a source instance and a target instance showing the desired outcome of performing data exchange using the schema mapping being designed. One of the central parts of the system is a module that, given a set of data examples, either returns a "best" fitting schema mapping, or reports that no fitting schema mapping exists.

1. INTRODUCTION

A schema mapping is a high-level declarative specification of the relationship between two database schemas, which we refer to as a source schema and a target schema. Designing schema mappings is a fundamental step in integrating data from different, heterogeneous sources. In many real-life scenarios, the schemas involved are complex, and designing the right schema mappings is often a daunting task.

Several mapping-design systems, including research prototypes such as Clio [5] and HePToX [3] and commercial offerings such as Altova MapForce[1] and Stylus Studio[2], have been developed to facilitate the process of designing schema mappings. All these systems are based on a design methodology where the mapping designer is required to provide a visual specification of the relationship between the source and target schemas, before a schema mapping can be generated. Visual specifications of the relationship between schemas are inherently ambiguous, in the sense that a visual specification alone is not enough to uniquely determine a schema mapping [1]. Indeed, in many cases, several logically inequivalent schema mappings can be derived from a visual specification. Hence, it is often the case that the mapping designer will still need to study and understand the details of the generated schema mapping in order to ensure that it has the intended transformation semantics.

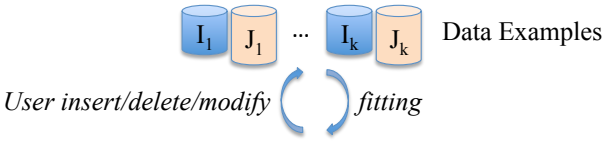

Figure 1: Eirene workflow for interactive design and refinement of schema mappings via data examples.

In this demonstration, we showcase Eirene, a system that supports a methodology for designing schema mappings that departs significantly from the methodology used by existing systems that we described above. In Eirene, a schema mapping is derived from data examples that are provided by the mapping designer, following the approach that was developed in [2]. A data example is a pair of source and target instances that conform to the source and target schemas respectively. In each data example that is passed to Eirene, the target instance represents the desired outcome of performing data exchange on the source instance using the schema mapping being designed. Hence, each data example represents a partial specification of the semantics of the schema mapping that is to be designed. Eirene interactively solicits data examples from the mapping designer and either returns a schema mapping that "best fits" the given set of data examples or reports that no fitting schema mapping exists.

Eirene is tailored for generating schema mappings specified by *GLAV (Global-and-Local-As-View) constraints* (also known in the literature as *source-to-target tuple generating dependencies*, or *s-t tgds*). We call such schema mappings *GLAV schema mappings*. GLAV schema mappings have been extensively studied in the context of data exchange and data integration [6, 7]. The GLAV schema mappings contain, as important special cases, *Local-As-View (LAV)* schema mappings and *Global-As-View (GAV)* schema mappings. Furthermore, they are used in such systems as Clio [5] and HePToX [3].

In what follows, we will describe the workflow of Eirene shown in Figure 1. In Section 2, we will describe the individual features of Eirene that we plan to demonstrate.

Workflow of Eirene Assuming that the goal is to design a schema mapping from a relational source schema $\mathbf{S}$ to a relational target schema $\mathbf{T}$, the interaction between the mapping designer and

*Research on this paper was supported by NSF grant IIS-0905276 and NSF Career Award IIS-0347065.

[1] www.altova.com/mapforce.html

[2] www.stylusstudio.com/xml_mapper.html

Proceedings of the VLDB Endowment, Vol. 4, No. 12

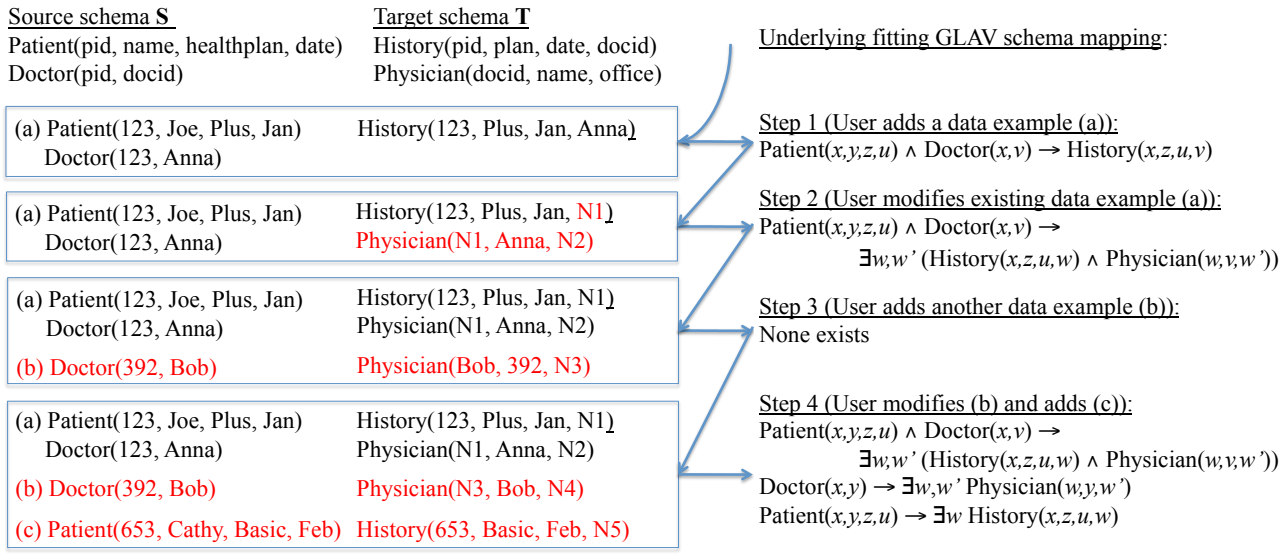

Figure 2: An example of Figure 1. Left: Each box is an input set of data examples. Right: Derived schema mapping, or none exists.

Eirene begins with the mapping designer providing an initial set $\mathcal{E}$ of data examples through Eirene's user interface. Recall that a data example is a pair (I, J), where I is an instance that conforms to $\mathbf{S}$ and J is an instance that conforms to $\mathbf{T}$.

After the set of data examples has been produced, the mapping designer can invoke Eirene to generate a GLAV schema mapping that *fits* the data examples in $\mathcal{E}$ or to determine and report that no such GLAV schema mapping exists. We say a GLAV schema mapping $\mathcal{M}$ *fits* the set $\mathcal{E}$ of data examples if for every $(I, J) \in \mathcal{E}$, J is a universal solution for I w.r.t. $\mathcal{M}$. Universal solutions were introduced in [4]. They formalize what is, intuitively, the natural outcome of transforming source data with respect to a schema mapping, and they are regarded as the preferred solutions in data exchange. More formally, a universal solution of a source instance I with respect to a schema mapping $\mathcal{M}$ is a "most general" target instance that, together with I, satisfies the specifications of $\mathcal{M}$, where "most general" is defined in terms of homomorphisms (see [4] for the precise definition).

Depending on the output of the system, the mapping designer may go back to $\mathcal{E}$ and make modifications, such as adding new data examples, removing some data examples, or inserting new tuples, or deleting or modifying existing tuples in the data examples. After this, the mapping designer can invoke Eirene again to test whether or not there is a GLAV schema mapping that fits the new set $\mathcal{E}'$ of data examples. As before, Eirene returns such a GLAV schema mapping, if one exists, or reports that none exists, otherwise. The cycle of generating schema mappings and modifying data examples can be repeated until the mapping designer is satisfied. We call this process the *interactive refinement* of a schema mapping via data examples.

Technical Features of Eirene As shown in [2], Eirene is a sound and complete system for determining the existence of a fitting GLAV schema mapping, given a finite set of data examples. In other words, Eirene finds a fitting GLAV schema mapping for a given finite set of data examples precisely when one exists. In addition, the schema mapping returned by Eirene is guaranteed to be the *most general* fitting GLAV schema mapping. That is, the GLAV schema mapping $\mathcal{M}$ returned by Eirene has the property that for every other GLAV schema mapping $\mathcal{M}'$ that fits the same set of data examples, we have that the constraints of $\mathcal{M}'$ logically imply those of $\mathcal{M}$. Furthermore, Eirene is "complete-for-design" in the sense that for any GLAV schema mapping $\mathcal{M}$, there is always a finite set of data examples that can be passed as input to Eirene to derive a GLAV schema mapping that is logically equivalent to $\mathcal{M}$.

We omit here the technical details of the algorithm at the core of Eirene. For a complete discussion of the properties, including a complexity analysis showing that the algorithm is worst-case optimal, we refer the interested reader to [2].

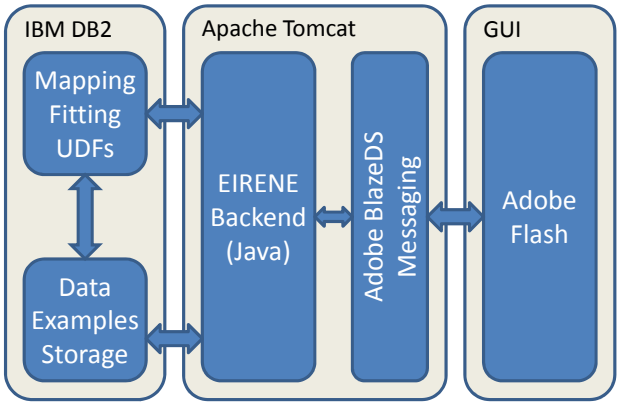

Figure 3: Eirene: System Architecture.

Implementation Overview In Figure 3, we present a high-level architecture diagram of our system. The core algorithms behind Eirene are implemented in Java 6. Our data examples are stored in an IBM DB2 Express-C 9.7 database. Some parts of our algorithm that help determine whether or not there is a fitting GLAV schema mapping are implemented as a set of user defined functions within the IBM DB2 database engine. The graphical user interface of Eirene for entering the data examples, as well as other features for analyzing the data examples and the derived schema mappings, is deployed as a Flash component developed using Adobe Flex 4.

The communication between the user inteface and the core of the system is performed via the Adobe BlazeDS messaging component, embedded within Apache Tomcat.

2. DEMONSTRATION OVERVIEW

We describe here the set of features of Eirene that we will demonstrate. For simplicity, we assume that the mapping designer wishes to design a schema mapping between two simple source and target schemas shown in the top-left corner of Figure 2, where the source schema has two relations: Patient and Doctor, and the target schema has two relations: History and Physician. In our actual demonstration, we plan to showcase Eirene on real-life schemas as well, such as Mondial[3], DBLP[4] , and Amalgam[5].

Designing a schema mapping from a given set of data examples: Figure 2 illustrates a sequence of data examples that a mapping designer may enter into Eirene. Due to space limitations, we do not show the sequence of screenshots from Eirene for entering the data examples. However, in the actual demonstration, each data example can either be entered directly through Eirene's user interface or a set of data examples can be created offline in a text file and imported in Eirene. Suppose that, as a first step, the mapping designer specifies a single data example, shown in the first box, which essentially states that `Anna` is the doctor of patient `Joe`, whose health plan is `Plus`, and date-of-visit is `Jan`. In the target relation of the data example, there is a single fact, also entered by the mapping designer, that consolidates information from the source instance and omits the name of the patient.

Testing for the existence of a fitting GLAV schema mapping: At this point, the mapping designer may decide to invoke Eirene's fitting algorithm by clicking on the "Schema Mappings / Fitting Diagnostics" tab on the user interface. (This tab can be seen on top of Figure 4.) Eirene returns a fitting GLAV schema mapping in this case, which is shown on under "Step 1" on the right of Figure 2. Note that the GLAV schema mapping is shown as a first-order formula, where all the universal quantifers have been omitted. This schema mapping states that whenever a Patient tuple and Doctor tuple agree on the `pid` value (i.e., a natural join between Patient and Doctor), create a target tuple with the `pid`, `healthplan`, `date`, and `docid` values from Patient and Doctor.

Refining the data examples: It is possible that the mapping designer may realize that there was a typographical error in the specified data example after this step. Indeed, `Anna` should not appear under `docid` in the target History tuple. At this point, the mapping designer may refine the data example into the one shown in the second box of Figure 2 through Eirene's user interface. The mapping designer may modify the target instance to consist of two tuples: a History tuple and a Physician tuple which are "connected" through the value `N1`. In addition, the office of `Anna` is given by the value `N2`. Observe that the values `N1` and `N2` in the target instance do not occur among the values of the source instance and they, intuitively, represent unknown and possibly different values.

Again, the mapping designer may decide to invoke Eirene to fit a GLAV schema mapping by hitting on "Schema Mappings / Fitting Diagnostics" tab on the user interface and the schema mapping will be refined into the one shown under Step 2 in Figure 2. Intuitively, this schema mapping states that information from the inner join of Patient and Doctor should be migrated to the target History and Physician relations, with appropriate labels to represent unknown

[3] `dbis.informatik.uni-goettingen.de/Mondial`
[4] `dblp.uni-trier.de/db`
[5] `dblab.cs.toronto.edu/~miller/amalgam`

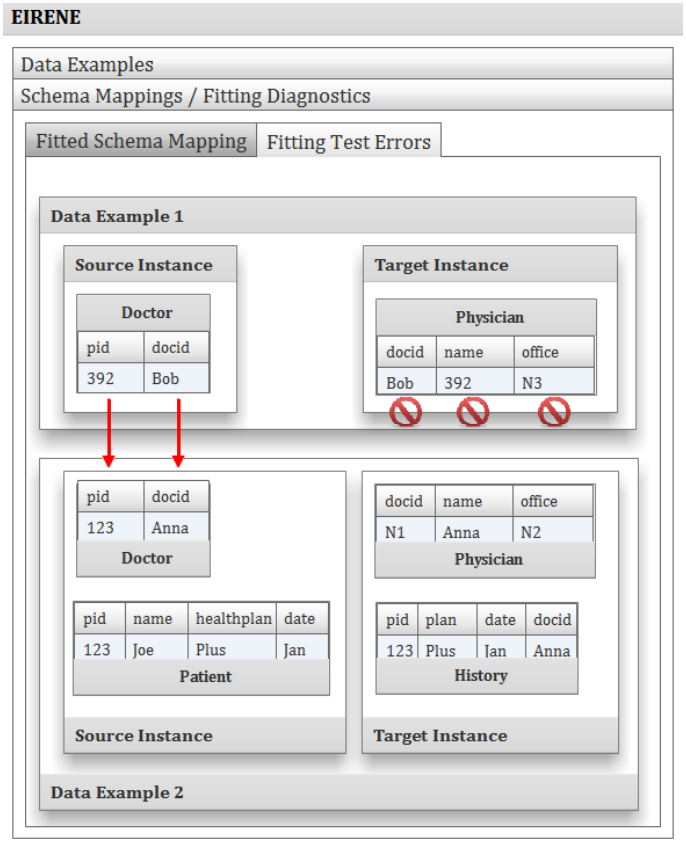

Figure 4: A visual explanation on why no fitting schema mapping exists: under the interpretation that Doctor(392,Bob) in data example 1 corresponds to Doctor(123, Anna) in data example 2, there is no way that the target fact Physician(Bob, 392, N3) can be mapped consistently to any target Physician fact in the second data example.

and possibly different values. The GLAV schema mapping under Step 2 states exactly this: for every Patient and Doctor facts that join on `pid`, there must exist History and Physician facts in the target with appropriate `pid`, `plan`, `date`, and `name` values copied from the source, as well as possibly different values w and w' for `docid` and `office` respectively.

Understanding why no fitting GLAV mapping exists: Further refinement steps can occur on the data examples. In the third box of Figure 2, the mapping designer adds a second data example (b) to the existing data example, and Eirene now reports that no GLAV schema mapping can fit. This is because data example (b) describes a pattern of data migration that is inconsistent with data example (a): According to (b), every Doctor(pid,docid) fact in the source must have a corresponding Physician(docid,pid,office) fact in the target. Observe that the `pid` value is copied to the second column of the corresponding Physician fact. However, this is inconsistent with what (a) states, where a Doctor(pid, docid) has a corresponding Physician(_,docid,_) fact in the target, and `docid` gets copied to the second column of the corresponding Physician fact instead.

Eirene visually presents the inconsistency that we have just described in Figure 4. Our visual presentation allows the mapping designer to gain insight into why no fitting GLAV schema mapping exist. Intuitively, there is no fitting GLAV schema mapping because under the interpretation that Doctor(392,Bob) in data exam-

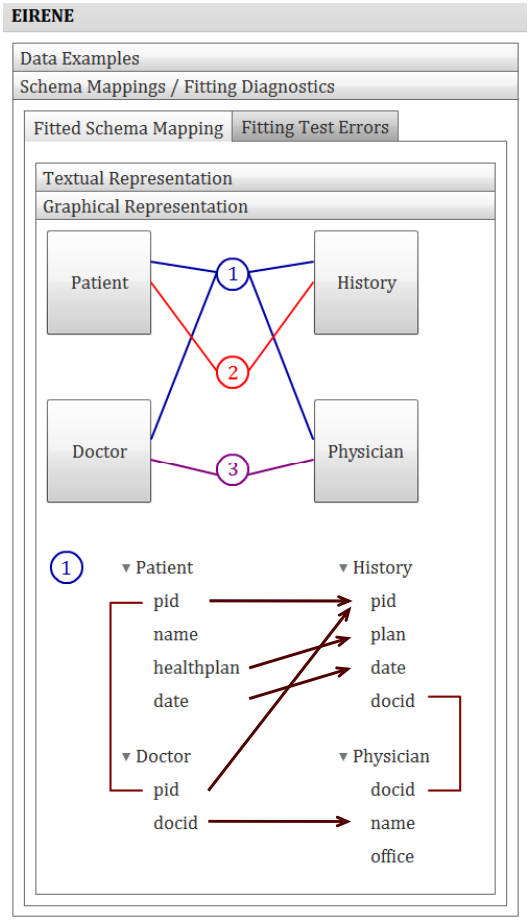

Figure 5: Graphical representation of schema mappings in Eirene. Top: high-level display of relationships specified by the schema mappings in Step 4 of Figure 2. Bottom: Zoom-in on the correspondences asserted by the first schema mapping.

ple 1 corresponds to Doctor(123, Anna) in data example 2, there is no way that the target fact Physician(Bob, 392, N3) can be mapped consistently to any target Physician fact the second data example.

In general, whenever no fitting GLAV schema mapping exists for a given set of data examples, Eirene will depict the inconsistency it has detected based on two data examples by showing how an interpretation of source facts from the first data example into the second data example cannot be extended consistently to the facts in the target instances of the two data examples.

Graphical representation of schema mappings: To continue with our running example in Figure 2, the mapping designer modifies the data example (b) and adds a third data example (c) (shown in the fourth box). Based on these data examples, Eirene reports the schema mapping shown under Step 4 on the right. Essentially, this schema mapping migrates information from the outer join of Doctor and Patient to the corresponding relations in the target. In particular, Doctors tuples, which may not join with any Patient tuples, are migrated according to the pattern depicted by data example (b), and Patient tuples, which may not join with any Doctor tuples, are migrated according to the pattern depicted by data example (c).

In addition to the textual representation of Figure 2, Eirene has the ability to present to the mapping designer the generated schema mapping in a graphical form, which is shown in Figure 5. This display consists of two parts. The top half of the display shows an overview of the relationships between source and target relations, as asserted by each of the schema mapping formulas. In the bottom half of the display, the mapping designer can zoom in on one of the schema mapping formulas. The mapping designer can explore how data values are exported from source to target according to the schema mapping, as well as the equality conditions that must hold within the source (or target) facts in order for the data migration to occur. For instance, the first schema mapping in Step 4 of Figure 2, displayed in the bottom half of Figure 5, shows where the `pid, healthplan, date` attributes from Patient and the `pid,docid` attributes from Doctor are exported to in the target. In addition, it also shows that the source Patient and Doctor tuples must have the same `pid` value, and the target History and Physician tuples must agree on the `docid` value.

Refining an existing schema mapping: Another feature of Eirene that we will demonstrate is how Eirene facilitates the refinement of an *existing* schema mapping. In this case, the mapping designer no longer inputs a set of data examples to Eirene but instead, she inputs the schema mapping that is to be refined by Eirene. Given a schema mapping, Eirene can automatically generate a set of "canonical" data examples that follows closely the structure of the schema mapping, as an initial illustration of the semantics of the schema mapping. Furthermore, if one or more "real" source instances are available, then, whenever possible, the source instances of the data examples generated by the system will make use of tuples from these real source instances.

After this, the mapping designer may modify the data examples and use Eirene to fit a schema mapping on the modified set of data examples.

It is worth mentioning that this feature allows Eirene to be used as a component of existing mapping-design systems, such as Clio, HePToX, Altova Mapforce, or Stylus Studio. The initial schema mapping that is generated by one of these systems can be passed to Eirene, where data examples can then be used to drive the process of refining the schema mapping.

3. REFERENCES

[1] B. Alexe, W. C. Tan, and Y. Velegrakis. STBenchmark: Towards a Benchmark for Mapping Systems. *PVLDB*, 1(1):230–244, 2008.

[2] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan. Designing and Refining Schema Mappings via Data Examples. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2011 (to appear).

[3] A. Bonifati, E. Q. Chang, T. Ho, V. S. Lakshmanan, and R. Pottinger. HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In *VLDB*, pages 1267–1270, 2005.

[4] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.

[5] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *ACM SIGMOD*, pages 805–810, 2005.

[6] P. G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *ACM PODS*, pages 61–75, 2005.

[7] M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM PODS*, pages 233–246, 2002.