# Mapping XML and Relational Schemas with Clio

Lucian Popa[†] Mauricio A. Hernández[†] Yannis Velegrakis[‡] Renée J. Miller[‡]
Felix Naumann[†] Howard Ho[†]

[†]IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

[‡]Dept. of Computer Science - University of Toronto
6 King's College Rd.
Toronto, ON, Canada M5S 3H5

## 1. Mapping Schemas with Clio

Merging and coalescing data from multiple and diverse sources into different data formats continues to be an important problem in modern information systems. *Schema Matching*, the process of matching elements of a source schema with elements of a target schema, and *Schema Mapping*, the process of creating a query that maps between two disparate schemas, are at the heart of data integration systems. We demonstrate *Clio*, a semi-automatic schema mapping tool developed at the IBM Almaden Research Center. In this demonstration we showcase Clio's mapping engine that allows mapping to and from relational and XML schemas, and takes advantage of data constraints in order to preserve data associations.

The semantically correct and complete creation and interpretation of mappings is a highly nontrivial process. Current tools in the market generate only trivial mappings across relational schemas or nested schemas, leaving it to the user to manually identify and specify the intricate details of a mapping, such as the generation of keys, references, join conditions, etc. To shield the user from writing complex queries or programs for every translation problem at hand, we advocate the use of a high-level schema mapping tool like *Clio*[1], where users are guided towards the specification of a high-level mapping using *value correspondences*. Informally, value correspondences specify how values for a target attribute are generated by one or more source attributes. Given this high-level mapping, Clio's mapping engine "discovers" a likely implementation of that mapping as a query (e.g., SQL, XQuery). In effect, Clio's mapping engine compiles the given high-level mapping (value correspondences) into a low-level representation (a query).

The compilation proceeds in two steps. In the *semantic translation* step, a precise and faithful understanding of the given value correspondences (the high-level mapping) is inferred. In other words, the *semantics* of the high-level mapping must be understood and converted into a precise *logical mapping*. In the *data translation* step, the logical mapping is converted into a low-level mapping (a query) that captures the data and schema transformation procedure. We explore some of the issues that must be tackled in each step by means of an example. The full details of the mapping engine algorithm can be found in [2].

**An Illustrating Scenario:** To understand the data translation implied by the given value correspondences, one must first identify *all* the different attributes that form a real-world object at the source and target. We discover such sets of attributes, which are not necessarily in the same table or nested schema, by chasing over the schema's constraints. We refer to these sets of attributes as *associations*. Associations are computed, all at once, when schemas and their constrains are loaded into the mapping tool. Value correspondences are then grouped by the associations that they affect and are interpreted as a whole, not individually. In effect, the mapping is viewed at an association-level rather than at an attribute-level. The result of this phase is a *logical mapping* that reflects the many ways the target associations can be constructed from the source associations through the given value correspondences.

Consider the two schemas shown in Figure 1. The source relational schema on the left, `expenseDB`, contains three tables: `company`, `grant`, and `project`. The nested target schema on the right, `statisticsDB`(specified either as a DTD or an XML Schema), groups information about companies and their funding by cities. In order to handle relational and XML schemas, Clio has an internal nested schema model that is expressive enough to capture both structures.

Given these two schemas, the user defines *value correspondences* from source to target. Consider, for example, the value correspondence $V_1$ in Figure 1. The meaning we associate with this (individual) correspondence is that for each company name in the source, one organization with the same name must exist in the target. In the same manner, $V_2$ indicates that for each principal investigator (`pi`) in the source, there must exist a funding of *some* organization in the target that has the same `pi`. The lines marked $r_1$, $r_2$, and $r_3$ in the figure specify foreign key constraints. According to the foreign key $r_1$, each `grant` tuple in the source is as-

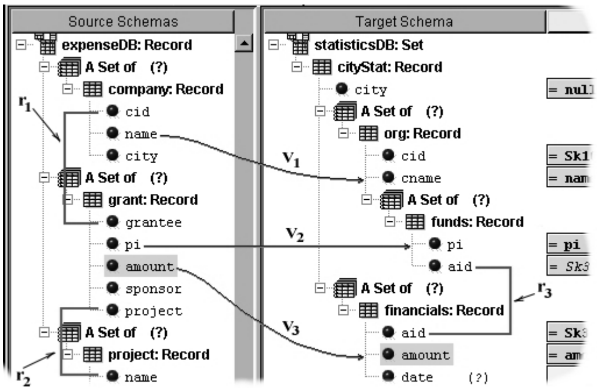

**Figure 1. A relational to XML schema mapping**

sociated (assuming non-null foreign keys) with a `company` tuple. In the target, `funds` are nested under the `orgh` elements. Therefore, a very likely interpretation of the group $\{V_1, V_2\}$ is to map, via $V_2$, each `pi` in the source to a `pi` in the target that is nested under precisely the same organization that is generated, via $V_1$, by the company associated to the source `pi`. Such semantics cannot be achieved by naively looking at $V_1$ and $V_2$ separately: a target instance built in such a way may lose the association that exists between principal investigators and companies in the source. For example, by using $V_1$, we can construct a set of organizations (with no funds) with all the company names in the source. On the other hand, by using $V_2$ (and independently of $V_1$) we can construct a set of funds (for each `pi` in the source) within some organization. But the latter organization may have nothing in common with the organizations that have been constructed in the first step. The conclusion of the above example is that in many cases value correspondences should be grouped together in order to produce a lossless mapping.

Consider now the third value correspondence $V_3$ in Figure 1. In the source, `amount` and `pi` of a grant are both in the same tuple. However, in the target, those two pieces of information are located in different elements. `pi` is in `funds` and `amount` in `financials`. The association between `amount` and `pi` of grant is achieved through the foreign key $r_3$. The value correspondences $V_2$ and $V_3$ indicate that for each `grant` tuple in the source, there must be a `financials` tuple in the target with the same `amount` and a `funds` tuple with the same `pi`. Even if $V_2$ and $V_3$ are considered together for the generation of a mapping, the association between `amount` and `pi` of a grant will be lost in the target unless the appropriate `aid` value is generated in the target `funds` and the `financials` tuples that have the `pi` and `amount` values of the source. Constraints in the target are equally important for a correct mapping generation.

The second phase of schema mapping is the data translation phase which produces the query implied by the logical mapping created by the previous phase. This query, which is currently expressed in SQL or XQuery, must satisfy a number of requirements. First, the appropriate grouping of the data retrieved from the source must be produced. This grouping is dependent on the shape of the target schema. In our example, the query generation phase should be able to create a query that: retrieves company names from the source, then, for each such company, retrieves the principal investigators (`pi`) of its grants, and nests them under the generated organization elements. Second, new values may need to be generated in the target. Recall that `aid` in the target is required in order to maintain the association between the amount of a grant and its `pi`. As it is often the case with such elements (carrying structural information but no real data), there is no correspondence to determine how that value will be generated from the source. Thus, Clio must invent a value that is the same for `aid` of `financials` and `aid` of `funds` given the same mapped source values. Clio achieves this by placing the appropriate *skolem* functions in the query. Finally, the correct unnesting and, if needed, join of source data must be expressed in the query.

## 2. The Demo

In this demo we showcase Clio's XML mapping engine algorithm and show some practical applications of the tool. This engine features full support for mappings in any direction and in any combination of relational and XML schemas. The engine takes into consideration **both** source and target data constraints when constructing the resulting query and creates missing values at the target whenever needed. Further, we illustrate the completeness and correctness of our mapping algorithm. The algorithm has the completeness guarantee that all semantic relationships between the schema attributes will be discovered by our semantic translation phase, and a correctness guarantee in that the produced query will preserve all relevant source information. Moreover, since the algorithm produces **all** possible interpretations (mappings) for the given set of correspondences, we show how Clio's GUI helps the user browse and select the appropriate interpretation. We will use the schemas shown on Figure 1 to demonstrate the basic concepts and schemas from a Life Science problem to demonstrate Clio on real-world schemas. To the best of our knowledge, no commercial tool is able to correctly generate XML transformations at the level of complexity of Clio.

## References

[1] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema Mapping as Query Discovery. In *VLDB 2000, Cairo, Egypt*, 2000.

[2] L. Popa, Y. Velegrakis, M. A. Hernández, R. J. Miller, and R. Fagin. XML Schema Mapping and Data Translation. Submitted for publication (see http://www.almaden.ibm.com/cs/clio for details), 2002.