SMART: A Tool for Semantic-Driven Creation of Complex XML Mappings

Atsuyuki Morishima Toshiaki Okawara Jun'ichi Tanaka Ken'ichi Ishikawa
University of Tsukuba
{mori, okwr, m188, m112}@slis.tsukuba.ac.jp

1. INTRODUCTION AND THE KEY IDEA

We focus on the problem of *data transformations*, i.e., how to transform data to another structure to adapt it to new application requirements or given environments. Here, we define data transformation as the process of taking as input two schemas A and B and an instance of A, and producing an instance of B. Today, data transformations are required in many situations: to integrate multiple information sources, to construct and receive data for Web services, and to migrate data from legacy systems to new systems, from local databases to data warehouses. This demonstration focuses on XML transformations, since XML is the de facto standard for data exchange.

Data transformation requires data transformation programs, which are often implemented by declarative queries. We call such queries *mapping queries*. The queries are presently being developed manually and the development requires tremendous costs, since the development is a non-trivial task that requires a deep understanding of the data. And the mapping query can be very complex. Creating mappings between independently created schemas is inherently different from the problem of mappings required in classical schema integration [5].

The SMART system aims at reducing the development cost of mapping queries by providing a tool for semi-automatic generation of mapping queries. Although many tools have been proposed related to the mapping creation [3][5][6][7][8], they share the common key idea that the mapping creation process consists of the following two phases[1]: (1) Receiving from the user (or semi-automatically finding) correspondences between (values or components of) the *schemas* to be mapped, and (2) Generating mapping queries consistent with the given correspondences. In general, finding correspondences between complex schemas is not easy, because this requires deep understanding of the schemas. We argue and demonstrate with our SMART system that introducing *reverse engineering techniques* to the query creation can dramatically facilitate the creation process. Reverse engineering in our context is a process to recover conceptual schemas from given database/XML schemas [1]. This approach makes the SMART system unique and we believe that the system has a practical impact.

[1]Some of such tools allow the phases to be interleaved for the incremental creation of mapping queries.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland, USA.
Copyright 2005 ACM 1-59593-060-4/05/06 $5.00.

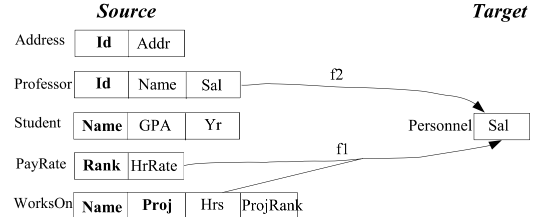

Figure 1: Mapping specification for Clio

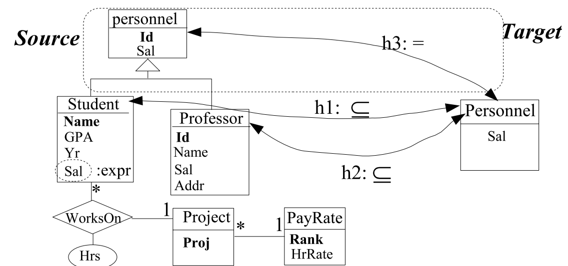

Figure 2: Mapping specification for SMART

This idea is best understood by comparing Figure 1 with Figure 2. Figure 1 is taken from [5] to explain the well-known Clio system, which is a semi-automatic tool to generate mapping queries. The figure shows example schemas to be mapped (The example uses relational schemas, but the same discussion applies to XML data.). The target schema defines a unary relation to contain salaries of both professors and students. To create mappings, the user gives *value correspondences* (denoted by $f1$ and $f2$ here) to Clio as clues to find queries implementing intended mappings. A value correspondence consists of (1) a function defining how a value (or combination of values) from a source database can be used to form a value in the target, and (2) a filter, indicating which source values should be used (i.e., the selection condition). In Figure 1, $f1$ and $f2$ are value correspondences: $f1 : PayRate(HrRate) \times WorksOn(Hrs) \rightarrow Personnel(Sal)$ and $f2 : Professor(Sal) \rightarrow Personnel(Sal)$ where we assume their filters are "true" (meaning, all values should be used). Clio enumerates possible queries consistent with the given value correspondences, and ranks the queries to find likely queries.

We can make the following observations from the example:

Observation 1. Finding appropriate value correspondences is not easy, because the user must understand the source schema. Professor salaries may be easy to find, but it is difficult to know $PayRate(HrRate) \times WorksOn(Hrs)$ computes those

of students until foreign key constraints on attributes of relations $Student$, $PayRate$, and $WorksOn$ are found. If the source schema is large, it has a huge number of attributes. So the job becomes more difficult.

Observation 2. It is possible that some supporting tools suggest possible or plausible value correspondences by using techniques for schema matching [9], but the search space for finding matching attributes from the source (with n attributes) and the target (with m attributes) is $2^n \times m$. In addition, it is unlikely that automatic schema matching techniques find value correspondences like $f1$.

Observation 3. Verifying whether the value correspondences and the (semi-automatically generated) query are correct is not easy, because verification also requires a deep understanding of the source schema (and the application domain). Clio has a mechanism to give example values so that the user can check if the created query is correct. Although this may allow the user to evaluate queries without examining query details, the same problem of being required to understand the source data remains.

Observation 4. In value correspondences, there is a tight connection between the inter-schema *relationships* and the *computation expressions* (functions and filters). This close connection makes it difficult to write and understand value correspondences. For example, the value correspondence $f1$ says the following two things at once: (1) *Relationship:* a salary of a student in the source is used as a value for $Personnel(Sal)$ in the target, and (2) *Computation:* each salary is computed by $PayRate(HrRate) \times WorksOn(Hrs)$. Also, the connection forces the user to write the same computation many times (as shown later).

Figure 2 shows how the user specifies the same mapping with our SMART system. First, SMART takes as inputs the schemas to be mapped, reverse-engineers the schemas, and shows the user the resulting conceptual schemas (outside the dotted boxes in Figure 2). The user can modify the conceptual schemas or, otherwise, give relationships (in Figure 2, $h1$ and $h2$) between classes from two schemas, annotated with inclusion labels (e.g., $\subseteq$). Next, SMART uses the inputs to automatically generate and add new information (inside the two dotted boxes) to conceptual schemas. In the example, SMART first finds that there should be a super class of `Student` and `Professor` that is equivalent to the target's `Personnel` class (which is represented by the relationship $h3$), according to the given relationships $h1$ and $h2$. It also finds that `Student` class should have a Sal attribute, after trying to match attributes of the source's `Student` class with those of the target's `Personnel` class. Note that reverse engineering makes the search space for matching attributes small, since only attributes of the related classes can be candidates. SMART then requests the user to specify how to compute the student's salary, and the user specifies that it is computed by $expr : Student.Sal = sum(this.WorksOn.[Hrs \times Project.PayRate.HrRate])$. Note that the specification is easy to understand because it only specifies computation of the student's salary, ignoring any relationship with the target schema. If necessary, the user can always modify the class diagrams during the process. Finally, the SMART system outputs a query that is consistent with the user's specification.

Comparing the two figures, the advantages of introducting reverse engineering frameworks to the mapping creation are clear. We enumerate and contrast them with the above observations: First, the introduction of the explicit reverse engineering into the creation of mappings helps the user to understand schemas and to specify relationships between the schemas to be mapped. In Clio, the user should understand schemas implicitly. In SMART, the relationships can be given as those between *classes*, not attributes, which we believe are easier to find. The relationships can be used as hints for SMART to find matching attributes. Second, the reverse engineering process makes the search space for matching attributes smaller, and gives chances to use domain ontologies to match attributes since the conceptual schemas are closer to such ontologies compared to database (or XML) schemas. More important, opportunities arise for SMART to find *missing attributes* of classes (like Sal of `Student`), which is otherwise impossible. Third, verifying correspondences by the user is much easier since conceptual schemas are semantically richer. And finally, separation of computations from relationships between schema components, along with the reverse engineering process, has the following advantages: (1) The user can use the *is-a hierarchy* to reduce the number of required computation expressions. Let's assume that the user wants salaries whose values are more than 2000. In Clio, the computation (filter) must be *distributed* to every value correspondence; the user must add the filter > 2000 to each of correspondences $f1$ and $f2$. In SMART, the user can specify that a subclass of `Personnel` of the source schema having restriction $Sal > 2000$ is equivalent to the `Personnel` of the target. (2) The user can specify each computation only in the source schema world, *without worrying about the relationship with the target schema* (Please remember the computation specification of Sal of `Student` for SMART).

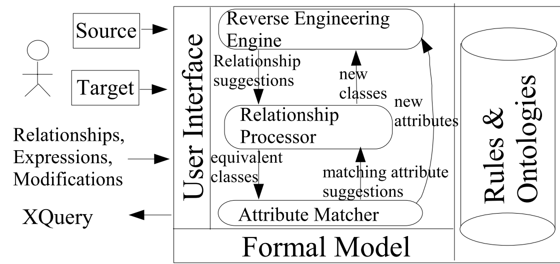

Figure 3: Architecture

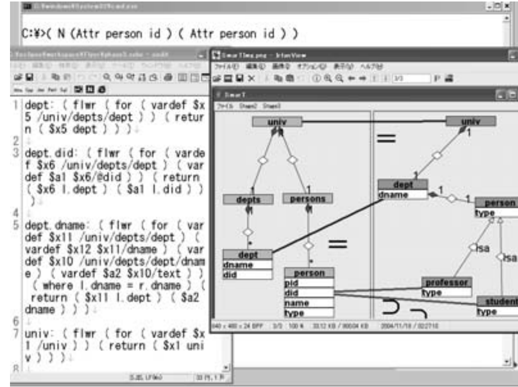

Figure 4: Prototype System

2. ARCHITECTURE

Figure 3 shows the architecture of the SMART system (Figure 4 is a screen shot of the prototype system.). The SMART system first takes as input the source and target schemas and activates the reverse engineering engine to produce conceptual schemas for the two given schemas. Next, the system interacts with the user to get instructions. After relationships between two schema components are given, the relationship processors and the attribute matcher are activated. These modules interact with each other by exchanging useful information, including equivalent classes, newly-found

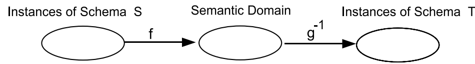

Figure 5: Mapping through Semantic Domain

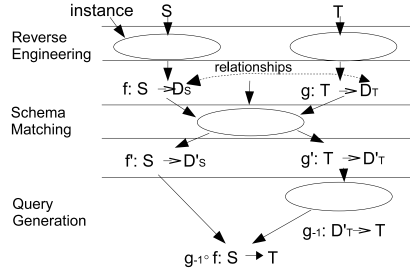

Figure 6: Three Stages

classes and attributes, and suggestions on likely relationships and matching attributes. Finally, SMART outputs XQuery query to transform instances of the source schema into those of the target schema. The system components are based on, and/or communicate with each other through, a formal model we have defined. Also, SMART utilizes a repository that stores rules and ontologies to be used by these components.

3. FORMAL MODEL FOR THE CREATION OF XML MAPPING QUERIES

An important contribution of our project is that we defined a formal model as a language to describe and discuss the development process of XML mapping queries. We model a mapping query as a mathematical mapping $F : S \to T$, which maps instances of an XML schema S to instances of T. And we define manipulations of such mappings for incremental construction of the final mapping queries. The presence of a formal framework not only gives us the formal basis, but works as an interface among various software components of our system. In our model, we manipulate mappings, not data. Mappings are dealt with as *first class citizens*.

Semantic Mappings. To incorporate the concept of reverse engineering into the problem of mapping creation, our model introduces *semantic mappings* as key components and realizes data transformations through semantic domains. A semantic domain is a set of instances of a conceptual schema (e.g., Figure 2(Source)) of data. Let S and T be two XML schemas. In our model, a mapping $F : S \to T$ is expressed as a composition of two semantic mappings $g^{-1} \circ f$ (Figure 5), where $f : S \to D_S$ and $g : T \to D_T$, if semantic domains D_S and D_T are compatible with each other, i.e., if D_S has classes and attributes equivalent to all of those in D_T. If they are not compatible, the conflict must be resolved before the composition.

4. DEMONSTRATION

Our method has the following three stages to generate mapping queries: the reverse engineering stage, the schema matching stage, and the query generation stage (Figure 6). The inputs of the system are two schemas S and T, and an instance of S. Relationships between the target and source conceptual schemas are given in the schema matching stage. As suggested in Section 1, the schema matching stage and the reverse engineering stage can be interleaved and affect each other, and the two stages are repeated until the SMART system obtains classes and attributes (in the source conceptual schema) equivalent to all of those in the target schema.

In our demonstration, the following issues are explained and demonstrated.

(1) Two Different Schemas. We show two different XML schemas having implicit constraints and structures, whose correspondence is not straightforward.

(2) Reverse Engineering Stage. Given the two XML schemas, the system tries to recover their conceptual schemas. Note that it is impossible to recover *one right* conceptual schema for each XML schema, because (a) there is more than one interpretation of schemas, and (b) schema definitions do not necesarily have all the constraints and semantic information. Therefore, the user is allowed to modify the recovered conceptual schemas if needed, by applying operators of the data transformation model or using higher-level functions provided by the system.

(3) Schema Matching Stage. This is an interesting stage. Points include: (1) By specifying relationships between classes first, the number of possible combinations of attributes becomes smaller, which helps both the user and the tool itself to find appropriate matching between schema attributes. (2) A description logic engine [2][4] is embedded and used to help the user to specify class relationships. This makes it possible for the SMART system to infer equivalence relationships from the incomplete information on the relationships (i.e., inclusion relationships) given by the user.

(4) Query Generation Stage. We show how the SMART system generates mapping queries based on the inputs. In general, the relationships given by the user, on their own, are insufficient to identify the intended query. We let the system ask the user to supply missing expressions, instead of enumerating possible queries based on the incomplete information. We believe our approach is more desirable than the enumeration approach in practice, since it is difficult for the latter approach to guarantee that the intended query is included in the set of enumerated queries.

(5) Other issues. We explain the model and the architecture in more detail, and would like to discuss related issues, including automatic verification of mappings and applications of domain ontologies.

5. ACKNOWLEDGEMENTS

We would like to thank Natsuko Furukawa and Tadatoshi Sekiguchi for their contributions to the early stages of this project. We would also like to thank Prof. Shigeo Sugimoto for his kind support. This research was partially supported by the Ministry of Education, Culture, Sports, Science, and Technology, Grant-in-Aid for Young Scientists (B) (15700108).

6. REFERENCES

[1] P. Aiken. Data Reverse Engineering Staying the Legacy Dragon, McGraw-Hill, Inc., New York, 1996.
[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (Eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press 2003.
[3] A. Doan, P. Domingos, A. Halevy: Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. SIGMOD Conference 2001
[4] Volker Haarslev, Ralf Möller: Description of the RACER System and its Applications. International Workshop on Description Logics (DL-2001).
[5] R. J. Miller, L. M. Haas, M. A. Hernandez: Schema Mapping as Query Discovery. VLDB 2000: 77-88
[6] A. Morishima, H. Kitagawa, A. Matsumoto: A Machine Learning Approach to Rapid Development of XML Mapping Queries. ICDE 2004: 276-287
[7] T. Milo, S. Zohar: Using Schema Matching to Simplify Heterogeneous Data Translation. VLDB 1998: 122-133
[8] L. Popa, Y. Velegrakis, R. Miller, M. Hernandez, R. Fagin: Translating Web Data. VLDB 2002: 598-609
[9] E. Rahm, P. A. Bernstein: A survey of approaches to automatic schema matching. VLDB J. 10(4): 334-350,2001.