A Support Tool for XML Schema Matching and Its Implementation

Toshiaki Okawara Jun'ichi Tanaka Atsuyuki Morishima Shigeo Sugimoto
University of Tsukuba, 1–2 Kasuga, Tsukuba, Japan
{okwr,m188,mori,sugimoto}@slis.tsukuba.ac.jp

Abstract

We are developing a software tool to support schema matching for data transformations. Schema matching is the process of finding relationships between components of two given database schemas. The tool is unique in that it first extracts conceptual schemas from the two database schemas and allows the user to use the extracted conceptual schemas as clues to perform schema matching. This paper overviews the tool and explains its implementation.

1 Introduction

We focus on the problem of data transformations whose importance has increased in recent years. In the context of databases, a *data transformation* is represented by an expression $I_B = F(I_A)$, where I_A is an instance of schema S_A of the source database, and I_B is an instance of schema S_B of the target database. We encounter data transformations in many situations, such as in migrating data from the source to the target database, and in publishing data from databases in the given XML format.

In such situations, function F is often implemented as a query against the source database. Since the implementation of such a function is labor-intensive and has a high overhead in cost, many systems to support data transformations have been developed. In general, it is impossible to develop F from information on schemas S_A and S_B only. Therefore, existing systems often take as input *relationships* between schema components. The process to find relationships between components of two given schemas is called *schema matching* [10].

Figure 1 illustrates an example. Assume that we have source schema A, which is the schema of the database used at a university [1]. Schema B is the target schema designed to have information on salaries (Sal) of professors (Prof) and students. We'd like to match the two schemas A and B.

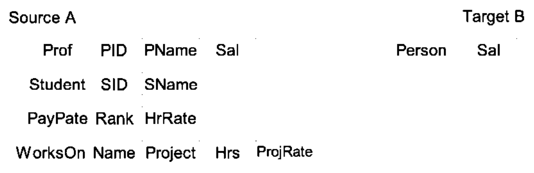


Figure 1. Two Database Schemas

Actually, it is not easy to match the schema for the following reasons. (1) It requires deep understanding of the schema structure. In this example, the Sal attribute of schema B must match attributes for salaries of professors and students in schema A. It is easy to identify the Sal of professors in the source schema, but it is difficult for the user to find the salary of students consisting of attributes $HrRate$ and Hrs until he knows there are foreign key constraints among the relations Student, PayRate, and WorksOn. (2) When the number of attributes increases, the process of schema matching becomes much more difficult, since possible combinations of attributes significantly increase.

We are developing a system called *SMART* to support data transformations [2] [7] [9]. This paper overviews the schema matching functions of SMART and explains their implementation.

2 Schema Matching in SMART

A feature of schema matching functions in SMART is that matching is not done in terms of schema attributes. Rather, matching is done between components of the conceptual schemas, which are extracted from the source and target schemas in advance. Figure 2 illustrates the matching process in SMART for the database schemas given in Figure 1. The matching process is done in the following way:

[1]To keep things simple, they are not XML schemas. The same discussion applies to XML schemas.

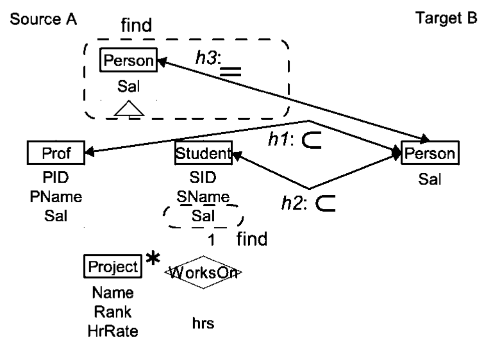


Figure 2. A Schema Matching Process

(1) The system first extracts the conceptual schemas from the given database (XML) schemas, and shows the user the conceptual schemas (outside the dashed box in Figure 2). Each conceptual schema is expressed as a class diagram.

(2) The user gives relationships between classes of the shown conceptual schemas ($h1$ and $h2$ in Figure 2). Each relationship has a label. In this example, the user uses "$\subset$" as a label of $h1$ and $h2$ since the instances of the Person class includes the instances of the Prof and Student classes. The relationships between classes with "=" as a label are called *equivalence relationships.*

(3) The system makes an inference from these relationships, and finds information inside the dashed box in Figure 2. In this example, the system finds the following: (3a) There should be a superclass of Prof and Student in the source conceptual schema, which corresponds to the Person class in the target. This is inferred from $h1$ and $h2$. (3b) The Student class in the source should have attribute Sal. This is inferred because the system can find no attribute in Student corresponding to the Person class in the target.

(4) The system requests the user to specify an expression to compute values of the new attribute Sal (in this example, Student.Sal=sum(this.WorksOn.[Hrs×Project.HrRate]) is given).

(5) Based on interaction with the user, the system computes and outputs an XQuery query to implement data transformation F.

Our approach has the following advantages, compared to other approaches that perform direct schema matching between database schemas. (1) The system helps the user to understand the data by extracting the conceptual schemas. We believe matching between classes is easier than matching between database attributes because we can expect the conceptual schemas of the source and the target to be similar in our data transformation applications. (2) Giving relationships between classes in advance reduces search space when finding relationships between attributes.

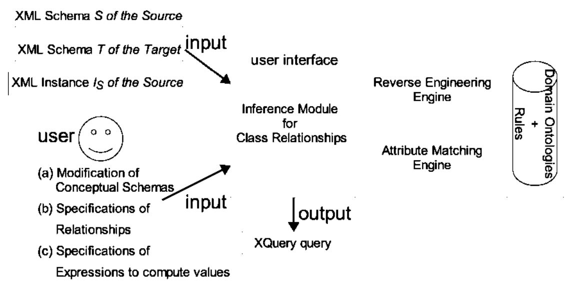

XML Schema S of the Source
XML Schema T of the Target
input
XML Instance I_S of the Source
user interface
Inference Module for Class Relationships
Reverse Engineering Engine
Attribute Matching Engine
Domain Ontologies + Rules
user
(a) Modification of Conceptual Schemas
(b) Specifications of Relationships
(c) Specifications of Expressions to compute values
input
output
XQuery query

Figure 3. Architecture

2.1 Related Work

Cupid [6] is a system that automatically performs schema matching between two schemas. It adopts an algorithm that combines methods for matching based on the attribute names and matching based on the schema structure. The algorithm used in Cupid can be used for our system, so we think it can be used to complement our tool.

The extraction of conceptual schemas from database schemas is known as *data reverse engineering* [1], and there are many algorithms and tools, including DB-MAIN [4].

3 Design and Implementation

Figure 3 illustrates the SMART architecture. The tool takes as input XML schema S of the source, XML schema T of the target, and XML instance I_S of S. It then outputs an XQuery query to compute $I_T = F(I_S)$. The tool has three main components:

Reverse Engineering Engine: The reverse engineering engine extracts the conceptual schema from two input XML schemas S and T.

We assume that this component incorporates existing data reverse engineering techniques for the conceptual schema extraction. In general, however, results of applying data reverse engineering techniques to database schemas are not perfect, because all integrity constraints are not necessarily coded in the database schemas. Even if some of the constraints can be inferred from the database instances, it is not possible to get the constraints from only a limited number of instances. This component, therefore, is designed to allow the user to modify the result conceptual schemas through prepared operations (Figure 3 (a)).

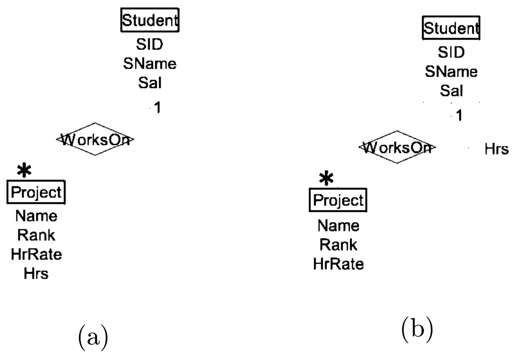
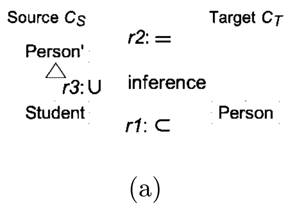


Figure 4. Modification Support for Conceptual Schemas

For example, this component provides an automatic modification function based on key restrictions (Figure 4). Let's assume that a part of the result of reverse engineering the given source schema in Figure 1 is the one shown in Figure 4 (a). In this example, Hrs is an attribute of the Project class, but its value actually represents how long a student works on a project. In other words, the values of hours depend on the combination of a student and a project. In our system, the user specifies that the key to the Project class is $Name$. The system then examines the instances of Project and checks if the functional dependencies $Name \rightarrow A_i$ hold where A_i is each attribute of Project. If a $Name \rightarrow A_i$ does not hold, A_i is moved to the right position, which is not inconsistent with the key constraint. In this example, attribute Hrs is moved to the attribute of the relationship (WorksOn) between the Student and Project classes because $Name \rightarrow Hrs$ and $SID \rightarrow Hrs$ do not hold but $Name, SID \rightarrow Hrs$ holds (Figure 4 (b)).

Inference Module for Class Relationships: The user gives to this component relationships (with any label) between classes of the source and target conceptual schemas (Figure 3 (b)). The system then computes and finds equivalence relationships (with the label"=") based on the given relationships. In some cases, it may find new classes like the Person class in the source in Figure 2. This process is done using the RACER inference engine [5] for the coding logic [3], which is an applied logic for relationships between classes.

Internally, the component transforms the given relationships into the description logic expressions and submits them to RACER. This transformation is done in the following way (See Figure 5 for the explanation).



$(r1)$ Student $\sqsubseteq$ Person
$(r2)$ Person' $\equiv$ Person
$\downarrow$ Inference
$(r3)$ Student $\sqsubseteq$ Person'

(b)

Figure 5. (a) Relationships between Classes and (b) Corresponding Expressions in the Description Logic

1. Conceptual schema C_S of source XML schema S, conceptual schema C_T of target XML schema T, and the relationships between their classes are transformed into description logic expressions ($r1$ and $r2$ in Figure 5).

2. The description logic expressions are submitted to RACER, and the inference result is obtained ($r3$ in Figure 5 (b)).

3. The inference result is reflected to conceptual schema C_S of the source ($r3$ in Figure 5 (a)).

Attribute Matching Engine: The attribute matching engine takes as input two classes, $C1$ and $C2$, connected by the equivalence relationship, and outputs relationships between attributes of $C1$ and $C2$.

The equivalence relationships may be given by the user directly, or can be the results of the relationship inferences produced by the inference module explained above. Existing techniques for schema matching [10] can be used to help the process of matching between attributes. Also, the user can modify the results after the matching results are given by the system. If necessary, the system requires the user to enter expressions to compute attribute values, as explained in (4) of Section 2 (Figure 3 (c)).

The prototype system we developed is written in Java and consists of 30 classes and about 10.000 lines of code. The database schemas are given in RALAX NG [8]. Figure 6 is a screenshot of this prototype system. Here, relationships between classes are entered by the user and the system makes inferences from them.

4 Conclusion

This paper overviewed the schema matching support in our SMART data transformation tool and explained its implementation. A feature of the tool is that matching is performed at the level of conceptual schemas. Future work includes the development of support mechanisms to cope with huge schemas having a large number of components.

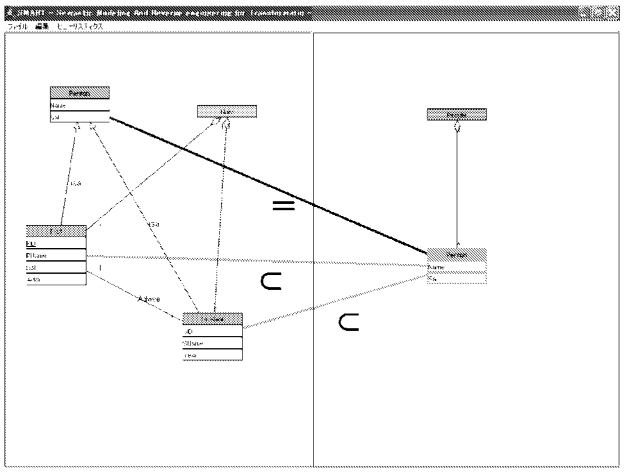

SMART – Semantic Modeling And Reverse engineering for Transformation –
ファイル 編集 ヒューリスティクス
Person
Name
Tel
Person
Name
Tel
=
⊂
⊂
Adviser

Figure 6. A Screenshot

Acknowledgments

We would like to thank Koichi Tabata, Tetsuo Sakaguchi, and Mitsuharu Nagamori for discussions. This research was partially supported by the Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Young Scientists (B) (15700108).

References

[1] P. Aiken. Data Reverse Engineering Staying the Legacy Dragon, McGraw-Hill, Inc., New York, 1996.

[2] Natsuko Furukawa, Tadatoshi Kamimura, Toshiaki Okawara, Atsuyuki Morishima, Shigeo Sugimoto: Implementation of a Prototype System to Extract Semantic Information from XML Data. DBSJ Letters, Vol.3, No.1, June 2004: 129-132.

[3] Benjamin N. Grosof, Lan Horrocks, Raphael Volz, Stefan Decker: Description Logic Programs: Combining Logic Programs with Description Logic. WWW 2003: 48-57.

[4] Jean-Luc Hainaut: Research in Database Engineering at the University of Namur. SIGMOD Record Vol.32, No.4, December 2003: 124-128.

[5] Volker Haarslev, Ralf Möller: Description of the RACER System and its Applications. International Workshop on Description Logics (DL-2001).

[6] Jayant Madhavan, Philip A. Bernstein, Erhard Rahm: Generic Schema Matching with Cupid. VLDB 2001: 49-58

[7] Atsuyuki Morishima, Toshiaki Okawara, Jun'ichi Tanaka, Ken'ichi Ishikawa: SMART: A Tool for Semantic-Driven Creation of Complex XML Mappings. SIGMOD Conference 2005 (DEMO), June 2005. (to appear)

[8] OASIS. RELAX NG Specification. http://www.oasis-open.org/committees/relax-ng/spec.html.

[9] Toshiaki Okawara, Atsuyuki Morishima, Sigeo Sugimoto: Extraction of Conceptual Models from XML Data by User-Defined Rules. DBSJ Letters, Vol.3, No.2, September 2004: 53-56.

[10] Erhard Rahm, Philip A. Bernstein: A survey of approaches to automatic schema matching. VLDB J. 10(4): 334-350, 2001.