MVT: A Schema Mapping Validation Tool

Guillem Rull[1,2] Carles Farré[2] Ernest Teniente[2] Toni Urpí[2]
Universitat Politècnica de Catalunya
1-3 Jordi Girona, Barcelona 08034, Catalonia, Spain
{grull | farre | teniente | urpi}@lsi.upc.edu

ABSTRACT

Schema mappings define relationships between schemas in a declarative way. We demonstrate MVT, a mapping validation tool that allows the designer to ask whether the mapping has certain desirable properties. The answers to these questions will provide information on whether the mapping adequately matches the intended needs and requirements. MVT is able to deal with a highly expressive class of mappings and database schemas, which allows the use of negations, order comparisons and null values. The tool does not only provide a Boolean answer as test result, but also a feedback for that result. Depending on the tested property and on the test result, the provided feedback can be in the form of example schema instances, or in the form of an explanation, that is, highlighting the mapping assertions and schema constraints responsible for getting such a result.

1. INTRODUCTION

A schema mapping is a declarative specification that models a relationship between two schemas. Defining a mapping is a key task in contexts such as information integration or data exchange. A lot of research has focused on finding correspondences between schemas [9], and on generating mapping candidates from these correspondences [8]. Nevertheless, generating a mapping is a semi-automatic process, which always requires feedback from a human designer to solve semantic heterogeneities, choose between different mapping candidates, and refine the mapping [3]. The designer thus needs to check whether the mapping produced is in fact what was intended, that is, he must find a way to validate the mapping.

Motivated by that, we present MVT, a prototype tool that implements the mapping validation approach we presented in [11]. MVT allows the designer to ask whether the mapping has certain desirable properties. The answers to these questions will provide information on whether the mapping adequately matches the intended needs and requirements.

As an example, consider the following database schema S1:

```
CREATE TABLE Category (
  name   char(20) PRIMARY KEY,
  salary real     NOT NULL,
  CHECK (salary >= 700), CHECK (salary <= 2000) )

CREATE TABLE Employee (
  name     char(30) PRIMARY KEY,
  category char(20) NOT NULL,
  address  char(50),
  CHECK (category <> 'exec'),
  FOREIGN KEY(category) REFERENCES Category(name))

CREATE TABLE WorksFor (
  emp  char(30) PRIMARY KEY,
  boss char(30) NOT NULL,
  CHECK(emp <> boss),
  FOREIGN KEY (emp)  REFERENCES Employee(name),
  FOREIGN KEY (boss) REFERENCES Employee(name) )
```

the following database schema S2:

```
CREATE TABLE Persons (
  id      int      PRIMARY KEY,
  name    char(30) NOT NULL,
  address char(50) )

CREATE TABLE Emps (
  empId  int  PRIMARY KEY,
  salary real NOT NULL,
  boss   int,
  CHECK (salary BETWEEN 1000 AND 5000),
  CHECK (empId <> boss),
  FOREIGN KEY (empId) REFERENCES Persons(id),
  FOREIGN KEY (boss)  REFERENCES Emps(empId) )
```

and the following mapping assertions between S1 and S2:

```
 MAPPING ASSERTION m1
(SELECT e.name, c.salary
 FROM employee e, category c
 WHERE e.category = c.name and c.salary >= 10000)
   SUBSET OF
(SELECT p.name, e.salary
 FROM persons p, emps e
 WHERE p.id = e.empId)

 MAPPING ASSERTION m2
(SELECT wf.emp, wf.boss
 FROM worksFor wf, employee e, category c
 WHERE wf.emp = e.name and e.category = c.name
       and c.salary >= 1000)
   SUBSET OF
(SELECT pEmp.name, pBoss.name
 FROM emps e, persons pEmp, persons pBoss
 WHERE e.empId = pEmp.id and e.boss = pBoss.id)
```

The mapping defined by these two assertions states that the *employees* of S1 that have a salary above a certain threshold are a subset of the *emps* of S2. Assertion *m1* captures information of employees that may or may not have a boss, while assertion *m2* takes care of specific information of employees that have a boss.

EDBT'09, March 24-26, 2009, Saint Petersburg, Russia.

[1] This work was supported in part by Microsoft Research through the European PhD Scholarship Programme.
[2] This work has been partly supported by the Ministerio de Ciencia e Innovación under project TIN2008-03863.

Mapped schemas S1 and S2 are themselves correct in the sense that their constraints are not contradictory. However, when the mapping is considered, it turns out that assertion *m1* can only be satisfied trivially. That is, if we want to satisfy *m1* without violating the constraints in S1, the first query of *m1* must get an empty answer (recall that the empty set is a subset of any set).

MVT allows the designer to detect that problem by means of running a mapping satisfiability test. Moreover, it highlights the schema constraints and mapping assertions responsible for the problem. In this case, the problem is in the interaction between *m1* and the constraint `CHECK(salary <= 2000)` from S1. That *explanation* may help the designer to realize that *m1* was probably miswritten, and that it should be mapping those employees with a salary above one thousand, instead of ten thousand.

In this paper, we demonstrate how MVT can be used to test the desirable properties of mappings considered in [11]: mapping satisfiability (strong and weak), mapping inference and query answerability (firstly identified in [7]), and mapping losslessness.

MVT considers a class of schemas and mappings defined by means of a subset of the SQL language. Namely:

- Primary key, foreign key, Boolean check constraints.
- SPJ views, negation, subselects (exists, in), union, outer joins (left, right, full).
- Data types: integer, real, string.
- Null values.

Mapping assertions are in the form of Q_1 *op* Q_2, where Q_1 and Q_2 are queries over the mapped schemas, and *op* is $=$, $\subseteq$ or $\supseteq$.

We also demonstrate that MVT always provides some *feedback*, in addition to the Boolean answer to whether the tested desirable property holds or not. Depending on the tested property and on the test result, the feedback can be in the form of example instances of the mapped schemas, or in the form of an *explanation* [10, 12]. An explanation is a set of schema constraints and mapping assertions that is responsible for the test result (e.g., the explanation *{mapping assertion m1, constraint* `CHECK(salary <= 2000)` *from S1}* provided as feedback for the satisfiability test in the example above). Providing this feedback is important since it may make easier to the designer to identify problems and fix them, especially when the schemas and the mapping are large.

Contributions. The main contribution of this work is that MVT is, to the best of our knowledge, the first implemented tool able to check this kind of properties in the context of mappings. Moreover, it does not only implement the validation approach of [11], but also integrates it with the work about computing explanations for failed database schema validation tests of [10, 12]. Implementing the extended validation method we presented in [12] allows MVT to compute one approximated explanation in the case in which example schema instances are not a suitable feedback for the test result. The explanation is approximated in the sense that it may be not minimal. Implementing the black-box method of [10] allows MVT to offer the designer the possibility of refining this approximated explanation into an exact one, and compute also all the additional possible explanations. Finally, MVT incorporates the treatment of null values, which was not considered in [10, 11, 12]. Allowing null values in the schema instances is significant since a single validation test may have a certain result when nulls are not allowed, and a different result when they are.

2. DEMONSTRATION OVERVIEW

We illustrate the use of MVT using the example scenario introduced in the previous section. Along the demonstration we modify the mapping according to the results of the different validation tests. The schemas and the mapping are kept small for the sake of clarity.

Testing mapping satisfiability. We begin the demonstration with the satisfiability test we already discussed in the introduction. Just say here that we define a mapping to be strongly (weakly) satisfiable if there exists a pair of instances of the mapped schemas that satisfy all (at least one) mapping assertion(s) in a non-trivial way. A trivial case would be when the queries in the mapping have an empty answer. Let us assume that we decide to fix *m1* as indicated in the previous section. We load the updated mapping into MVT, perform the satisfiability test again, and show that now *m1* is satisfiable. This time, the feedback the tool provides consists in one instance of each mapped schema that indeed satisfies both *m1* and *m2* non-trivially (we omit it here).

Testing mapping assertion redundancy. The next test we demonstrate uses the mapping inference property [7] to detect redundant assertions in the mapping. An assertion is inferred from a mapping if all pairs of schema instances that satisfy the mapping also satisfy the assertion. Based on that, a mapping assertion is redundant if it can be inferred from the other assertions in the mapping (taking into account the mapped schema constraints). Therefore, the expected feedback for a mapping assertion that is redundant is the set of schema constraints and other mapping assertions the tested assertion is inferred from. If the tested assertion is not redundant, it is better to illustrate that by means of providing a pair of mapped schema instances that satisfy all mapping assertions except the tested one.

To illustrate this test, let us assume that we have come up with an alternative mapping, more compact that the one we already had. It consists in the following single assertion:

```
 MAPPING ASSERTION m3
(SELECT e.name, c.salary, wf.boss
 FROM employee e LEFT OUTER JOIN worksFor wf
         ON wf.emp = e.name, category c
 WHERE e.category = c.name and c.salary >= 1000)
   SUBSET OF
(SELECT pEmp.name, e.salary, pBoss.name
 FROM emps e LEFT OUTER JOIN persons pBoss
         ON e.boss = pBoss.id, persons pEmp
 WHERE e.empId = pEmp.id)
```

The main difference with respect to *m1* and *m2* is that *m3* uses left outer join to capture both the employees with and without boss at the same time. Now, we may want to know how this assertion relates with the other two. Therefore, we load the schemas and the three assertions into MVT, and run the assertion redundancy test. We get the following results. Assertions *m1* and *m2* are both redundant, with explanations as feedback. The explanation for *m1* is {*m3*}. The one for *m2* is {*m3*, `WorksFor.boss NOT NULL`}. However, *m3* is not redundant, and the feedback provided by MVT is the following pair of schema instances:

Instance of S1:

```
Category('execA', 1000)
Employee('A', 'execA', null)
Employee('AA','execA', null)
WorksFor('A', 'AA')
```

Instance of S2:

```
Persons(0, 'A',  null)
Persons(1, 'A',  null)
Persons(2, 'AA', null)
Emps(0, 1000, null)
Emps(1, 2000, 2)
Emps(2, 1000, null)
```

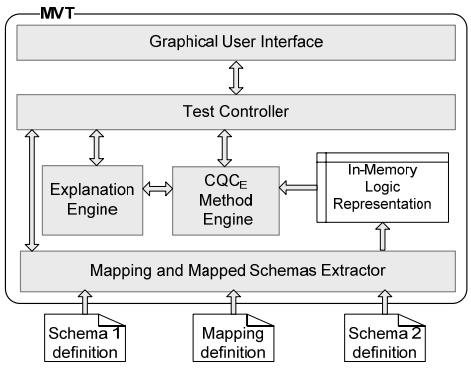

Figure 1: Architecture of MVT.

These schema instances show that *m3* is not only more compact but also more accurate. Assertions *m1* and *m2* allow a single *employee* from S1 to be mapped to two *persons* with different *ids*. Assertion *m3* prevents that by means of the outer join (other formalisms allow expressing this kind of correlations by means of Skolem functions [8]).

Testing mapping losslessness. A mapping is said to be lossless [11] with respect to a given query if the information needed to answer that query is captured by the mapping. More formally, the mapping $\{V_1\ op\ W_1, \ldots, V_n\ op\ W_n\}$ is lossless w.r.t. query Q defined over S1 (S2) if Q is determined by the extension of the V_i (W_i) queries (these query extensions must satisfy the mapping assertions). The purpose of this property is to allow the designer to test whether a mapping that may be partial or incomplete is enough for the intended purpose.

When a mapping turns out to be lossy, MVT provides a counterexample as feedback (see below). When the mapping is indeed lossless, the provided feedback is the explanation (schema constraints and mapping assertions) that prevents such a counterexample from being constructed.

We illustrate the property with the following example. Let us assume that after replacing the mapping {*m1*, *m2*} with {*m3*} we want to know whether the names and addresses of all employees with a salary of at least 1000 are mapped. We perform a mapping losslessness test with the following query as parameter:

```
SELECT e.name, e.address
FROM employee e, category c
WHERE e.category = c.name and c.salary >= 1000
```

The result of the test indicates that the mapping is not lossless with respect to that query, and provides the following schema instances as feedback:

Instance 1 of S1:
```
Category('execA', 1000)
Employee('A', 'execA', null)
```
Instance 2 of S1:
```
Category('execA', 1000)
Employee('A', 'execA', 'A')
```
Instance of S2:
```
Persons(0, 'A', null)
Emps(0, 1000, null)
```

The above counterexample shows two instances of S1 that differ in the address of the employee, but are mapped to the same instance of S2, and have the same extension for the queries in the mapping. Seeing this, the designer can realize that the address of the employees is not captured by the mapping. This result does not mean necessarily that the current mapping is wrong. That depends on the intended semantics. For example, if the address of the employees in S1 was considered classified for some reason, then a lossy mapping would be what the designer wanted. Let us assume that this is not the case, and that the designer decides to modify m3 in order to capture the addresses. Then, it suffices adding `e.address` and `pEmp.address`, respectively, to the select clauses of the queries in *m3*.

MVT also allows testing the property of query answerability (identified in [7] together with mapping inference). We omit its discussion here due to space reasons.

3. SYSTEM ARCHITECTURE

MVT extends our database schema validation tool [13] to the context of mappings. The architecture of MVT is depicted in Figure 1.

The GUI component allows using MVT in an easy and intuitive way. To perform the different available tests, users go along the following interaction pattern:

1. Load the mapping and the mapped schemas.
2. Select one of the available validation tests.
3. Enter the test parameters (if required).
4. Execute the test.
5. Obtain the test result and its feedback, which can be in the form of example schema instances, or in the form of highlighting the schema constraints and mapping assertions responsible for the test result.

The Test Controller processes the commands and data provided by users through the GUI, and transfers back the obtained results.

The Mapping and Mapped Schemas Extractor is responsible of translating the loaded mapping and mapped schemas into a format that is tractable by the CQC$_E$ Method Engine. In this way, it generates an in-memory representation where both the mapping and the schemas are integrated into a single logic database schema that is expressed in terms of deductive rules.

According to the approach we presented in [11], the Test Controller and the Mapping and Mapped Schemas Extractor work together to reformulate the problem of validating the selected mapping property in terms of the problem of testing whether a query is satisfiable over a database schema. The resulting query satisfiability test is performed by the CQC$_E$ Method Engine.

The CQC$_E$ Method Engine implements the CQC$_E$ method [12], an extended version of the CQC method [6]. The original CQC method can be used to check whether a certain query is satisfiable over a given database schema. It provides an example database instance when the query is indeed satisfiable. However, it does not provide any kind of explanation for why the tested query is not satisfiable. Other validation methods do not provide an explanation for this case either. The CQC$_E$ method addressed this issue. It extends the CQC method so this is able to provide an approximated explanation for the unsatisfiability of the tested query. The provided explanation is the subset of constraints that prevented the method from finding a solution. It is approximated in the sense that it may be not minimal.

The Text Controller may ask the Explanation Engine to check whether the explanation provided by the CQC$_E$ Method Engine is minimal, and to find the other possible minimal explanations (if

any). In order to do that, the Explanation Engine implements the black-box method presented in [10].

The feedback is translated back to the original SQL representation by the Test Controller and the Mapping and Mapped Schemas Extractor, and shown to the user through the GUI. If the CQC_E Method Engine provides a database instance, and since this instance corresponds to the integrated schema that resulted from the problem reformulation, it has to be translated in terms of the original mapped schemas. Similarly, if the feedback is an explanation (a set of constraints) and since these constraints belong to the integrated schema, they have to be translated in terms of the original mapped schema constraints and mapping assertions.

The whole MVT tool has been implemented in the C# language using Microsoft Visual Studio as a development tool. Our implementation can be executed in any system that features the .NET 2.0 framework. Some screenshots are shown in Figure 2.

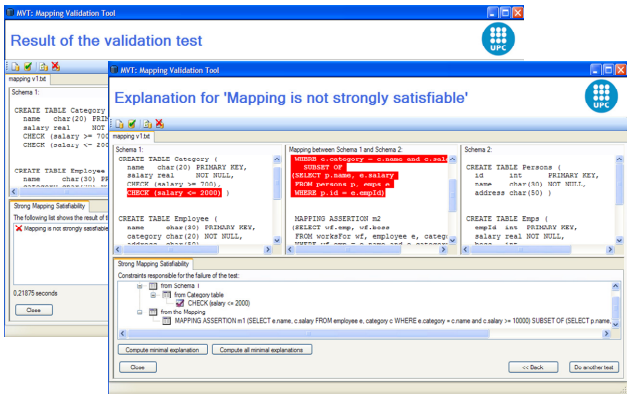

Figure 2: Screenshots of MVT.

4. RELATED WORK

Recently, other tools related with the mapping validation problem have been presented [2, 4]. The main difference of our tool with respect to them is that they need schema instances in order to perform the validation. Our tool only requires the mapping and mapped schemas definitions to be provided, and it is therefore able to reason over the mapping itself rather than relying on specific instances that may not reveal all the potential pitfalls.

The SPIDER tool demonstrated in [2] is a mapping debugger for source-to-target tuple-generating dependencies mappings, based on the computation of *routes* [5]. Basically, the user can select a set of target tuples, and see how this tuples were obtained from the source instance through the mapping. Since routes are intended to allow the user to explore and understand a given schema mapping, this work can be seen as complementary to ours. As we demonstrate here, our tool sometimes provides schema instances as feedback for a certain validation test. Therefore, routes could be used to help the designer to understand this feedback, and to make easier the detection and fixing of the problems.

The Spicy system [4] is aimed at helping the designer to choose among the different candidate mappings (tuple-generating dependencies) the ones that represent better transformations of the source into the target. Source and target schema instances are required. Each candidate mapping is executed over the source instance in such a way that a new instance for the target schema is obtained, and this new instance is compared with the available target instance. At the end, the user gets a ranked list of mappings, suggesting which ones are believed to better reproduce the target. At this point, the validation information provided by our tool, combined with the similarity measure attached by Spicy, might help the designer to choose and refine the final mapping.

Another tool that has appeared recently is Muse [1], a mapping design wizard that assists designers in understanding and refining schema mappings. In particular, it guides the designer on the choice among alternative mapping definitions by constructing synthetic examples that illustrate the differences among them. However, the construction of such examples is not aimed at revealing potential mapping-definition flaws such as redundancies and information loses. In this sense, our tool complements Muse by enabling the designer to validate and refine the chosen mapping definitions. Although our current version of the tool does not deal with nested mappings and nested relational schemas as Muse does, we are already working to include such features in a next release. Instead, our tool does handle mapping definitions featuring negations and order comparisons, which Muse does not consider.

5. REFERENCES

[1] B. Alexe, L. Chiticariu, R. J. Miller, D. Pepper, W.-C. Tan: Muse: a system for understanding and designing mappings. In SIGMOD Conference, pp. 1281-1284, 2008.

[2] B. Alexe, L. Chiticariu, W.-C. Tan: SPIDER: a Schema mapPIng DEbuggeR. In VLDB, pp. 1179-1182, 2006.

[3] P. A. Bernstein, L. Haas: Information integration in the enterprise. Commun. ACM 51(9), pp. 72-79, 2008.

[4] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, G. Summa: The Spicy system: towards a notion of mapping quality. In SIGMOD Conference, pp. 1289-1294, 2008.

[5] L. Chiticariu, W.-C. Tan: Debugging Schema Mappings with Routes. In VLDB, pp. 79-90, 2006.

[6] C. Farré, E. Teniente, T. Urpí: Checking query containment with the CQC method. Data Knowl. Eng. 53(2), pp. 163-223, 2005.

[7] J. Madhavan, P. A. Bernstein, P. Domingos, A. Y. Halevy: Representing and Reasoning about Mappings between Domain Models. In AAAI/IAAI, pp. 80-86, 2002.

[8] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, R. Fagin: Translating Web Data. In VLDB, pp. 598-609, 2002.

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

[10] G. Rull, C. Farré, E. Teniente, T. Urpí: Computing explanations for unlively queries in databases. In CIKM, pp. 955-958, 2007.

[11] G. Rull, C. Farré, E. Teniente, T. Urpí: Validation of mappings between schemas. Data Knowl. Eng. 66(3), pp. 414-437, 2008.

[12] G. Rull, C. Farré, E. Teniente, T. Urpí: Providing Explanations for Database Schema Validation. In DEXA, pp. 660-667, 2008.

[13] E. Teniente, C. Farré, T. Urpí, C. Beltrán, D. Gañán: SVT: Schema Validation Tool for Microsoft SQL-Server. In VLDB, pp. 1349-1352, 2004.