A Classification of Schema Mappings and Analysis of Mapping Tools

Frank Legler
IBM Deutschland Entwicklung GmbH
flegler@de.ibm.com

Felix Naumann
Hasso-Plattner-Institut, Potsdam
naumann@hpi.uni-potsdam.de

Abstract: Schema mapping techniques for data exchange have become popular and useful tools both in research and industry. A schema mapping relates a source schema with a target schema via correspondences, which are specified by a domain expert possibly supported by automated schema matching algorithms. The set of correspondences, i.e., the mapping, is interpreted as a data transformation usually expressed as a query. These queries transform data from the source schema to conform to the target schema. They can be used to materialize data at the target or used as views in a virtually integrated system.

We present a classification of mapping situations that can occur when mapping between two relational or nested (XML) schemata. Our classification takes into consideration 1:1 and n:m correspondences, attribute-level and higher-level mappings, and special constructs, such as choice constraints, cardinality constraints, and data types. Based on this classification, we have developed a general suite of schemata, data, and correspondences to test the ability of tools to cope with the different mapping situations. We evaluated several commercial and research tools that support the definition of schema mappings and interpret this mapping as a data transformation. We found that no tool performs well in all mapping situations and that many tools produce incorrect data transformations. The test suite can serve as a benchmark for future improvements and developments of schema mapping tools.

1 Schema Mappings, Data Exchange, and Mapping Tools

The problem of information integration, i.e., enabling access to multiple distributed, autonomous, and heterogeneous data sources through a common interface (common schema, common query language), is eminent in database research and information systems development. Apart from technical challenges the problem of schematic heterogeneity is particularly obtrusive. Even when using a common data model there are many different ways of modeling the same real world entities and relationships. In this article we focus on the technique of *schema mapping* to describe the relationships between two schemata in the context of data transformation. Schema mappings and their use for data exchange has recently become a popular notion as exemplified in [AL05, FKP05, MBHR05] and many other projects and tools. Schema mappings have a wide array of further usages, such as schema translation [SL90] and schema integration [BLN86], not considered in this article.

Regard the simple example of Fig. 1. It shows two nested schemata, both modeling persons and their membership in teams. In the first schema, the membership of a person to a

team is modeled as a foreign key constraint, in the second, the membership is modeled by nesting person elements under team elements. Also observe that different facts are represented in the different schemata. The first schema models the address of a person while the second schema does not. And vice versa, the second schema models the person's date of birth (DOB) while the first does not. Even when elements with the same meaning are included, their labels need not coincide. For instance, teamURL and website have the same semantics but different labels.

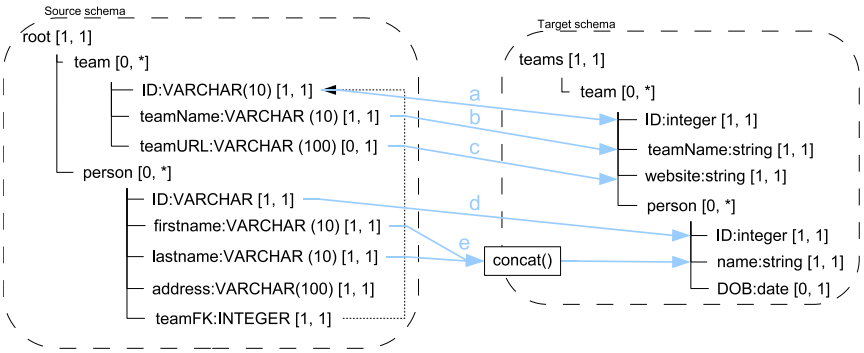


Figure 1: A mapping between two nested schemata

Given these and many more heterogeneities among schemata, schema mapping describes the general technique of relating elements between two heterogeneous schemata. The relationships are based on some semantic similarity of the elements and are usually called correspondences; graphically, correspondences are modeled as arrows from an element in a source schema to an element in a target schema; conceptually, a correspondence states that data for the target element can be obtained by fetching the data stored at the corresponding source element. Formally, a correspondence is a relation between a set of elements in the source schema and a set of elements in the target schema. This relation is identified by a transformation function that transforms source data into target data or a filter that selects elements of the source schema [Leg05]. This definition also allows m:n correspondences. A schema mapping is a set of correspondences between a source schema and a target schema. Schema mappings are used to transform data stored under the source schema so that it conforms to the target schema. In a typical situation, the source schema might be that of a data source and the target schema is the federated schema of an integrated system. In a data exchange scenario the two schemata are those of peers willing to exchange data.

Figure 1 shows a schema mapping that relates attributes of the left-hand schema, the source schema, with the right-hand schema, the target schema. This situation raises two important questions that have been extensively dealt with in recent literature and in recent products: (i) How can one obtain the correspondences between two schemata and (ii) how can one interpret a set of correspondences to actually transform source data so that it conforms to the target schema. In this article we focus on the second question and assume that the correspondences have been established.

Informally, the interpretation of a schema mapping faces several difficulties:

Source schema interpretation: Associations of data elements (relations, nesting, foreign keys) in the source schema should be recognized and preserved during transformation.

Target schema conformance: The result of a transformation should conform to the target schema.

User intention: The expert users merely provide informal correspondences ("arrows") between the schemata. Even under the demand that an interpretation of such a mapping should cover all correspondences there usually remain several alternatives. Correctly guessing the intended one can rely only on suitable heuristics.

In this article we classify these difficulties and evaluate several schema mapping tools in their ability to overcome them. We observe that surprisingly not even the first two difficulties are adequately addressed in many of the tools. We limit the scope of our evaluation to the relational and XML data models, because of their widespread usage and the fact that they are supported by most schema mapping tools. Hence, we also limited the number of query languages that are used to transform data from the source to the target. For simplicity in this article we restrict ourselves to a graphical notation for schemata and mappings. Even though we only consider the relational and XML data model in this article, the basis for our research is a schema definition language that is independent of the data model. Both the relational data model and the XML data model can be transferred into this schema definition language. The schema definition language, the transformation of the relational and the XML model into the schema definition language, and the formal definition of each mapping situation can be found in [Leg05].

2 A Classification of Mapping Situations

Figure 2 shows our classification of schema mapping situations, distinguishing three main classes: mapping situations related to *missing correspondences* displayed in the left subtree, mapping situations related to *single correspondences* in the middle, and mapping situations belonging to *multiple correspondences* on the right. The following sections introduce each of the classes and outline possible ways for mapping tools to interpret them.

A classification similar to ours was presented by Kim et al., who classify conflicts between relational schemata based on their structure [KS91]. We used this classification as a basis for ours and extended it to include non-relational features. Previous research in schematic heterogeneity yielded several classifications of correspondences and mapping conflicts. Batini et al., for example, distinguish four types of semantic relationships: identical, equivalent, compatible, and incompatible [BLN86]. Every non-identical relationship implies a conflict. Their classification requires the existence of semantic knowledge about the participating schemata. This requirement means that users know exactly which real world object is modeled by which schema element. Our classification, on the other hand, just utilizes the structural information given by the schemata and the mapping between them.

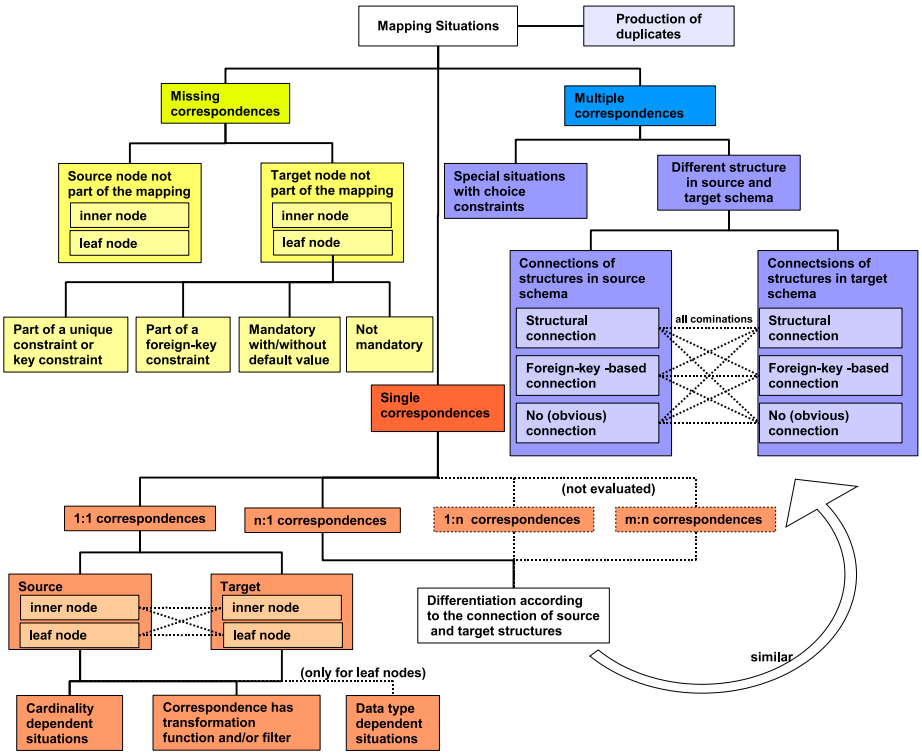


Figure 2: Classification of schema mapping situations

Spaccapietra et al. classify conflicts between schemata into semantic conflicts, descriptive conflicts, heterogeneity conflicts, and structural conflicts [SPD92]. Because our classification assumes a given schema mapping, we assume both semantic conflicts and descriptive conflicts as resolved. The two remaining classes contain the conflicts that are interesting for our purpose, but were not sufficiently analyzed.

2.1 Missing correspondences

This class contains mapping situations that occur if leaf nodes or inner nodes of the schema are not part of the schema mapping, i.e., no correspondence is connected to the nodes.

If leaf nodes of the source schema are not part of the mapping, the only important consideration is a loss of information, meaning that the source data cannot be recreated from the target data after the transformation. On the contrary, if leaf nodes of the target schema are left out of the mapping, constraints of the target schema could be violated:

- If the node is part of a **key or unique constraint** (not null) a mapping tool could automatically generate a value, as Popa et al. demonstrate [PVM+02]. Another possible solution is to reject the mapping and ask the user for manual resolution.
- If the node is part of a **foreign-key constraint**, a tool could automatically detect a connection between the key and the foreign-key using the information from the source schema. If that is not possible, a manual resolution is necessary.
- If the node is **mandatory** and a default value is given, a tool should use this information. Otherwise, a manual resolution would be the best option. A random value is also an acceptable but not optimal solution. The fact that the value is required implies that it is relevant for the application; inserting a random value contradicts this intuition.
- If the node is **not mandatory**, a tool can neglect it, preferably with some warning.

Whether a missing correspondence to and from inner nodes of a schema tree influences the produced transformation query or not depends on the definitions of correspondences and mappings. Problems occur depending on whether a tool allows and interprets correspondences between inner nodes or not (see later).

2.2 Single correspondences

Mapping situations related to single correspondences are outlined in this section. The classification is based on the properties of correspondences along different dimensions:

1. Cardinality of the correspondence. According to the number of the participating schema elements in a correspondence, 1:1, 1:n, n:1, and n:m correspondences can be identified. Only 1:1 correspondences are further analyzed in this section. The remaining three types are analogous to problems at inner node level: the question on how to combine multiple input values and how to produce multiple output values are similar to the questions discussed for multiple correspondences (see Sec. 2.3).

2. Type of the participating schema elements. Elements associated with a correspondence can either be leaf nodes or inner nodes.

- **C**orrespondences between **leaf nodes** are the most common kind of correspondences and are supported by every schema mapping tool. Their purpose is to define which source data to transform into which target data.
- Correspondences between **inner nodes** are not supported by all mapping tools. If they are supported, then for every element in the source instance an element in the target instance could be created.
- When connecting **inner nodes with leaf nodes**, the data and metadata levels are mixed. If no additional transformation function is given, a mapping tool should reject this correspondence. Figure 3 shows an example of metadata in the source schema corresponding to data in the target schema.

3. Properties of the participating schema elements and constraints. Correspondences can also be classified according to the properties that participating schema elements have (see bottom left of Fig. 2).

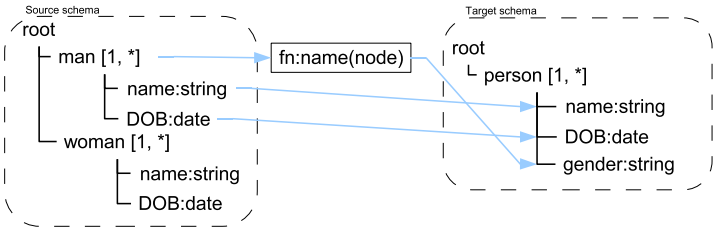


Figure 3: Data vs. metadata

There are several **cardinality-related situations**, which can be classified as situations with a *loss of information* (when source data cannot be transferred into the target schema due to a higher cardinality in the source schema) and situations with a *lack of information* (when the source instance does not include enough information to build a valid target schema instance due to a lower cardinality in the source schema). Both situations should be recognized by a mapping tool and treated accordingly. For example, if in a 1:1 correspondence the source node is nullable and the target node is mandatory with no default value, a mapping tool should ask for a manual resolution for cases where the source value is null.

Conversion tables or cast tables define how **data types** can be transformed into each other (also spanning different data models). They can be used to resolve mapping situations related to different data types. According to these tables there are three classes of compatibility: *compatible*, *partly compatible* (depending on the concrete value), and *incompatible data types*. For compatible data types, mapping tools could insert casts according to these conversion tables, and warn users if data types are not or only partly compatible.

2.3 Multiple correspondences

This class comprises mapping situations related to multiple correspondences. The categories in this class are based on the structure of the source and target schema rather than on the properties of individual elements.

Research projects addressing this topic developed sophisticated algorithms to discover clusters of semantically connected schema elements (e.g., [MHH00, PVM+02]). To be able to better comprehend and compare the tests and results, a simplified approach was chosen. According to this approach, three different kinds of relationships (associations) between elements of a schema can be distinguished:

- Elements have a **structural connection** if they have at least one common ancestor, whose cardinality is greater than one. A mapping tool should recognize *structurally connected elements in a source schema* and maintain their semantic relationship. This can be done by transforming structurally connected elements simultaneously, e.g., leaf nodes with the same parent node. Furthermore, a mapping tool should be able to combine nested elements by un-nesting them where appropriate.

  Regarding *structurally connected elements in the target schema*, a tool has the

choice whether to group the data or not. This question also leads to the problem whether an aggregation of the data is possible. Figure 4 shows an example where a grouping of families under the same lastname and an aggregation of the family-income is desirable.

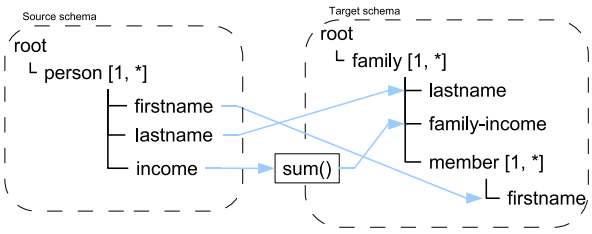


Figure 4: Grouping and aggregation

- A **foreign-key based connection** between elements is given if the elements reside in subtrees, that are inter connected with a foreign-key relationship. This class also includes explicitly defined joins between subtrees, which can be captured with some mapping tools. A mapping tool should recognize *source schema elements, that are connected by a foreign-key* and maintain their semantic relationship. Hence, a transformation query should contain a (outer-)join over the connected subtrees. For *target schema elements with a foreign-key connection* two situations can be distinguished: Either the key and foreign-key elements are part of the mapping or they are not part of the mapping. In case they are part of the mapping, the user already provides the information on how to establish a semantic relationship between the target values. If they are not part of the mapping, a schema mapping tool could automatically create key/foreign-key pairs for related elements. This can be realized with skolem-functions (see [PVM+02]).
- Nodes that are connected neither structurally nor foreign-key-based are **connectionless nodes**. Connectionless in this context means that they can be arbitrarily combined in a transformation from the source to the target. Connectionless nodes could, for example, be combined with a Cartesian product, an outer join, or listed in the order of their appearance in the instance of the source schema.

Additionally, we classified mapping situations, that result from **choice constraints**. A choice constraint specifies that in an instance a node might have only one child node from the set of nodes listed in the schema. A schema mapping tool should treat *source schema nodes with a choice constraint* similarly to nodes that are not mandatory. Unless all child nodes of a source node with a choice constraint are associated with the same target schema node, it cannot be guaranteed that a value is transferred into the target. The problem is then a lack of information, similar to the cardinality-related situations where the cardinality of the element in the source schema is lower than the cardinality of the element in the target schema. A schema mapping tool should recognize these situations and either reject the mapping or inform the user about the problem.

For a *target schema node with a choice constraint* it must be guaranteed that exactly one child node is produced. Accordingly, a mapping tool should recognize situations, where no

child node or more than one child node could be created and either reject those situations or warn the user.

A characteristic that is independent of a certain mapping situation, is the **production of duplicates**. A tool can either produce duplicate values or produce *distinct values*. Even though both solutions are correct, it would be desirable to give the user the choice, whether a tool should produce duplicates or not. For brevity, we foreclose the results of this aspect of schema mapping: All tools, except for one, produce per default **duplicate values**. Only two tools give the user the choice, whether to produce duplicates or not. One of them provides a special operator, whereas the other one requires the manual insertion of script.

3 Schema Mapping Tools

In this section we briefly describe the selection of six schema mapping tools from research and industry that we have evaluated. All tools feature a graphical user interface displaying source and target schema as a tree and allow to draw lines between elements of the schemata. The evaluation with respect to their capability to interpret schema mappings in the next section is anonymized, so this section serves only as a pointer to different tools and a comparison of their general capabilities.

BizTalk Mapper 2004 is part of Microsoft's BizTalk Server to manage business processes (`http://www.microsoft.com/biztalk/`). Messages and data from different sources are converted to XML and the BizTalk mapper allows to specify transformations on this data. In addition to the pure mapping functionality, BizTalk Mapper offers a large library of "functoids" to transform data values. The mappings are interpreted as XSLT scripts.

Clio is not a commercially available tool, but a research prototype developed at IBM's Almaden Research Center (`http://www.almaden.ibm.com/software/km/clio/`). It was one of the first and most sophisticated tools motivating many theoretical and practical research results. The product version of Clio is now part of Rational Data Architect.

MapForce 2005 is part of Altova's XML suite of tools to help users develop XML applications, which also includes XMLSpy (`http://www.altova.com/products/mapforce/`). Like Clio, MapForce is a tool developed solely for the purpose of schema mapping and deriving transformation queries.

Oracle Warehouse Builder 10*g* Release 1 is a tool to develop data warehouses based on the Oracle 10*g* database system (`http://www.oracle.com/technology/products/warehouse/`). Part of this development is the ETL process (Extract, Transform, Load), which in turn includes a schema mapping step, among many others. We discuss this tool as a representative for many ETL tools offered by database and data warehouse vendors and companies specializing in ETL.

Stylus Studio 6 is an XML integrated development environment by Progress Software specializing on XQuery / XSLT creation and visualization (`www.stylusstudio.com`). With its XQuery Mapper developers can create and visualize mappings between XML schemata, which are in turn interpreted as XQueries or XSLT transformation queries.

WebSphere Studio Application Developer Integration Edition is IBM's integrated development environment including a collection of tools to develop Web Services, Portals, etc. (`http://www.ibm.com/software/integration/wsadie/`). One feature is a tool to map between XML Schemata, namely the XML-to-XML Mapping Editor.

Schema mapping is a research and development field of growing importance. Apart from the tools described here, there are several others including most ETL tools, the schema management prototype Rondo [MRB03], the SAP Exchange Infrastructure, etc. We also note that the tools are also still under development or are research prototypes that do not aim at full coverage of all possible situations. We have noted the versions we used and are aware of the fact that many of the observed problems are in fact bugs and are likely to be corrected in later versions. However, other problems reach deeper into the semantics of mapping and there is no easy or obvious "fix". We believe that the set of experiments described in the next section are well-suited not only to document the state-of-the-art, but also to track progress in their development.

4 Evaluation

This section presents a set of tests to evaluate a schema mapping tools capability to cope with thementioned situations. Section 4.1 introduces the overall evaluation procedure and criteria. Sections 4.2 – 4.4 then present a brief overview of the evaluation results. The evaluation is anonymized for legal reasons. In Sec. 4.2 schema mappings and evaluation results are exemplarily shown in more detail for situations where target leaf nodes are not part of the mapping. The schema mappings and detailed evaluation results for all other mapping situations are listed in [LN06].

4.1 Evaluation procedure

To avoid side effects and to increase understandability, all mapping situations are tested in an as isolated as possible way. I.e., we have taken care that the situation under scrutiny is the *only* difficult situation in the schemata. To this end we developed 43 small schemata and mappings between them. All tools were exposed to these situations, if the situation was applicable to the features they support. For instance, the Oracle Warehouse Builder supports only relational data, so tests regarding nesting were not performed for that tool.

For each tool and each situation we loaded the two schemata, manually entered the appropriate correspondences, let the tool produce a transformation query, evaluated the query, and regarded the query result. To increase the comparability between the different mapping tools, all but the Oracle Warehouse Builder were tested using XML. The query language of the transformation query depends on the features of the tools. Most tools support more than one query language to express a transformation query. Wherever available we used XQuery.

We evaluate the results of the different tools with "+" for a satisfactory outcome, "o" for

an acceptable outcome, and "–" for an insufficient or incorrect outcome. Depending on the individual tests the scores were given based on different criteria: In general we favored results where little user-intervention was necessary. Tools that use all available schema and mapping information and generate a correct outcome are marked with a "+". In many situations tools unnecessarily ask for user input, for instance asking how to join two elements despite an existing foreign-key relationship. In such cases, and when tools generate syntactically correct but semantically questionable results, we noted a "o". Finally, incorrect results, i.e., transformed data that does not conform to the target schema, were marked with "–".

4.2 Missing Correspondences

This class was tested with two sets of mappings: the first leaving out leaf nodes of the source schema (Fig. 5a) and the second leaving out leaf nodes of the target schema (Fig. 5b-5d). For the second set we present the full set of results as an example of the level of detail and the types of outcomes. Missing inner nodes were evaluated as part of the tests in Section 4.3. This section summarizes the results.

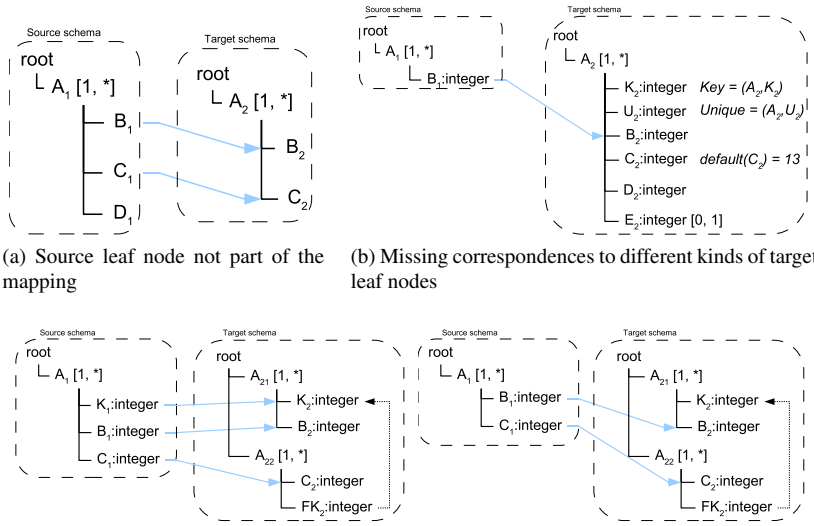
1. Leaf node of the source schema is not part of the mapping. As expected, all tools (correctly) ignore such leaf nodes if those nodes do not participate in the mapping.

2. Leaf node of the target schema is not part of the mapping. Such mapping situations were tested using three different schema mappings as shown in Fig. 5. The mapping in Fig. 5b was used to test mapping situations with key constraints, unique constraints, default values, and target leaf nodes that are not mandatory. The two other mappings in Fig. 5c and 5d were used to test mapping situations, where target leaf nodes with a foreign-key constraint are not part of any correspondence. The evaluation results are listed in Tab. 1. To summarize, only a single tool (Tool 1) performs well in all situations. This tool follows a passive conflict solving strategy by giving warnings to the user for almost all critical situations.

A second tool (Tool 2), on the other hand, follows an active conflict solving strategy by automatically solving the issues. However, its solutions are not always perfect. In particular, an available default value was ignored (see Row 4a in Tab. 1). The remaining four tools ignore the target schema attributes and create transformation queries that produce data not conforming to the target schema. They either do not produce the node at all (Tools 3 and 4) or produce it with a NULL value (Tools 5 and 6).

Inner node is not part of the mapping. Whether problems arise if inner nodes of a schema are not part of the mapping, mainly depends on the way a tool defines correspondences and interprets them.

Two of the tools do not allow inner nodes to be part of the mapping at all. Whereas three tools *require* correspondences between inner nodes in order to transform all instances of repeatable elements. If repeatable inner nodes are not connected, these tools transform only the first instance of this node from the source to the target.



(a) Source leaf node not part of the mapping

(b) Missing correspondences to different kinds of target leaf nodes

(c) Key is given

(d) Neither key nor foreign-key are given

Figure 5: Leaf nodes not part of the mapping

4.3 Single correspondences

For the remainder of the evaluation we only briefly describe experiments and results here. The full evaluation results are listed in [LN06].

1:1 correspondences. The tests in this category are not exhaustive, because the number of all possible combinations between the different properties of a schema element is very high. Therefore we chose to test cardinality-related situations with correspondences only between leaf nodes and include correspondences between inner nodes only when required by the tool to properly generate results. The same was done with data type-related situations, because only leaf nodes can contain data. Furthermore, transformation functions were included only when testing data type-related mapping situations.

1. **Type of participating schema elements.** Of course, all tools support *correspondences between leaf nodes* of the source schema and the target schema. This is not true for *correspondences between inner nodes* of the participating schemata. As mentioned earlier, three tools require these kinds of correspondences in order to produce the correct result. The other three tools do not require them (two do not even allow them). The interpretations of correspondences between inner nodes in turn are very varied: One tool warns the user, if no child-element of the inner node is part of the mapping. Another tool forces the user to connect the child-leaf nodes in this case. Two tools produce subtrees without content in the target schema and the last tool just produces an instance of the inner node, without adhering to the target schema.

   The connection of data level and metadata level via *correspondences between inner*

Situation	Tool 1	Tool 2	Tool 3	Tool 4	Tool 5	Tool 6
1 (Key)	(+) warning	(+) produced with skolem-function	(–) no node produced	(–) no node produced	(–) node with NULL value	(–) empty node
2 (UNIQUE)	(+) warning	(+) produced with skolem-function	(+) no node produced	(+) no node produced	(+) node with NULL value	(+) empty node
3 (Foreign-key)	(+) warning	(+) produced with skolem-function	(–) no node produced	(–) no node produced	(–) node with NULL value	(–) empty node
4a (Mandatory with default value)	(+) used default value	(–) produced with skolem-function	(–) no node produced	(–) no node produced	(–) node with NULL value	(–) empty node
4b (Mandatory without default value)	(+) warning	(o) produced with skolem-function	(–) no node produced	(–) no node produced	(–) node with NULL value	(–) empty node
5 (Not mandatory)	(+) no node produced	(+) no node produced	(+) no node produced	(+) no node produced	(+) node with NULL value	(–) empty node

Table 1: Target leaf node not part of the mapping

nodes and leaf nodes or vice versa is not supported by two schema mapping tools. All other tools are able to model and interpret the example shown in Fig. 3.

2. **Cardinality of the participating schema elements.** All tested tools have severe problems with cardinality-related mapping situations. Surprisingly, two schema mapping tools are not even aware of cardinalities. They produce the same transformation query regardless of possible conflicts. The other four tools recognize only few of the conflict situations.

3. **Data type of the participating schema elements.** The evaluation reveals that three schema mapping tools are wholly unaware of data types. Hence, they do not recognize problems with incompatible and partly compatible data types. The tools that are aware of data types in turn do recognize most conflict situations.

1:n, n:1, n:m correspondences. The evaluation was limited to n:1 correspondences, because none of the tools supports more than one target schema element as part of a single correspondence. For n:1 correspondences the main question is, how to combine the input values of the correspondence. The interpretations provided for these situations are similar to the interpretations given for multiple correspondences. They depend on the kind of connection between the participating source schema elements, which can be distinguished in structural connection, foreign-key-based connection and no connection (see Sec. 2.3).

Only one tool provides an algorithm that automatically detects foreign-key relationships. With another tool the user is forced to manually join the related subtrees. Three tools ignore the relationship and the last tool at least warns the user.

An automatic un-nesting of relationships between elements, that are structurally connected is performed by three tools where necessary. The other tools allow to manually un-nest these subtrees. For example, this can be done by connecting inner nodes in the source

schema with inner nodes in the target schema. This way some tools are forced to generate for each node in the source instance a respective node in the target instance.

Not all tools support n:1 correspondences in which the participating source schema nodes are *not obviously connected* (see sample mapping in Fig. 6). Additionally, the tools that allow these correspondences do not provide satisfying solutions. They either produce non-target schema compliant data or do not transfer all source data into the target data.

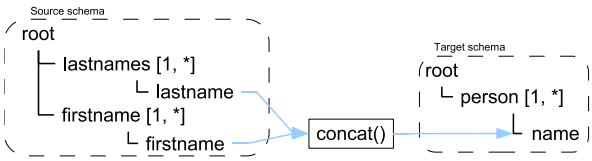


Figure 6: A 2:1 correspondence with not obviously connected source schema nodes

4.4 Multiple correspondences

For mapping situations with multiple correspondences we distinguish structurally connected, foreign-key-based connected, and connectionless elements in source and target schema. For reasons of simplification we do not evaluate situations, where these kinds of connection between nodes are nested into each other. Nevertheless, even with these simple situations no schema mapping tool performs well.

The evaluation shows, that the combination of input values for multiple correspondences is almost identical to the results for n:1 correspondences (with a transformation function). Therefore this section regards only the combination of output values.

1. **Structurally connected target schema elements.** Only one tool gives the user the choice to decide whether to group the target data or not. One tool automatically groups the target data and three tools do not group the data, thus losing associations among the data values. The aggregation shown in Fig. 4 (Page 7) could not be modeled by any of the tools.
2. **Foreign-key-based connected target schema elements.** If the target nodes are connected with a foreign-key relationship and both the key and the foreign-key are not mapped, only one tool automatically creates the key/foreign-key pair. Another tool warns the user, that some information is missing to create valid target data. All other tools ignore the relationship and create data that is not compliant with the target schema.
3. **Connectionless target schema elements.** All tools treat connectionless target schema elements independently and transfer all source data into each of the target schema elements (according to the correspondences).
4. **Choice constraint.** Only two of the five mapping tools that support XML are aware of the choice content model defined in XML Schema. The other tools create the same transformation query for the choice model and the sequence model and therefore do not recognize possible issues.

5 Discussion and Outlook

In this article we presented two main contributions: A *classification* of schema mapping situations for relational and nested schemata and a *comparison* of a set of current schema mapping tools from research and industry using a *benchmark-suite* of schemata and correspondences. While this is only a snapshot of the state of the art of schema mapping and new versions of tools have already appeared during our study, the contributions have a lasting character and can be used as reference for future schema mapping tool development and evaluation.

At this point it is worthwhile to discuss the broad range between users creating a mapping by editing correspondences and the applications "intelligently" guessing what kind of mapping the user might intend. This trade-off is best exemplified in a mapping from two relations to a single relation in the target. Some tools can guess and suggest to join the two source tables using a known foreign key constraint between the two. Other tools might deliberately refrain from such interference and let only users explicitly state joins between tables. Therefore, the question is how much intelligence and automation is needed in a schema mapping tools.

In general, tools provide three methods for automation and thus to ease the burden of a user: First, schema matching techniques can suggest correspondences between source and target schema. Despite much research in the field of schema matching, the automatically detected correspondences can merely be suggestions to be confirmed or rejected by the user. Second, tools interpret a set of correspondences as a mapping making decisions whether to join two elements, whether to use inner or outer joins, how to group elements etc. I.e., the tools help *specify* the mapping in more detail, again allowing users to interfere and change this specification. Third, the tools automatically compile this specification into a transformation query in some query language. This compilation is always fully automatic – users can only manually adapt the resulting query if necessary.

We believe that for scenarios with very large schemata or with very many schemata a high degree of automation is indispensable. This automation must be combined with a high degree of flexibility, i.e., users must be able to correct and adapt the mapping at any point and time. With a schema mapping debugger recent work already points in this direction [CT06]. In this article we have highlighted the necessary abilities of tools to enable this degree of automation and have pointed out the current limitations.

Acknowledgments. This research was supported in part by the German Research Society (DFG grant no. NA 432).

References

[AL05] Marcelo Arenas and Leonid Libkin. XML data exchange: consistency and query answering. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 13–24, Baltimore, MD, 2005.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies

for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[CT06] Laura Chiticariu and Wang-Chiew Tan. A Tool for Debugging Schema Mappings for Data Exchange. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Seoul, Soth Korea, 2006.

[FKP05] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Transactions on Database Systems (TODS)*, 30(1):174–210, 2005.

[KS91] Won Kim and Jungyun Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Journal*, 24(12):12–18, 1991.

[Leg05] Frank Legler. Datentransformation mittels Schema Mapping. Master's thesis, Humboldt-Universität zu Berlin, 2005.

[LN06] Frank Legler and Felix Naumann. A Classification of Schema Mappings and Analysis of Mapping Tools (full version). Technical report, Hasso-Plattner-Institut, Universität Potsdam, 2006. available at `http://www.hpi.uni-potsdam.de/fileadmin/hpi/FG_Naumann/publications/MappingTools_extended.pdf`.

[MBHR05] Sergey Melnik, Philip A. Bernstein, Alon Halevy, and Erhard Rahm. Supporting executable mappings in model management. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 167–178, New York, NY, USA, 2005. ACM Press.

[MHH00] Renée J. Miller, Laura M. Haas, and Mauricio A. Hernández. Schema Mapping as Query Discovery. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 77–88, San Francisco, CA, USA, 2000. Morgan Kaufmann.

[MRB03] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: a programming platform for generic model management. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 193–204, New York, NY, USA, 2003. ACM Press.

[PVM+02] Lucian Popa, Yannis Velegrakis, Renee J. Miller, Mauricio A. Hernandez, and Ronald Fagin. Translating Web Data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 598–609, 2002.

[SL90] Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SPD92] Stefano Spaccapietra, Christine Parent, and Yann Dupont. Model independent assertions for integration of heterogeneous schemas. *VLDB Journal*, 1(1):81–126, 1992.