AUTOMATED DATA MAPPING FOR CROSS ENTERPRISE DATA INTEGRATION

Stefan Böttcher and Sven Groppe
University of Paderborn , Computer Science and C-LAB , Fürstenallee 11 , D-33102 Paderborn , Germany
Email: stb@uni-paderborn.de, sg@uni-paderborn.de

Keywords: E-procurement catalogues, enterprise application integration, automated generation of data and query mappings

Abstract: There are currently multiple different classifications of product descriptions used in enterprise-internal applications and cross-enterprise applications, e.g. E-procurement systems. A key problem is the running of applications developed for one catalogue on product descriptions that are stored in a different classification. A common solution is that a catalogue specialist manually maps different classifications onto each other. Our approach avoids unnecessary manual work for mapping and automatically generates mappings between different classifications wherever possible. This allows us to run E-procurement applications on different catalogues with significantly reduced manual work needed for mapping, what we consider to be an important step towards enterprise application integration.

1 INTRODUCTION

1.1 Problem Origin and Motivation

Product catalogues of engineering companies with a wide product portfolio store products according to classifications, which are also used by enterprise applications which access such a catalogue. However, these catalogues are usually not just placed in isolation as online catalogue on the company's website, but are also integrated into customers' E-procurement systems or market places, together with the products of other manufacturers.

Our research originates from the development of an integrated electronic market place at INCONY AG in Paderborn as part of the B2BECOM project of the European Community, which aims to integrate inhomogeneous product catalogues and can access and can be accessed by multiple different E-procurement systems.

E-procurement systems (e.g. as used in electronic market places) handle their product data in XML formats like BMEcat (Schmitz et al., 2001), GOM (CEN/ISSS, 2000), cXML (Ariba, 2001), xCBL (Commerce One, 2002). These XML formats provide a field for each product data that contains a code for its product group called its category. Categories are arranged in a structure called a *classification* (c.f. figure 1). A classification is usually defined hierarchically, but the context of this paper does not require us to define the classification in such a way. In the figures here, however, classifications are presented hierarchically.

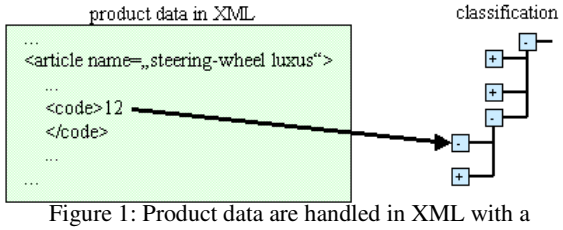

Figure 1: Product data are handled in XML with a reference to a position in a classification

A variety of standards have been suggested for classifications, e.g. ETIM (ETIM, 2002), eCl@ss (eCl@ss e.V., 2002), Edibatec (Edibatec, 2002), UN/SPSC (ECCMA, 2002), NACE (Eurostat, 1985), PRODCOM (Eurostat, 2002), IEC 61360 (IEC, 2002) and UniClass (ISO, 1994). Additionally, many companies have their own classification for historic reasons. In order to interchange data between different E-procurement systems, two tasks have to be carried out: a conversion between the XML formats and a conversion of the product data according to the classifications are needed. While XML transformers (e.g. XSLT) can be used to complete the first task, our work contributes to solving the second task, i.e. to *map* different classifications for product data onto each other (c.f. figure 2), which we call *data mapping*.

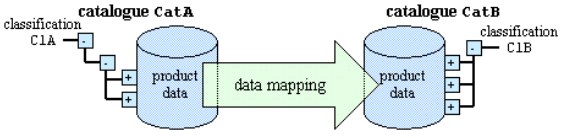

Figure 2: Data mapping from catalogue `CatA` to `CatB`

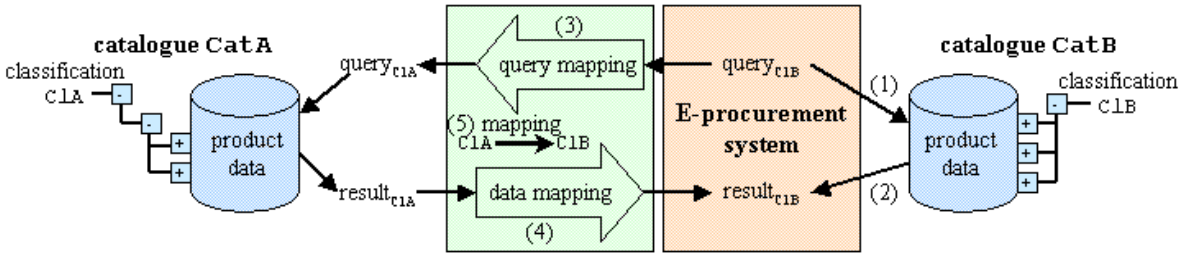

Figure 3: Combination of query and data mapping

When E-procurement systems query databases to get product data (c.f. figure 3 (1)), they get answers in the classification of the used catalogue (c.f. figure 3 (2)). When another catalogue with a different classification is integrated in such an E-procurement system, there are two main tasks to be carried out. The first main task is to map such queries (c.f. figure 3 (3)) according to classifications of the other catalogue, which we call *query mapping*. The second main task is to map the resulting product data according to the classification of the original catalogue (c.f. figure 3 (4)). Our approach applies a *mapping of classifications* to transform given program queries that generates queries to another product database using a different catalogue without any modification of the program itself (c.f. figure 3). Note that in our approach, query mapping and data mapping both use the same mapping definition from a classification `ClA` to a classification `ClB` (c.f. figure 3 (5)).

Because the mapping of classifications plays such a central role in integrating E-procurement systems and often the costs incurred prohibit the manual development of a mapping, our goal is to minimize the manual work needed in mapping definitions by reversing and chaining mappings already given for product data and queries. Thereby, clients of integrated E-procurement systems can access more product offers than in a stand-alone E-procurement system and providers in integrated E-procurement systems can offer their products to more clients than in a stand-alone E-procurement system.

For example, figure 4 shows four E-procurement systems (`EpsA`, `EpsB`, `EpsC` and `EpsD`) each using its own classification (`ClA`, `ClB`, `ClC` and `ClD`) and three given mappings between these applications. Assume a client using `EpsA` with classification `ClA` wants to query for and use product data in `EpsD` with classification `ClD`, then we need a mapping from `ClD` to `ClA`. Within our approach, the mapping from `ClD` to `ClA` can be generated by reversing the given mapping from classification `ClA` to classification `ClB` to get a mapping from `ClB` to `ClA`, and chain the mappings from `ClD` to `ClC`, from `ClC` to `ClB` and from `ClB` to `ClA` (c.f. figure 4).

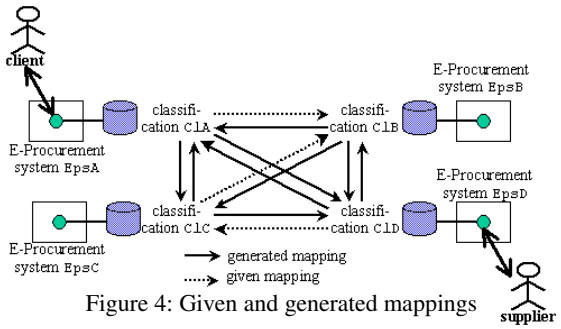

Figure 4: Given and generated mappings

1.2 Relation to other Work and our Focus

Mapping is related to contributions to database schema integration, views and query processing in federated databases, data warehouses and XML documents.

For the mapping of XML queries to other data storage formats at least two major research directions can be distinguished: firstly, to map XML queries to object oriented or relational databases (e.g. (Bourret et al., 2000)), and secondly, to map XML queries or documents to other XML documents (e.g. (Abiteboul, 1999). We follow the second approach, however, we additionally focus on mappings that overcome the problems of mismatching domains for data and queries.

Within related contributions to schema integration two approaches to data and query translation can be distinguished. While the majority of contributions (e.g. (Cluel et al., 1998), (Abiteboul, 1997), (Sciore et al., 1994)) map the data to a unique representation, we follow (Chang & Garcia-Molina., 2000) and (Chang & Garcia-Molina, 1999) to map the queries to those domains (or classifications) where the data resides. Similar to further work on query reformulation for federated databases (Calmet et al., 1997) and data warehouses (Yan et al., 2001), (Miller et al., 2000), we use mappings that map schema information from one product catalogue to another.

In contrast to all these contributions, we use such a mapping not only for data exchange but also for the automated generation of further mappings, i.e. reversed mappings and chained mappings for data and queries.

The reversion of mappings has been discussed in the context of updating XML (Tatarinov et al., 2001), (Zhang et al., 2001). Common to the database-XML mapping approaches, we also support data transfer in both directions, however, we use a direct mapping from XML data to XML data, based on classifications that are used in catalogue systems, without the need to map the data to a database. Another system containing a component for the mapping of XML data between different XML structures is the BizTalk Mapper (Microsoft Corporation, 2002), a part of the BizTalk Server. Like BizTalk, we support flexible and complex functions within the mapping definitions, but we additionally consider classifications and reverse mapping definitions. A further contribution (Duschka et al., 2000) reverses the rules of logic programming languages. However, this is done in the context of query reformulation and not in order to reverse a mapping definition. We additionally examine how to handle functions within the rules, how to add chaining of mappings, and how to reverse a mapping definition.

In contrast to all other approaches, we focus on the automated generation of mappings (i.e. to compute reversed and chained mappings) to integrate different classifications into enterprise applications that require another classification and use this as a basis for a free exchange of catalogue data and queries in a distributed E-procurement system.

2 CATALOGUE

2.1 Catalogue = Classification + Product Data

E-procurement systems usually classify products according to a given *classification* which is stored together with the product data in a product *catalogue*. Therefore, a product catalogue consists of two things: a classification which (comparable to a schema of a database) defines (product) *categories* and associated *attributes* of a specific type, and *product data* containing attribute *values* that are assigned to the attributes.

An example of a catalogue `CatA` is given in figure 5, where `ClA` is the *classification*, `vw_steering-wheel` and `opel_steering-wheel` are *categories*. Both categories have an *attribute* `diameter` of type `float` with the associated unit `cm`.

Furthermore, figure 5 shows *product data* `ProA` (and `ProB` respectively), where `50` (and `40` respectively) is an *attribute value*, which is assigned to the *attribute* diameter of the category `vw_steering-wheel` (and `opel_steering-wheel` respectively). In real systems and XML formats like BMEcat (Schmitz et al., 2001), GOM (CEN/ISSS, 2000), cXML (Ariba, 2001) and xCBL (Commerce One, 2002) the association is handled with a reference to the category by using an identification string.

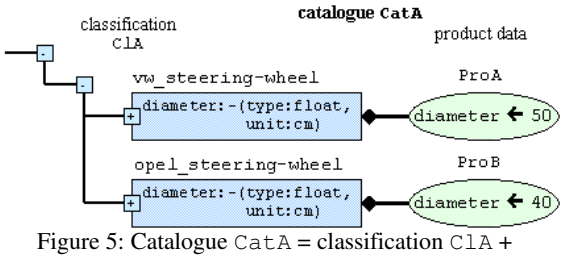

Figure 5: Catalogue `CatA` = classification `ClA` + product data `ProA` and `ProB`

Note that the classification determines how applications like query tools and services can search and find the data they need, i.e. when the classification changes, the query tools or services have to be adapted accordingly.

2.2 Multiple Catalogues and the Need for Data Mapping

Since there are different product classifications in use and the applications of many companies rely on their own product classification, free data exchange between the applications of different enterprises requires an integration of the underlying product classifications. For example, consider a catalogue `CatB` that uses the classification `ClB`. The classification `ClB` introduces a category `car_wheel` which has two attributes, say `diam` of type `float` and `manufacturer` of a catalogue-defined enumeration type containing the values `"VW"` and `"Opel"`. In classification `ClB`, the attribute `diam` represents the diameter of a steering-wheel and the unit of `diam` is `mm`.

Whenever applications implemented for catalogue `CatB` want to use product data from catalogue `CatA`, this is described in terms of classification `ClA`, they need a product data representation according to `ClB`, i.e. the data stored in catalogue `CatA` should be transformed to product data with an attribute `diam` set to `500` and an attribute `manufacturer` set to `"VW"`.

Transforming product data from a *source classification* `ClA` to a *target classification* `ClB` involves assigning the attribute `diam` of `ClB` with the attribute value of `diameter` of the product data of `ClA` multiplied by `10` and assigning the attribute `manufacturer` of `ClB` with `"VW"` (c.f. figure 6).

The problem considered is the definition of mappings for product data and queries. In order to avoid unnecessary manual work which is needed for defining the mapping of data and queries, we have to generate new mappings from given mappings. This includes automatically generating a reversed mapping from a given mapping and chaining mappings, i.e. computing transitive mappings.

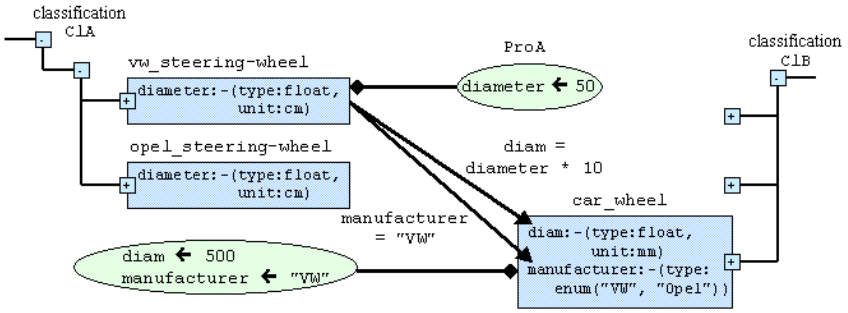

Figure 6: Data mapping of product data ProA from classification ClA to ClB

3 DATA AND QUERY MAPPING

In the following subsections, we describe how mappings between different catalogues for product data and queries can be defined and how a set of given mappings can be used in order to generate new mappings for product data and queries.

3.1 Four Nested Layers of Data Mapping

Let us assume, a given *mapping rule*, which allows us to transform product data from a *source classification* ClA to a *target classification* ClB whenever needed, is defined as follows (For simplicity we use a notation similar to Datalog, although the implementation is done in XML and Java.):

Example 1:

```
ClB.car_wheel(
    diam(Diam) ,
    manufacturer(Manufacturer)
):- ClA.vw_steering-wheel(
      diameter(Diameter) ) ,
    set Diam = Diameter * 10 ,
    set Manufacturer="VW".
```

Since the manual definition of such mapping rules is time consuming and error prone, i.e., it should be reduced to a minimum, we present a technique that uses given mappings to generate new mappings. For this purpose, we regard such a data mapping as containing four *nested mapping layers*, which we call *value layer*, *attribute layer*, *category layer* and *mapping* from a source classification to a target classification, which are described next.

Firstly, this mapping rule applies a unit transformation from cm to mm for the attribute value of the attribute diameter by multiplying it by 10. For this unit transformation, we use a function f_1(Diameter)=Diameter*10 that maps a source attribute value for Diameter to the corresponding target attribute value. We say that a function that maps attribute values belongs to the *value layer* of our mapping.

Furthermore, the mapping rule assigns the attribute value "VW" to the attribute Manufacturer. (Our mapping rule reflects the fact that products of the category vw_steering-wheel are manufactured by VW.) This can be described as a function with no parameters, i.e. $f_2()$="VW", because it is not necessary to inspect catalogue data of catalogue CatA in order to determine the Manufacturer. In general, the functions of the value layer may take an arbitrary combination of *source attributes* (i.e. of attributes found in the source classification here classification ClA) as arguments.

Secondly, it is necessary to map attributes that may have different names onto each other, e.g. the result of f(Diameter) = Diameter*10 is assigned to a target attribute Diam. We say, the assignment of f(Diameter) to a *target attribute* (i.e. an attribute of a category of the target classification) belongs to the *attribute layer* of our mapping. In general, in the attribute layer, source attributes $q_{i1},\ldots,q_{im_i}$ are mapped to a target attribute t_i by an assignment set $t_i=f_i(q_{i1},\ldots,q_{im_i})$, e.g. set Diam=f(Diameter).

Note that the attribute layer and the value layer of a mapping are nested, i.e., the attribute layer uses the value layer.

Thirdly, in our example, the *source category* vw_steering-wheel is mapped to the *target category* car_wheel. We say, this mapping of categories belongs to the *category layer*.

In its most general form, a mapping rule is defined as (*)

```
target(t1,…,tk):- source(q1,…,qp),
                  set t1=f1(q11,…,q1m1),…,
                  set tk=fk(qk1,…,qkmk),
                  check condition.
```

where each q_{ij} must be one of the source attributes $q_1, \ldots, q_p$, that can be retrieved from the source product catalogue.

Furthermore, $\{t_1, \ldots, t_k\}$, occurring on the left side of the rule, must be a subset of the attributes of the `target` category and each t_i must be assigned within the attribute layer (i.e. on the right side of the rule). Similarly, $\{q_1, \ldots, q_p\}$ must be a subset of the attributes of the `source` category.

Finally, a mapping rule may contain a `condition` which restricts the application of the mapping rule and can also be considered as part of the category layer. For example, in order to map only those steering-wheels with `Diameter` less than `50 cm` from the category `opel_steering-wheel` to the category `car_wheel`, we add the following rule to the mapping from `ClA` to `ClB`, that contains a condition `check Diameter<50` which restricts the mapping.

Example 2:

```
ClB.car_wheel(
    diam(Diam) ,
    manufacturer(Manufacturer)
):- ClA.opel_steering-wheel(
       diameter(Diameter) ),
    set Diam = Diameter * 10,
    set Manufacturer="Opel",
    check Diameter<50.
```

In general, a *condition* of the mapping rule (*) is defined as follows. `true` and `false` are conditions. If `Attr` is an attribute defined for a category `source`, `const` is a constant value, and `op` is a comparison operator ($<, >, =, \neq, \leq, \geq$), then `Attr op const` is a condition. If `B1` and `B2` are conditions, then `B1 and B2`, `B1 or B2` and `not B1` are conditions too.

Altogether, we can distinguish four nested layers of mapping: The top layer (or fourth layer) itself, called the *mapping* from a source classification `ClA` to a target classification `ClB` consists of the set of all mapping rules from (source) categories of `ClA` to (target) categories of `ClB`. The *category layer* (or third layer) determines which source category `CatA` of classification `ClA` is mapped to which target category `CatB` of `ClB`. Furthermore, the *attribute layer* (or second layer) determines which attributes $\texttt{AttrA}_1$, ..., $\texttt{AttrA}_n$ of `CatA` are mapped to which attribute `AttrB` of `CatB`. Finally, the *value layer* (or inner-most layer) determines which function has to be applied in order to compute `AttrB` from $\texttt{AttrA}_1$, ..., $\texttt{AttrA}_n$. The distinction between these four layers helps us to automate the generation of new mappings for data and queries.

3.2 Query Mapping

We use the following example in order to explain query mapping.

In a working online-system using catalogue `CatB`, a user may navigate within the classification `ClB` and search for product data by specifying some conditions on attributes as described above, i.e., the front-end of the online-system may e.g. submit a query

```
query( Diam ) :-
  ClB.car_wheel(
    manufacturer(Manufacturer),
    diam(Diam) ) ,
  check Manufacturer="VW" and Diam<800.
```

In order to include product data from catalogue `CatA` which uses `ClA` within an application based on `ClB`, the query has to be transformed into the corresponding query using `ClA`, by applying the mapping rules of Example 1 and 2 to the query.

The mapping rule of Example 1 applied to the given query yields the following *mapped query*:

```
query( Diameter ) :-
  ClA.vw_steering-wheel(
    diameter(Diameter) ) ,
  check true and Diameter<80.
```

Note that the data mapping (of Example 1) contains an assignment `set Manufacturer="VW"` in the mapping rule, which guarantees that the part of the condition `Manufacturer="VW"` is always `true`.

The mapped query can be carried out on catalogue `CatA`, and thereafter the results of query can be mapped by data mapping and transferred back to the application which uses `ClB`.

Since the condition `Manufacturer="VW"` of the given query is insatisfiable after the assignment `set Manufacturer ="Opel"` of the mapping rule of Example 2, this mapping rule cannot generate further results from the given query.

The general approach to query mapping is discussed in Appendix A.

3.3 Chaining of Mappings

Assume, we have two mapping rules, one as defined in Example 1 mapping a `vw_steering-wheel` of classification `ClA` to a `car_wheel` as defined in classification `ClB`, and a further mapping rule which maps a `car_wheel` in classification `ClB` to an `automobile_wheel` in a catalogue using a

classification ClC:

```
ClC.automobile_wheel(
    manu(Manufacturered_By) ,
    peri(Perimeter)
):- ClB.car_wheel(
      manufacturer(Manufacturer) ,
      diam(Diam) ) ,
    set Manufactured_By = Manufacturer,
    set Perimeter = Diam * 3.14 .
```

then we can chain both mapping rules to a transitive rule

```
ClC.automobile_wheel(
    manu(Manufactured_By) ,
    peri(Perimeter)
):- ClA.vw_steering-wheel(
     diameter(Diameter) ),
    set Manufactured_By = "VW",
    set Perimeter = (Diameter*10)*3.14.
```

The general approach to the chaining of mapping is discussed in Appendix B.

3.4 Reversion of Mapping

Assume, an application uses classification ClB to query for product data and also uses a given mapping. Applying the given mapping transforms the query into another query that can be used in a different catalogue CatA based on a classification ClA. This application uses the same given mapping definition to transfer product data found in CatA for that query back to ClB (c.f. figure 2).

We describe next, how to automatically generate a so called *reversed mapping* that allows us to map a query for product data using ClA to a query for product data in a catalogue using ClB, and to map the results found in the catalogue back to ClA (c.f. figure 8).

Such a reversed mapping rule of the mapping rule given in Example 1 is

Example 3 (reversed mapping rule of Example 1):

```
ClA.vw_steering-wheel(
    diameter(Diameter)
):- ClB.car_wheel( diam(Diam) ,
      manufacturer(Manufacturer) ),
    set Diameter=Diam/10,
    check Manufacturer="VW".
```

The assignment `set Diam=Diameter*10` of Example 1 has been transformed into the reversed assignment `set Diameter=Diam/10`. In this case, the reverse function can be generated, however in general, in the *value layer,* the reverse function of $f(q_1,\ldots,q_m)$ must be given.

Note however that the assignment `set Manufacturer="VW"` of Example 1 has been transformed into `check Manufacturer="VW"`. We have to restrict the reverse mapping to wheels manufactured by "VW", since car_wheel contains different manufacturers, but the classification vw_steering-wheel only contains products manufactured by "VW". In general, every original mapping rule containing an assignment `set t=const` with const being a constant is transformed into what we call a *category condition*, i.e. `check t=const`.

Now, let us compute the reversed mapping rule of Example 2:

Example 4 (reversed mapping rule of Example 2):

```
ClA.opel_steering-wheel(
     diameter(Diameter)
):- ClB.car_wheel( diam(Diam) ,
      manufacturer(Manufacturer) ) ,
    set Diameter=Diam/10,
    check (Manufacturer="Opel") and
      (Diam<500).
```

Again, `set Diam=Diameter*10` is transformed into `set Diameter=Diam/10`. Similar to the last example, the assignment `set Manufacturer="Opel"` is transformed into the condition `check Manufacturer="Opel"`. Furthermore, by applying query mapping the condition `check Diameter<50` is transformed into a condition `check Diam<500`, which we call a *mapped condition*. This is because only small wheels shall be mapped in both directions.

The general approach to the reversion of mappings is discussed in Appendix C.

4 CONSEQUENCES FOR PRACTICAL CLASSIFICATIONS

The automated computation of reversed mappings has been implemented and evaluated at INCONY AG, Paderborn, as part of their electronic market places for the electrical industry. For example, an online-system based on two manually developed mappings has been completed: one maps EDIBATEC classifications to ETIM, the other maps classifications between ETIM 1.0 and ETIM 1.1. For each classification, original product data is available. Furthermore, each classification can be used through its own graphical user interface which supports browsing according to the given catalogue structure.

Within this application, approximately 100

categories have to be mapped from each classification to the other classifications – with an average time of 1 day needed for each manual category mapping definition. The automated computation of reverse mappings alone allows us to reduce the manual work by 50%. The combination of computing reversed and chained mappings saves even more work (c.f. figure 4), since N suitable mappings between N+1 different classifications can be used to generate up to N^2 mappings without any manual work.

Although, in general, not every mapping can be reversed, our approach can generate most of the reversible mappings that occur in practice (in our application this was more than 95%).

For the application, the mapping between different incompatible versions of a product catalogue (ETIM 1.0 and ETIM 1.1) enables an incremental catalogue upgrade without interrupting the service of a running online-system. Therefore, our approach meets a key requirement of applications running on an integrated market place, i.e. the flexible exchange of product data beyond the limits of different classifications. Furthermore, our approach to generating reversed and chained mappings for data and queries appears to be an important step towards enterprise application integration.

5 SUMMARY AND CONCLUSIONS

Cross company E-procurement systems use different classifications. Therefore product catalogue integration requires a mapping between the classifications. Since the manual map definition between different classifications requires considerable work and is error prone, we suggest instead generating mappings automatically wherever possible. Our solution includes the automated generation of reversed mappings and chained mappings.

This allows an E-procurement system, which has been implemented for a single classification, to use data of all other catalogues for which a mapping exists. We consider this exchange of data across the boundaries of company specific classifications to be an important step towards enterprise application integration.

We developed our approach in the context of electronic market places, however, it may be promising to investigate how this can be extended to other areas of enterprise application integration.

ACKNOWLEDGEMENTS

This work was funded by the B2B-ECOM project (IST-1999-10281) and by the MEMPHIS project (IST-2000-25045).

REFERENCES

Abiteboul, S., 1999: On views and XML. In *PODS*, pages 1-9.

Abiteboul, S., Cluet, S., and Milo, T., 1997. Correspondence and translation for heterogeneous data. In *Proc. of the 6th ICDT.*

Ariba, 2001: CXML User's Guide Version 1.2.007. http://www.cXML.org/, Ariba, Inc..

Bourret, R., Bornhövd, C., Buchmann, A.P., 2000: A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases. *2nd Int. Workshop on Advanced Issues of EC and Web-based Information Systems (WECWIS)*, San Jose, California.

Calmet, J., Jekutsch, S., and Schu, J., 1997: A generic query-translation framework for a mediator architecture. In *ICDE*, pages 434-443.

CEN/ISSS, 2000: Definition of an information model to enable the interchange of engineering information on a marketplace and an appropriate description of content in a system-neutral format. *Draft, CEN/ISSS Workshop on Global Engineering Networking*.

Chang, C.-C. K., and Garcia-Molina, H., 1999. Mind your vocabulary: Query mapping across heterogeneous information sources. In *Proc. of the 1999 ACM SIGMOD Conf.*, Philadelphia, 1999. ACM Press, NY.

Chang, C.-C. K., and Garcia-Molina, H., 2000: Approximate Query Translation Across Heterogeneous Information Sources. *VLDB 2000.*

Cluet, S., Delobel, C., Simon, J., and Smaga, K., 1998. Your mediators need data conversion! In *Proc. of the 1998 ACM SIGMOD Conf.*

Commerce One, 2002: XML Common Business Library. http://www.xCBL.org/.

Duschka, O. M., Genesereth, M. R., and Levy, A. Y., 2000: Recursive query plans for data integration. *Journal of Logic Programming*, 43(1):49-73.

ECCMA (Electronic Commerce Code Management Association), 2002: Universal Standard Products and Services Classification (UNSPSC). http://www.unspsc.org/.

eCl@ss e.V., 2002: eCl@ss - Standard für Materialklassen und Warengruppen. http://www.eClass.de. eCl@ss e.V. c/o Institut der deutschen Wirtschaft, Köln.

Edibatec, 2002. Bienvenue sur le site Edibatec. http://www.edibatec.org/.

ETIM, 2002: ETIM. http://etim.de/.

Eurostat (Commission of the European Communities (Statistical Office/Eurostat)), 1985: General Industrial Classification of Economic Activities within the European Communities. Luxembourg: Office for Official Publications of the European Communities.

Eurostat (Commission of the European Communities (Statistical Office/Eurostat)), 2002: PRODCOM list (List of PRODucts of the European COMmunity). Luxembourg: Office for Official Publications of the European Communities.

Groppe, S., 2002: Online-Mapping und Mappingumkehr für Internet-Marktplätze zur Anbindung von Produktkatalogen. *Diplomarbeit*, Universität-Gesamthochschule Paderborn.

IEC (International Electrotechnical Commission), 2002: IEC 61360-1 to 4, Standard data element types with associated classification scheme for electric components. http://www.iec.ch/.

ISO (International Organization for Standardization), 1994: Classification of information in the construction industry. *Technical Report ISO 14177*, Geneva: International Organization for Standardization.

Levy, A. Y., and Weld, D. S., 2000: Intelligent internet-systems. *Artificial Intelligence*, 118(1-2).

Microsoft Corporation, 2002: Microsoft BizTalk Server 2002 Enterprise Edition. *Documentation*, http://www.microsoft.com/biztalk/, Microsoft Corporation.

Miller, R. J., Haas, L. M., and Hernández, M. A., 2000: Schema mapping as query discovery. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases,* September 10-14, 2000. Cairo, Egypt, pages 77-88, Morgan Kaufmann.

Schmitz, V., Kelkar, O., Pastoors, T., Renner, T., and Hümpel, C., 2001: Spezifikation BMEcat Version 1.2. http://www.bmecat.org/, Fraunhofer IAO, Universität Esssen BLI.

Sciore, E., Siegel, M., and Rosenthal, A., 1994. Using semantic values to facilitate interoperability among heterogeneous information systems. *Trans. on Database Systems*, 19(2).

Tatarinov, I., Ives, Z.G., Halevy, A.Y., Weld, D.S., 2001: Updating XML. *ACM SIGMOD Int. Conf. on Management of Data*.

W3C, 2001: Extensible Stylesheet Language (XSL). http://www.w3.org/Style/XSL/.

Yan, L., Miller, R. J., Haas, L. M., and Fagin, R., 2001: Data-driven understanding and refinement of schema mappings. In *ACM SIGMOD, Int. Conf.*, Santa Barbara.

Zhang, X., Mitchell, G., Lee, W.C., Rundensteiner, E.A., 2001: Clock: Synchronizing Internal Relational Storage with External XML Documents, *RIDE-DM 2001*.

APPENDIX A – GENERAL QUERY MAPPING

Assume, a given query

```
query(t1,…,tk) :- target(t1,…,tk),
                  check query_condition.
```

queries for product data in the `target` category using a constraint `query_condition`. In practice, queries may have fewer parameters than the category `target`. In this case, a subset of the attributes of our query will answer the given mapping. Query mapping transforms this given query according to a mapping rule

```
target(t1,…,tk):- source(q1,…,qp),
                  set t1=f1(q11,…,q1m1),…,
                  set tk=fk(qk1,…,qkmk),
                  check condition.
```

into a *mapped query*

```
query(q1,…,qp) :- source(q1,…,qp),
  check mapped_query_condition and
        condition.
```

where `mapped_query_condition` has the following property P:

For each product data and each mapping rule, the following must hold: Applying the mapped query to the product data (given for the `source` category) yields the same query result as first mapping the product data to the `target` category and then applying the given query to the mapped data.

In order to calculate `mapped_query_condition` we regard condition as a tree (with inner nodes `and`, `or` or `not`). We replace all leaf nodes, which are not trivial (like `true` and `false`), according to the following rules:

The leafs of the form t_i `op const` are replaced with a condition of the `source` category according to the function $t_i=f_i(q_{i1},…,q_{im_k})$ of the assignment `set` $t_i=f_i(q_{i1},…,q_{im_k})$ to t_i of the mapping rule as described below in two special cases.

If f_i has one parameter q_{i1} and is strictly increasing, we replace `const` by the result of the inverted function f_i^{-1}`(const)` and t_i by q_i.

If f_i returns a constant value, say $f_i()= const_{f_i}$, then we replace the leaf with `true` or `false` depending on the results of the evaluation of `const op` $const_{f_i}$.

There are further rules, e.g. rules using value tables or nested functions that we skip here due to space limitations.

APPENDIX B – GENERAL CHAINING OF MAPPINGS

In general, chaining of mapping rules can be computed as follows: In order to chain the mapping rules (1) and (2) in figure 7 the source category `intermediate(i1,…, ia)` of mapping rule (1) is replaced with the source category `source(q1,…, qp)` of mapping rule (2). Each attribute i_{de} of category `intermediate`, which corresponds to an attribute i_j in `intermediate(i1,…, ia)`, is then replaced with the right side $g_j(q_{j1},…, q_{jm_j})$ of the assignment `set` $i_j = g_j(q_{j1},…, q_{jm_j})$ in mapping rule (2). Finally, `intermediate_condition` is replaced with the conjunction of

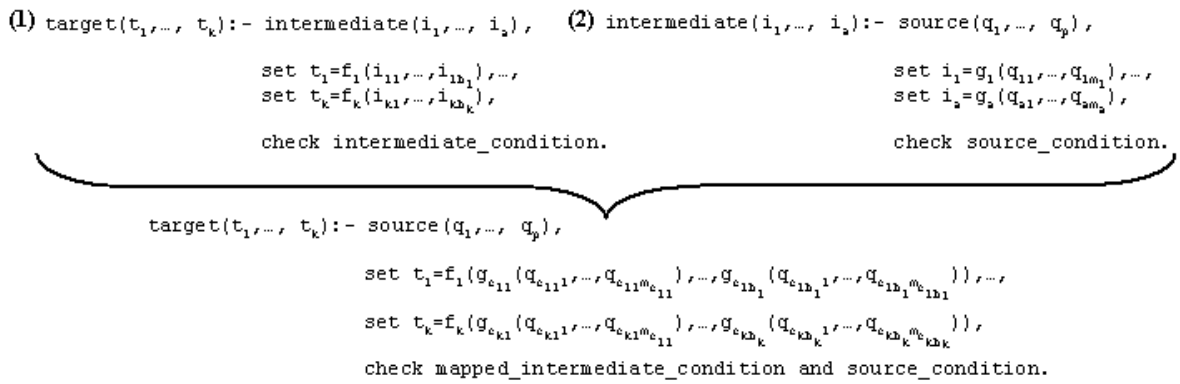

Figure 7: Chaining mapping rules

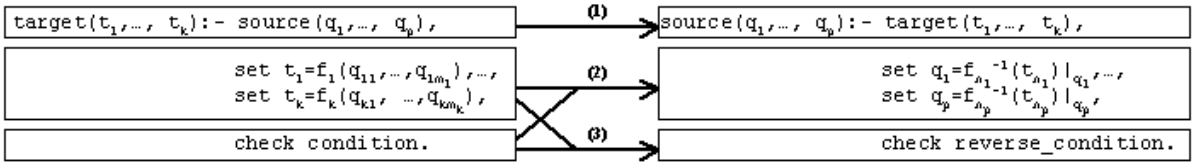

Figure 8: Steps for reversing a mapping rule

source_condition and the result of query mapping of intermediate_condition with mapping rule (2), which we call mapped_intermediate_condition.

APPENDIX C – GENERAL REVERSION OF MAPPING

In general, a mapping rule of the *category layer* is reversed by the following three steps (c.f. figure 9):

1. Swap target(t1,…, tk) and source(q1,…, qp).
2. Calculate the reverse mapping for each qi.
3. Calculate the *reverse condition* reverse_condition.

For the second step, i.e. to calculate the reversed mapping for each qi, we can determine qi using one of two different possible ways.

Firstly, if exactly one value, say e, for qi fulfills condition, as e.g. for the condition Manufacturer="VW", then the reverse mapping for qi yields an assignment set qi=e, e.g. set Manufacturer="VW". This will occur when we use the mapping of Example 3 as input and compute the mapping of Example 1 as reversed mapping.

Secondly, we use inverted functions $f_{n_i}^{-1}(t_{n_i})=(q_{n_i1},\ldots,q_{n_im_{n_i}})$ on the *value layer*. For example the inverted function $f^{-1}(Diam)=Diam/10$ is used to compute the value of Diameter in the assignment set Diameter=Diam/10. The reverse mapping in the *attribute layer* which reverses $t=f(q_1,\ldots,q_m)$ and computes each attribute qi is set $q_i=f^{-1}(t)|_{q_i}$, i.e. the assignment of the attribute qi to $f^{-1}(t)=(q_1,\ldots,q_m)$ restricted to the i-th component. At the end of this step we compute the reversed mapping rule (**)

```
source(q1,…, qp):- target(t1,…, tk),
                   set q1=fn1-1(tn1)|q1,…,
                   set qp=fnp-1(tnp)|qp.
```

without reverse_condition , because the latter is computed in the following third step.

For the third step, our goal is to compute a reverse_condition that is as restrictive as possible. For this purpose two constraints of the category target have to be computed:

The first constraint of the category target, which we call the *mapped condition* mapped_condition (c.f. figure 10), is computed by applying query mapping to condition. Note, that query mapping of condition requires the reversed mapping rule (**) from the source to target category without reverse_condition, as computed in the second step.

The second constraint is a *category condition* $category\_condition_i$. Category conditions are computed from assignments to constants, i.e. each assignment to a constant set t=const is transformed into a restriction check t=const.

In principle, the *reverse condition* is the conjunction of all category conditions and the mapped condition received by query mapping, i.e. the reverse condition reverse_condition is equal to

$category\_condition_1$ and … and
$category\_condition_n$ and
mapped_condition.