Defining and Using Schematic Correspondences for Automatically Generating Schema Mappings

Lu Mao, Khalid Belhajjame, Norman W. Paton, and Alvaro A. A. Fernandes

School of Computer Science
University of Manchester
Oxford Road, Manchester, UK
{lmao,khalidb,norm,alvaro}@cs.man.ac.uk

Abstract. Mapping specification has been recognised as a critical bottleneck to the large scale deployment of data integration systems. A mapping is a description using which data structured under one schema are transformed into data structured under a different schema, and is central to data integration and data exchange systems. In this paper, we argue that the classical approach of correspondence identification followed by (manual) mapping generation can be simplified through the removal of the second step by judicious refinement of the correspondences captured. As a step in this direction, we present in this paper a model for schematic correspondences that builds on and extends the classification proposed by Kim *et al.* to cater for the automatic derivation of mappings, and present an algorithm that shows how correspondences specified in the model proposed can be used for deriving schema mappings. The approach is illustrated using a case study from integration in proteomics.
Key words: Schematic correspondences, schema mappings, mapping generation.

1 Introduction

Data integration has for the last two decades been the subject of active investigations within the database and the artificial intelligence communities [2, 4]. This is testified partly by the number of research papers, projects and prototypes that tackle data integration related issues [9]. The aim is to provide users with integrated access to data sets that reside in multiple sources and are stored using heterogeneous representations [13]. A data integration system can play a central role in multiple applications, e.g., it can be used for cross-querying of data stored in databases that belong to independent companies, or to promote collaboration in large scientific projects by providing investigators with a means for querying and combining results produced by multiple research labs [21]. The components at the heart of a data integration system are: the schemas of the sources, the data sets to be integrated, an integration schema over which users pose queries, and mappings that specify how data structured under the schemas of the sources can be transformed and combined into data structured according to the integration schema [6].

Despite the advances made, data integration seems to have had a limited impact in practice: existing data integration systems are mostly research prototypes. The limited adoption of this technology is partly due to its cost ineffectiveness [8]. In particular,

the specification of the mappings between the schemas of the sources and the integration schema has proved to be both time and resource consuming, and was recently recognised as a critical bottleneck to the large scale deployment of data integration systems [8, 15].

Mapping specification is generally a two-phase process. In the first phase, the correspondences specifying how the elements of the integration schema relate to the elements of the sources' schemas are specified. Correspondences are primarily used to identify the elements of the integration schema and the source schemas that are semantically equivalent, i.e., represent information that belongs to the same domain concept. For example, the correspondence $\langle S_{int}.\mathit{Staff}, S_1.\mathit{Employee}\rangle$ specifies that the relation *Staff* in the integration schema S_{int} is semantically equivalent to the relation *Employee* of the source schema S_1. Different types of correspondences may be drawn between two schemas [19, 5]. For example, correspondences can be used to relate one element of a given schema to one element of another schema, e.g., $\langle S_{int}.\mathit{Staff}, S_1.\mathit{Employee}\rangle$, or to relate multiple elements of one schema to one element of another schema, e.g., the correspondence $\langle S_{int}.\mathit{Staff}, \{S_1.\mathit{Employee}, S_1.\mathit{Department}\}\rangle$ states that the relation *Staff* in the integration schema is semantically equivalent to some combination of the relations *Employee* and *Department* of the source schema S_1. There are several models for drawing schematic correspondences in the literature [17, 7, 14, 12], of which the model proposed by Kim *et al.* [12] is perhaps the most comprehensive. In the second phase of mapping specification, the views that implement the mappings necessary for rewriting the queries issued against the integration schema into queries over the schemas of the sources are specified. Examples of techniques that can be used for specifying the mappings based on identified correspondences were informally described by Kim *et al.* [11].

The specification of schema mappings is largely a manual activity. In this paper, we show that the views implementing the mappings can be automatically derived if the correspondences defined by Kim *et al.* [12] are refined in a number of carefully targeted ways. In essence, it is the argument of this paper that the classical approach to correspondence identification followed by (manual) mapping generation can be made more cost-effective through the removal of the second step based on judicious refinement of the correspondences captured. Take for example the correspondence $\langle S_{int}.\mathit{Staff}, \{S_1.\mathit{Employee}, S_1.\mathit{Department}\}\rangle$, presented earlier. This correspondence does not specify the correspondences between the attributes of *Staff* and those of *Employee* and *Department*. Neither does it specify how the tuples of the *Staff* and *Department* relations should be combined. Because of this, an algorithm would not be able to derive the view that can be used for populating the relation *Staff* using the tuples in *Employee* and *Department*. The main contributions of this paper are therefore:

- A model for schematic correspondences that builds on and extends the classification proposed by Kim *et al.* to enable the automatic derivation of mappings.
- An algorithm for automatically generating the views that implement the mappings between two schemas based on these more expressive schematic correspondences.

Accordingly, the paper is organised as follows. We begin by presenting the model for specifying schematic correspondences in Section 2. We go on to show how the

correspondences specified in this model can be used for automatically generating the views that implement the mappings between two schemas in Section 3. We show how schematic correspondences can be used for deriving mappings between proteomics data sources in Section 4. We analyse existing proposals for modelling schematic correspondences and automating mapping specification in Section 5. Finally, we close the paper by underlying our main contributions in Section 6.

2 Schematic Correspondences

Schematic correspondences are used to associate elements of one schema (referred to in what follows as the *source schema*) to elements that are semantically equivalent in another schema (referred to as the *target schema*). For the purpose of this paper we assume that source and target schemas are specified using the relational model [3]. Therefore, the elements connected by schematic correspondences are relations and/or attributes. As mentioned earlier, the model of schematic correspondences presented in this paper is built on the classification proposed by Kim *et al.* [12]. This classification identifies a *wide range* of correspondences that may occur between two schemas. Nevertheless, the correspondences as defined by Kim *et al.* do not convey sufficient information for deriving the views that express the mappings between schemas. Consider, for example, a correspondence that connects the relations *Employee* and *Address* in the source schema to the relation *Staff* in the target schema. To be able to specify the view for populating the relation *Staff* using the tuples in both *Employee* and *Address*, information specifying how the tuples in *Employee* and *Address* are to be combined is needed. In the following, we augment the model of correspondences proposed by Kim *et al.* to support the automatic generation of mappings. In doing so, we distinguish between two kinds of correspondences: *relation correspondences* and *attribute correspondences*.

2.1 Relation Correspondences

This family of correspondences associates relations from the source schema with relations that are semantically equivalent in the target schema. We distinguish between four kinds of relation correspondences depending on the number of relations used to represent a given concept in the source and target schemas, namely *one-to-one relation correspondence*, *many-to-many relation correspondence*, *many-to-one relation correspondence* and *one-to-many relation correspondence*.

One-To-One Relation Correspondence This kind of correspondence associates one relation in the source schema with one relation that is semantically equivalent to it in the target schema. For example, a one-to-one relation correspondence can be used to associate the *Company* relation in the source schema with the *Corporation* relation in the target schema: the tuples of both relations represent business organisations. We define a one-to-one relation correspondence by the tuple:

$$\langle r_s, r_t, p_s, p_t, AC \rangle$$

where r_s is a relation in the source schema and r_t is a relation in the target schema. *AC* is a set of attribute correspondences that specifies the relationships between the attributes of the relation r_s and those of r_t. Attribute correspondences will be presented in Section 2.2. p_s and p_t are two selection predicates specifying which instances of r_t can be used to populate r_s and which instances of r_s can be used to populate r_t, respectively. Note that r_s and r_t can have different names and different structures, i.e., one or more attributes in r_s may not have any corresponding attributes in r_t, or vice versa.

Many-To-Many Relation Correspondence This kind of correspondence associates multiple relations from the source schema R_s with multiple relations in the target schema R_t. It specifies that the combination of the relations in R_s is semantically equivalent to the combination of the relations in R_t. We define a many-to-many relation correspondence by the tuple:

$$\langle R_s, R_t, type_s, type_t, JP_s, JP_t, HPP_s, HPP_t, AC \rangle$$

where, R_s is a set of relations in the source schema and R_t is a set of relations in the target schema. To specify the mapping from the relations in R_s to the relations in R_t, we need information specifying how the relations in R_s and the relations in R_t are to be combined, respectively. This is specified using the fields $type_s$ and $type_t$ which take either the value *Vertical partitioning* or the value *Horizontal partitioning*. $type_s$ specifies how the relations in R_s are to be combined, whereas $type_t$ specifies how the relations in R_t are to be combine. Vertical partitioning indicates that the relations in R_s (resp. in R_t) should be combined by joining them using JP_s (resp. JP_t), a conjunction of attribute comparison predicates. Figure 1 illustrates an example of vertical partitioning in which the combination of the relations *Department* and *Grant* in the source schema corresponds to the combination of the relations *Fund* and *Person* in the target schema. The relations *Department* and *Grant* are combined using the join predicate '$Department.did = Grant.did$', and the relations *Fund* and *Person* are combined using the join predicate '$Fund.PI = Person.person\_id$'. Note that the attribute PI represents the identifier of the principal investigator. As we shall see later in Section 3, we use outer join predicate instead of natural join to avoid any loss of information.

A horizontal partitioning indicates that the relations in $R_s \cup R_t$ represent the same domain concept. As an example, consider the many-to-many relation correspondence that connects the relations *Undergraduate* and *Postgraduate* from the source schema to the relations *LocalStudent* and *OverseasStudent* in the target schema (see Figure 2). The relations *Undergraduate*, *Postgraduate*, *LocalStudent* and *OverseasStudent* capture information about the same semantic domain, namely *Student*. Differently from vertical partitioning in which the relations are combined using the join relational operator, in the case of horizontal partitioning the relations in the source schema (resp. in the target schema) have compatible structures and are combined using the union relational operator. Note that this assumes that semantically corresponding attributes in the relations to be combined are given the same name. It is worth noting that horizontal partitioning may be defined between relations that do not have identical structures. Consider the example of horizontal partitioning presented earlier in Figure 2. The relation *PostgraduateStudent* may have, in addition to the attributes of *UndergraduateStudent*, the attributes

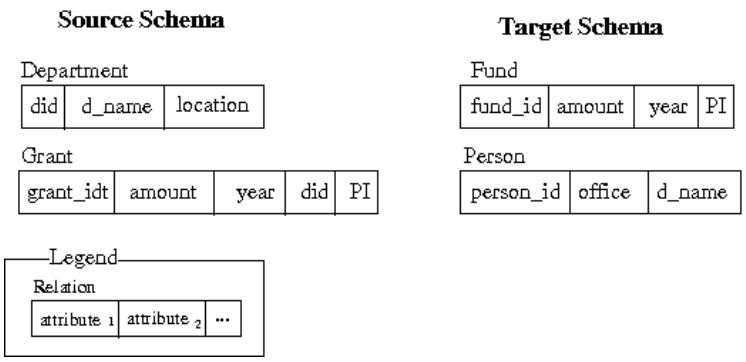

Fig. 1. Example of many-to-many vertical partitioning

supervisor and *advisor*. Applying a union operator in this case is not possible since the two relations have different structures. In this case, it is possible to use the outer union operator to combine the relations *UndergraduateStudent* and *PostgraduateStudent* [3]. This operator deals with the problem of inconsistency in structure between two relations r_1 and r_2 by applying the union operator over the relations r'_1 and r'_2, where r'_1 (resp. r'_2) is obtained by adding the attributes in r_2 (resp. r_1) that are missing in r_1 (resp. r_2) and padding them with null values. For example, the outer union of the *UndergraduateStudent* and *PostgraduateStudent* relations is a relation that has the same structure as *PostgraduateStudent* and in which the tuples representing undergraduate students have null values for the attributes *supervisor* and *advisor*.

To specify the view for populating a relation in the target schema we need information specifying which tuples obtained by the union of the relations in the source schema are to be considered. For example, we know that not all local and overseas students are undergraduate students, and, therefore, we cannot populate the relation *UndergraduateStudent* using all the tuples obtained by the union of the relations *LocalStudent* and *OverseasStudent*. HPP_s and HPP_t are used for this purpose. These are two sets, the elements of which are predicates: an element of HPP_t is a conjunction of predicates that specifies which tuples should be used for populating a given relation in the target schema; similarly an element of HPP_t is a conjunction of predicates that specifies which tuples should be used for populating a given relation in the source schema. We assume in the following the existence of the function *getHPPredicate(corr,r)* that returns the conjunction in $corr.HPP_t$ that is associated with the relation *r*. For example, $getHPPredicate(corr_{student}.HPP_s, UndergraduateStudent) = `(category = "undergrad")'$, where $corr_{student}$ is the correspondence that relates the relations *LocalStudent* and *OverseasStudent* to the relations *Undergraduate* and *Postgraduate*.

As in one-to-one relation correspondences, *AC* is a set specifying the correspondences between the attributes of the relations in R_s and those of the relations in R_t.

Many-To-One and One-To-Many Relation Correspondences These can be seen as specific cases of many-to-many relation correspondences. A many-to-one relation correspondence associates multiple relations from the source schema R_s to one relation r_t

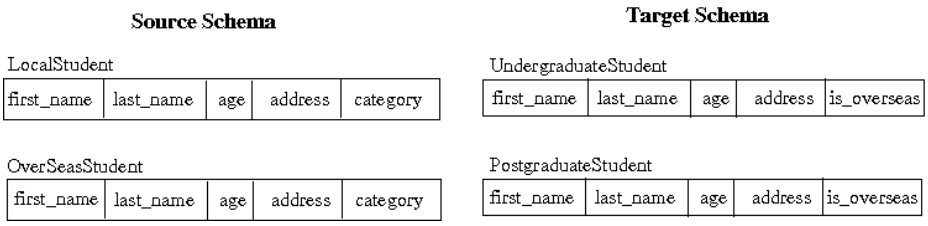

Fig. 2. Example of many-to-many horizontal partitioning

in the target schema. It specifies that the relation obtained by combining the relations in R_s is semantically equivalent to r_s. As for many-to-many relation correspondences, the relations in R_s can be combined by joining them in the case of vertical partitioning, or by using the union operator in the case of horizontal partitioning. A many-to-one relation correspondence can be defined by the tuple:

$$\langle R_s, r_t, type_s, JP_s, HPP_s, p_t AC \rangle$$

where, R_s is a set of relations in the source schema and r_t is a relation in the target schema. *type*$_s$*, JP*$_s$*, HPP*$_s$ and *AC* have the same meaning as in the definition of many-to-many relation correspondence. p_t is a selection predicate specifying which tuples can be used to populate r_t.

A one-to-many relation correspondence associates one relation from the source schema to multiple relations in the target schema. It is defined by the tuple:

$$\langle r_s, R_t, type_t, JP_t, HPP_t, AC \rangle$$

where, r_s is a relation in the source schema and R_t is a set of relations in the target schema. *type*$_t$*, JP*$_s$*, HPP*$_s$ and *AC* have the same meaning as in the definition of many-to-many relation correspondence. p_s is a selection predicate specifying which tuples can be used to populate r_s.

2.2 Attribute Correspondences

This family of correspondences associates attributes of relations that belong to the source schema with attributes that are semantically equivalent in the target schema. As for relation correspondences, attribute correspondences can be classified into four kinds depending on the number of attributes involved from the source and target schemas.

One-To-One Attribute Correspondence This correspondence associates one attribute in the source schema with one attribute that is semantically equivalent in the target schema. We define a one-to-one attribute correspondence by the tuple:

$$\langle r_s.att_s, r_t.att_t, f_{s \to t}, f_{t \to s} \rangle$$

where att_s and att_t are attributes of the relations r_s and r_t, respectively. $f_{s \to t}$ (resp. $f_{t \to s}$) is a function specifying how a value of att_t (resp. att_s) can be computed using a value of att_s (resp. att_s).

Many-To-Many Attribute Correspondence This correspondence associates multiple attributes Att_s in the source schema with multiple attributes Att_t in the target schema. It specifies that the combination of the attributes in Att_s is semantically equivalent to the combination of the attributes in Att_t. To specify the mapping between Att_s and Att_t we need information specifying how the values of the attributes in Att_s are to be combined to obtain the values of the attributes in Att_t, and vice versa. We therefore define many-to-many attribute correspondence by the tuple:

$$\langle Att_s, Att_t, f_{s\to t}, f_{t\to s}\rangle$$

where Att_s is a set of attributes in the source schema and Att_t is a set of attributes in the target schema. The correspondence also specifies two functions: $f_{s\to t}$ for computing the values of the attributes in Att_t given the values of the attributes in Att_s, and $f_{t\to s}$ for computing the values of the attributes in Att_s given the values of the attributes in Att_t.

Notice that the attributes in Att_s do not have to belong to the same relation, i.e. attributes from different relations can be involved. The same observation applies to the attributes in Att_t.

Many-to-one and one-to-many attribute correspondences These are specific cases of many-to-many attribute correspondences. A many-to-one attribute correspondence associates multiple attributes in the source schema with one attribute in the target schema, whereas a one-to-many attribute correspondence associates one attribute in the source schema with multiple attributes in the target schema. Both correspondences can be specified using the same tuple we used for defining many-to-many attribute correspondence, i.e., $\langle Att_s, Att_t, f_{s\to t}, f_{t\to s}\rangle$: Att_s (resp. Att_t) is a singleton in the case of one-to-many (resp. many-to-one) attribute correspondence. Below is an example of a many-to-one attribute correspondence that associates the attributes *fare* and *tax* in the source schema to the attribute *full_price* in the target schema.

$$\langle\{fare, tax\}, \{full\_price\}, price(), getPriceDetails()\rangle$$

price() is a function with the signature given below. Given a fare *f*, and a tax *t*, *price(f,t)* returns the full price, i.e., the sum of *f* and *t*. *getPriceDetails()* is a function with the signature given below. Given a full price *p*, *getPriceDetails(p)* returns a pair $\langle f, t\rangle$, where *f* is the fare and *t* is the tax. In the following, we use *getPriceDetails(p)*$\downarrow$*fare* and *getPriceDetails(p)*$\downarrow$*tax* to denote *f* and *t*, respectively.

$$price : domain(fare) \times domain(tax) \to domain(full\_prise)$$
$$getPriceDetails : domain(full\_price) \to domain(fare) \times domain(tax)$$

Schematic correspondences as defined above can be used, as we shall see in the next section, for automatically generating the mappings between source and target schemas. To avoid (or minimise) human intervention during this process, every relation in the target schema should be involved in zero or one schematic correspondence, and every attribute of a relation in the target schema should be involved in zero or one schematic

correspondence. Failure to meet the above criteria will lead to conflicts between correspondences, the resolution of which requires the intervention of a human user. The resolution of conflicts between correspondences is outside the scope of this paper. Therefore, we assume henceforth that the above conditions are satisfied by the schematic correspondences between source and target schemas.

3 Mapping Generation

Given the model for schematic correspondences just presented, we can construct an algorithm (shown in Figure 3) for automatically generating the mappings. The outcome of a mapping generation process are the views specifying how data described using a source schema could be used to populate a target schema. Before presenting the details of this algorithm, we outline the notation that we will use:

- The algorithm for mapping generation outputs a set of views. A view is a named query. For the purpose of this work, we define a view, *v*, by the pair $\langle name_v, query_v \rangle$, where $name_v$ is a string that identifies the view, and $query_v$ is a query specified using the relational algebra.
- Given a relation *r*, *r.name* denotes the name of *r* and *r.attributes* the list of attributes of *r*.
- Given a relation correspondence *corr*, *corr.source* denotes the set of relations in the source schema that are involved in *corr*, and *corr.target* denotes the set of relations in the target schema that are involved in *corr*.
- Given a schematic correspondence *corr*, be it a relation or an attribute correspondence, *getCorrCardinality(corr)* is a function that specifies whether *corr* is a one-to-one, many-to-one, one-to-many or many-to-many correspondence.
- Given a set of attributes correspondences *AC*, *getAttCorrepondence(AC,r.att)* returns the correspondence in which the attribute *r.att* is involved if such a correspondence exists, and returns null otherwise.
- Given a set of relations $R = \{r_1, \ldots, r_m\}$, and a conjunction of predicates *JP*, we use $\sqsupset\bowtie\sqsubset_{JP} R$ to denote the full outer join of the relations in *R* using *JP*, and $\biguplus R$ to denote the union of the relations in R using the outer union operator.
- *append(l,e)* is an operation that adds the element *e* to the end of the list *l*.

The mapping generation algorithm takes as input a target schema together with a set of relation correspondences. It iterates over the relations present in the target schema, retrieving for each of them the associated correspondences *(Figure 3, line 3)*. If a relation has no correspondence *(Figure 3, line 5)* then no view is generated for that relation. If, on the other hand, it is associated with more than one correspondence, then this conflict in correspondences is reported to the user *(Figure 3, line 7)*. Otherwise, the algorithm derives a view for the relation in question using the subroutine presented in Figure 4. This subroutine operates in two steps. Firstly, it specifies the relations taking part in the view and the way they are to be combined. If the correspondence *corr* used as input is a one-to-one or one-to-many relation correspondence then the view is assigned the source relation specified by the correspondence *(Figure 4, lines 5 and 6)*. If, on the other hand, *corr* is a many-to-many or many-to-one relation correspondence then the following cases are possible:

Algorithm GenerateMappings
inputs Sc_t: a target schema.
Corr: a set of schematic relation correspondences.
outputs *View*: a set of views.
begin
1 *View* = $\emptyset$
2 **FOR EACH** $r_t \in Sc_t.relations$ **DO**;
3 $Corr_{r_t}$ = *getCorrespondences(Corr,*r_t*)*
4 **IF** $|Corr_{r_t}| = 0$ **THEN**
5 *Signal('No view is generated for the relation'* $+ r_t$*)*
6 **IF** $|Corr_{r_t}| > 1$ **THEN**
7 *Signal('The relation'* $+ r_t +$ *' has more than one correspondence.')*
8 **IF** $|Corr_{r_t}| = 1$ **THEN**
9 *Let corr* be the only element of $Corr_{r_t}$
10 *v = DeriveView(*r_t*,corr)*
11 *View = View* $\cup \{v\}$
12 **RETURN** View
end

Fig. 3. Algorithm used for generating mappings

– *corr specifies a vertical partitioning between the source relations*: The view is assigned the relation obtained by applying an outer join to the source relations in *corr* using *corr.*JP_s *(Figure 4, lines 8 and 9)*. We use the outer join for combining the relations in the source schema instead of a natural join to avoid any loss of information. To further illustrate this, consider the example of vertical partitioning presented earlier in Figure 1. A department may not have any associated grants. Therefore, applying a natural join, instead of an outer join, for combining *Department* and *Grant* means losing available information about those departments when joining the two relations.
– *corr specifies a horizontal partitioning between the source relations*: The view is assigned the union of the source relations in *corr*. As pointed out earlier, we use the outer union operator instead of the union operator as the relations may not be union incompatible. In addition, the view is augmented with a selection specifying which tuples in the source relations are to be used for populating the target relation *(Figure 4, lines 11 and 12)*.

Secondly, the subroutine specifies the elements that constitute the columns of the view. To do this, it iterates over the attributes of the target relation r_t, retrieving for each attribute att_t the associated attribute correspondence among the correspondences in *corr.AC*. The following four cases are possible:

– att_t *has no associated correspondence among the set of attribute correspondences specified by corr*. In this case, att_t is considered as a missing attribute. If att_t is a

Algorithm DeriveView
inputs r_t: a relation in the target schema.
corr: a schematic relation correspondence that involves r_t.
outputs *view*: a view for populating r_t.
begin
1 *view = new View()*
2 *view.name = r_t.name*
3 *q = new Query()*
4 **IF** (*getCorrCardinality(corr)* $\in \{`one-to-one', `one-to-many'\}$) **THEN**
5 **Let** r_s be the source relation of *corr*
6 $q \leftarrow \sigma_{corr.p_s}\ r_s$
7 **IF** (*getCorrCardinality(corr)* $\in \{`many-to-many', `many-to-one'\}$) **THEN**
8 **IF** (*corr.type$_s$ = 'VerticalPartitioning'*) **THEN**
9 $q \leftarrow \sqsupset\bowtie\sqsubset_{corr.JP_s}\ corr.source$
10 **IF** (*corr.type$_s$ = 'HorizontalPartitioning'*) **THEN**
11 *p = getHPPredicate(corr,r_t)*
12 $q \leftarrow \biguplus corr.source$
13 *ViewColumns = new List()*
14 **FOR EACH** (*att$_t$* $\in$ *r_t.attributes)* **DO**
15 *corr$_{att}$ = getAttCorrepondence(corr.AC, att$_t$)*
16 **IF** (*corr$_{att}$ = null)* **THEN** // That is, if *att$_t$* is a missing attribute
17 **IF** (*att$_t$* **IS** *a key attribute*) **THEN**
18 *append(ViewColumns, generateKey())*;
19 **ELSE** *append(ViewColumns,* $`null'$*)*;
20 **IF** (*getCorrCardinality(corr$_{att}$)* = $`one-to-one'$) **THEN**
21 **Let** *att$_s$ be the source attribute in corr$_{att}$*
22 $append(ViewColumns, corr_{att}.f_{s\rightarrow t} + `('+att_s+`)')$
23 **IF** (*getCorrCardinality(corr$_{att}$)* $\neq `many-to-many'$) **THEN**
24 **Let** $att_{s_1}, \ldots, att_{s_m}$ *be the attributes in the source schema that are associated with* att_t
25 $append(ViewColumns, corr_{att}.f_{s\rightarrow t} + `('+att_{s_1}+`,'+\ldots+`,'+att_{s_m}+`)\downarrow'+att_t)$
26 $q \leftarrow \Pi_{ViewColumns}\ q$
27 *view.query = q*
28 **RETURN** *view*
end

Fig. 4. Algorithm for deriving the view for populating a given relation in the target schema

key attribute then the call to function *generateKeyValue()* is appended to the list of columns of the view, otherwise, the value *null* is appended to the list of columns of the view *(Figure 4, lines 16-19). generateKeyValue()* is a function for generating unique identifiers: it is used to enable the mapping in the absence of attribute cor-

respondences that specify the values of key attributes of the relation in the target schema.
- *att_t is involved in a one-to-one attribute correspondence*. In this case, the corresponding source attribute is appended to the list of columns of the view *(Figure 4, line 22)*.
- *att_t is involved in a many-to-many, many-to-one or one-to-many attribute correspondence*. In this case, a call to the function specified by the attribute correspondence with the source attribute(s) used as input, is appended to the list of columns of the view *(Figure 4, lines 24 and 25)*.

The next section describes an extended, realistic example of automatic mapping generation in the area of proteomics data.

4 Using Schematic Correspondences for Deriving Mappings Between Proteomics Data Sources

Proteomics is the study of the set of proteins produced by an organism with the aim of understanding the behaviour of these proteins under varying environments and conditions. There is a growing number of resources that offer a range of approaches for the capture, storage and dissemination of proteome experimental data. While the existence of such resources opens up possibilities for the proteomics community, the diversity of data models creates schema integration challenges. In this respect, we have integrated, in previous work [21], the schemas of four major proteomics data sources, namely PedroDB[1], GPMDB[2], PepSeeker[3] and Pride[4]. To do this, we manually specified the mappings between the schemas of these sources and the integration schema using the Automed toolkit [1]. In this section, we show that these mappings can be automatically generated if schematic correspondences of the form presented in this paper are used as input. Due to space limitation, we will present one example of a one-to-one relation correspondence and one example of a many-to-many relation correspondence. Three relations from the integration schema are involved in these examples, namely *IntProtein*, *IntProteinHit* and *IntPeak* (see Figure 5). The *IntProtein* relation describes a protein using an accession number, the name of the gene, its synonyms, the organism in which the protein is to be found, a textual description, the protein amino-acid sequence, *in vivo* modification and the reading frame *rf*. The *IntProteinHit* relation is used to store information about the protein against which all or some of the peptides have been aligned, and links to some information about the protein itself. A protein is (experimentally) identified using a mass spectrometer which produces a spectrum composed of a list of *peaks*. A peak is described by the *IntPeak* relation using the mass-to-charge ratio of the protein ions, *m-to-z*, the peak height, *abundance* and the isotopic pattern around the main peak, *multiplicity*[5].

[1] http://pedro.cs.manchester.ac.uk
[2] http://gpmdb.thegpm.org
[3] http://nwsr.smith.man.ac.uk/pepseeker
[4] http://www.ebi.ac.uk/pride
[5] For further information about the integration schema, the reader is referred to [21].

Integration Schema

IntPeak

pk_id	m_to_z	abundance	multiplicity

IntProtein

p_id	accession_number	gene_name	synonyms	organism	description	sequence	modifications	rf

IntProteinHit

hit_id	all_peptide_matched	expect	score	threshold	p_id

PepSeeker Schema

SearchMasses

sid	products	title	mass_min	mass_max	int_min	int_max	num_vals

GPMDB Schema

Protein

proid	expct	uid	pida	pidb	proseqid

ProSeq

proseqid	seq	label	label_aux	ref

Fig. 5. Examples of relations from the integration schema, GPMDB schema and PepSeeker schema

Example 1 The correspondence $corr_1$ presented below is an example of a one-to-one relation correspondence that associates the *SearchMasses* relation in the schema of GPMDB with the *IntPeak* relation in the integration schema. It specifies that the attributes *pid* and *m-to-z* of the *IntPeak* relation are associated with the attributes *sid* and *products* of the *SearchMasses* relation, respectively, whereas *abundance* and *multiplicity* are missing attributes in the sense that they are not involved in any attribute correspondence.

$$
\begin{aligned}
\mathtt{corr_1.source} &= \{\mathtt{SearchMasses}\} \\
\mathtt{corr_1.target} &= \{\mathtt{IntPeak}\} \\
\mathtt{corr_1.AC} &= \{\langle \mathtt{sid}, \mathtt{pk\_id}, \mathtt{false}\rangle,\ \langle \mathtt{products}, \mathtt{m-to-z}, \mathtt{false}\rangle\} \\
\mathtt{corr_1.p_s} &= \mathtt{true} \\
\mathtt{corr_1.p_t} &= \mathtt{true}
\end{aligned}
$$

Using the schematic correspondence $corr_1$ together with the *IntPeak* as input to the algorithm presented in Section 3, we obtained the view $v_{IntPeak}$ presented below for populating the *IntPeak* relation of the integration schema. Notice that the missing attributes *abundance* and *multiplicity* are assigned the null value.

$$\mathtt{v_{IntPeak}.query} \leftarrow \Pi_{\mathtt{sid,products,null,null}}\mathtt{SearchMasses}$$

Example 2 The schematic correspondence $corr_2$ presented below is an example of a many-to-many relation correspondence that associates the relations *Protein* and *ProSeq*

in the schema of GPMDB with the relations *IntProtein* and *IntProteinHit* in the integration schema. It specifies that the relations *Protein* and *ProSeq* should be combined by joining them using the attribute comparison predicate *'Protein.proseqid = ProSeq.proseqid'*, and that the relations *IntProtein* and *IntProteinHit* should be combined by joining them using the attribute comparison predicate *'IntProtein.p_id = IntProteinHit.p_id'*.

$\texttt{corr}_2.\texttt{source} = \{\texttt{Protein}, \texttt{ProSeq}\}$
$\texttt{corr}_2.\texttt{target} = \{\texttt{IntProtein}, \texttt{IntProteinHit}\}$
$\texttt{corr}_2.\texttt{type}_\texttt{s} = \text{`}\texttt{VerticalPartitioning}'$
$\texttt{corr}_2.\texttt{AC} = \{\langle \texttt{proid}, \texttt{p\_id}, \texttt{false}\rangle,\ \langle \texttt{proseqid}, \texttt{hit\_id}, \texttt{false}\rangle,\ \langle \texttt{expct}, \texttt{expect}, \texttt{false}\rangle,$
$\langle \texttt{seq}, \texttt{sequence}, \texttt{false}\rangle,\ \langle \texttt{label}, \texttt{accession\_number}, \texttt{false}\rangle,\ \langle \texttt{rf}, \texttt{rf}, \texttt{false}\rangle\}$
$\texttt{corr}_2.\texttt{JP}_\texttt{s} = \text{`}\texttt{Protein.proseqid} = \texttt{ProSeq.proseqid}'$
$\texttt{corr}_2.\texttt{JP}_\texttt{t} = \text{`}\texttt{IntProtein.p\_id} = \texttt{IntProteinHit.p\_id}'$

As with $corr_1$, we used the schematic correspondence $corr_2$ as input to the algorithm *DeriveView* and obtained the views $v_{IntProtein}$ and $v_{IntProteinHit}$ presented below for populating the relations *IntProtein* and *IntProteinHit*, respectively.

$\texttt{v}_{\texttt{IntProtein}}.\texttt{query} \leftarrow \Pi_{\texttt{proid,label,null,null,null,null,seq,null,rf}}(\texttt{Protein} \sqsupset\bowtie\sqsubset_{\texttt{proseqid}} \texttt{ProSeq})$
$\texttt{v}_{\texttt{IntProteinHit}}.\texttt{query} \leftarrow \Pi_{\texttt{proseqid,null,expect,null,null,proid}}(\texttt{Protein} \sqsupset\bowtie\sqsubset_{\texttt{proseqid}} \texttt{ProSeq})$

5 Related Work

A number of authors have proposed models for specifying schematic correspondences between heterogeneous schemas (e.g., in [19, 20]), in most of which a correspondence is defined as an association between attributes of two schemas, i.e., one-to-one or many-to-many attribute correspondences. Schematic correspondences of this form do not provide the necessary information for deriving the mappings between schemas. This is the case, for example of mappings that involve more than one relation in the source or target schemas. To overcome this problem, the authors in Clio [20] exploit functional dependencies and referential integrity constraints between the relations in the source and target schemas. For example, they rely on referential integrity constraints to specify the join predicates for joining relations in the case of vertical partitioning. This is a partial solution in that it does not always lead to the desired schema mapping. For example, referential integrity constraints that can be used for joining the relations involved in a mapping may not exist. And, even if they did exist, they may not encode the join condition that meets the semantics of the desired schema mapping.

Richer models for schematic correspondences were proposed by Pottinger *et al.* [16] and Quix *et al.* [18]. A schematic correspondence in these models can be used to model many-to-many relation and attribute correspondences, for instance. Yet, the information they provide is not sufficient, and human intervention may be needed for deriving the mappings. For example, they do not distinguish between horizontal and vertical partitioning, and they do not specify the condition that can be used for combining relations.

In other proposals, such as those by Magnani *et al.* [14] and Hakimpour [7] *et al.*, correspondences are defined as extentional constraints between the elements of two

schemas. For example, correspondences of this form can be used to specify that the extent of a relation is covered by the extent of another relation. While correspondences of this form are useful for merging schemas, as shown by the authors of these proposals, they do not provide the information necessary for deriving schema mappings.

Kedad *et al.* [10] proposed a solution similar to that proposed in Clio, for generating views that populate the elements of an integration schema. In doing so, they use as input extentional constraints such as those defined by Magnani *et al.* [14] together with functional dependencies and referential integrity constraints specified within the schemas of the sources. Because the correspondences used as input do not provide information that is sufficient for deriving the mappings that encode the semantics desired by the user, the solution proposed by the authors attempts to generate multiple alternatives for populating a given relation in the integration schema. The user is then responsible for choosing the view that implements the desired mapping. In our approach, we use input correspondences that provide information that is sufficient for determining and deriving the desired schema mappings.

6 Conclusions

We presented in this paper a model for schematic correspondences that extends and enriches those proposed by Kim *et al.* [12]. The model proposed outperforms existing correspondence models in that it covers the information necessary to cater for automatic derivation of schema mappings. Note that the model of schematic correspondences presented in this paper does not handle all the kinds of correspondences between schemas identified by Kim *et al.*, e.g., we did not consider schematic correspondences associating relations that are semantically not equivalent, i.e., that refer to different semantic domains. Instead, we focused on refining the correspondences that are useful for generating schema mappings. In this respect, we presented an algorithm for automatically generating the views implementing schema mappings between two schemas.

References

1. Michael Boyd, Sasivimol Kittivoravitkul, Charalambos Lazanitis, Peter McBrien, and Nikos Rizopoulos. Automed: A bav data integration system for heterogeneous data sources. In *CAiSE*, pages 82–97. Springer, 2004.
2. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information integration: Conceptual modeling and reasoning support. In *CoopIS*, pages 280–291. IEEE Computer Society, 1998.
3. E. F. Codd. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4(4):397–434, 1979.
4. AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, pages 509–520, 2001.
5. AnHai Doan and Alon Y. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 26(1):83–94, 2005.
6. Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

7. Farshad Hakimpour and Andreas Geppert. Global schema generation using formal ontologies. In *21st International Conference on Conceptual Modeling*, pages 307–321. Springer, 2002.
8. Alon Y. Halevy, Michael J. Franklin, and David Maier. Principles of dataspace systems. In Stijn Vansummeren, editor, *PODS*, pages 1–9. ACM, 2006.
9. Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Data integration: The teenage years. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *VLDB*, pages 9–16. ACM, 2006.
10. Zoubida Kedad and Mokrane Bouzeghoub. Discovering view expressions from a multi-source information system. In *CoopIS*, pages 57–68. IEEE Computer Society, 1999.
11. Won Kim, Injun Choi, Sunit K. Gala, and Mark Scheevel. On resolving schematic heterogeneity in multidatabase systems. In *Modern Database Systems*, pages 521–550. 1995.
12. Won Kim and Jungyun Seo. Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer*, 24(12):12–18, 1991.
13. Maurizio Lenzerini. Data integration: A theoretical perspective. In Lucian Popa, editor, *PODS*, pages 233–246. ACM, 2002.
14. Matteo Magnani, Nikos Rizopoulos, Peter McBrien, and Danilo Montesi. Schema integration based on uncertain semantic mappings. In *24th International Conference on Conceptual Modeling*, pages 31–46. Springer, 2005.
15. Robert McCann, Bedoor K. AlShebli, Quoc Le, Hoa Nguyen, Long Vu, and AnHai Doan. Mapping maintenance for data integration systems. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 1018–1030. ACM, 2005.
16. Rachel Pottinger and Philip A. Bernstein. Creating a mediated schema based on initial correspondences. *IEEE Data Eng. Bull.*, 25(3):26–31, 2002.
17. Rachel Pottinger and Philip A. Bernstein. Merging models based on given correspondences. In *VLDB*, pages 826–873, 2003.
18. Christoph Quix, David Kensche, and Xiang Li 0002. Generic schema merging. In *9th International Conference on Advanced Information Systems Engineering*, pages 127–141. Springer, 2007.
19. Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
20. Ling-Ling Yan, Renée J. Miller, Laura M. Haas, and Ronald Fagin. Data-driven understanding and refinement of schema mappings. In *SIGMOD Conference*, pages 485–496, 2001.
21. Lucas Zamboulis, Hao Fan, Khalid Belhajjame, Jennifer A. Siepen, Andrew C. Jones, Nigel J. Martin, Alexandra Poulovassilis, Simon J. Hubbard, Suzanne M. Embury, and Norman W. Paton. Data access and integration in the ispider proteomics grid. In *DILS*, pages 3–18. Springer, 2006.